IONDV Documentation

Release latest

IONDV LLC and The IONDV Community

Feb 19, 2021

Оглавление

ГЛАВА 1

1. О IONDV. Framework

IONDV. Framework - это опенсорный фреймворк на node.js для разработки учетных приложений на основе метаданных в формате JSON/YAML и отдельных функциональных модулей. Визуальный редактор IONDV. Studio позволяет создавать приложения по технологии “no code” и собирать приложение с веб-сервисами REST-API (модуль rest). Ключевой модуль registry является универсальным средством представления и редактирования данных, обработки их по бизнес-процессам.

На видео технология разработки и сборки приложения

1.1 Бесплатные demo-версии

Посмотрите наши демо уже сейчас:

- Studio - специализированная IDE, созданная как приложение iondv для визуальной (no code) разработки приложений на IONDV. Framework. Инструкция и видео по созданию приложения с помощью IONDV. Studio. См. на GitHub.
- Telecom - приложение по организации учета, хранения и отображения данных о наличии услуг связи (интернет, сотовая связь, телевидение, почта и др.) в населенных пукнтах региона. См. на GitHub
- DNT - приложение для разработки и тестирования функциональности фреймворка, в котором каждая учетная сущность отражает тим метаданных, например класс “строка”, или класс “коллекция”. Это позволяет изучать возможности фреймворка через приложение. См. на GitHub.
- War Archive - это программное решение на основе IONDV. Framework, реализованное для действующего проекта “Вспомнить каждого”, цель которого оцифровать архивные документы, внести информацию в базу и обеспечить к ним свободный доступ. См. на GitHub.
- Project Management - приложение по организации проектной деятельности региональных ОГВ , целью которой является контроль результатов, соблюдение и сокращение сроков их достижения, эффективное использование временных, человеческих и финансовых ресурсов, принятие своевременных и обоснованных управленческих решений. См. на GitHub

- CRM - это программное решение, реализованное для организации регистрации, учета, хранения и отображения бизнес-данных (входящие заявки, звонки, посетители, продукция, услуги). См. на GitHub

Логин для доступа - demo, пароль - ion-demo. Регистрация не требуется.

1.2 Типовые приложения

IONDV. Framework - конструктор веб-приложений широкой специфики, так как преметная область определяется структурой метаданных, описывающих поведение приложение. Например, можно создать приложения:

- CRM - управление отношениями с клиентами;
- учет и управление ресурсами предприятия;
- автоматизация бизнес-процессов организации и документооборота;
- сбор и хранение любых данных, например метрик оборудования (IoT);
- представление данных в виде порталов;
- REST-API для SPA приложений;
- REST-API и бэкграунд для мобильных приложений;

1.3 Структура фреймворка

Схема приложения на основе фреймворка: core + metadata + modules = application

На рисунке отражены компоненты:

- ION Core - это ядро приложения в виде IONDV. фреймворка;
- meta class, meta view, meta navigation, meta workflow, meta security - это функциональные метаданные приложения - структуры, представления, навигации, бизнес-процессов и безопасности соответственно;
- registry module - отражает подключаемые функциональные модули, например модуль registry для просмотра и редактирования данных;

Чуть ниже представлены дополнительные типы меты и модули. Они представляют собой дополнительную функциональность и применяются в соответствии со спецификой приложения. Зависимости приложения представлены в файле package.json.

Приложение - это метаописание его поведения в файлах формата JSON (YAML) + функциональный код + HTML шаблоны, расширяющие типовую функциональность -> с ним удобно работать через репозиторий версий git. Посмотрите примеры на Github .

Подробнее о функциональных возможностях фреймворка IONDV. Framework и его модулей в документации.

1.4 Функциональные возможности

IONDV. Framework обеспечивает реализацию следующей функциональности:

- обеспечение трансляции описательных метаданных в структуру хранения данных в СУБД;

- обеспечение функциональности работы с различными СУБД (ORM технологию);
- обеспечение авторизации в системе с различными политиками, по умолчанию oath2, с открытым конфигурируемым API для подключения авторизационных модулей библиотеки passport обеспечивает до 500 различных политик авторизации;
- обеспечение безопасности доступа к данным – статической к типам данных, к навигации, к этапам бизнес-процессов, к действиям на форме; динамической – через условия в данных, которым должен соответствовать профиль текущего пользователя (принадлежность к подразделению или организации указанной в объекте, группе или другим условиям); через url; обеспечение исключения в авторизации и безопасности по url или для специального пользователя;
- подключение модулей, обеспечивающих дополнительную функциональность и реализуемую через доступ к интерфейсам (API) ядра;
- обеспечение импорта, экспорта данных в системе, метаданных, безопасности из файлов;
- обеспечение взаимодействия с файловой системой для хранения данных, в том числе с внешними файловыми хранилищами, такими как nextcloud;
- расчет значения с формулами и кэширование этих данных;
- обеспечение жадной загрузки данных и их фильтрации в связанных коллекциях;
- кэширование запросов и сессий в memcached, redis;
- выполнение задач по расписанию;
- уведомление пользователей по событиям.

1.5 Документация

Документация по платформе IONDV.Framework доступна на двух языках - русский и english.

1.6 Ссылки

Ссылки на дополнительную информацию по разработке приложений с использованием IONDV. Framework.

- Документация
- Web-сайт
- Обучающие видеоролики на Youtube
- Обратная связь на Facebook

License Contact us English

ГЛАВА 2

2. Начало работы

2.1 Как развернуть

2.1.1 Шаг 1 Установка окружения

Окружение - это список программ необходимых для запуска.

Окружение, необходимое для запуска платформы IONDV. Framework с приложениями:

- СУБД MongoDb версии 3.6.
- Среда разработки Node.js версии 10.x.x.

СУБД

1. Необходимо установить СУБД MongoDB. Проверенная версия 3.6.9 и 4.0.0.
2. Далее создаем папку data на диске C: и в ней подпапку db.
3. Для запуска базы данных переходим в папку расположения MongoDb, далее в папку server\bin и запускаем файл mongod.exe. Если есть необходимость использовать каталог с БД отличный от c:\data\db, тогда файл mongod.exe необходимо запустить с параметром --dbpath после которого указать путь к каталогу.

Среда выполнения Node.js

Node.js - является средой, в которой осуществляется выполнение компонентов.

1. Необходимо установить среду разработки Node.js. Проверенная версия node.JS 10.14.2.
2. По умолчанию установщик сам прописывает пути к Node.js в PATH, а также устанавливает менеджер пакетов npm.

Установка глобальных зависимостей

Установливайте глобальные зависимости в командной строке cmd.exe, запущенной от имени администратора, после установки node.js.

NB: команда node -v - показывает версию node.js.

Установка среды сборки под Windows

1. Установите глобально пакет node-gyp командой npm install -g node-gyp необходимый для сборки различных библиотек.
2. Для работы библиотеки под операционной системой семейства Windows дополнительно необходимо установить пакет windows-build-tools - npm install -g --production windows-build-tools.

Пакет сборщика проектов

Для организации тестирования и сборки дистрибутивов при разработке используется Gulp. Установите глобально командой npm install -g gulp@4.0. 4.0 - поддерживаемая версия Gulp.

Установщик фронтенд библиотек

Для установки библиотек фронтенд используется bower. Установите глобально командой npm install -g bower.

2.1.2 Шаг 2 Установка ядра, модулей и приложения

Клонирование приложения и его компонентов

NB: пути не должны содержать русских букв и пробелов. Мы советуем размещать приложение в c:\workspace.

Рассматриваем формирование проекта с модулями на примере приложения develop-and-test.

1. Находим приложение в репозитории github. Набираем искомое приложение develop-and-test в поле поиска и переходим на него.
2. Переходим в репозиторий файлов на ветку версии.
3. Открываем файл package.json в котором смотрим зависимости.

```
"engines": {
  "ion": "3.0.0"
},
"ionModulesDependencies": {
  "registry": "3.0.0",
  "geomap": "1.5.0",
  "portal": "1.4.0",
  "report": "2.0.0",
  "ionadmin": "2.0.0",
  "dashboard": "1.1.0",
  "soap": "1.1.2"
```

(continues on next page)

(continued from previous page)

```
},
"ionMetaDependencies": {
  "viewlib": "0.9.1"
}
```

1. engines": "ion": 3.0.0 - версия ядра 3.0.0.
2. ionModulesDependencies - список модулей и их версий.
3. ionMetaDependencies - список других метаданных, необходимых для проекта, в данном случае исключение viewlib - библиотека представлений.

NB: для переключения на tag номера версии - смотрите версии в файле package.json.

Получение репозитория ядра

Ядро находится в репозитории framework. На главной странице есть поле с путем к репозиторию.

1. Запустите командную строку от имени администратора.
2. Скопируйте адрес репозитория, перейдите в папку workspace командой cd c:\workspace и выполните команду git clone https://github.com/iondv/framework. Эта команда создает папку framework и в неё клонирует репозиторий.

Получение модулей

1. Переходим в папку модулей командой cd framework\modules.
2. Для каждого модуля из списка package.json в свойстве ionModulesDependencies - находим репозиторий модуля среди группы модулей https://github.com/iondv/ION-MODULES.
3. Клонируйте все модули из списка ionModulesDependencies командой git clone https://github.com/iondv/registry.
4. Перейдите в папку установленного модуля, переключитесь на tag номера версии git checkout tags/v1.27.1. Например 1.27.1 - это номер версии модуля registry.
5. Повторите для всех модулей.

Получение приложения

1. Переходим в папку приложения. Если вы находитесь в папке модулей выполните команду cd ..\applications.
2. Далее вернитесь на страницу репозитория develop-and-test, скопируйте путь и клонируйте его командой git clone https://github.com/iondv/develop-and-test.
3. Перейдите в папку установленного приложения, переключитесь на tag номера версии git checkout tags/v1.17.0.
4. Установка зависимостей в ionMetaDependencies осуществляется в папку applications, для установки необходимо убедиться, что находитесь в папке приложений. Клонируем приложения из списка в параметре ionMetaDependencies. Для приложения viewlib клонируйте командой git clone https://github.com/iondv/viewlib.
5. Перейдите в папку установленного приложения, переключитесь на tag номера версии git checkout tags/v0.9.1. Повторите для каждого приложения.

6. Приложение скомпоновано.

NB: мы советуем создать для него проект в IDE, например, в Visual Studio Code, и в нём создать конфигурациионный файл.

Конфигурациионный файл

Конфигурационный файл служит для задания основных параметров окружения приложения и настройки дополнительных параметров запуска.

1. Создайте конфигурационный файл setup с расширением ini в папке config.
2. Открываем файл в редакторе и вставляем содержимое.

```
auth.denyTop=false
auth.registration=false
auth.exclude[]=/files/**
auth.exclude[]=/images/**
db.uri=mongodb://127.0.0.1:27017/iondv-dnt-db
server.ports[]=8888
module.default=registry
fs.storageRoot=./files
fs.urlBase=/files
```

Самый главный параметр - db.uri=mongodb://127.0.0.1:27017/db. Он указывает на название базы которую мы будем использовать для приложения. База данных будет создана автоматически.

2.1.3 Шаг 3 Сборка и запуск приложения

Для всех дальнейших команд, необходимо запустить командную строку от имени администратора.

Перейдите в папку приложения cd c:\workspace\framework и задайте переменную окружения NODE_PATH равной пути к приложению. Для Windows команда - set NODE_PATH=c:\workspace\framework, для Linux - export NODE_PATH=/workspace/framework.

Сборка приложения

Сборка приложения обеспечивает установку всех библиотек, импорт данных в базу данных и подготовку приложения для запуска.

1. При первом запуске необходимо выполнить npm install - она поставит ключевые зависимости, в том числе локально сборщик gulp. Убедитесь, что версия Gulp - 4.0. Эта команда ставит все библиотеки из свойства dependencies файла package.json ядра.
2. После этого, а также все последующие разы выполняйте команду сборки приложения gulp assemble.

   NB: Убедитесь, что стоит переменная окружения NODE_PATH, запущена база MongoDB, Gulp установлен глобально и локально и его версия 4.0.
3. Перед непосредственным запуском приложения необходимо добавить базового пользователя для входа. Откройте программу Mongo Compass и в базе данных найдите таблицу ion-user. Удалите все записи, которые увидите там. Далее вернитесь в консоль и выполните указанные ниже команды. Добавьте пользователя admin с паролем 123 командой node bin/adduser.js --name admin --pwd 123. Добавьте пользователю права администратора командой node bin/acl.js --u admin@local --role admin --p full.

Запуск приложения с использованием скриптов

В папке “bin” содержатся скрипты запуска приложения, реализованного на IONDV. Framework, такие как:

- acl.js
- export.js
- import.js и import-data.js
- setup.js
- www.js

NB. Запускаются локально из папки framework, шаблоны команд указаны в разделах с описанием назначения скрипта.

Подробнее о скриптах запуска приложения

Запуск приложения

После окончания сборки можно запускать приложение. Убедитесь, что стоит переменная окружения NODE_PATH. Без этого система выдаст ошибку - об отсутствии компонентов.

Запуск системы осуществляется командой npm start, альтернативой является запуск node bin/www.

После запуска системы, откройте браузер с адресом http://localhost:8888 и авторизуйтесь в приложении, где 8888 - порт указанный в параметре server.ports конфигурации запуска.

2.1.4 Скрипты запуска приложения

В папке “bin” содержатся скрипты запуска приложения, реализованного на IONDV. Framework, такие как:

- acl.js
- export.js
- import.js и import-data.js
- setup.js

NB. Запускаются локально из папки framework, шаблоны команд указаны в разделах с описанием назначения скрипта.

acl.js

Шаблон команды запуска: node %NODE_PATH%\bin\acl.js --d %NODE_PATH%\applications\%IONAPP%\acl где NODE_PATH - путь к директории платформы, %IONAPP% - наименование приложения.

Добавляет права на объекты системы, указанные в папке acl.

Так же доступны команды для создания роли и прав для нее в системе, в случае если в приложении отсутствуют настройки в папке acl:

- Настройка логина и пароль - node bin/adduser.js --name admin --pwd 123

- Настройка доступа - node bin/acl.js --u admin@local --role admin --p full
- Права на генерацию токена для сервиса rest/token - node bin/acl.js --role admin --p USE --res ws:::gen-ws-token

По такому же принципу можно задавать пользователей и права на отдельные ресурсы системы

export.js

Шаблон команды запуска: node bin/export --ns %IONAPP%, где NODE_PATH - путь к директории платформы, %IONAPP% - наименование приложения (namespace).

Выполняет экспорт данных и меты из приложения, которое в данный момент собрано и запущено локально. Файлы экспорта формируются в структуру папок, которые расположены на одном уровне с директорией платформы в папке out.

import.js и import-data.js

При импорте меты, по умолчанию, импорт данных не выполняется. Поэтому, для импорта меты вместе с данными вызываем команду:

node bin/import.js --src %NODE_PATH%/applications/%IONAPP% --with-data --ns %IONAPP% где, NODE_PATH - путь к директории платформы, %IONAPP% - наименование приложения (namespace).

А для импорта непосредственно данных вызываем:

node %NODE_PATH%/bin/import-data.js --src %NODE_PATH%/applications/%IONAPP%/data --ns %IONAPP%

При этом, если импортируем мету и данные, указываем директорию приложения (при этом данные для импорта будут искаться в поддиректории data), если импортируем данные, нужно указывать непосредственно директорию с данными.

setup.js

Шаблон команды запуска: node %NODE_PATH%\bin\setup %IONAPP% где, NODE_PATH - путь к директории платформы, %IONAPP% - наименование приложения.

Выполняет установку приложения, запускает скрипт развертывания приложения, что включает в себя импорт и запись в базу данных меты модулей приложения.

В данном разделе - о том, как развернуть систему. Для развертывания системы необходимо выполнить несколько шагов.

2.1.5 Шаги

1. Установка окружения
2. Установка ядра, модулей и приложения
3. Сборка и запуск приложения

2.2 Быстрый запуск

2.2.1 IONDV. Studio

IONDV. Studio является приложением IONDV. Framework. Может использоваться как отдельное приложение node.js или как десктоп-приложение. Развернуть приложение можно с помощью демо-версии студии. Регистрация и учетная запись не требуются.

Описание

IONDV. Studio - приложение для визуального создания и редактирования метаданных (таких как классы, навигация, представления, бизнес-процессы, портальные формы), которые разворачиваются как веб-приложение IONDV. Framework.

Как запустить приложение в Студии

Смотрите видео, о том, как развернуть в IONDV. Studio уже готовое приложение из репозитория.

- Выберите пример готового приложения с github. Например, Nutrition-Tickets.
- Скачайте архивный файл приложения и откройте его в приложении.
- Нажмите кнопку воспроизведения в правом верхнем углу экрана, чтобы запустить приложение.
- Приложение начнет развертывание. Это занимает примерно 80 секунд.
- Получите ссылку на приложение и нажмите на нее.
- Войдите в приложение, используя логин demo и пароль ion-demo.

Приложение запущено! Приложение в демо-Студии хранится в течение 1-го часа, Вы можете ознакомиться с технологией и попробовать самостоятельно внести изменения в структуру приложение и увидеть результат. После этого приложение будет удалено с сервера.

Как создать приложение в студии

В Студии также можно создавать приложения. Смотрите видео о создании простого приложения IONDV. Nutrition-tickets в IONDV. Studio. Инструкция доступна в репозитории IONDV. Nutrition-Tickets.

- Нажмите на + чтобы приступить к созданию приложения. В всплывающем окне заполняем обязательные поля. Вкладка приложения, которое вы создали, появляется в левом верхнем углу и управляется по типу браузера.
- Появляется боковое меню - это рабочая панель приложения. Раздел Классы нужен для создания Классов и Атрибутов.
- Создание приложения начинается с Класса. Нажимаем на Класс и в рабочем пространстве нажимаем +Класс. В всплывающем окне заполняем обязательные поля. Описание полей тут.
- У нас появился Класс и автоматически создался Атрибут ID.
- Когда Класс выделен, можно добавить к нему Атрибуты, нажав на +Атрибут. Описание свойств и типов Атрибута тут.
- Когда у вас есть минимум 2 Класса, можно настроить между ними связи. Это выполняется через настройку типа данных при создании атрибута Класса. Основные типы - Коллекция и Ссылка. Заданный тип данных у Атрибута отразится в рабочей области в виде связующей линии.

В разделе Класс появятся созданные вами Классы и их Атрибуты - это называется дерево проекта, по которому будет легко ориентироваться, когда количество Классов приложения увеличится. Это - базовые пункты для создания приложения.

Приложение в IONDV. Studio

Для того чтобы разработать приложение в Студии, нужно предварительно запустить ее одним из следующих способов:

- Используйте демо-версию студии.
- Запустите локально как приложение IONDV. Framework, получив исходные коды на github. После чего соберите и разверните его согласно инструкции приложений Framework.
- Откройте в браузере по ссылке http://localhost:8888.
- Запустите локального как отдельное приложение node.js согласно инструкции ниже.
- Запустите как десктоп приложение, согласно инструкции ниже.
- Запустите в docker-конейтнере выполнив docker run -d -p 8888:8888 –name studio iondv/studio.

Откройте в браузере по ссылке http://localhost:8888.

Затем:

- В Студии разработайте Ваше приложение, путем создания классов, навигации.
- Обратите внимание, что данные хранятся в локальном репозитории браузера. Экспортируйте приложение как zip-архив.
- Скачайте последнюю версию IONDV. Framework и модуля IONDV. Registry module - получить их можно с репозиториев GitHub Framework и Registry.
- Следуйте типовой инструкции развертывания приложения из git, за исключением приложения - вместо приложения разверните в папку applictions ваш архив с приложением.
- Далее необходимо собрать и развернуть приложение, согласно инструкции IONDV. Framework.

Варианты использования Студии

Отдельное приложение node.js

Преимуществами использования отдельного приложения является отсутствие необходимости в базе данных и в IONDV. Framework.

- Выполните команду git clone https://github.com/iondv/studio.git. Имените локальную дирректорию на studio.
- Выполните команду npm install для установки всех необходимых зависимостей, включая локальное приложение сборки gulp.
- Пожалуйста проверьте, что глобально установлен Gulp версии 4.0.
- Выполните команду gulp build для сборки приложения.
- Запустите приложение командой npm start или node www (node standalone для запуска приложения как [standalone](/readme-standalone_ru.md).)
- Перейдите в браузере по адресу http://localhost:8888.

Десктоп приложение Студии (node-webkit)

Перед формированием десктоп приложения Студии, соберите Отдельное приложение node.js

Запуск новой студии на локальном сервере node-webkit

1. Скачайте последнюю NORMAL версию node-webkit с сайта https://nwjs.io/.
2. Распакуйте содержимое архива в любую удобную папку.
3. Воспользуйтесь одним из имеющихся способов для того, чтобы соеденить приложение и node-webkit.

Примеры описаны в статье https://github.com/nwjs/nw.js/wiki/How-to-package-and-distribute-your-apps в пунктах 2a и 2b.

Более удобным является вариант воспользоваться пакетом nw-builder:https://github.com/nwjs-community/nw-builder. Пример команды: nwbuild ./studio -p win64 -v 0.34.0 -o ./destination. Стоит отметить, nw-builder сам скачает необходимую версию node-webkit.

В результате вы получите ваше приложение в папке с dll, которые использует nwjs. Запустить приложение можно с помощью nw.exe файла (название может отличаться).

Формирование одного единственного исполняемого файла

1. Скачайте Enigma virtual box с сайта https://enigmaprotector.com/en/downloads.html, установите и запустите
2. Занесите в первое поле путь к исполняемому файлу вашего приложения. (Можно выбрать)
3. Занесите во второе поле путь сохранения исполняемого файла.
4. Занесите в поле Files ВСЕ файлы и папки из директории вашего приложения кроме исполняемого файла .exe.
5. В меню Files options, поставьте галочку на путкте Compress.
6. Нажмите Process и дождитесь результата.

External App Tracker

Вся настройка в deploy.json -> globals -> externalAppTracker

```
{
  "items": [{
    "name": "dnt",
    "title": "Develop and test",
    "url": "https://github.com/iondv/develop-and-test/archive/master.zip"
  }, {
    "name": "crm-en",
    "title": "CRM EN",
    "url": "https://github.com/iondv/crm-en/archive/master.zip",
    "language": "en"
  }, {
    "name": "crm-ru",
    "title": "CRM RU",
```

(continues on next page)

(continued from previous page)

```
    "url": "https://github.com/iondv/crm-ru/archive/master.zip",
    "language": "ru"
  }],
  "front": "/themes/portal/static/archives/",
  "storage": "applications/studio/themes/portal/static/archives/",
  "tempZip": "applications/studio/temp.zip",
  "enableUpdate": false,
  "updateInterval": 86400
}
```

- item.name - задает имя файла при сохранении архива
- item.title - отображается на клиенте при выборе приложения
- item.url - удаленный адрес архива приложения
- item.front - адрес архива для клиента, если не указан, создается по общей настройке и имени
- item.language - если не указан, то приложение отобразится в любом языке
- storage - место сохранения архивов приложений
- front - ссылка до архивов с клиента
- tempzip - временный файл при удаленной закачки с другого сервера
- enableUpdate - вкл/выкл синхронизацию с удаленным сервером. При старте сервера проверяется наличие архивов, и если нет, то скачиваются с указанных URL. По истечению периода updateInterval архивы обновляются
- updateInterval - период повторной загрузки архива на сервер (секунды)

На клиенте можно указать custom URL, но нужно иметь в виду, что браузер разрешает загрузки с чужих хостов только явно разрешенные через Access-Control-Allow-Origin

Оригинальная инструкция на английском представлена на сайте https://github.com/nwjs/nw.js/wiki/How-to-package-and-distribute-your-apps в пункте An alternative way to make an executable file in Windows

Ссылки

- Репозиторий приложения
- Node-webkit
- Node-webkit вики
- Пакет для формирования исполняемого файла
- Программа для линковки dll
- Руководство пользователя
- Запуск приложения как standalone
- Инструкция по созданию ИС при помощи ION. Studio

2.2.2 Сборка приложения из репозитория

Для быстрого запуска приложений с использованием репозитория GitHub выполните следующие шаги:

1. Установите системное окружение и глобальные зависимости.
2. Клонируйте ядро, модуль и приложение.
3. Соберите и разверните приложение.
4. Запустите.

Ниже - развернутая инструкция, как собрать приложение из репозитория GitHub.

Развернутая инструкция

Системное окружение и глобальные зависимости

Запуск фреймворка осуществляется в среде Node.js версии 10.x.x.

Для хранения данных необходимо установить и запустить MongoDb версии старше 3.6.

Для сборки компонентов и библиотек фреймворка необходимо установить глобально:

- пакет node-gyp npm install -g node-gyp. Для работы библиотеки под операционной системой семейства windows дополнительно необходимо установить пакет windows-build-tools npm install -g --production windows-build-tools.
- пакет сборщика проектов Gulp npm install -g gulp@4.0. 4.0 - поддерживаемая версия Gulp.
- для версий IONDV. Framework 3.x.x и более ранних нужен менджер пакетов фронтенд библиотек Bower npm install -g bower. Для версия 4.x.x и старше не требуется.

Ручная установка ядра, модулей и приложения

Рассматриваем на примере приложения develop-and-test. Находим приложение develop-and-test в репозитории. Смотрим зависимости указаные в файле package.json.

```
 "engines": {
  "ion": "1.24.1"
},
"ionModulesDependencies": {
  "registry": "1.27.1",
  "geomap": "1.5.0",
  "graph": "1.3.2",
  "portal": "1.3.0",
  "report": "1.9.2",
  "ionadmin": "1.4.0",
  "dashboard": "1.1.0",
  "lk": "1.0.1",
  "soap": "1.1.2",
  "gantt-chart": "0.8.0"
},
"ionMetaDependencies": {
  "viewlib": "0.9.1"
  "viewlib-extra": "0.1.0"
```

- Начинаем установку с ядра, версия которого указана в парметре "engines": {"ion": "1.24.1"}. Скопируйте адрес репозитория ядра и в командной строке выполните комманду git clone https://github.com/iondv/framework. Перейдите в папку ядра, переключитесь на tag номера версии git

checkout tags/v1.24.1. Так как совместимость обеспечивается на уровне метаданных, а новые версии выпускались из-за изменения технологии сборки, то вы можете использовать последние, например 4.0.0.

- После этого устанавливаются необходимые для приложения модули, указанные в параметре "ionModulesDependencies". Модули устанавливаются в папку modules ядра, для этого перейдите в неё командой cd modules. Клонируем модули из списка "ionModulesDependencies", для модуля registry это осуществляется командой git clone https://github.com/iondv/registry. Перейдите в папку установленного модуля, переключитесь на tag номера версии git checkout tags/v1.27.1. Повторите для каждого модуля. Для большинства приложений, можно использовать последние совместимые с ядром модули.
- Установка самого приложения осуществляется в папку applications, для этого перейдите в неё командой cd ..\applications, если вы находитесь в папке модулей. Установку выполните клонированием репозитория коммандой git clone https://github.com/iondv/dnt_ru.
- После этого установите дополнительно необходимые приложения из параметра "ionMetaDependencies". Установка осуществляется в папку applications, для установки необходимо убедиться, что находитесь в папке приложений. Клонируем приложения из списка в параметре "ionMetaDependencies", для приложения viewlib осуществляется командой https://github.com/iondv/viewlib. Перейдите в папку установленного приложения, переключитесь на tag номера версии git checkout tags/v0.9.1. Повторите для каждого приложения.

Сборка, конфигурирование и развертывание приложения

Сборка приложения обеспечивает установку всех зависимых библиотек, импорт данных в базу данных и подготовку приложения для запуска.

Создайте конфигурационный файл setup.ini в папке config, куда был клонирован фреймворк для задания основных параметров окружения приложения.

```
auth.denyTop=false
auth.registration=false
db.uri=mongodb://127.0.0.1:27017/db
server.ports[]=8888
module.default=registry
fs.storageRoot=./files
fs.urlBase=/files
```

Открываем файл в редакторе и вставляем содержимое. Самый главный параметр db.uri=mongodb://127.0.0.1:27017/ion-dnt - он указывает на название базы, которую мы будем использовать для приложения. База данных будет создана автоматически.

Задайте переменную окружения NODE_PATH равной пути к ядру приложения следующей командой set NODE_PATH=c:\workspace\dnt для Windows и export NODE_PATH=/workspace/dnt для Linux, где workspace\dnt - папка куда склонирован фреймворк.

При первом запуске необходимо выполнить npm install - она поставит ключевые зависимости, в том числе локально сборщик gulp.

Далее выполните команду сборки приложения gulp assemble.

Если вы хотите выполнить импорт данных в вашем проекте, проверьте папку data в приложении и выполните команду: node bin/import-data --src ./applications/develop-and-test/data --ns develop-and-test

Добавьте пользователя admin с паролем 123 командой node bin\adduser.js --name admin --pwd 123.

Добавьте пользователю права администратора командой node bin/acl.js --u admin@local --role admin --p full.

Запуск

Запустите приложение командой в папке ядра npm start или node bin\www.

Откройте браузер с адресом http://localhost:8888 и авторизуйтесь в приложении, где 8888 - порт указанный в параметре server.ports конфигурации запуска.

2.2.3 Docker-контейнер

Приложения также можно запустить с использованием докер контейнера.

Docker – это программная платформа для быстрой разработки, тестирования и развертывания приложений.

Шаги

1. Запустите СУБД mongodb: docker run --name mongodb -v mongodb_data:/data/db -p 27017:27017 -d mongo
2. Запустите IONDV. App docker run -d -p 80:8888 --link mongodb iondv/app, где App - название приложения и путь к соответствующему репозиторию.
3. Откройте ссылку http://localhost в браузере через минуту (время требуется для инициализации данных). Для авторизации используйте типовой логин: demo, пароль: ion-demo

2.2.4 Установщик для Linux

Вы также можете использовать установщик приложений IONDV. Framework для Linux, требующий установленных node, mongodb и git. В ходе установки будет проверены и установлены все остальные зависимости, а также собрано и запущено само приложение.

Установка в одну строку:

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) -t git -q -i -m␣
↪localhost:27017 app
```

Где параметры для iondv-app localhost:27017 адрес MongoDB, а app название приложения. После запуска открыть ссылку http://localhost:8888, учетная запись бек офиса demo, пароль ion-demo.

Также другой способ заключается в клонировании - git clone https://github.com/iondv/iondv-app.git и установите приложение с помощью команды bash iondv-app -m localhost:27017 app, где app - название приложения.

Можно также собрать приложение в докер конейтнерах, тогда из окружения нужен только docker и СУБД mongodb в докер контейнере.

Требования к окружению

Для работы системы должна быть запущена СУБД Монго ДБ. В docker-контейнере её можно запустить командой:

docker run --name mongodb -v mongodb_data:/data/db -p 27017:27017 -d mongo

Описание подготовки системы приведено ниже.

Установка, сборка и запуск приложения одной командой в докер-контейнере

Примеры сборок ниже, подразумевают что СУБД запущена в контейнере с именем mongodb.

На примере простейшего приложения Nutrition-Tickets, команда скачивает установщик с репозитория github, который собирает и запускает приложение в docker-контейнере в текущей директории.

- из репозтория git

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) \   -t docker -q -i -
↪l mongodb nutrition-tickets
```

- из архива zip, например полученного с гитхаб curl -L https://github.com/iondv/nutrition-tickets/archive/master.zip > ./nutrition-tickets.zip или созданного в IONDV. Studio. Обратите внимание, что в package.json созданного приложения в атрибуте "ionModulesDependencies" нужно указать модуль для отображения данным - обычно это "registry".

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) \   -t docker -q -i -
↪l mongodb ./nutrition-tickets.zip
```

- из папки, при этом оригинальная папка приложения не модифицируется. Обратите внимание, что название папки должно соответствовать неймспейсу приложения (если папка распакована с архива github - то в названии обычно добавляется код ветки - нужно переименовать)

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) \   -t docker -q -i -
↪l mongodb ./nutrition-tickets
```

Адрес собранного приложения по умолчанию http://localhost:8888, пользователь demo, пароль iondemo.

Установка, сборка и запуск приложения в локальной файловой системе

Примеры сборок ниже, подразумевают что СУБД запущена локально и доступна по адресу localhost:27017.

Установка в локальной файловой системе позволяет получить приложение готовое к модификации и доработкам средствами разработчика, например в IDE - достаточно открыть папку приложения. Папка приложения создается в папке запуска, либо в папке заданной параметром -p

- из репозитория git в папке /workspace

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) \   -t git -p /
↪workspace -m localhost:27017 https://github.com/iondv/nutrition-tickets.git
```

- из архива zip в текущей папке, например полученного с гитхаб curl -L https://github.com/iondv/nutrition-tickets/archive/master.zip > ./nutrition-tickets.zip или созданного в IONDV. Studio. Обратите внимание, что в package.json созданного приложения в атрибуте "ionModulesDependencies" нужно указать модуль для отображения данным - обычно это "registry"

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) \ -t git -
↪p /workspace -m localhost:27017 ./nutrition-tickets.zip
```

- из папки, при этом оригинальная папка приложения не модифицируется. Обратите внимание, что название папки должно соответствовать неймспейсу приложения (если папка распакована с архива github - то в названии обычно добавляется код ветки - нужно переименовать)

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) \    -t git -q -i -m␣
↪localhost:27017 ./nutrition-tickets
```

Параметры запуска

iondv-app [OPTION]... IONDV_APP_NAME|IONDV_APP_NAME@VERSION|GIT_URL|IONDV_APP_ZIP|IONDV_

Параметры	Тип сборки приложения
-t [value]	git: клонирование репозиториев в файловую систему (требуется установленный git) docker: сборка в докер-контейнерах, не требует установки окружения на хост-машину
-c [value]	запуск приложения как кластера с кол-вом инсталляций [value]
-m [value]	uri для монгодб, примеры: mongodb:27017. localhost:27017 - по умолчанию (при сборке в докере выдаст ошибку подключения к БД(!). Для докера используйте параметр -l, либо укажите внешний адрес СУБД
-r	проверка и удаление папки с именем приложения в директории сборки
-i	импорт данных при инициализации приложения
-a	импорт ролей и учетных записей пользователей при инициализации приложения
-y	применение всех значений по умолчанию (yes to all)
-q	тихий режим. Показывается только основная информация, предупреждения и ошибки
-l [value]	имя контейнера MongoDB для линковки к собранному контейнеру (тип сборки docker или параметр -d при типе сборки git), также формирует конфигурацию с указанием значения mongo uri как [value]:27017
-p [value]	путь к директории в которой будет создавать папка с именем приложения и осуществляться сборка
-s [value]	полный путь к скрипту, запускаемому в папке приложения после сборки, но до деплоя приложения. Может использоваться для дополнительной обработки файлов приложения
-n [value]	параметр определяющий запуск изменение неймспейса приложения на новое, до деплоя
-h	пропуск переключения на версии зависимостей приложения, установка последних версий
-x	выход без запуска приложения
Параметры для метода git:	
-d	на основе собранной версии подготовить также docker-контейнер. Также остановить и удалить контейнер, образ с таким именем
-k	пропустить проверку окружения
Параметры для метода сборки docker:	сохранять временные версии контейнеров - позволяет ускорить последующие сборки. Но кэширование пропускается, если установлен флаг игнорировать версии зависимостей
-v	на основе собранной версии подготовить также docker-контейнер. Также остановить и удалить контейнер, образ с таким именем
Переменные окружения:	
IONDVUrlGitFramework	URL репозитория фреймворка, по умолчанию https://github.com/iondv/framework.git Вы можете задать логин и пароль к своей версии в приватном репозитории. Например: https://login:password@git.company-name.com/iondv/framework.git
IONDVUrlGitModules	URL к модулям, по умолчанию by default https://github.com/iondv
IONDVUrlGitApp	URL к приложениям - используется если для сборки указано только имя приложения, по умолчанию https://github.com/iondv
IONDVUrlGitExtApp	URL к приложениям-расширениям, по умолчанию https://github.com/iondv

Подготовка окружения

Установка docker

Рекомендуется делать не под root

- Установка последней версии docker для CentOS:

1. Обновляем систему sudo yum update

2. Устанавливаем необходимые библиотеки yum install -y yum-utils device-mapper-persistent-data lvm2

3. Регистрируем репозиторий yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

4. Установка последней версии yum -y install docker-ce docker-ce-cli containerd.io

5. Запускаем докер systemctl start docker

6. Для автоматического запуска докера systemctl enable docker

- Установка последней версии docker для Ubuntu:

1. Добавляем ключ GDP curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
2. Проверяем ключ apt-key fingerprint 0EBFCD88
3. Добавляем репозиторий

```
sudo add-apt-repository \    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"
```

4. Обновляем репозитории sudo apt-get update
5. Ставим последнюю версию sudo apt-get install docker-ce docker-ce-cli containerd.io

Добавляем текущего пользователя в группу docker:

sudo groupadd docker sudo usermod -aG docker $USER

Проверить можно docker run hello-world

Запуск Mongo в докере

Запускаем с маппингом на локальный порт:

docker run --name mongodb -v mongodb_data:/data/db -p 27017:27017 -d mongo

Установка node

Для ускорения сборки, рекомендуется предварительно скачать локально docker-образ node:10, т.к. он занимает 900Мб.

docker pull node:10

Проверить можно командой docker images | grep node - будет отображён списокок локальных образов node.

Начать знакомство с IONDV. Framework рекомендуем с быстрого запуска приложений.

В этом разделе приведены инструкции для разворачивания приложений IONDV. Framework. Существует несколько способов быстрого запуска приложений. Вы можете определиться со способом, который удобен для Вас.

Развернуть приложения Вы можете следующими способами:

- с помощью IONDV. Studio
- клонирование репозитория приложения и установка всех компонентов - смотреть
- с помощью docker-контейнера с собранными приложениями
- с помощью установщика для операционной системы Linux

2.2.5 Ссылки

Ниже представлены ссылки на дополнительную информацию по разработке приложений с использованием IONDV.Framework.

- Документация
- Домашняя страница фреймворка
- Обратная связь на Facebook
- Обучающие видеоролики на youtube

ГЛАВА 3

3. Разработка

3.1 Расширение функциональности

3.1.1 Разработка функциональных утилит в приложении

Функциональные утилиты приложения предназначены для решения специфичных задач приложения, без реализации отдельной логики в виде модуля.

Например - утилиты вызываемой задачи по расписанию или утилиты вызываемой при переходе в бизнес-процессе.

Утилиты для кнопок действия

Утилиты для кнопок действия предназначены для автоматизации выполнения некоторых действий при нажатии кнопки в веб-форме. Кнопка подключается для веб-формы создания и редактирования объекта через массив commands основого объекта формы:

```
{
  "commands": [
    {
      "id": "SAVE",
      "caption": "Save",
      "visibilityCondition": null,
      "enableCondition": null,
      "needSelectedItem": false,
      "signBefore": false,
      "signAfter": false,
      "isBulk": false
    },
    {
      "id": "SAVEANDCLOSE",
      "caption": "Save and close",
```

(continues on next page)

(continued from previous page)

```
    "visibilityCondition": null,
    "enableCondition": null,
    "needSelectedItem": false,
    "signBefore": false,
    "signAfter": false,
    "isBulk": false
  },
  {
    "id": "CREATE_INDICATOR_VALUE",
    "caption": "Form the collected values",
    "visibilityCondition": null,
    "enableCondition": null,
    "needSelectedItem": false,
    "signBefore": false,
    "signAfter": false,
    "isBulk": false
  }
],
```

На этой форме доступно три кнопки: SAVE, SAVEANDCLOSE и CREATE_INDICATOR_VALUE.

Пользовательские кнопки перед подключением к форме необходимо задать в deploy.json в объекте modules.registry.globals.di.actions.options.actions:

```
{
  "modules": {
    "registry": {
      "globals" : {
        "di": {
          "actions": {
            "options": {
              "actions": [
                {
                  "code": "CREATE_INDICATOR_VALUE",
                  "handler": "ion://createIndicatorValueHandler"
                },
                {
                  "code": "ASSIGNMENT_TO_EVENT_ONLY",
                  "handler": "ion://assignmentToEventOnly"
                },
                {
                  "code": "CREATE_PROJECT_REPORTS",
                  "handler": "ion://createProjectReportsHandler"
                }
              ]
```

А также указать параметры используемого кнопкой модуля-обработчика:

```
{
  "modules": {
    "registry": {
      "globals" : {
        "di": {
          "createIndicatorValueHandler": {
            "module": "applications/sakh-pm/lib/actions/createIndicatorValueHandler",
            "initMethod": "init",
            "initLevel": 2,
```

(continues on next page)

(continued from previous page)

```
        "options": {
          "data": "ion://securedDataRepo",
          "workflows": "ion://workflows",
          "log": "ion://sysLog",
          "changelogFactory": "ion://changelogFactory",
          "state": "onapp"
        }
      },
```

В этом примере нажатие на кнопку CREATE_INDICATOR_VALUE запускает скрипт ./applications/sakh-pm/lib/actions/createIndicatorValueHandler.js.

Содержание скрипта:

```
/**
 * Created by kras on 08.09.16.
 */
'use strict';

const ActionHandler = require('modules/registry/backend/ActionHandler');
const edit = require('modules/registry/backend/items').saveItem;
const ivc = require('../indicator-value-creator');

/**
 * @constructor
 * @param {{}} options
 * @param {DataRepository} options.data
 * @param {WorkflowProvider} options.workflows
 * @param {Logger} options.log
 * @param {ChangelogFactory} [options.changelogFactory]
 * @param {String} [options.state]
 */
function CreateIndicatorValueHandler(options) {

  options = options || {};

  const work = ivc(options);

  this.init = function () {
    if (options.workflows && options.state) {
      options.workflows.on(
        'indicatorBasic@sakh-pm.' + options.state,
        (e) => {
          let logger = null;
          if (options.changelogFactory && e.user) {
            logger = options.changelogFactory.logger(() => e.user.id());
          }
          return work(e.item, e.user, logger).then(() => null);
        }
      );
    }
  };

  /**
   * @param {{metaRepo: MetaRepository, securedDataRepo: SecuredDataRepository}} scope
   * @param {ChangelogFactory} scope.changelogFactory
   * @param {Request} req
```

(continues on next page)

(continued from previous page)

```
   * @returns {Promise}
   */
  this._exec = function (scope, req) {
    let logger;
    let user = scope.auth.getUser(req);
    if (options.changelogFactory) {
      logger = options.changelogFactory.logger(() => user.id());
    }
    return edit(scope, req, null, logger, true)
      .then(item => scope.dataRepo.getItem(item, null))
      .then((item) => {
        if (item.get('status') !== 'edit') {
          throw new Error('Создать значения показателей, можно только при редактировании!');
        }
        return work(item, user, logger);
      })
      .then((count) => {
        return {$message: 'Создано ' + count + ' значений для ввода по периодам!'};
      });
  };
}

CreateIndicatorValueHandler.prototype = new ActionHandler();

module.exports = CreateIndicatorValueHandler;
```

Утилиты для задач по расписанию

Утилиты для задач по расписанию (jobs) предназначены для автоматизации регулярного выполнения некоторых действий через определенные промежутки времени.

Для этого каждая утилита должна быть определена в deploy.json приложения в объекте globals.jobs, например:

```
{
  "globals": {
    "jobs": {
      "ticketClose": {
        "description": "Ночной перевод билетов в статус \"проверен\"",
          "launch": {
            "timeout": 3600000,
            "hour": 24
          },
          "worker": "ticketCloser",
          "di": {
            "ticketCloser": {
              "executable": "applications/khv-ticket-discount/lib/overnightTicketClose",
              "options": {
                "dataRepo": "ion://dataRepo",
                "log": "ion://sysLog",
                "workflows": "ion://workflows"
...
```

В di должно содержаться поле с именем, равном значению worker - это задача, которая будет запускаться.

Здесь по расписанию выполняется скрипт applications/khv-ticket-discount/lib/overnightTicketClose.js. launch может быть объектом, содержащим следующие поля:

month, week, day, dayOfYear, weekday, hour, min, minute, sec, second - задают интервал между выполнениями задачи;

check - интервал проверки условия выполнения, в милисекундах, по умолчанию - 1000.

Например если check равен 5000, а sec - 2, задание будет выполняться лишь каждые 10 секунд, когда интервал между проверками совпадет с интервалом выполнения

Если интервал выполнения задачи не задан, то она будет выполнена при запуске приложения и через каждый интервал проверки.

timeout - время в милисекундах, после которого запущенная задача прерывается по таймауту;

launch также может равняться целому числу - интервалу выполнения задания в милисекундах, при этом задача также будет выполнена сразу при запуске приложения. Таймаут будет установлен равным интервалу выполнения.

В поле options могут быть указаны любые переменные и их значения, которые станут доступны в скрипте через поля объекта, передаваемого как аргумент основной функции модуля.

Скрипт составляется в формате модуля, например так:

```
"use strict";
const Logger = require("core/interfaces/Logger");

module.exports = function (options) {
  return options.dataRepo
    .getList(
      "ticket@khv-ticket-discount",
      "ticketYear@khv-ticket-discount"
    )
    .then((tickets) => {
      let p = Promise.resolve();
      tickets.forEach((ticket) => {
        p = p
          .then(() => options.dataRepo.editItem(ticket.getClassName(), ticket.getItemId(), {"state": "close"}))
          .then(item => (item.name === "ticketYear" ?
            options.workflows.pushToState(item, "ticketYear@khv-ticket-discount", "close") :
            options.workflows.pushToState(item, "ticket@khv-ticket-discount", "close")))
          .catch((err) => {
            if(options.log instanceof Logger) {
              options.log.error(err);
            } else {
              console.error(err);
            }
          });
      });
      return p;
    });
};
```

Утилиты для печатных форм

Утилиты для печатных форм (инжекторы) предназначены для обработки выводимых в шаблон данных, в том числе проведения промежуточных расчетов и форматирования.

Печатную форму, для которой будет использоваться инжектор, необходимо определить в deploy.json, например:

```
"registry": {
   "globals": {
      "di": {
         "pmListToDocx": {
            "module": "modules/registry/export/listToDocx",
            "initMethod": "init",
            "initLevel": 0,
            "options": {
               "tplDir": "applications/khv-ticket-discount/export/list",
               "log": "ion://sysLog",
            }
            ...
         }
         ...
      }
      ...
   }
   ...
}
```

В данном случае используется модуль listToDocx, следовательно в печатную форму будет выгружен список всех объектов определенного класса.

Для каждого такого класса в tplDir нужно создать папку с названием пространства имен, в которую затем поместить файл с названием нужного для выгрузки класса из этого пространства имен, например:

```
...\applications\khv-ticket-discount\export\list\khv-ticket-discount\ticketYear.docx
```

Таким образом в документ будет выгружен список всех объектов класса ticketYear@khv-ticket-discount.

Сама утилита представляет собой .js скрипт, подключаемый к приложению в формате модуля в deploy.json, например так:

```
"registry": {
   "globals": {
      "di": {
         "weekTicketStatsInjector": {
            "module": "applications/khv-ticket-discount/export/injectors/monthTicketStats",
            "options": {
            "dataRepo": "ion://dataRepo"
            }
            ...
         }
         ...
      }
      ...
   }
   ...
}
```

.js файл здесь находится по пути “module”.

После подключения утилиту необходимо включить в опции печатной формы:

```
"registry": {
   "globals": {
      "di": {
         "pmListToDocx": {
            "module": "modules/registry/export/listToDocx",
            "initMethod": "init",
            "initLevel": 0,
            "options": {
               "tplDir": "applications/khv-ticket-discount/export/list",
               "log": "ion://sysLog",
               "injectors": [
                  "ion://monthTicketStatsInjector"
               ]
            }
            ...
         }
         ...
      }
      ...
   }
   ...
}
```

Скрипт инжектора составляется в формате модуля, с тем условием, что он должен содержать функцию this.inject, в параметр которой будет передан объект с вложенным в него списком объектов заданного ранее класса, для примера из этой справки:

```
ticketYear@khv-ticket-discount
```

Пример файла monthTicketStats.js:

```
function monthTicketStatsInjector() {
  this.inject = function (value) {
    if (value && value.className === "ticketYear") {
      let expValueList = [];
      const periodBegF = value.periodBegF;
      const periodEndF = value.periodEndF;
      const areaF = value.areaF;
      let i = 0;
      value.list.forEach((vectorparams) => {
        if (vectorparams.person.area.code === areaF && vectorparams.dateAirGo >= periodBegF &&
↪vectorparams.dateAirGo <= periodEndF && ((vectorparams.state !== "canceled") && (vectorparams.state !
↪== "returned"))) {
          expValueList[i++] = vectorparams;
        }
      });
      value.list = expValueList;
    }
    return value;
  };
}

module.exports = monthTicketStatsInjector;
```

Пример конфигурации экспорта для данной формы в deploy.js:

```
"registry": {
```

(continues on next page)

(continued from previous page)

```
  "globals": {
    "di": {
      "export": {
        "options": {
          "configs": {
            "ticketYear@khv-ticket-discount": {
              "pmListToDocx": {
                "type": "list",
                "caption": "Ежемесячный отчет",
                "mimeType": "application/vnd.openxmlformats-officedocument.wordprocessingml.
↪document",
                "extension": "docx",
                "params": {
                  "periodBegF": {
                  "caption": "Период с",
                  "type": "date"
                },
                "periodEndF": {
                  "caption": "по",
                  "type": "date"
                },
                "areaF": {
                  "caption": "Район",
                  "type": "reference",
                  "className": "area@khv-ticket-discount"
                }
              },
              "preprocessor": "ion://pmListToDocx",
              "eagerLoading": [
                "person",
                "person.documents",
                "person.area",
                "route.pointDeparture",
                "route.pointArrival",
                "route.flight"
              ],
              "fileNameTemplate": "Ежемесячный отчет"
            }
          }
        }
      }
      ...
    }
    ...
  }
  ...
}
```

Здесь следует обратить внимание на поле params - в нем можно указать параметры, доступные в форме экспорта в веб сервисе приложения. Возможны следующие типы параметров:

“string” - строка для ввода текста,

“date” - интерактивный календарь, в котором можно выбрать интересующую дату

“reference” - ссылка на класс, в данном случае в окне экспорта будет отображен выпадающий список всех объектов класса.

Переданные параметры будут доступны в скрипте через параметр функции this.inject.

Утилиты для веб-сервиса (rest)

Утилиты веб-сервиса предназначены для реализации обработки различных видов запросов к серверу.

Сервис подключается к приложению в deploy.json в объекте modules.rest.globals.di, например так:

```
{
  "modules": {
    "rest": {
      "globals": {
        "di": {
          "acceptor": {
            "module": "modules/rest/lib/impl/acceptor",
            "options": {
              "dataRepo": "ion://dataRepo",
              "metaRepo": "ion://metaRepo"
...
```

В этом случае подключается сервис acceptor, он станет доступен для запросов по url https://dnt.iondv.com/rest/acceptor.

Функциональное описание взаимодействия с запросами должно содержаться в скрипте modules/rest/lib/impl/acceptor.js.

В поле options могут быть указаны любые переменные и их значения, которые станут доступны в скрипте через поля объекта, передаваемого как аргумент основной функции модуля.

Скрипт составляется в формате модуля, например так:

```
const Service = require('modules/rest/lib/interfaces/Service');

/** Simple app service - REST module
 * @param {{dataRepo: DataRepository, metaRepo: MetaRepository}} options
 * @constructor
 */
function EchoRest(options) {
  this._route = function(router) {
    this.addHandler(router, '/', 'POST', (req) => {
      return Promise.resolve({
        echo: 'peekaboo'
      });
    });
    this.addHandler(router, '/', 'GET', (req) => {
      return Promise.resolve({
        echo: 'peekaboo'
      });
    });
  };
}
EchoRest.prototype = new Service();
module.exports = EchoRest;
```

Подробное описание принципов создания сервиса можно найти в https://github.com/iondv/rest/blob/master/README_RU.md/ раздел Разработка обработчика сервиса в приложении

Утилиты для бизнес-процесса

Утилиты для бизнес-процесса предназначены для автоматизации выполнения некоторых действий при изменении статуса бизнес процесса. Утилита подключается к приложению в deploy.json в объекте globals.plugins, например так:

```
{
  "globals": {
    "plugins": {
      "wfEvents": {
        "module": "applications/sakh-pm/lib/wfEvents",
        "initMethod": "init",
        "initLevel": 1,
        "options": {
          "workflows": "ion://workflows",
          "metaRepo": "ion://metaRepo",
          "dataRepo": "ion://dataRepo",
          "log": "ion://sysLog",
          "AIPConfigPath": "applications/sakh-pm/paths_config/eventOnlyAIP.json"
```

Здесь подключается утилита wfEvents. Скрипт, содержащий описание действий при изменении статуса бизнес-процесса находится по пути applications/sakh-pm/lib/wfEvents.js.

В поле options могут быть указаны любые переменные и их значения, которые станут доступны в скрипте через поля объекта, передаваемого как аргумент основной функции модуля.

Скрипт составляется в формате модуля, при условии, что он должен включать метод init, например так:

```
'use strict';
const ivc = require('./indicator-value-creator');

function WorkflowEvents(options) {
  this.init = function () {
    options.workflows.on(
      ['assignmentBasic@sakh-pm.fin'],
      (e) => {
        if (e.transition === 'toKT') {
          return options.dataRepo.getItem(e.item, null, {forceEnrichment: [['meeting','basicObj'],['basicObj']]})
            .then((item) => {
              const data = {
                basicObj: item.property('meeting.basicObj').evaluate() || item.property('basicObj').evaluate(),
                name: item.get('name'),
                owner: item.get('owner'),
                datePlannedEnd: e.item.get('datePlannedEnd'),
                priority: e.item.get('priority'),
                descript: e.item.get('descript')
              };
              return options.dataRepo.createItem('eventControl@sakh-pm', data, null, {user: e.user});
            });
        }
        return Promise.resolve();
      }
    );
    options.workflows.on(
      ['proposal@sakh-pm.cancel'],
      (e) => {
        if (e.transition === 'curatorToCancel') {
```

(continues on next page)

(continued from previous page)

```
        return options.dataRepo.editItem(e.item.getMetaClass().getCanonicalName(), e.item.getItemId(),
↪{archive: true})
          .then(
            item => collectionToArchive(item, 'proposals')
              .then(() => collectionToArchive(item, 'eventBlock'))
              .then(() => collectionToArchive(item, 'project'))
          );
      }
      return Promise.resolve();
    }
  );
 };
}

module.exports = WorkflowEvents;
```

В этом скрипте описаны действия, которые нужно выполнить при изменении статуса бизнес-процесса assignmentBasic@sakh-pm на fin, и статуса бизнес-процесса proposal@sakh-pm на cancel.

3.1.2 Ключевые функции ядра

dataRepo

//dataRepo info from

1. coreimpldatarepositoryionDataRepository.js
2. coreinterfacesDataRepositoryDataRepository.js
3. coreinterfacesMetaRepositoryMetaRepository.js
4. coreimplmetaDsMetaRepository.js

#. coreiterfacesDataSource.js #.

coreimpldatasourcemongodb.js // supported calls:

1. wrap(className, data, [version], [options]) supported options: user ...
2. setValidators(validators[]) ...
3. getCount(obj, [options]) supported options: filter

Возвращает количество объектов класса obj в базе данных. ...

1. getList(obj, [options]) supported options: filter offset count sort countTotal nestingDepth env user

Возвращает список объектов класса obj в базе данных. ...

1. getIterator(obj, [options]) supported options: filter offset count sort countTotal nestingDepth env user

предположительно https://docs.mongodb.com/manual/tutorial/iterate-a-cursor/ ...

1. aggregate(className, [options]) supported options: user expressions filter groupBy

предположительно https://docs.mongodb.com/manual/aggregation/

...

1. rawData(className, [options]) supported options: user filter attributes distinct

https://docs.mongodb.com/manual/reference/method/db.collection.find/ ...

1. getItem(obj, [id], [options]) supported options: filter nestingDepth user ...
2. createItem(className, data, [version], [changeLogger], [options]) supported options: nestingDepth skipResult adjustAutoInc user ...
3. editItem(className, id, data, [changeLogger], [options]) supported options: nestingDepth skipResult adjustAutoInc user ...
4. saveItem(className, id, data, [version], [changeLogger], [options]) supported options: nestingDepth autoAssign skipResult adjustAutoInc user ...
5. deleteItem(className, id, [changeLogger], [options]) supported options: user ...
6. put(master, collection, details, [changeLogger], [options]) supported options: user ...
7. eject(master, collection, details, [changeLogger], [options]) supported options: user ...
8. getAssociationsList(master, collection, [options]) supported options: filter offset count sort countTotal nestingDepth user ...
9. getAssociationsCount(master, collection, [options]) supported options: filter offset count sort countTotal nestingDepth user ...
10. bulkEdit(classname, data, [options]) supported options: filter nestingDepth forceEnrichment user ...
11. bulkDelete(classname, [options]) supported options: filter user
12. recache(item, [options]) ...

MetaRepo

TODO

Workflow

TODO

3.1.3 Использование шаблонов для полей ввода данных в веб-форме

Шаблоны предназначены для задания пользовательских параметров построения полей ввода данных в веб-форме.

Для подключения шаблона к полю веб-формы, необходимо указать его в опциях соответствующего поля json формы:

```
{
  "tabs": [
    {
      "caption": "General info",
      "fullFields": [
        {
          "property": "surname",
          "caption": "Surname",
          //...
          "options": {
            "template": "capitalize"
          },
          "tags": ""
        },
```

Шаблоны в формате .ejs загружаются по пути, указанном в modules.registry.globals.templates из deploy.json.

Например, если указан путь applications/khv-ticket-discount/templates/registry, то в прошлом примере для поля property будет загружен скрипт applications/khv-ticket-discount/templates/registry/capitalize.ejs.

Скрипт будет выполнен при загрузке поля в веб-форме. Синтаксис стандартный для ejs.

Внутри скрипта доступны некоторые элементы веб-формы, подробнее: Опции.

Пример скрипта для автоматической замены в текстовом поле ввода нижнего регистра букв на верхний и буквы “ö” на “o”:

```
<% wfState = item.base.state %>
<div class="form-group <%= wfState === 'edit' || item.id === null ? (field.required ? 'required' : '') : ''
↪%> " style data-type="<%= wfState === 'edit' || item.id === null ? 'input' : 'static' %>" data-name="
↪<%= prop.getName().toLowerCase() %>" data-prop="<%= JSON.stringify(field) %>" >
  <label for="a_khv-ticket-discount_<%= item.getClassName().split('@')[0] %>_<%= prop.getName().
↪toLowerCase() %>" class="col-md-2 col-sm-3 control-label"><%= prop.getCaption() %>
  </label>
   <div class="col-sm-9">
      <input id="a_khv-ticket-discount_<%= item.getClassName().split('@')[0] %>_<%= prop.getName().
↪toLowerCase() %>" type="<%= wfState === 'edit' || item.id === null ? 'text' : 'hidden' %>" class=
↪"form-control attr-value" name="<%= prop.getName().toLowerCase() %>" data-mask="{&amp;quot;regex&amp;quot;:&
↪quot;[öÖa-zA-Z .-]{1,50}&amp;quot;}" placeholder="<%= prop.getCaption() %>" value="<%= prop.getValue() !
↪== null ? prop.getValue() : "" %>" im-insert="true">
      <% if(wfState === 'done' && item.id !== null) { %>
         <div class="form-control-static"><%= prop.getValue() %></div>
      <% } %>
      <script>
         if (typeof inputField !== 'object') {inputField = [];}
         propName = '<%= prop.getName().toLowerCase() %>';
         inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'] =␣
↪document.getElementById(`a_khv-ticket-discount_<%= item.getClassName().split('@')[0] %>_${propName}
↪`);
         inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'].
↪addEventListener('focusout',applyStyle)
         inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'].
↪addEventListener('keyup', applyStyle)
         inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'].
↪addEventListener('keydown', applyStyle)
         inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'].
↪addEventListener('paste', applyStyle)
         function applyStyle() {
            while ((/[Ö]/).test(inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().
↪toLowerCase() %>'].value)) {
               inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>
↪'].value = inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'].
↪value.replace(/[Ö]/, 'O');
            }
            while ((/[ö]/).test(inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().
↪toLowerCase() %>'].value)) {
               inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>
↪'].value = inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'].
↪value.replace(/[ö]/, 'o');
            }
            while ((/[a-z]/).test(inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().
↪toLowerCase() %>'].value)) {
```

(continues on next page)

(continued from previous page)

```
                inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>
↪'].value = inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'].
↪value.toUpperCase();
            }
            //(/[а-я]/).test(inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().
↪toLowerCase() %>'].value[0]) ? inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().
↪toLowerCase() %>'].value = inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().
↪toLowerCase() %>'].value[0].toUpperCase() + inputField['<%= item.getClassName().split('@')[0] %><%=␣
↪prop.getName().toLowerCase() %>'].value.substring(1) : ""; - только первая буква}
        }
    </script>
  </div>
</div>
```

3.2 Функциональность

Функциональность приложения - это набор возможностей (функций), которые реализуются отдельно и независимо от метаданных, но в рамках приложения.

Наименование	Описание
Шаблоны модулей	Разделяются на системные и проектные шаблоны.
Печатные формы	Передача форматирования в прописном виде.
Задания по расписанию	Подсистемы запуска заданий по расписанию.
Уведомления	Лента новых уведомлений.
Кеширование данных	Кеширование данных обеспечивает быстрый доступ к запрашиваемой информации.
Фильтры на форме	Запрос для фильтра на форме представления списка.
Связь двух неймспейсов	Для связывания двух проектов.
ЭЦП	Электронно-цифровая подпись.
Виртуальные атрибуты	Специальный тип атрибутов, который позволяет выводить полный код и наименование класса.
Утилиты	Дополнительные программы для более специализированного применения

3.2.1 Шаблоны модулей

Тема оформления

Тема оформления - директория следующей структуры:

```
/static/css     директория стилей
/static/js      директория скриптов
/templates      директория шаблонов ejs
```

Темы оформления могут располагаться:

- В директории view модуля и платформы - тогда это системные темы оформления
- В директории applications платформы как приложения - тогда это проектные темы оформления
- В директории themes приложения - тогда это проектные темы офрмления

Настройка текущей темы оформления:

1. Для платформы
    - Настройка theme в config.json платформы
    - Настройка globals.theme в deploy.json приложения
2. Для модуля
    - Настройка theme в config.json модуля
    - Настройка Имя модуля.globals.theme в deploy.json приложения

По умолчанию используется системная тема default (в платформе и модулях registry, geomap, report, offline-sync).

В настроке theme указывается путь до директории темы, он разрешается в соответствии с правилами:

1. Абсолютный путь берется как есть
2. Относительный путь разрешается относительно системных путей в следующем порядке:
    - Относительно директории view модуля или платформы
    - Относительно директории applications платформы
    - Относительно директории платформы

Пример в dnt

```
"geomap": {
    "globals": {
      "theme": "develop-and-test/themes/geomap",
```

3.2.2 Печатные формы

Печатные формы Word

- Используется библиотека docxtemplater
- Примеры подключения и использования docxtemplater здесь.

Параметры передачи форматирования

Для table_col параметр для передачи форматирования см. правила/примеры здесь.

Вид:

```
${table_col:коллекция:разделитель:формат}
```

Пример:

```
${table_col:list.instructions.limit:;:DD.MM.YYYY}
```

Результат:

```
30.08.2017;06.09.2017
```

В формате допускается использование символа :.

Вывод значения суммы в прописном варианте

Для docx-шаблонов есть фильтр toWords, который по умолчанию преобразует просто в текст, если добавить вторым параметром true, то будет добавляться рублевый формат (рубли - копейки).

Пример:

```
{costing.costExp | toWords:true}
```

В результате значение атрибута “costExp” = 345,52. Результат в печатной форме будет = Триста сорок пять рублей пятьдесят две копейки.

Преобразования между датой и строкой

Доступны к применению следующие функции:

- date - преобразование строки в дату
- upper - строку к верхнему регистру
- lower - строку к нижнему регистру

В экспорте в docx в выражениях доступны фильтры:

- lower - к нижнему регистру
- upper - к верхнему регистру
- dateFormat - дата к строке, примеры применения:
  - {now | dateFormat:ru}
  - {since | dateFormat:ru}
  - {date | dateFormat:ru:YYYYMMDD}
- toDate - строка к дате

Значение текущей даты _now

```
{_now} г.
```

Настройка для отображения значения полей из массива объектов

Если необходимо отобразить поля из массива объектов (коллекция например) используется тэг:

```
${table_col:list.collection.attrFromCollection}
```

По умолчанию значения будут объеденины через точку с запятой. Чтобы указать другой разделитель, укажите его после второго двоеточия:

```
${table_col:list.collection.attrFromCollection:разделитель}
```

3.2.3 Подсистема задания по расписанию

Запуск подсистемы заданий по расписанию выполняется двумя способами:

1. в отдельном процессе посредством скрипта bin/schedule.js
2. внутри процесса веб-приложения ION (bin/www) путем указания в ini-файле опции jobs.enabled=true

Во втором случае управление заданиями возможно реализовать в веб-приложении. Задания по расписанию настраиваются в deploy.json приложений в разделе глобальных настроек как параметр jobs.

Пример:

```
"jobs": {
  "dummy": {
    "launch": { // Периодичность запуска задания
      "month": [2,5], // в феврале и мае
      "week": 3, // каждую третью неделю (month и week  - взаимоисключающие настройки),
      "weekday": [1, 3, 5], // по понедельникам, средам и пятницам
      "dayOfYear": 5, // раз в 5 дней в течение года,
      "day": 10, // раз в 10 дней в течение месяца
      "hour": 3, // раз в 3 часа
      "minute": [10, 15, 35], // на 10-ой, 15-ой и 35-ой минуте
      "sec": 10 // раз в 10 секунд
    },
    "di": { // скоуп задания
      "dummy": {
        "module": "applications/develop-and-test/jobs/dummy",
        "options": {
        }
      }
    },
    "worker": "dummy", // имя компонента из скоупа задания, который будет исполняться
  }
}
```

В качестве запускаемого задания может быть указан компонент, в этом случае он должен иметь метод run. В качестве запускаемого задания может быть указана и функция. Тогда в di она описывается аналогично компоненту, но с использованием настройки executable:

```
"di": {
 "dummy": {
   "executable": "applications/develop-and-test/jobs/dummy",
   "options": {}
 }
}
```

Пример:

Раздел jobs в глобальных настройках.

```
...
"jobs": {
     "dummy": {
       "launch": {
         "sec": 30
       },
       "worker": "dummy",
       "di": {
         "dummy": {
           "executable": "applications/develop-and-test/jobs/dummy"
         }
       }
     }
...
```

3.2.4 Уведомления

Уведомления - на форме отображются в виде красного колокольчика, при клике на котором отображается лента уведомлений (10 последних непрочитанных), при клике на уведомлении оно скрывается как прочитанное. Лента обновляется раз в 15 секунд. Если колокольчик отсутсвует это значит что уведомлений нет.

Настройка уведомлений

Реализована отправка уведомлений на почту. Настраивается в проекте - TODO.

Реализовано API для отправки уведомлений программно. т.е. можно в обработчике события отправить уведомление. В админке при отправке уведомлений в поле “Получатели” указывается список полных имен пользователей (user@local) разделенных пробелом.

Хранение уведомлений происходит в коллекциях: ion_notification, ion_notification_recievers.

3.2.5 Кеширование данных

Логика работы

Между базовым репозиторием данных dataRepo и репозиторием данных с проверкой безопасности securedDataRepo внедрен репозиторий данных с поддержкой кеширования cachedDataRepo. Данные загружаются по цепочке: БД -> dataRepo -> cachedDataRepo -> securedDataRepo.

При этом, cachedDataRepo, при запросе, сначала проверяет наличие данных в кеше (если кеширование активировано), и если данных в кеше нет, запрашивает их у dataRepo, после чего помещает в кеш и отдает securedDataRepo.

Настройка кеширования объектов отдельных классов

Можно индивидуально в файле настроек конфигурации приложения (deploy.json), в опциях cachedDataRepo указать список классов, данные которых нужно хранить в кеше.

Пример:

```
...
"options": {
  "cachedClasses": ["class1@namespace", "class2@namespace", "class3@namespace"]
}
...
```

Если cachedClasses не указан, то кешируются все данные.

Кеширование списка объектов

Кешируются как индивидуальные обьекты, так и списки.

При кешировании списка на основании его условий выборки (в т.ч. пагинации) формируется ключ, выполняется выборка и все полученные обьекты кешируются индивидуально, а список кешируется как массив идентификаторов обьектов. При выборке списка из кеша на основании этого массива из кеша выбираются соответствующие обьекты и формируется результирующий список.

Аналогично кешируются все объекты по ссылке и коллекции загружаемые жадно.

Настройка гибкого кеширования

Для расширенной конфигурации настроек кеширования (таких как, время хранения запросов, глубина хранения объектов в кеше и др.) применяется настройка гибкого кеширования объектов. Настройка гибкого кеширования задается в файле deploy.json в свойстве "di", в компонентах memcached и redis в опции connectOptions. Все нужные плейсхолдеры прописаны в файле config.json репозитория платформы.

NB: Глубина хранения обьектов в кеше соответствует глубине запроса обьектов из БД, то есть в обьекте хранится информация о его ссылках, а дальше в кеше получается граф обьектов, а не дерево.

Настройка кеширования через ini файл приложения

Параметры ini файла:

```
...
cache.module=redis // - модуль используемый репозиторием данных для кеширования.
//(варианты: ion://memcached, ion://redis, ion://innerCache)

cache.memcached.enabled=true // активируем memcached
cache.memcached.location1 // сервер1
cache.memcached.location2 // сервер2
cache.memcached.timeout // таймаут обращения к кэшу

cache.redis.enabled=true // активируем redis
cache.redis.host=127.0.0.1 // хост redis
cache.redis.port=6379 // порт redis
```

(continues on next page)

(continued from previous page)

```
session.type=redis // хранение сессий авторизации в redis
```

По умолчанию кеширование не используется.

3.2.6 Фильтры на форме представления списка

Если для даты значение в поле фильтра и значение в поле атрибута имеют разный формат, то фильтр по такому полю работать НЕ БУДЕТ

Запрос для фильтра задается выражением (поисковым запросом).

Поддерживает следующие операции:

- группировки скобками
- логических: AND, OR, NOT
- сравнения: =, <, >, <=, >=, <>
- арифметических: +, -, *, /
- строковых: like
- над коллекциями: size

Создание запроса

Выбор атрибута выполняется из выпадающего списка, с помощью кнопки >_, расположенной у основания поля запроса для фильтра. Наименование атрибута оборочивается в “backticks” т.е.:

```
`Наименование атрибута` != 2
```

Параметры комбинации значений атрибутов для запроса:

- and - И - т.е. обязательно оба (или больше) значаний,
- or - ИЛИ - т.е. любое из значений оба (или больше) значаний.

пример комбинации:

```
`Атрибут1` = 1 AND `Атрибут2` != 2
```

Строковые значения атрибутов при формировании запроса оборачиваются в двойные кавычки:

```
`Название поля` != "привет"
```

Обращение к атрибутам по ссылке:

```
`Атрибут1`.`Атрибут по ссылке из Атрибут 1` = "значение"
```

Подсказки:

В конце поля запроса для фильтра расположен знак ?, при клике на который откроется модельное окно с описанием принципа работы фильтра и синтаксиса запроса к нему.

Для парсинга поисковых выражений используется библиотека https://nearley.js.org/

Варианты использования

Помимо кнопки рядом со строкой поиска в верхней части страницы, фильтр можно вызывать кликом на аналогичного вида значек, расположенный в каждом столбце таблицы.

Для создания запроса для фильтра необходимо выбрать из выпадающего списка значение, или же начать ввод значения в строку. Как только значение выбрано необходимо нажать клавишу Enter - в столбце со значениями отобразится результат запроса.

3.2.7 Связь двух неймспейсов

Связь нескольких проектов с использованием пространства имен “namespace”

Применяется при необходимости связывать между собой 2 проекта и разворачивать их на одной платформе, с указанием ссылок и коллекций не только внутри проекта, но и на другие, развернутые в этом же контексте. Реализуется с применением пространства имен проекта, т.е. что бы было понятно к какому проекту относятся классы по ссылке. Так, например, связываем тестовый проект develop-and-test с проектом fias:

Пример

```
{
  "namespace": "develop-and-test",
  "code": "projectJoin.addressExt",
  "orderNumber": 0,
  "type": 1,
  "title": "",
  "caption": "Адрес ФИАС",
  "classname": "address@fias",
  "container": null,
  "collection": null,
  "url": null,
  "hint": null,
  "conditions": [],
  "sorting": [],
  "pathChains": [],
  "metaVersion": "2.0.61.21119"
}
```

В меню проекта develop-and-test есть узел навигации, который является представлением класса из проекта fias. Таким образом при переходе по пункту меню одного проекта - получаем данные из другого проекта, при указании необходимого пространства имен.

Описание

Для меты:

При импорте неймспейс берется не из параметра, а из меты класса. Убираем неймспейс как параметр навигации и т.д.

При импорте меты навигации, при совпадении системных имен приоритет имеет мета, накатываемая последней, но с учетом заполненности атрибутов. т.е. пустое значение атрибута не переопределяет непустое.

Для студии:

В мете класса добавляется свойство namespace. В студии в настройках проекта ION добавлен параметр - “пространство имен”. Это пространство имен по умолчанию проставляется в соответствующее поле на форме создания класса.

Для атрибута “пространство имен” также заданы отдельные поля на формах создания и редактирования класса. В студии теперь системное имя класса везде отображается с неймспейсом через знак @, если класс объявлен в неймспейсе отличном от неймспейса текущего проекта.

Поле неймспейса в виде выпадающего списка, откуда можно выбрать либо неймспейс проекта, либо неймспейсы связанных проектов (Project references). При этом есть возможность и вбить его руками.

У остальных объектов меты неймспейса нет, но ссылки на классы везде должны проставляться с учетом неймспейса (в ссылочных атрибутах тоже). Это значит, что в списках выбора классов должно быть полное имя класса с неймспейсом.

Также в списки выбора классов должны попадать классы из всех Project References. На физическом уровне неймспейс включается в имя файла меты класса (и директорий, если они участвуют в формировании логики привязки к классу, например, у меты представления).

3.2.8 Электронно-цифровая подпись

Описание

Электронно-цифровая подпись (ЭЦП) - это реквизит электронного документа, предназначенный для защиты данного электронного документа от подделки, полученный в результате криптографического преобразования информации с использованием закрытого ключа электронной цифровой подписи. Он позволяет идентифицировать владельца сертификата ключа подписи и установить отсутствие искажения информации в электронном документе.

Цель использования

В приложении ЭЦП может быть нужна для:

- Проверки целостности данных
- Определения авторства данных

Существует три вида электронной цифровой подписи, которые отличаются по своему применению:

- Простая электронно-цифровая подпись
  - для определения авторства данных
  - создается с использованием кодов, паролей или иных средств
- Усиленная неквалифицированная электронно-цифровая подпись
  - для проверки целостности данных
  - для определения авторства данных

  - создается с использованием средств электронной подписи
- Усиленная квалифицированная электронно-цифровая подпись
  - для проверки целостности данных
  - для определения авторства данных
  - для создания и проверки электронной подписи используются средства электронной подписи, получившие подтверждение соответствия требованиям законодательства

Специфика работы

Работает утилита ЭЦП на основе КриптоПро, поэтому он должен быть установлен на одном компьютере:

- ставим крипто про
- ставим плагин
- для тестирования, выпускаем сертификат

Реализация

ЭЦП можно отнести к утилитам для приложения, так как основная ее реализация находится в приложении. Обычно реализация ЭЦП находиться в папке приложения lib/digest (на примере приложения project-management):

- lib/digest/digestData.js - проверка при загрузке формы объекта на необходимость в электронной подписи (_applicable) и процесс подписи при выполнении перехода БП (_process)
- lib/digest/signSaver.js - прикрепление подписи к объекту

Для того, чтобы статус ЭП запрашивался/отображался, для registry добавляем в deploy настройку signedClasses.

Пример

```
"modules": {
   "registry": {
     "globals": {
       "signedClasses": [
        "class@application"
       ],
...
```

В БП workflows/indicatorValueBasic.wf.json добавляем переход со свойством "signBefore": true.

Пример

```
{
    "name": "needAppTrs_sign",
    "caption": "На утверждение",
    "startState": "edit",
    "finishState": "onapp",
```

(continues on next page)

(continued from previous page)

```
  "signBefore": true,
  "signAfter": false,
  "roles": [],
  "assignments": [
    {
      "key": "state",
      "value": "onapp"
    }
  ],
  "conditions": []
}
```

3.2.9 Виртуальные атрибуты

Описание

Виртуальные атрибуты - специальный тип атрибутов, указывается в property наименование атрибута, начинающееся с __. Функционал каждого виртуального атрибута различен, существуют следующие виртуальные атрибуты:

- __class - выводит полный код класса, в котором заведен атрибут
- __classTitle - выводит наименование класса, в котором заведен атрибут

3.2.10 Утилиты

Описание

Утилиты - дополнительные программы для более специализированного применения.

Утилиты ядра

Утилиты ядра являются самыми необходимыми утилитами от установки до эксплуатации приложения и лежат в папке bin. Перед их запуском необходимо указать переменную окружения NODE_PATH, если она не задана заранее (с указанием на папку ядра).

В состав ядра входят утилиты:

- bin/acl.js - для импорта и редактирования настроек безопасности приложения (ресурсы, роли, пользователи, права доступа). Параметры запуска:
  - --u - указывается пользователь
  - --res - указывается ресурс
  - --role - указывается роль
  - --p - указывается доступ
  - --m - указывается метод применения доступа
  - --d - указывается для импорта директория с настройками безопасности

- bin/adduser.js - для добавления новых пользователей в настройки безопасности приложения. Параметры запуска:
  - --name - указывается логин пользователя (по умолчанию admin)
  - --pwd - указывается пароль пользователя (по умолчанию admin)
- bin/bg.js - для запуска фоновых процедур с низким приоритетом, требующих больших мощностей от процессора или выполняющихся относительно длительно по времени
- bin/export.js - для экспорта приложения в локальную директорию. Параметры запуска:
  - --dst - путь к директории, в которую будет записан результат экспорта (по умолчанию ../out)
  - --ns - указывается неймспейс приложения
  - --file-dir - путь к директории, в которую будут экспортированы файлы из файловых атрибутов
  - --acl - дополнительно экспортировать настройки безопасности
  - --nodata - пропустить экспорт для всех созданных объектов в приложении
  - --nofiles - пропустить экспорт файловых атрибутов
  - --ver - версия (последняя версия -last)
- bin/import.js - для импорта метаданных приложения. Параметры запуска:
  - --src - путь к директории, из которой будет происходить импорт (по умолчанию ../in)
  - --ns - неймспейс приложения
  - --ignoreIntegrityCheck - при импорте игнорируется целостность данных
- bin/import-data.js - для импорта данных приложения. Параметры запуска:
  - --src - путь к директории, из которой будет происходить импорт (по умолчанию ../in)
  - --ns - неймспейс приложения
- bin/job-runner.js - для запуска задач по расписанию
- bin/job.js - для запуска рабочего компонента задания из утилиты job-runner
- bin/meta-update.js - для конвертации меты приложения из одной версии в другую
- bin/schedule.js - для ручного запуска задач по расписанию
- bin/setup.js - для установки настроек deploy из приложения. Параметры запуска:
  - --reset - предварительный сброс всех настроек deploy в приложении
  - --sms - при сбросе настроек, не зачищаются настройки deploy помеченные как важные
  - --rwa - переопределять, а не дополнять массивы в настройках deploy

Утилиты приложения

Утилиты приложения реализуют специфичный функционал приложения на этапе эксплуатации, которое пока не реализовано в ядре в унифицированном виде для различных приложений. Как правило, данные утилиты находятся в директории lib и подключаются к приложению через deploy.

Примеры реализованных утилит для приложения sakh-pm:

- lib/actions/createIndicatorValueHandler.js - утилита создания показателей в коллекции за выбранный период

- lib/actions/createProjectReportsHandler.js - утилита автоматического создания печатных форм по проекту с сохранением файлов в облако
- lib/actions/assignmentToEventOnly.js - утилита, формирующая из поручения контрольную точку

Создание утилиты для приложения на примере createIndicatorValueHandler

Реализация

Для реализации как правило используется язык JavaScript с использованием доступного функционала модулей, входящих в приложение.

При реализации утилит в приложении с относительно большим функционалом сами файлы могут быть разбиты на несколько зависимых файлов.

В приведенном примере в главном файле утилиты lib/actions/createPlanIndicatorsHandler.js должен быть экспорт последней строкой:

```
module.exports = CreatePlanIndicatorsHandler;
```

Подключение к приложению

Для запуска утилиты при эксплуатации приложения нужно настроить параметры подключения в deploy.

Для примера нужно сначала для утилиты добавить элемент интерфейса в представлении, который будет запускать утилиту. Для этого в файле views/indicatorFinancial/item.json нужно добавить кнопку CREATE_INDICATOR_VALUE:

```
{
    "id": "CREATE_INDICATOR_VALUE",
    "caption": "Сформировать собираемые значения",
    "visibilityCondition": null,
    "enableCondition": null,
    "needSelectedItem": false,
    "signBefore": false,
    "signAfter": false,
    "isBulk": false
}
```

Затем нужно добавить настройки в deploy, чтобы связать кнопку в интерфейсе CREATE_INDICATOR_VALUE и утилиту createIndicatorValueHandler:

```
"modules": {
   "registry": {
     "globals": {
        "di": {
          "createIndicatorValueHandler": {
             "module": "applications/sakh-pm/lib/actions/createIndicatorValueHandler",
             "initMethod": "init",
             "initLevel": 2,
             "options": {
             "data": "ion://securedDataRepo",
             "workflows": "ion://workflows",
             "log": "ion://sysLog",
```

(continues on next page)

(continued from previous page)

```
            "changelogFactory": "ion://changelogFactory",
            "state": "onapp"
            }
        },
        "actions": {
            "options": {
            "actions": [
                {
                "code": "CREATE_INDICATOR_VALUE",
                "handler": "ion://createIndicatorValueHandler"
                }
            ]
        }
    }
```

В примере все настройки хранятся для модуля registry, так как из него будет вызвана утилита при нажатии на кнопку CREATE_INDICATOR_VALUE в форме объекта класса indicatorFinancial.

Дополнительная информация

Настройки модулей в deploy.json

Мета представлений - Действия

3.3 Локализация

Локализация фреймворка, модулей и приложений организована по папкам i18n в соответствующих директориях.

Чтобы перевести строку для определенной локали, нужно добавить ее и ее перевод в:

```
<корень фреймворка, приложения или модуля>/i18n/<код локали>/index.js
```

в экспортируемый объект, например:

```
module.exports = {
    'Контрольная панель': `Control panel`
};
```

Здесь строка ‘Контрольная панель’, при ее загрузке, будет переведена как ‘Control Panel’

3.4 Структура метаданных

3.4.1 Мета классов - общая часть

Признак абстрактности класса

Признак абстрактности класса - требуется в случае, когда необходимо по ссылке атрибута на базовый класс отображать список выбора его наследников. При формировании списка выбора классов для создания обьекта абстрактные классы не включены. Выставите true в поле “abstract”.

Пример

```
{
  "name": "SomeClassName",
  "abstract": true
}
```

Класс становится недоступным для инициализации на уровне UI.

Пример

```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "value|[|plannedValue|](|dateStart|-|dateEnd|)",
  "name": "indicatorValueBasic",
  "version": "",
  "caption": "Значения показателей на период",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": true,
  "abstract": true,
  "compositeIndexes": [
    {
      "properties": [
        "indicatorBasic",
        "dateStart",
        "dateEnd"
      ],
      "unique": true
    }
  ],
  "properties": [
...
```

Наследование

Наследование - позволяет создать новый класс на основе уже существующего (родительского), при этом атрибуты родительского класса заимствуются новым классом. Родительский класс может иметь несколько наследников с различным атрибутивным составом. Каждый наследник может включать в себя свой индивидуальный состав атрибутов + атрибуты родительского класса.

Атрибутивный состав

В мете родительского класса атрибутивный состав формируется таким образом, чтобы каждый из них можно было использовать для какого-либо из классов наследников. В то время как в мете класса наследника атрибутивный состав для каждого свой и применяется только на класс, в котором находится.

Пример

Родительский класс:

```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "name",
  "name": "organization",
  "version": "",
  "caption": "Организация",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": true,
  "compositeIndexes": [],
  "properties": [
```

Класс-наследник:

```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "medicalOrg",
  "version": "",
  "caption": "Медицинская организация",
  "ancestor": "organization",
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": true,
  "compositeIndexes": [],
  "properties": [
```

"id" - является уникальным идентификатором для всех его наследников. Хранится в родительском классе.

"name" - в классе есть атрибут “Наименование”, который может отображаться в любом из наследников, или в одном из них. То есть, если атрибут "name" задается в родительском классе, то он может отображаться в любом из наследников. Но если "name" задается в классе наследнике, то отображается только в том классе, в котором он был задан.

Представление класса:

Если для родительского класса задан признак абстрактности, то представление для него задавать не обязательно.

Представления класса задается для каждого наследника по отдельности, с указанием необходимого атрибутивного состава (из класса, для которого создается + необходимые из родительского класса).

Настройка списка классов наследников для создания объектов по ссылке

Задается в мете класса для атрибута типа "Ссылка"/"Коллекция" после указания ссылочного класса/класса коллекции:

```
"itemsClass": "event",
"allowedSubclasses": [
        "Subclasses1",
        "Subclasses2"
    ],
```

itemsClass - коллекция на базовый класс [event]

Subclasses1 - дочерний класс базого класса [event], который будет отображаться в списке, при создании объекта по ссылке (далее перечисляем все дочерние классы, которые нужно отображать в списке).

NB: Если данная настройка не задана - при создании, в списке отображаются все дочерние классы.

Условия для применения настройки:

- Тип атрибута “Коллекция” или “Ссылка”;
- Для атрибута типа “Коллекция”, “Ссылка” указан класс ссылки/коллекции на родительский (базовый) класс (при создании объекта ссылочного класса выводиться окно выбора нескольких классов);
- Помимо скрытия базового класса, при создании объекта не нужно отображать все дочерние классы в списке выбора классов для создания объекта по ссылке.

Пример

Родительский класс [Мероприятия] имеет несколько классов наследников ([Мероприятие1], [Мероприятие3], [Мероприятие2]). В классе [Проект] есть атрибут типа “Коллекция”, который ссылается на родительский класс [Мероприятие] :

```
{
    "namespace": "ns",
    "isStruct": false,
    "key": [],
    "semantic": "",
    "name": "project",
    "version": "",
    "caption": "Проект",
    "ancestor": "",
    "container": null,
    "creationTracker": "",
```

(continues on next page)

(continued from previous page)


```
  "changeTracker": "",
  "creatorTracker": "",
  "editorTracker": "",
  "history": 0,
  "journaling": true,
  "compositeIndexes": [],
  "properties": [
   {
    "orderNumber": 80,
    "name": "event",
    "caption": "Мероприятия",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": true,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "event@ns",
    "allowedSubclasses": [
        "event1",
        "event2"
    ],
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
   }
 ...
```


В случае, если для класса задана настройка абстрактности, то при создании объекта класса [Мероприятие] в коллекцию, в списке выбора отобразятся те наследники класса [event], которые указаны в свойстве "allowedSubclasses". То есть, исходя из примера, в коллекцию “Мероприятия” можно создать только объекты класса “Мероприятие1” и “Мероприятие2”.

Многоуровневая иерархия

Дочерние классы могут унаследовать атрибутивный состав не только от своих прямых родительских классов, но и от тех, которые находятся выше по иерархии наследования.

Пример

[basicObj] - родительский класс ->> [eventBasic] - наследник класса [basicObj] ->> [eventBlock] - наследник класса [eventBasic].


```
{
  "isStruct": false,
  "key": [
    "guid"
  ],
  "semantic": "name",
  "name": "basicObj",
  "abstract": true,
  "version": "31",
  "caption": "Учетный объект",
  "ancestor": null,
  "container": null,
  "creationTracker": "createDatet",
  "changeTracker": "modifeDate",
  "creatorTracker": "creator",
  "editorTracker": "editor",
  "history": 0,
  "journaling": true,
  "compositeIndexes": null,
  "properties": [
```

```
{
  "isStruct": false,
  "key": [
    "guid"
  ],
  "semantic": "name| ( |code| )",
  "name": "eventBasic",
  "version": "31",
  "caption": "Базовое мероприятие",
  "ancestor": "basicObj",
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": true,
  "compositeIndexes": null,
  "abstract": true,
  "properties": [
```

```
{
  "isStruct": false,
  "key": [
    "guid"
  ],
  "semantic": "name| ( |code| )",
  "name": "eventBlock",
  "version": "31",
  "caption": "Блок мероприятий",
  "ancestor": "eventBasic",
  "container": null,
  "creationTracker": "",
```


(continues on next page)

(continued from previous page)

```
  "changeTracker": "",
  "history": 0,
  "journaling": true,
  "compositeIndexes": null,
  "properties": [
```

Наследник [eventBlock] будет так же наследовать атрибутивный состав родительского класса [basicObj], как и наследник [eventBasic].

Индексация

Индексация - составные уникальные поля. Используется для поиска и контроля целостности данных.

В общей части меты классов есть поле "compositeIndexes", которое позволяет задать требование уникальности сочетания полей.

Описание

Составной индекс задается перечислением входящих в него атрибутов и указанием признака уникальности. Когда в классе присутсвует составной индекс, при сохранении объекта в базе проверяется отсутвие одинаковых сочетаний перечисленных полей. То есть значения полей "protocol" и "family" из примера ниже могут повторятся, но пара значений - всегда уникальна.

Пример:

```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "protocol|family",
  "name": "refusal",
  "version": "",
  "caption": "Письменные отказы граждан",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": [
    {
      "properties": [
        "protocol",
        "family"
      ],
      "unique": true
    }
  ],
  "properties": [
```

(continues on next page)

(continued from previous page)


```
  {
    "orderNumber": 10,
    "name": "id",
    "caption": "Идентификатор",
    "type": 0,
    "size": 24,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": false,
    "readonly": true,
    "indexed": false,
    "unique": true,
    "autoassigned": true,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 20,
    "name": "protocol",
    "caption": "Протокол заседания комиссии",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
```


(continues on next page)

(continued from previous page)


```
    },
    {
      "orderNumber": 31,
      "name": "family",
      "caption": "Семья, поставленная на учет",
      "type": 13,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": true,
      "readonly": false,
      "indexed": false,
      "unique": false,
      "autoassigned": false,
      "hint": null,
      "defaultValue": null,
      "refClass": "family",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    }
  ]
}
```

Журналирование

Журналирование - указывает на необходимость журналирования всех действий произведенных над объектом. Находится в поле journaling основной части меты класса. Значение true указывает на необходимость журналирования. Если в поле указано значение false или отсутсвует, то журналирование изменений объекта не требуется.

Пример:


```
{
  "isStruct": false,
  "key": "okato",
  "semantic": "name",
  "name": "naselenniyPunkt",
  "caption": "Населенный пункт",
  "journaling": true,
  "ancestor": null,
  "container": "",
  "creationTracker": "",
```


(continues on next page)

(continued from previous page)

```
  "changeTracker": "",
  "properties": []
}
```

Ключевые поля

В каждом классе обязательно должен быть задан ключевой атрибут (поле "key" основной части меты класса). Без этого приложение не будет функционировать.

Типы ключевых полей

1. Guid - “Глобальный идентификатор [12]”.
2. “Строка [0]”.
3. “Целое [6]”.

Требования к ключевому атрибуту

При формировании меты ключевого атрибута, поля "unique" и "autoassigned" выставляются в true. Необходимо запретить пустое значение, выставляя "nullable" в false.

Если атрибут не генерируется автоматически, то "autoassigned" можно поставить в false, тогда поле должно быть задано оператором при создании. Например, если это код классификатора или регистрационный номер задаваемый внешним способом (на бумаге).

Версионирование

Версионирование - позволяет хранить в системе сразу несколько версий метаданных. В каждый объект при изменении и сохранении проставляется его версия. Таким образом, версионирование предоставляет возможность работы с различными версиями одних объектов.

Версионирование задается в поле "version" основной части меты классов. Чтобы изменить версию меты, необходимо добавить атрибут version(“version” : 2).

Механизм работы

При загрузке метаданных, если есть атрибут version (“version” : 2) - будет закачена мета с версией, иначе версия = 1.

```
{
  "isStruct": false,
  "key": "id",
  "semantic": "caption",
  "name": "ion_filter",
  "caption": "Фильтры",
```

(continues on next page)

(continued from previous page)

```
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "version" : 2
}
```

При создании объектов к ним будет добавляться последняя версия метаданных из текущих в базе, а при редактировании объектов они будут редактироваться на основании сохраненной версии.

Пример сохраненных объектов с разными версиями в базе:

```
{
    "_id" : ObjectId("567cfa1eb869fc2833690ea4"),
    "id" : 24006,
    "class" : "ALL",
    "caption" : "11",
    "html" : "",
    "filter" : "[\"{\\\"property\\\":\\\"contact\\\",\\\"operation\\\":20,\\\"value\\\":\\\"11\\\",\\\"title\\\
↪":\\\"Контактная информация содержит 11\\\",\\\"type\\\":7}\"]",
    "period" : "2015-12-08,2016-02-05",
    "version" : 1,
    "semanticTitle" : "11 "
}

{
    "_id" : ObjectId("56944e5cb73f51ec182c7369"),
    "class" : "ALL",
    "caption" : "ffffff",
    "filter" : "[\"{\\\"property\\\":\\\"class\\\",\\\"operation\\\":0,\\\"value\\\":\\\"fff\\\",\\\"title\\\":\\\
↪"Класс фильтра равно fff\\\",\\\"type\\\":1}\"]",
    "version" : 2,
    "id" : NaN,
    "semanticTitle" : "ffffff "
}
```

Работа в коде

При считывании меты классов, данные разделяются по версиям. Имена версионированных классов имеют имена следующего вида “<имякласса><номер версии>”. Например, ion_filter_1, ion_filter_2 – Класс ion_filter версии 1 и 2 соответственно.

При выборке объектов, данные берутся с учётом версии. Версия объекта передаётся в виде параметра version запроса на открытие объекта.

Семантика

Семантика - используется для вывода объекта класса в качестве одной строки заголовка класса.

В мете классов поле "semantic" встречается дважды:

1. в общей части меты класса, где формирует строковое представление для данного класса,

2. в мете атрибута класса, где формирует строковое представление объектов класса, на который ссылается атрибут, т.е. используется для ссылочных атрибутов.

Цель использования

Используется для корректировки отображения атрибутов и значений атрибутов в списке. В атрибутах, которые выводят табличные данные, семантика используется для ограничения вывода колонок.

Примеры использования в ссылочных атрибутах

Например, есть класс class, у которого есть атрибуты: id, name, link, date. Есть второй класс classTable, у которого есть ссылочный атрибут table на класс class. Без использования семантики в объектах класса classTable в атрибуте table будут выводится значения идентификаторов объектов класса class. Атрибуты, используемые как идентификаторы, указаны в мете класса class.

Чтобы вывести значения атрибутов name и link в атрибуте table, а не значения идентификаторов, нужно прописать "semantic": "name|link". В зависимости от типа атрибута результат будет разный:

- Если атрибут table является ссылкой, то в нем будут заполнены значения атрибутов name и link через пробел. Тут можно использовать дополнительные слова и выражения через знак |, например "semantic": "name|, |link" или "semantic": "У объекта есть 2 атрибута:|name|, |link".
- Если атрибут table является коллекцией объектов класса class, то в нем будут выведеные колонки name и link.

Формат отображения в семантике

- Можно обрезать вывод с помощью: [].

```
"name[0,50]|..."
Указываем позицию и количество выводимых букв из семантики атрибута. Из атрибута name␣
↪выводим 50 символов семантики (значение атрибута), начиная с первого.
```

- Доступно разыменование через . т.е. доступ во вложенный объект.

```
"semantic": "digitalTV|kachestvoCTB|analogTV.name|kachestvoAnal|period"
где ``analogTV`` - ссылочный атрибут класса, для которого задается семантика, а ``name`` -␣
↪атрибут класса по ссылке.
```

Отображение семантики на форме

1. В списках первого уровня (открываемые непосредственно по узлу навигации), в качестве заголовка выводим только значение из поля “caption” узла навигации.
2. В списках выбора в заголовок выводим только значение из поля “caption” класса объектов списка.
3. В форме редактирования в заголовок выводим только семантику объекта.
4. В форме создания в заголовок выводим только значение из поля “caption” класса.
5. В списках выбора над заголовком мелким шрифтом выводим строку “Выбор значения атрибута <...> объекта <...>”.

6. В форме создания, если создается объект в коллекции или ссылке, над заголовком мелким шрифтом выводим строку “Создание объекта в коллекции/по ссылке <...> объекта <...>”.

Метки времени создания и изменения

Речь идет о следующих полях общей части меты классов:

1. "creationTracker" - Метка времени создания: позволяет сохранять в классе дату/время создания объекта, требует наличия соответствующего атрибута класса, "name" которого и вносится в данное поле.
2. "changeTracker" - Метка времени изменения: позволяет сохранять в классе дату/время изменения объекта, требует наличия соответствующего атрибута класса, "name" которого и вносится в данное поле.

Пример


```
{
  "isStruct": false,
  "key": "id",
  "semantic": "rtrs",
  "name": "digitTv",
  "caption": "Цифровое ТВ",
  "ancestor": null,
  "container": null,
  "creationTracker": "createDate",
  "changeTracker": "modifeDate",
  "properties": [
    {
      "orderNumber": 60,
      "name": "createDate",
      "caption": "Метка времени создания",
      "type": 9,
      "size": null,
      "decimals": 0,
      "nullable": false,
      "readonly": false,
      "indexed": false,
      "unique": true,
      "autoassigned": true,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "selConditions": [],
      "selSorting": [],
      "selection_provider": null,
      "indexSearch": false,
      "eagerLoading": false
    },
    {
```


(continues on next page)

(continued from previous page)


```
    "orderNumber": 70,
    "name": "modifeDate",
    "caption": "Метка времени изменения",
    "type": 9,
    "size": null,
    "decimals": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "selConditions": [],
    "selSorting": [],
    "selection_provider": null,
    "indexSearch": false,
    "eagerLoading": false
   }
  ]
}
```


Метки пользователя создавшего и измененившего объект

1. "creatorTracker" - Метка пользователя создавшего: позволяет сохранять в классе имя пользователя, создавшего объект, требует наличия соответствующего атрибута класса, "name" которого и вносится в данное поле.
2. "editorTracker" - Метка пользователя изменившего: позволяет сохранять в классе имя пользователя, изменившего объект, требует наличия соответствующего атрибута класса, "name" которого и вносится в данное поле.


```
{
  "isStruct": false,
  "key": [
    "guid"
  ],
  "semantic": "name",
  "name": "basicObj",
  "abstract": true,
  "version": "31",
  "caption": "Учетный объект",
  "ancestor": null,
  "cacheDependencies": [
    "basicObj"
  ],
  "container": null,
  "creatorTracker": "creator",
  "editorTracker": "editor",
  "history": 0,
  "journaling": true,
```


(continues on next page)

(continued from previous page)


```
"compositeIndexes": null,
"properties": [
  ...
  {
    "orderNumber": 20,
    "name": "creator",
    "caption": "Метка пользователя, создавшего объект",
    "type": 18,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": true,
    "hint": "",
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "editor",
    "caption": "Метка пользователя, изменившего объект",
    "type": 18,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": true,
    "hint": "",
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
```


(continues on next page)

(continued from previous page)

```
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  }
 ],
 "metaVersion": "2.0.61"
   }
```

Тип Структура [16]

Структура - способ отображения связанных объектов (ссылок). Если у поля указано структура, то данный тип атрибута нужен для уменьшения действий при заведении модели, при заранее известных типовых классах, атрибуты которых используются во многих других классах. Для класса-структуры в основной части меты класса задается значение true в поле "isStruct".

Для атрибутов типа “Структура [16]” в мете представлений создания и изменения используется тип “Группа [0]”. Если не указывать поля у группы, они создаются автоматически по мете. В представлениях списка нет необходимости делать колонки для атрибутов-структур, в объекте не будет такого свойства, а будут соответствующие атрибуты класса-структуры. Для них можно добавить колонки.

NB: Объекты класса-структуры не создаются!

Пример

Класс-структура:

```
{
 "isStruct": true,
 "key": [
  "id"
 ],
 "semantic": "",
 "name": "is_struct",
 "version": "",
 "caption": "\"isStruct\" класс-структура",
 "ancestor": null,
 "container": null,
 "creationTracker": "",
 "changeTracker": "",
 "history": 0,
 "journaling": false,
 "compositeIndexes": null,
 "properties": [
  {
   "orderNumber": 10,
   "name": "id",
   "caption": "Идентификатор",
   "type": 12,
   "size": null,
   "decimals": 0,
   "allowedFileTypes": null,
```

(continues on next page)

(continued from previous page)


```
    "maxFileCount": 0,
    "nullable": false,
    "readonly": false,
    "indexed": false,
    "unique": true,
    "autoassigned": true,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 20,
    "name": "last_name",
    "caption": "Фамилия",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "first_name",
    "caption": "Имя",
    "type": 0,
    "size": null,
```


(continues on next page)

(continued from previous page)


```
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 40,
    "name": "patronymic",
    "caption": "Отчество",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 50,
    "name": "date",
    "caption": "Дата рождения",
```


(continues on next page)

(continued from previous page)


```
    "type": 9,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  }
 ]
}
```


Класс с атрибутом типа “Структура [16]”:


```
{
 "isStruct": false,
 "key": [
  "id"
 ],
 "semantic": "",
 "name": "struct",
 "version": "",
 "caption": "Класс \"Структура [16]\" (класс с типом атрибута 16 - структура)",
 "ancestor": null,
 "container": null,
 "creationTracker": "",
 "changeTracker": "",
 "history": 0,
 "journaling": false,
 "compositeIndexes": null,
 "properties": [
  {
   "orderNumber": 10,
   "name": "id",
   "caption": "Идентификатор",
   "type": 12,
   "size": null,
   "decimals": 0,
   "allowedFileTypes": null,
   "maxFileCount": 0,
```


(continues on next page)

(continued from previous page)


```
      "nullable": false,
      "readonly": false,
      "indexed": false,
      "unique": true,
      "autoassigned": true,
      "hint": null,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    },
    {
      "orderNumber": 20,
      "name": "struct",
      "caption": "Класс \"Структура [16]\"",
      "type": 16,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": true,
      "readonly": false,
      "indexed": false,
      "unique": false,
      "autoassigned": false,
      "hint": null,
      "defaultValue": null,
      "refClass": "is_struct",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    }
  ]
}
```


Объект класса с атрибутом-структурой в базе:


```
{
    "_id" : ObjectId("57c3e46fd53ecd50123cc4f5"),
```


(continues on next page)

(continued from previous page)

```
    "struct$id" : "5f421610-6dba-11e6-874f-1b746e204b07",
    "struct$last_name" : "Мирошниченко",
    "struct$first_name" : "Ирина",
    "struct$patronymic" : "Львовна",
    "struct$date" : ISODate("1978-07-13T14:00:00.000Z"),
    "id" : "5f41ef00-6dba-11e6-874f-1b746e204b07",
    "_class" : "struct@develop-and-test",
    "_classVer" : ""
}
```

Тип Структура

Структура - способ отображения связанных объектов (ссылок). Если у поля указано структура, то данный тип атрибута нужен для уменьшения действий при заведении модели, при заранее известных типовых классах, атрибуты которых используются во многих других классах.

Для класса-структуры в основной части меты класса задается значение true в поле "isStruct".

Для атрибутов типа “Структура [16]” в мете представлений создания и изменения используется тип “Группа [0]”. Если не указывать поля у группы, они создаются автоматически по мете. В представлениях списка нет необходимости делать колонки для атрибутов-структур, в объекте не будет такого свойства, а будут соответствующие атрибуты класса-структуры. Для них можно добавить колонки.

NB: Объекты класса-структуры не создаются!

Пример

Класс-структура:

```
{
  "isStruct": true,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "is_struct",
  "version": "",
  "caption": "\"isStruct\" класс-структура",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
  "properties": [
    {
      "orderNumber": 10,
      "name": "id",
      "caption": "Идентификатор",
      "type": 12,
      "size": null,
```

(continues on next page)

(continued from previous page)


```
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": false,
    "readonly": false,
    "indexed": false,
    "unique": true,
    "autoassigned": true,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 20,
    "name": "last_name",
    "caption": "Фамилия",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "first_name",
    "caption": "Имя",
```


(continues on next page)

(continued from previous page)


```
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 40,
    "name": "patronymic",
    "caption": "Отчество",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 50,
```


(continues on next page)

(continued from previous page)


```
      "name": "date",
      "caption": "Дата рождения",
      "type": 9,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": true,
      "readonly": false,
      "indexed": false,
      "unique": false,
      "autoassigned": false,
      "hint": null,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    }
  ]
}
```


Класс с атрибутом типа “Структура [16]”


```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "struct",
  "version": "",
  "caption": "Класс \"Структура [16]\" (класс с типом атрибута 16 - структура)",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
  "properties": [
    {
      "orderNumber": 10,
      "name": "id",
      "caption": "Идентификатор",
      "type": 12,
      "size": null,
      "decimals": 0,
```


(continues on next page)

(continued from previous page)


```
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": false,
    "readonly": false,
    "indexed": false,
    "unique": true,
    "autoassigned": true,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
   },
   {
    "orderNumber": 20,
    "name": "struct",
    "caption": "Класс \"Структура [16]\"",
    "type": 16,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "is_struct",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
   }
  ]
}
```


Объект класса с атрибутом-структурой в базе:

```
{
    "_id" : ObjectId("57c3e46fd53ecd50123cc4f5"),
    "struct$id" : "5f421610-6dba-11e6-874f-1b746e204b07",
    "struct$last_name" : "Мирошниченко",
    "struct$first_name" : "Ирина",
    "struct$patronymic" : "Львовна",
    "struct$date" : ISODate("1978-07-13T14:00:00.000Z"),
    "id" : "5f41ef00-6dba-11e6-874f-1b746e204b07",
    "_class" : "struct@develop-and-test",
    "_classVer" : ""
}
```

Общая часть меты классов - содержит поля параметров класса, которые имеют отношения к самой структуре и способам оперирования данными в ней.

JSON


```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "class_integer",
  "abstract": true,
  "version": "",
  "caption": "Класс \"Целое [6]\"",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "creatorTracker": "",
  "editorTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
  "properties": [...]
}
```

Описание полей

Код	Имя	Допустимые значения	Описание
"isStruct"	Является структурой	Логическое.	Если выставлено “true” - это говорит о том, что данный класс является структурой, и может быть использован в прочих классах в атрибутах особого типа - Структура [16]
"key"	Ключевые атрибуты	Массив строк, минимум одно значение.	Для функционирования приложения, в каждом классе должно быть задано ключевое поле, однозначно идентифицирующее объект в коллекции.
"semantic"	Семантический атрибут	Строка.	Задает семантику - правило формирования строкового представления для данного класса.
"name"	Системное имя	Строка, только латиница, без пробелов.	Задает в том числе первую часть имени файла меты класса, служебное имя.
"abstract"	Признак абстрактности для класса	Логическое	Используется только для родительских (базовых) классов.
"version"	Версионирование	Строка.	Позволяет задавать версионирвоание меты, для возможности оперирования данными созданными в разных версиях меты в рамках одной коллекции.
"caption"	Логическое имя	Строка.	Отображаемое в пользовательском интерфейсе имя класса
"ancestor"	Наследование	Null либо строка.	Набор атрибутов, заведенных в данном классе, наследуется классами-наследниками. Является способом сократить количество сущностей, когда для них можно использовать одинаковый набор атрибутов. Все классы-наследники будут наследовать атрибутивный состав родителя + можно завести атрибуты, принадлежащие индивидуально данному классу-наследнику (при необходимости).
"container"	Атрибут ссылки на контейнер	Null либо строка.	Здесь можно выбрать ссылочный атрибут, который будет использован для автоматического построения иерархической навигации. Объект, на который будет ссылаться выбранный атрибут, будет восприниматься средой как контейнер экземпляра доменного класса, и автоматически выстроит иерархию объектов.
"creationTracker"	Метка времени создания	Строка	Позволяет сохранять в классе дату/время создания объекта, требует наличия соответствующего атрибута класса, "name" которого и вносится в данное поле.

3.4.2 Мета классов - атрибутивная часть

Типы атрибутов

Коллекция

Общее описание

Коллекция - тип данных, позволяющий выводить в объекте списки других объектов. Данные объекты могут быть объектами любого класса включая исходный.

Способы задания коллекций:

с точки зрения используемых полей атрибутивной части меты классов

1. один-ко-многим - означает наличие контейнера и вложенного объекта с ссылкой на контейнер. В контейнере необходимо указать коллекцию, а у нее указать ссылочный атрибут вложенного объекта по которому формируется связь. См. Обратные ссылки в контексте коллекций.
2. многие-ко-многим - необходимо определить коллекцию без ссылок и указать класс вложенных элементов - связи создаются при помещении в коллекцию и хранятся как отдельные сущности в БД. См. Коллекции.
3. обратная коллекция - если коллекция многие ко многим задается на основании коллекции в другом объекте, то другая сторона связи задается через backColl. См. Обратные коллекции.

См. полное описание типа Коллекция в атрибутивной части меты класса - Атрибут коллекции и обратной коллекции.

Тип атрибута дата/время

Описание

Тип атрибута дата/время - представляет собой дату в формате ISODate. Может быть отображена либо как дата, либо как дата-время.

Режимы даты и времени

Режимы хранения даты задаются в параметре mode атрибутивной части меты класса. Доступно 3 режима хранения даты - реальный, локализованный, универсальный:

- 0 - Реальный (по умолчанию). Дата хранится без информации о часовом поясе. При отображении приводится к часовому поясу пользователя. Т.е. 01.01.2017 заданное на Камчатке в Хабаровске отобразится как 31.12.2016.

- 1 - Локализованный. Дата хранится вместе с часовым поясом, в котором была задана. При отображении приводится к этому часовому поясу. Т.е. 01.01.2017 заданное на Камчатке, в Хабаровске отобразиться так же - 01.01.2017. Но правильно его отображать с указанием часового пояса, т.е. 01.01.2017 (+11). При редактировании часовой пояс обновляется часовым поясом новой даты. При этом в БД хранится РЕАЛЬНЫЙ момент времени, что нужно учитывать в условиях выборки задаваемых хардкодом в мете.
- 2 - Универсальный. Дата сохраняется как если бы она задавалась в UTC. Т.е. не приводится к UTC, а сохраняется в UTC так же, как была введена. Т.е. если мы в Хабаровске ввели 01.01.2017 по Хабаровску, то сохранится она как “01.01.2017 00:00 UTC”, а не как “31.12.2016 14:00 UTC”. Отображается в любом часовом поясе так же, как была введена. Использовать для дат в реквизитах (дата рождения, дата выдачи документа и т.д.), т.е. когда важен не реальный, а формальный момент времени. Либо, когда не нужно учитывать время вообще.

Пример


```
{
  "isStruct": false,
  "metaVersion": "2.0.7",
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "class_datetime",
  "version": "",
  "caption": "Класс \"Дата/Время [9]\"",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
  "properties": [
    {
      "orderNumber": 10,
      "name": "id",
      "caption": "Идентификатор",
      "type": 12,
      "size": 24,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": false,
      "readonly": false,
      "indexed": false,
      "unique": true,
      "autoassigned": true,
      "hint": null,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
```


(continues on next page)

(continued from previous page)


```
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 20,
    "name": "data_data",
    "caption": "Выбор даты [120]",
    "type": 9,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "data_datatime",
    "caption": "Реальная дата",
    "type": 9,
    "mode": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
```


(continues on next page)

(continued from previous page)


```
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "data_datatime1",
    "caption": "Дата с часовым поясом",
    "type": 9,
    "mode": 1,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "data_datatime2",
    "caption": "Универсальная дата",
    "type": 9,
    "mode": 2,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
```


(continues on next page)

(continued from previous page)


```
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    }
  ]
}
```


Геоданные

Тип атрибута геокоординаты - является атрибутом, который хранит координаты с уникальными представлениями для создания и редактирования. Значение свойства “type” для атрибута, имеющего тип геокоординаты равно 100.

Описание:

Если атрибут имеет тип геокоординаты и на форме создания/редактирования объекта атрибут заполнен (координаты заданы), то отображается поле карты с координатами и кнопка “Изменить”. Если атрибут не заполнен – то отображается только кнопка “Задать координаты”. Доступно только в студии

Особенности работы

Если задано свойство "autoassigned": true и не заданы данные при создании формы - то нужно определять координаты автоматически по данным адреса из атрибутов указанных в свойствах геометки.

Способы отображения на форме

Для отображения атрибута с типом геокоординаты используется тип представления геообъект. Геообъект может быть отображён на форме в одном из трёх режимов:

- Карта (0) - геообъект отображается на карте;
- Поиск по адресу (1) - геообъект отображается на карте, на которой по адресу можно найти место и переместить туда геообъект. Или же можно просто задать координаты геообъекта;
- Холст (2) - геообъект отображается на карте, можно задать интерактивное отображение геообъекта на карте.

Режим отображения задаётся указанием соответствующей цифры в поле "mode" в мете представлений класса:

“mode”: 0, – Режим отображения Карта

“mode”: 1, – Режим отображения поиск по адресу

“mode”: 2, – Режим отображения Холст

Способы хранения в БД

Данные атрибута с типом геокоординаты сохраняются в БД в виде строки со значениями через запятую, строки в формате JSON-массива или строки в формате JSON-объекта.

Пример структуры атрибута в режиме Карта (0) в JSON:

В логическом имени атрибута (caption) указан режим отображения (mode) для разделения атрибутов с одинаковым типом.


```
{
    "orderNumber": 20,
    "name": "class_geodata",
    "caption": "Геоданные [100] mode [0] - Карта",
    "type": 100,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  }
```


Пример структуры атрибута в режиме Поиск по адресу (1) в JSON:


```
{
    "orderNumber": 30,
    "name": "class_geodata1",
    "caption": "Геоданные [100] mode [1] - Поиск по адресу",
    "type": 100,
```


(continues on next page)

(continued from previous page)


```
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  }
```


Пример структуры атрибута в режиме Холст (2) в JSON:


```
{
    "orderNumber": 40,
    "name": "class_geodata2",
    "caption": "Геоданные [100] mode [2] - Холст",
    "type": 100,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
```


(continues on next page)

(continued from previous page)

```
    "formula": null
  }
```

Тип “Структура [16]”

Структура - способ отображения связанных объектов (ссылок). Если у поля указана структура, то данный тип атрибута нужен для уменьшения действий при заведении модели, при заранее известных типовых классах, атрибуты которых используются во многих других классах.

Для класса-структуры в основной части меты класса задается значение true в поле "isStruct".

Для атрибутов типа “Структура [16]” в мете представлений создания и изменения используется тип “Группа [0]”. Если не указывать поля у группы, они создаются автоматически по мете. В представлениях списка нет необходимости делать колонки для атрибутов-структур, в объекте не будет такого свойства, а будут соответствующие атрибуты класса-структуры. Для них можно добавить колонки.

NB: Объекты класса-структуры не создаются!

Пример

Класс-структура:

```
{
  "isStruct": true,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "is_struct",
  "version": "",
  "caption": "\"isStruct\" класс-структура",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
  "properties": [
    {
      "orderNumber": 10,
      "name": "id",
      "caption": "Идентификатор",
      "type": 12,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": false,
      "readonly": false,
      "indexed": false,
      "unique": true,
```

(continues on next page)

(continued from previous page)


```
    "autoassigned": true,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 20,
    "name": "last_name",
    "caption": "Фамилия",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "first_name",
    "caption": "Имя",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
```


(continues on next page)

(continued from previous page)


```
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 40,
    "name": "patronymic",
    "caption": "Отчество",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 50,
    "name": "date",
    "caption": "Дата рождения",
    "type": 9,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
```


(continues on next page)

(continued from previous page)


```
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
   }
  ]
}
```


Класс с атрибутом типа “Структура [16]”


```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "struct",
  "version": "",
  "caption": "Класс \"Структура [16]\" (класс с типом атрибута 16 - структура)",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
  "properties": [
    {
      "orderNumber": 10,
      "name": "id",
      "caption": "Идентификатор",
      "type": 12,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": false,
      "readonly": false,
      "indexed": false,
      "unique": true,
      "autoassigned": true,
```


(continues on next page)

(continued from previous page)


```
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 20,
    "name": "struct",
    "caption": "Класс \"Структура [16]\"",
    "type": 16,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "is_struct",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  }
 ]
}
```


Объект класса с атрибутом-структурой в базе:


```
{
    "_id" : ObjectId("57c3e46fd53ecd50123cc4f5"),
    "struct$id" : "5f421610-6dba-11e6-874f-1b746e204b07",
    "struct$last_name" : "Мирошниченко",
    "struct$first_name" : "Ирина",
    "struct$patronymic" : "Львовна",
    "struct$date" : ISODate("1978-07-13T14:00:00.000Z"),
```


(continues on next page)

(continued from previous page)

```
    "id" : "5f41ef00-6dba-11e6-874f-1b746e204b07",
    "_class" : "struct@develop-and-test",
    "_classVer" : ""
}
```

Ссылки

Общее описание

Ссылка - тип данных, который хранит в себе ссылку на другой объект. Данный объект может быть объектом исходного класса, если ссылка указывает на класс, в котором она определена.

Значение в поле атрибута типа “Ссылка” выводятся в соответствии с семантикой, указанной для класса по ссылке. Главное отличие от типа “Коллекция” в том, что в поле атрибута типа “Ссылка” может отображаться и храниться только один объект класса по ссылке.

См. полное описание типа Ссылка в атрибутивной части меты класса - Атрибут ссылки и обратной ссылки.

Способы задания ссылок:

Способы задания ссылок с точки зрения используемых полей атрибутивной части меты классов:

1. один-к-одному - означает наличие ссылки и вложенного объекта с ссылкой-связкой на исходный объект. В ссылке необходимо указать ссылку-связку, а у нее указать ссылочный атрибут вложенного объекта по которому формируется связь. Обязательно в атрибуте-ссылке указать свойство уникальности - true. См. Обратные ссылки в контексте ссылок.
2. один-ко-многим - необходимо определить ссылку и указать класс вложенного элемента - связи создаются при помещении в ссылку и хранятся как отдельная сущность в БД. См. Ссылки.

Расписание

Расписание - тип данных реализующий хранение периодов времени или периодичность выполнения регулярных событий.

Режим отображения атрибута на форме:

В представлении для расписания предусмотрено два типа полей: SCHEDULE = 210 – расписание отображается в табличном виде CALENDAR = 220 – расписание отображается в виде календаря

Пример структуры атрибута в табличном виде:

```
{
        "caption": "Расписание [210]",
        "type": 210,
        "property": "schedule",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": null,
        "orderNumber": 20,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": "",
        "historyDisplayMode": 0,
        "tags": null
    }
```


Пример структуры атрибута в виде календаря:


```
{
        "caption": "Календарь [220]",
        "type": 220,
        "property": "calendar",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": null,
        "orderNumber": 30,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": "",
        "historyDisplayMode": 0,
        "tags": null
      }
```

Пример хранения расписания в БД:

Периоды времени задаются как объекты периодичности в атрибуте occurs с указанием атрибута duration. Пропуски во временном периоде указываются в атрибуте skipped.

```
[
  {
    "description": "Рабочие часы",
    "item": "develop-and-test@WorkTime@12345", // Ссылка на объект данных
    "occurs": [ // происходит
        {
          "hour": 9, // на 9 час суток
          "duration": 14400 // длится 4 часа (4 * 60 * 60)
        },
        {
          "hour": 14, // на 14 час суток
          "duration": 14400 // длится 4 часа (4 * 60 * 60)
        }
    ],
     "skipped": [ // пропускается
        {
          "weekday": 6 // по субботам
        },
        {
          "weekday": 7 // по воскресеньям
        }
    ]
  },
// ...
]
```

Периодичность

В объекте периодичности атрибуты задаются в рамках своих обычных значений, кроме атрибута year - год. Атрибут year, задаётся в виде частоты, так как является не периодической характеристикой.

Пример:

```
{
  "second": 30, // 1 - 60
  "minute": 20, // 1 - 60
  "hour": 9, // 0 - 23
  "day": 5, // 1 - 31
  "weekday": 1 // 1 - 7
  "month": 3 // 1 - 12
  "year": 2,
  "duration": 30 //
}
```

Описание примера:

В примере определён временной интервал длительностью 30 секунд, который повторяется один раз в два года, пятого марта в 9 часов 20 минут 30 секунд и только если день выпадает на понедельник.

Пользовательские типы

Пользовательский тип - "type": 17, задает значение пользовательского типа на основе базового типа. Находится в директории meta, types + [название типа].type.json. Используется в случаях, когда необходимо применить маску на значения определенного атрибута в различных классах.

Допустимые базовые типы

При создании пользовательского типа доступны следующие базовые типы:

- Строка [0]
- Целое [6]
- Действительное [7]
- Дата/Время [9]
- Десятичное [8]

Пример пользовательского типа userPassport.type.json

```
{
  "name": "userPassport",
  "caption": "Номер паспорта",
  "type": 0,
  "mask": "99 99 999999",
  "mask_name": "passport",
  "size": 12,
  "decimals": null
}
```

Применение

Пользовательские типы подключаются путем указания типа атрибута “Пользовательский [17]” - "type": 17 и указанием наименования пользовательского типа в поле “refClass”.

Пользовательский тип в JSON

```
{
    "orderNumber": 20,
    "name": "passport",
    "caption": "Номер паспорта (Пользовательский тип [17])",
    "type": 17,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
```

(continues on next page)

(continued from previous page)


```
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "userPassport",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  }
```


Таким образом, при вводе значения для атрибута "Номер паспорта (Пользовательский тип [17])" будет применяться маска, заданая для типа "userPassport" по ссылке свойства "refClass".

Тип атрибута - указывает на тип поддерживаемых атрибутом данных, как например размер допустимых значений и другие.

Код	Имя	Тип в БД	Описание
0	Строка	String	Тип данных, значениями которого является произвольная последовательность (строка) символов алфавита. Каждая переменная такого типа (строковая переменная) может быть представлена фиксированным количеством символов.
1	Текст	String	Хранит текстовые данные.
2	HTML	String	Форматированный текст, содержащий гипертекстовую разметку с возможностью редактирования с учетом возможных начертаний.
3	URL	String	Хранит ссылку, позволяет сохранить любую строку.
4	Изображение	String	Изображение, сохраняемое в файловом хранилище с предпросмотром в представлениях.
5	Файл	String	Файл, сохраняемый в файловом хранилище. В процессе реализации в регистри.
6	Целое	Int32	Целое число
7	Действительное	Double	Любое положительное число, отрицательное число или ноль.
9	Дата/время	Date	Дата в формате ISODate. Может быть отображена как дата, либо как дата-время.
8	Десятичное	Double	Число, представленное в десятичной системе счисления. Алфавит этой системы счисления состоит из 10 цифр от нуля до 9, отсюда и название - десятичная.
10	Логический	Boolean	Принимает два возможных значения, называемых истиной (true) и ложью (false).
11	Пароль	String	Хеш пароля
12	Глобальный идентификатор	String	Тип предназначенный для ключевого поля класса. Предполагает выставление атрибутов уникальности и автозаполнения.
13	Ссылка	String	Тип данных, хранящий в себе ссылку на объекты другого класса.
14	Коллекция	Array	Коллекция - тип данных, который хранит в себе ссылки на другие объекты. Каждая ссылка содержит значение идентификатора объекта, определенного в мете класса. Разделяются ссылки через запятую. Все значение из последовательности ссылок и запятых хранится строкой в базе.
15	Множество	String	Храним набор дискретных значений из предопределенного списка выбора.
16	Структура	String	Тип данных, хранящий в себе ссылку на объект класса-структуры.
17	Пользовательский тип	String	Дает возможность определения пользовательских типов на основе примитивных типов.
18	Пользователь	Строка	Хранит имя пользователя, для настройки безопасности, в формате имя@local
100	Геоданные	Object	Особый тип данных, хранящий координаты с уникальными представлениями для создания и редактирования.
110	Коллекция файлов	String	Тип атрибута для хранения комплекта файлов до 5 штук, с общим ограничением размера и возможностью задания допустимых расширений файлов
210	Расписание	Array	Тип данных, предназначеный для хранения данных календаря/расписания

Идентификаторы типов атрибутов:

```
module.exports = {
  STRING: 0,
  TEXT: 1,
  HTML: 2,
  URL: 3,
  IMAGE: 4,
  FILE: 5,
  INT: 6,
  REAL: 7,
  DECIMAL: 8,
  DATETIME: 9,
  BOOLEAN: 10,
  PASSWORD: 11,
  GUID: 12,
  REFERENCE: 13,
  COLLECTION: 14,
  SET: 15,
  STRUCT: 16,
  CUSTOM: 17,
  USER: 18,
  PERIOD: 60,
  GEO: 100,
  FILE_LIST: 110,
  SCHEDULE: 210
};
```

Автозаполняемые атрибуты

Тип автозаполняемые поля "autoassigned": true - указывает, что значение данного атрибута должно быть заполнено автоматически при создании экземпляра класса. Применяется в основном для атрибутов типа «Уникальный идентификатор» "unique": true для целосчисленных и строковых атрибутов, а также для атрибутов типа «Дата-время».

Принцип формирования:

1. Для атрибутов «Дата-время» атрибуту должно быть присвоено значение текущего момента времени. Используется для меток создания и изменения.
2. Для целочисленных атрибутов, если указано значение “Уникальный идентификатор” ("unique": true) при создании формы, заполняется случайным набором символов.
3. Для строк, если указано значение “Уникальный идентификатор” ("unique": true), то должно быть сгенерировано случайное значение hex - размером с длину строки - в примере ниже 20 символов.

   ```
   var crypto = require('crypto');
   ID = crypto.randomBytes(20).toString('hex');
   ```

4. Для типа “Глобальный идентификатор” - реализуется аналогично строке.

NB: Необходимо сделать проверку при сохранении. Поле должно генерироваться автоматически для пустых значений или даты. Для всех остальных (целое, строка) должны генерироваться ранее созданные значения.

Пример:

```
{
  "orderNumber": 50,
  "name": "auto",
  "caption": "auto",
  "type": 6,
  "size": null,
  "decimals": 0,
  "nullable": true,
  "readonly": false,
  "indexed": false,
  "unique": true,
  "autoassigned": true
}
```

Кеширование значения вычисляемого атрибута

Описание функционала

При применении функционала кеширования, значения атрибутов рассчитываются при создании и изменении объекта. При выборках берутся ранее рассчитанные значения.

Если есть два вычисляемых атрибута A и B обращающиеся к коллекции C, и при этом на A настроено кеширование, а на B не настроено, то при редактировании коллекция C будет выдергиваться 2 раза - один раз для атрибута B на уровне securedDataRepo для проверки доступа, второй раз для атрибута A при его пересчете уже в dataRepo. При чтении объекта из БД в данном случае кеш атрибута A просто не имеет смысла, так как коллекция в любом случае будет выбираться для атрибута B.

Кеширование семантик

Для кеширования семантик объектов в мете класса необходимо указать параметр:

```
semanticCached: true
```

Для атрибутов используемых в кешируемых семантиках не выполняется предварительная выборка объектов загрузки. Также в таких семантиках не рекомендуется использовать даты, т.к. они не будут приведены к формату часового пояса пользователя, так как кешируются при редактировании объекта на уровне DBAL.

Кеширование значения вычисляемого атрибута

Для кеширования значения вычисляемого атрибута в его мете указываем:

```
cached: true
```

Кроме того, реализована возможность обновлять кеши у объектов по ссылкам при редактировании основного объекта.

Для этого в мете класса указываем настройку:

```
cacheDependencies: ["refAttr1", "refAttr2.refAttr3", "refAttr2.collAttr4"]
```

В настройке необходимо указать ссылки и коллекции, кеши объектов в которых необходимо обновить при редактировании объекта данного класса. Обновления выполняестя рекурсивно, то есть если в классе объекта в refAttr1 настроено обновление кешей, оно будет запущено. Настройка наследуется в классах наследниках.

Пример настройки:

```
{
    "orderNumber": 40,
    "name": "kolStatOps",
    "caption": "Количество стационарных ОПС",
    "type": 6,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": true,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": "count($raionObslu.oktmo_nasPunkta.svyaz.ops,&eq($gops, b), 1)",
    "cached": true
}
```

Кешируется значение данного атрибута, получаемое из формулы. Дополнительно для обновления значения при редактирования объекта необходимо обновлять кеши объектов по ссылке: для этого в мете класса каждого объекта по ссылке указываем cacheDependencies:.

Пример:

```
{
  "isStruct": false,
  "key": [
    "okato"
  ],
  "semantic": "name",
  "name": "naselenniyPunkt",
```

(continues on next page)

(continued from previous page)


```
  "version": "",
  "caption": "Населенный пункт",
  "ancestor": null,
  "container": "",
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": true,
  "compositeIndexes": null,
  "cacheDependencies": ["supOktmo"],
  "properties": [
...
```


Значение по умолчанию

Значение по умолчанию задается при необходимости вывода значения в поле атрибута автоматически при открытии формы создания объекта. Значение по умолчанию задается присвоением значения default для свойства "defaultValue". Основное применение - для списков выбора допустимых значений.

Пример


```
{
    "orderNumber": 20,
    "name": "defaultValue",
    "caption": "Значение поля",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": "default",
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": {
      "type": "SIMPLE",
      "list": [
        {
          "key": "default",
          "value": "Значение, которое отображается по умолчанию при создании объекта"
```


(continues on next page)

(continued from previous page)

```
        },
        {
          "key": "other",
          "value": "Другое значение"
        }
      ],
      "matrix": [],
      "parameters": [],
      "hq": ""
    },
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
}
```

Для реализации автоматического расчета значения по умолчанию можно использовать функцию max:

```
"defaultValue": {max: ["className@namespace", "attr", {"filterAttr": "filterValue"}]}
```

Значение по умолчанию для атрибута типа “Ссылка” задается с помощью операции get следующими способами:

```
get(className) // возвращаем id случайно выбранного объекта класса
get(className, id) // проверяем наличие объекта в БД, если объект есть, возвращаем его id
get(className, attr1, val1, attr2, val2, ...) // возвращаем id первого объекта удовлетворяющего критериям
↪поиска: attr1=val1 и attr2=val2 и т.д.
```

Вычисляемые атрибуты (без кеширования)

Описание

Вычисляемые атрибуты (формулы) применяются для моментального формирования строкового выражения (результата) по заданному алгоритму при обращении к объекту класса через API. Например, при открытии объекта.

На уровне меты класса вычисляемые атрибуты хранят алгоритм формирования строкового выражения в свойстве formula.

Для примера можно получить все уникальные значения из атрибута name, значения которого хранятся в коллеции ownOrg. Затем уникальные значения с разделителями “,” между собой объединяются в одно строковое значение.

```
"formula": {
  "merge": [
    "$ownOrg",
    "name",
    null,
    1,
    ", "
  ]
}
```

Например, если есть коллекция с не уникальными значениями, например “ownOrg1”, “ownOrg2”. и снова “ownOrg1”, то чтобы получить только уникальные значения коллекции “Организация 1 и Организация 2”, пригодится выше описанная формула для вычисляемого атрибута с использованием функции merge.

В зависимости от функции можно обращаться к необходимому атрибуту для получения значения через атрибуты типа “Ссылка”, “Коллекция”.

При сохранении изменений и закрытии формы объекта результат не сохраняется в атрибуте, если не настроено кеширование.

Если в мете класса задано несколько вычисляемых атрибутов, то порядок их вычисления задается в свойстве orderNumber. При этом можно использовать результаты вычислений по заданному порядку orderNumber в последующих вычисляемых атрибутах.

В семантике класса или атрибута можно указывать вычисляемые атрибуты.

Если в свойстве formula задать Null, то атрибут не будет являться вычисляемым и кеширование к такому атрибуту не должно применяться.

Модель записи формулы

Каждая формула начинается с описания объекта и функции в формате JSON.

```
"formula": {
  "function1": [
    {
      "function2": [
        "operand3"
      ]
    },
    "operand1",
    "operand2"
  ]
}
```

В объекте обязательно нужно указать подходящую функцию необходимым количеством операндов для получения результата в function1.

В объекте содержится полное описание алгоритма, по которому будут происходить вычисления, за исключением тех функций, которые находятся в зависимых вычисляемых атрибутах.

Массив в функции хранит порядок операндов, которые передаются в функцию.

Операндами функции могут быть:

- Строковые значения, хранящие константу

```
"formula": {
  "function1": [
    "string"
  ]
}
```

- Строковые значения, хранящие переменные.

```
"formula": {
  "function1": [
    "$attr1"
```

(continues on next page)

(continued from previous page)

```
  ]
}
```

- Числовые значения

```
"formula": {
  "function1": [
    3.14
  ]
}
```

- Пустые значения

```
"formula": {
  "function1": [
    null
  ]
}
```

- Объекты

```
"formula": {
  "function1": [
    {
      "function2": [
        "operand1"
      ]
    }
  ]
}
```

Пример применения формулы:

```
{
    "orderNumber": 5,
    "name": "addressString",
    "caption": "",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": true,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
```

(continues on next page)

(continued from previous page)


```
"semantic": null,
"selConditions": [],
"selSorting": [],
"selectionProvider": null,
"indexSearch": false,
"eagerLoading": false,
"formula": {
 "concat": [
  {
   "if": [
    "$zipCode",
    {
     "concat": [
      "$zipCode"
     ]
    },
    ""
   ]
  },
  " ",
  {
   "if": [
    "$subjectFederation",
    "$subjectFederation",
    ""
   ]
  },
  {
   "if": [
    "$federationBorough",
    {
     "concat": [
      ", ",
      "$federationBorough"
     ]
    },
    ""
   ]
  },
  {
   "if": [
    {
     "and": [
      {
       "ne": [
        "$subjectFederation",
        "Санкт-Петербург г"
       ]
      },
      {
       "ne": [
        "$subjectFederation",
        "Москва г"
       ]
      }
     ]
```


(continues on next page)

(continued from previous page)

```
          },
          {
            "concat": [
              ", ",
              "$town"
            ]
          },
          ""
        ]
      },
      {
        "if": [
          "$street",
          {
            "concat": [
              ", ",
              "$street"
            ]
          },
          ""
        ]
      },
      {
        "if": [
          "$houseNumber",
          {
            "concat": [
              ", Дом ",
              "$houseNumber"
            ]
          },
          ""
        ]
      },
      {
        "if": [
          "$flatNumber",
          {
            "concat": [
              ", Квартира (офис) ",
              "$flatNumber"
            ]
          },
          ""
        ]
      }
    ]
  }
}
```

Результат: вывод адреса с пробелами и запятыми между значениями атрибутов

Поддерживаемые функции

eq - равно

ne - не равно

lt - меньше

gt - больше

lte - меньше либо равно

gte - больше, либо равно

and - и

or - или

not - не

add - арифметическое сложение

sub - арифметическое вычитание

mul - арифметическое умножение

div - арифметическое деление

nempty - не пусто

empty - пусто

pad - дополнение строки символами до нужной длины

next - извлекает новое значение последовательности

merge - конкатенация атрибутов в коллекции

size - принимает в качестве аргумента атрибуты типа строка и коллекция. Для строк возвращает длину, для коллекций - количество элементов

element - получение произвольного элемента из массива, индексирование с 0 ([массив значений], [индекс элемента: 0 - первый элемент, last - последний элемент])

dateAdd - добавление к дате (в нотации momentjs - [Дата], [добавляемый интервал (число)], [ед.изм (строка [d, m, y, h, min, s, ms)])

dateDiff - разница между датами (в нотации momentjs - [ед.изм], [Дата1], [Дата2])

now - текущая дата-время

concat - конкатенация строк

```
substring - получение подстроки ([Строка], [ с какого символа], [сколько символов])
```

obj - формирование объекта, нечетные аргументы - имена свойств, четные - значения

агрегация:

max, min, avg, sum, count

Все функции агрегации принимают следующие аргументы:

либо

```
[$Имя атрибута коллекции], [Имя агрегируемого атрибута], [функция фильтрации элементов коллекции]
```

либо

```
[Имя класса], [Имя агрегируемого атрибута], [Объект фильтра сформированный функцией obj
↪соответствующий нотации фильтров mongodb]
```

1 - указывает на уникальность объекта, то есть позволяет для функций агрегации производить подсчет только по уникальным объектам

\n - перенос на другую строку

Пример:

```
"formula": {
      "merge": [
        "$atr1",
        "atr2.name",
        null,
        1,
        "\n"
      ]
    },
```

Автоприсвоение и получение значения атрибута в вычисляемом выражении

1. Чтобы вычисляемое выражение не выполнялось при открытии формы создания, у атрибута надо выставить autoassigned: true. Тогда выражения будут вычислены непосредственно перед сохранением объекта. Это актуально при использовании функции next в вычислениях, так как не всегда необходимо извлекать очередное значение последовательности при каждом открытии формы создания.
2. Значения по умолчанию рассчитываются до записи объекта в БД, то есть на этапе их вычисления в простых автоприсваемых атрибутах еще ничего нет.
3. Функция next($id) (если в $id задано значение) будет всегда возвращать 1, так как для каждого объекта будет создаваться отдельная последовательность, из которой выбирается только первое значение.

Индексация атрибутов

Индексация атрибутов необходима для ускорения поиска. Индексация задается вручную, присвоением значения true для свойства "indexed", то есть:

```
"indexed": true
```

За исключением следующих типов атрибутов:

1. Индексация задается автоматически (независимо от того, что указано в индексации для поля):
   - для ключевого поля;
   - для объектов с полем "compositeIndexes": true в основной части меты класса.
2. При импорте меты принудительно индексируются все объекты атрибутов типа “Ссылка”.
3. Объекты атрибутов типа “Коллекция” не индексируются, так как коллекции с обратной ссылкой не хранятся в объекте, соответственно, в индексации нет необходимости.

NOTE. Запрещается индексация атрибутов типа "Текст" [1] и "HTML" [2].
Связано с ограничением MongoDB на размер индексируемого значения,
т.к. размер значения атрибутов данных типов может превышать допустимый размер.

Коллекция

Коллекция - тип данных, позволяющий выводить в объекте списки других объектов. Данные объекты могут быть объектами любого класса включая исходный.

Ссылки разделяются через запятую. Все значения из последовательности ссылок и запятых хранятся строкой в базе данных.

Способы задания коллекций:

с точки зрения используемых полей атрибутивной части меты классов

1. один-ко-многим - классическая связь дочернего объекта на родительский объект. Означает наличие контейнера и вложенного объекта с ссылкой на контейнер. В контейнере необходимо указать коллекцию, а у коллекции указать ссылочный атрибут вложенного объекта, по которому формируется связь. См. Обратные ссылки
2. многие-ко-многим - определяется через коллекцию без ссылок и класса вложенных элементов - связи создаются при помещении в коллекцию и хранятся как отдельные сущности в БД. См. Коллекции.
3. обратная коллекция - связь один-ко-многим, но в обратную сторону - со стороны объекта на который идут ссылки. Задается через backColl. См. Обратные коллекции.

Атрибут коллекция в формате JSON

Пример


```
{
    "orderNumber": 50,
    "name": "table",
    "caption": "Таблица",
    "type": 14,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "collRefCatalog@develop-and-test",
    "backRef": "",
```


(continues on next page)

(continued from previous page)


```
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
}
```


NB: Если коллекция ссылается на класс, содержащий множество потомков, то при заполнении коллекции будет возможность создавать объекты как родительского, так и дочерних классов.

Коллекции вместе с объектом грузятся по семантике, заданной в мете класса-коллекции или атрибута-коллекции.

Обратная ссылка в контексте коллекций

Обратная ссылка в контексте коллекций образуется следующим образом:

- Создается обычная коллекция с указанием ссылочного класса
- В ссылочном классе должен быть атрибут-ссылка, ссылающийся на исходный класс и имеющий свойство unique равным false . Значение атрибут-ссылке присваивается сразу при создании связи с коллекцией, без необходимости сохранения формы
- В исходном классе обычной коллекции заполняем свойство "backRef" - туда записывается код атрибута-ссылки из ссылочного класса

Атрибут обратная ссылка в формате JSON

Пример


```
{
    "orderNumber": 30,
    "name": "coll",
    "caption": "Коллекция с обратной ссылкой",
    "type": 14,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": null,
    "itemsClass": "ref_backcoll_ref@develop-and-test",
    "backRef": "ref_backcoll_ref",
    "backColl": "",
```


(continues on next page)

(continued from previous page)

```
    "binding": "",
    "semantic": "backcoll_data",
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": true,
    "formula": null
}
```

Режимы отображения атрибута типа “Коллекция” на форме:

Режимы отображения задаются в мете представления. Могут определяться при помощи свойства "mode", либо задаваться шаблоном в свойстве "options".

- mode: 4 - “Облако тегов” хранит значения одного или нескольких объектов по ссылке в виде тегов, наименование которых определяется семантикой объекта по ссылке.
- mode: 3 - “Таблица” хранит значения одного или нескольких объектов по ссылки в таблице, колонки которой предопределены для формы представления.

Пример

```
{
   "caption": "Таблица",
   "type": 3,
   "property": "table",
   "size": 2,
   "maskName": null,
   "mask": null,
   "mode": 3,
   "fields": [],
   "columns": [],
   ...
},
...
```

- “Комментарий” - задается аналогично режиму отображения “Таблица”, но с наложением шаблона, указанном в свойстве "options". Представляет собой поле, которое содержит данные, заранее предопределенные в свойстве "columns" для объекта по ссылке. Предназначено, в основном, для обсуждения информации по объекту на определенном этапе бизнес-процесса.

Пример

```
{
    "caption": "Коментарий",
    "type": 3,
    "property": "coment",
    "size": 2,
    "maskName": null,
    "mask": null,
```

(continues on next page)

(continued from previous page)

```
    "mode": 3,
    "fields": [],
    "columns": [
      {
        "sorted": true,
        "caption": "Дата",
        "type": 120,
        "property": "date",
        ...
      },
      {
        "sorted": true,
        "caption": "Подтверждение (Обоснование)",
        "type": 7,
        "property": "descript",
        ...
      },
      {
        "caption": "Ведущий",
        "type": 2,
        "property": "owner",
        ...
      }
    ],
    "actions": null,
    "commands": [
      {
        "id": "CREATE",
        "caption": "Создать",
        "visibilityCondition": null,
        "enableCondition": null,
        "needSelectedItem": false,
        "signBefore": false,
        "signAfter": false,
        "isBulk": false
      },
      {
        "id": "EDIT",
        "caption": "Править",
        "visibilityCondition": null,
        "enableCondition": null,
        "needSelectedItem": true,
        "signBefore": false,
        "signAfter": false,
        "isBulk": false
      }
    ],
    "orderNumber": 80,
    ...
    "tags": null,
    "options": {
      "template": "comments",
      "comments": {
        "textProperty": "descript",
        "userProperty": "owner",
        "parentProperty": "answlink",
```

(continues on next page)

(continued from previous page)

```
        "photoProperty": "owner_ref.foto.link",
        "dateProperty": "date"
      }
    }
}
```

Обратная коллекция

Пример коллекции выше преобразуется для обратной коллекции следующим образом:

```
{
    "orderNumber": 30,
    "name": "backcoll",
    "caption": "Обратная коллекции",
    "type": 14,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "coll_backcoll_coll",
    "backRef": "",
    "backColl": "coll",
    "binding": "",
    "semantic": "backcoll_data",
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": true,
    "formula": null
}
```

Обратите внимание на указание в свойстве "backColl" дополнительного значения - имя атрибута из класса в коллекции (из примера это coll)

Таким образом, реализуется связь многие-ко-многим без промежуточного класса. Не только атрибут "backcoll" с типом “Коллекция” может содержать несколько ссылок, но и объекты по ссылкам также могут содержать в своей коллекции “coll” несколько ссылок на объекты исходного класса.

Внимание:

- "type": 14 - тип атрибута “Коллекция”
- "backColl" - название ссылочного атрибута типа коллеции, ссылающегося на исходный класс с коллекцией

- "itemsClass" - название класса, объекты которого могут хранить свои идентификаторы в коллекции и, таким образом, формировать связь к объекту по идентификатору
- "backRef" - атрибут-ссылка из ссылочного класса, указанного в "itemsClass"
- При указании класса-родителя есть возможность создавать объекты родительского и дочерних классов
- Коллекции вместе с объектом грузятся по семантике, заданной в мете класса-коллекции или атрибута-коллекции

Схема обработки коллекций и формат хранения в БД

Для сохранения коллекции, необходимо передать в соответствующем ей атрибуте объекта массив действий вида:

```
"collection": [
  {"action": "put", "id": "1234"},
  {"action": "put", "id": "1235"},
  {"action": "put", "id": "1236"},
  {"action": "eject", "id": "1230"}
]
```

Порядок объектов должен соответствовать порядку выполнения соответствующих действий. Коды операций: put - добавление в коллекцию, eject - извлечение из коллекции. Алгоритм для создания и редактирования одинаков. Действия с коллекциями выполняются после создания или сохранения контейнера.

Принцип работы коллекций на форме создания и редактирования принципиально разный:

- На форме создания взаимодействие с сервером требуется лишь для получения и отображения в таблице выбранного/созданного объекта коллекции
- На форме редактирования реализована возможность получения ответа сервера при необходимости, и изменение параметров выборки при запросе, в зависимости от выполненных действий над коллекцией.

Ссылка

Описание

Ссылка - тип данных, который хранит простое значение и которое интерпретируется системой как ссылка на ключевой атрибут объекта другого класса. Данный объект может быть объектом любого класса, включая исходный. При указании класса-родителя есть возможность создавать объекты родительского и дочерних классов. Ссылки вместе с объектом грузятся по семантике, заданной в мете класса-ссылки или атрибута-ссылки.

Отображаемые значения в атрибуте ссылочного типа выводятся в соответствии с семантикой, указанной в ссылочном классе этого атрибута.

Возможность замены объекта по обратной ссылке определяется параметром nullable, связывающего ссылочный атрибут. При замене объекта связь будет потеряна, и объект по ссылке будет удален при попытке изменить связь из коллекции с обратной ссылкой.

Типы связей, реализуемые типом “Ссылка”:

Ссылочный тип с точки зрения используемых полей атрибутивной части меты классов:

1. один-ко-многим - классическая связь дочернего объекта на родительский объект. Необходимо определить ссылку и указать класс вложенного элемента - связи создаются при помещении в ссылку и хранятся как отдельная сущность в БД.
2. один-к-одному - аналогична связи один-ко-многим, означает наличие ссылки и вложенного объекта с ссылкой-связкой на исходный объект. В ссылке необходимо указать ссылку-связку, а у нее указать ссылочный атрибут вложенного объекта по которому формируется связь. Обязательно в атрибуте-ссылке указать свойство "unique": true.

Ссылка в формате JSON

Пример


```
{
    "orderNumber": 20,
    "name": "ssylka",
    "caption": "Ссылка",
    "type": 13,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "collRefCatalog@develop-and-test",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
}
```


Режимы отображения атрибута типа “Ссылка” на форме:

Режимы отображения задаются в мете представления. Могут определяться при помощи свойства "mode" либо задаваться шаблоном в свойстве "options".

- “mode”: 0 - отображение только семантики объекта по ссылке
- “mode”: 1 - отображение ссылки на форму объекта по ссылке

- “mode”: 3 - иерархический поиск объекта
- “mode”: 4 - уточняющий поиск объекта

Обратные ссылки

Обратная ссылка в контексте ссылок получается следующим образом:

- Создаётся атрибут с типом 13, указанием ссылочного класса refClass и указанием свойства "backRef" - куда записывается код атрибута из ссылочного класса.
- В ссылочном классе должен быть атрибут-ссылка, ссылающийся на исходный класс и имеющий свойство "unique": true.

Обратная ссылка в формате JSON

Пример


```
{
    "orderNumber": 30,
    "name": "backref",
    "caption": "Обратная ссылка",
    "type": 13,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "otorbrRef@develop-and-test",
    "itemsClass": "",
    "backRef": "ref",
    "backColl": "",
    "binding": "",
    "semantic": "data",
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": true,
    "formula": null
}
```


Внимание:

- "type": 13 - тип атрибута “Ссылка”
- "refClass" - название класса, объекты которого могут хранить свои идентификаторы в ссылке и, таким образом, формировать связь к объекту по идентификатору.

- "backRef" - указывается имя атрибута, который принадлежит классу, заданному в свойстве "refClass". Атрибут должен иметь тип “Ссылка” и ссылку на исходный класс.
- При указании класса-родителя есть возможность создавать объекты родительского и дочерних классов.
- Ссылки вместе с объектом грузятся по семантике, заданной в мете класса-ссылки или атрибута-ссылки.

Пример

```
Employee: {
    property: {
        aaa: {
            refClass: Post,
            backRef: bbb,
            ...
        },
        ...
    }
}

Post: {
    property: {
        bbb: {
            refClass: Employee,
            ...
        },
        ...
    }
}
```

Условия отбора допустимых значений

Описание

Условия отбора допустимых значений - условия, позволяющие ограничить выбор объектов по ссылке, допустимых для привязки в данном ссылочном атрибуте.

Фильтр списка допустимых значений используется в мете классов для атрибутов типа “Ссылка” и “Коллекция”. Фильтром накладываются условия ограничения выборки объектов. Условия накладываются как список последовательных операций.

Доступные операции:

- EQUAL: 0, // равно =
- NOT_EQUAL: 1, // не равно <>
- EMPTY: 2, // пусто '' или null
- NOT_EMPTY: 3, // не пусто !'' или !null
- LIKE: 4, // похож
- LESS: 5, // меньше <

(continues on next page)

(continued from previous page)

- MORE: 6, // больше >
- LESS_OR_EQUAL: 7, // меньше или равно <=
- MORE_OR_EQUAL: 8, // больше или равно >=
- IN: 9, // элемент входит в коллекцию/массив (IN)
- CONTAINS: 10 // содержит

```
module.exports = {
  AND: 0,
  OR: 1,
  NOT: 2,
  MIN: 3,
  MAX: 4,
  AVG: 5,
  SUM: 6,
  COUNT: 7
};
```

Описание операций

Операции могут быть разделены на группы по наличию свойств в условии:

Атрибут не указан в условии и условие - объект

- nestedConditions не содержит условий
  - Операции агрегации AgregOpers
    1. MIN
    2. MAX
    3. AVG
    4. SUM
    5. COUNT
- nestedConditions содержит условия
  - Логические операции сравнения вложенных условий BoolOpers
    1. OR
    2. NOT

Атрибут указан и условие - объект: операции сравнения значения артибута в условии со значением в value

1. EMPTY
2. NOT_EMPTY
3. CONTAINS
4. EQUAL
5. NOT_EQUAL
6. LESS
7. MORE
8. LESS_OR_EQUAL

9. MORE_OR_EQUAL
10. LIKE
11. IN

Условие в виде массива

- Применяется логическая операция AND для сравнения результатов условий (объектов в массиве).

В операции типа ключ-выражение ключом является имя атрибута в классе ссылки или в классе коллекции. Смежные условия объединяются логической операцией «И» (если не указана другая операция) - добавляются фильтры в свойство “selConditions”.

Применение операций и другие особенности

При выполнении запроса к атрибуту, необходимо использовать условие “nestedConditions”. Для каждого атрибута выполняется отдельная операция. Не указывайте вложенные ссылочные атрибуты через точку в поле “property”.

Для запроса значений атрибута, которые не равны нулю, необходимо выполнить операцию nempty, в поле “value” указываем null.

Операция CONTAINS применима к типам атрибута:

- строка - к строке данных применяется операция LIKE
- коллекция
  - применяется операция IN, если сравниваемое значение value является массивом и содержит хотя бы один элемент
  - происходит переход к вложенным условиям nestedConditions, если сравниваемое значение value не является массивом или не содержит хотя бы один элемент в массиве

JSON

```
{
  "selConditions": [
    {
      "property": "region",
      "operation": 10,
      "value": "Хабаровский край",
      "nestedConditions": [
        {
          "property": "town",
          "operation": 0,
          "value": "г Хабаровск",
          "nestedConditions": []
        }
      ]
    }
  ]
}
```

```
{
  "selConditions": [
    {
```

(continues on next page)

(continued from previous page)

```
        "property": "town",
        "operation": 3,
        "value": null,
        "nestedConditions": []
      }
    ]
}
```

Описание полей

Поле	Наименование	Допустимые значения	Описание
"property"	Атрибут	Строка, только латиница без пробелов	Атрибут класса ссылки, по которому производится фильтрация значений
"operation"	Операция	Код операции (см. выше)	Операция, согласно которой производится фильтрация
"value"	Значение	Зависит от типа операции	Второе значение для бинарных операций фильтрации
"nestedConditions"	Вложенные условия отбора	Объект, структура аналогична структуре самого объекта условий отбора.	

Пример

Внимание

Поле “selection_provider”. См. подробнее Список выбора допустимых значений.

- “type”: “SIMPLE” - простой тип,
- “list”: [] - массив допустимых значений

```
{
  "orderNumber": 80,
  "name": "type",
  "caption": "Тип организации",
  "type": 0,
  "size": null,
  "decimals": 0,
  "nullable": true,
  "readonly": false,
  "indexed": false,
  "unique": false,
  "autoassigned": false,
  "defaultValue": null,
  "refClass": "",
  "itemsClass": "",
  "backRef": "",
  "backColl": "",
  "binding": "",
```

(continues on next page)

(continued from previous page)


```
  "selConditions": [],
  "selSorting": [],
  "selectionProvider": {
    "type": "SIMPLE",
    "list": [
      {
        "key": "zakazchik",
        "value": "Заказчик"
      },
      {
        "key": "ispolnitel",
        "value": "Исполнитель"
      }
    ],
    "matrix": [],
    "parameters": [],
    "hq": ""
  },
  "indexSearch": false,
  "eagerLoading": false
}
```


Пример

В ссылочном атрибуте необходимо показать только те объекты, у которых в ссылочном классе задан атрибут “selConditions” и в поле property этого атрибута указано поле связанного класса,чьё значение в поле “value” соответствует условию “operation”.

В атрибуте организация, задача показать только организации (“refClass”: “organization”), у которых в поле тип (“property”: “type”) равно (“operation”: 0) значению zakazchik (“value”: “zakazchik”).

Все условия в "selConditions" объединяются по условию “И”.


```
{
  "orderNumber": 120,
  "name": "zakazchik",
  "caption": "Заказчик",
  "type": 13,
  "size": null,
  "decimals": 0,
  "nullable": true,
  "readonly": false,
  "indexed": false,
  "unique": false,
  "autoassigned": false,
  "defaultValue": null,
  "refClass": "organization",
  "itemsClass": "",
  "backRef": "",
  "backColl": "",
  "binding": "",
  "selConditions": [
    {
      "property": "type",
      "operation": 0,
```


(continues on next page)

(continued from previous page)


```
      "value": "zakazchik",
      "nestedConditions": []
    }
  ],
  "selSorting": [],
  "selectionProvider": null,
  "indexSearch": false,
  "eagerLoading": false
},
{
  "orderNumber": 130,
  "name": "ispolnitel",
  "caption": "Исполнитель",
  "type": 13,
  "size": null,
  "decimals": 0,
  "nullable": true,
  "readonly": false,
  "indexed": false,
  "unique": false,
  "autoassigned": false,
  "defaultValue": null,
  "refClass": "organization",
  "itemsClass": "",
  "backRef": "",
  "backColl": "",
  "binding": "",
  "selConditions": [
    {
      "property": "type",
      "operation": 0,
      "value": "ispolnitel",
      "nestedConditions": []
    }
  ],
  "selSorting": [],
  "selectionProvider": null,
  "indexSearch": false,
  "eagerLoading": false
}
```


Условия отбора допустимых значений для атрибутов с типом “Дата”

В ядре реализован атрибут контекста $$now, возвращающий текущую дату. $$now доступен везде при задании условий.

Подробнее см. переменные.

Пример

Условие: выводить объекты, у которых значение атрибута [dataStart] меньше текущей даты:

```
{
    "property": "dateStart",
    "operation": 5,
    "value": [
      "$$now"
    ],
    "nestedConditions": []
}
```

Список выбора допустимых значений

Список выбора допустимых значений - задает список выбора допустимых значений для поля ввода атрибута и находится в атрибутивной части меты класса - "selectionProvider". Список формируется в виде массива объектов типа «ключ-значения» и представляет собой список выбора значения для атрибута с типом «Строка», «Действительное», «Целое», «Десятичное», «Текст».

Есть три типа списка выбора, тип задается в поле ("type") одной из следующих ключевых фраз:

- "SIMPLE" - список выбора простого типа;
- "MATRIX" - список выбора матричного типа.

Описание полей структуры

Структура объекта списка выбора

```
"selectionProvider": {
  "type": "SIMPLE",
  "list": [...],
  "matrix": [],
  "parameters": [],
  "hq": ""
},
```

Поле	Наименование	Допустимые значения	Описание
"type"	Тип	"SIMPLE", "MATRIX", "HQL"	Тип списка выбора
"list"	Простой тип	Массив объектов типа “ключ-значение”.	Список выбора просто типа (“SIMPLE”) хранится здесь.
"matrix"	Матрица	Массив векторов, каждый из которых состоит из именования, комплекта условий выбора и комплекта пар “ключ-значение”.	Список выбора матричного типа (“MATRIX”).
"parameters"	Параметры запроса	Массив объектов типа “ключ-значение”.	Параметры запроса, не реализовано
"hq"	Запрос	Строка запроса в соответствии с форматом обработчика не используется в текущей версии	Строка запроса, не реализовано

Поле "list" - массив объектов следующей структуры

```
"list": [
  {
    "key": "2001-03-23 09:00:00.000Z",
    "value": "Затопление орбитальной станции «Мир» (23 марта 2001 г. 09:00 мск)"
  },
  {
    "key": "1957-10-04 19:28:00.000Z",
    "value": "Запуск первого в мире искусственного спутника (4 октября 1957 г. в 19:28 гринвич)"
  },
  {
    "key": "1970-04-17 12:07:00.000Z",
    "value": "Завершение полёта «Аполлон-13» (17 апреля 1970 г. 12:07 Хьюстон)"
  }
],
```

Поле	Наименование	Допустимые значения	Описание
"key"	Ключ	Любое значение, соответствующее типу атрибута для которого заведен данный список выбора	При сохранении объекта именно значение ключа записывается в базу
"value"	Значение	Любая строка, могут быть проблемы при наличии управляющих последовательностей	Значение этого поля выводится в пользовательском интерфейсе

Поле "matrix" - массив объектов следующей структуры

```
  "matrix": [
    {
      "comment": "Оба отрицательные",
      "conditions": [
        {
          "property": "matrix_base_1",
          "operation": 5,
          "value": "0",
          "nestedConditions": []
        },
        {
          "property": "matrix_base_2",
          "operation": 5,
          "value": "0",
          "nestedConditions": []
        }
      ],
      "result": [
        {
          "key": "Оба отрицательные",
          "value": "Оба отрицательные"
        }
      ]
    },
    {
```

(continues on next page)

(continued from previous page)


```
    "comment": "Оба неотрицательные",
    "conditions": [
      {
        "property": "matrix_base_1",
        "operation": 8,
        "value": "0",
        "nestedConditions": []
      },
      {
        "property": "matrix_base_2",
        "operation": 8,
        "value": "0",
        "nestedConditions": []
      }
    ],
    "result": [
      {
        "key": "Оба неотрицательные",
        "value": "Оба неотрицательные"
      }
    ]
  },
  {
    "comment": "Первое неотрицательное второе отрицательное",
    "conditions": [
      {
        "property": "matrix_base_1",
        "operation": 8,
        "value": "0",
        "nestedConditions": []
      },
      {
        "property": "matrix_base_2",
        "operation": 5,
        "value": "0",
        "nestedConditions": []
      }
    ],
    "result": [
      {
        "key": "Первое неотрицательное второе отрицательное",
        "value": "Первое неотрицательное второе отрицательное"
      }
    ]
  },
  {
    "comment": "Первое отрицательное, второе неотрицательное",
    "conditions": [
      {
        "property": "matrix_base_1",
        "operation": 5,
        "value": "0",
        "nestedConditions": []
      },
      {
        "property": "matrix_base_2",
```


(continues on next page)

(continued from previous page)


```
        "operation": 8,
        "value": "0",
        "nestedConditions": []
      }
    ],
    "result": [
      {
        "key": "Первое отрицательное, второе неотрицательное",
        "value": "Первое отрицательное, второе неотрицательное"
      }
    ]
  }
 ],
 "parameters": [],
 "hq": ""
},
```


Каждый объект массива "MATRIX" содержит следующие обязательные поля:

Поле	Наименование	Допустимые значения	Описание
"comment"	Комментарий	Любая строка	Комментарий к вектору
"conditions"	Условия	Массив объектов	Определяет условия при которых выводится список элементов описанный в "result" данного вектора
"result"	Результаты	Массив объектов, аналогичен структуре поля "list"	Задает список выбора, который выводится при соблюдении условий, перечисленных в "conditions" данного вектора

Поле "conditions" массива "MATRIX"

Поле	Наименование	Допустимые значения	Описание
"property"	Атрибут	Строка, только латиница без пробелов	Атрибут класса, значение поля которого проверяется на соответствие данному условию данного вектора
"operation"	Операция	Код операции	Операция, согласно которой производится определение
		0 - равно (И)	
		1 - не равно (ИЛИ)	
		2 - пусто (НЕ)	
		3 - не пусто (МИН ИЗ)	
		4 - (МАКС ИЗ)	
		5 - < ()	
		6 - >	
		7 - <=	
		8 - >=	
		9 - IN /Похож/	
		10 - содержит	
"value"	Значение	Зависит от типа операции	Второе значение для бинарных операций
"nestedConditions"	Вложенные условия отбора	Объект, структура аналогична структуре самого объекта условий отбора.	

NB: Код операции соответствует разным значениям операций, в зависимости от того, выбран атрибут или нет. Если поле "property" равно null, то кодируется логическое условие, по которому объединяются вложенные условия отбора. (Указаны в скобках в таблице выше)

Описание

Список выбора типа “SIMPLE”

Данный список выбора позволяет создать жестко зашитый в приложении пресет значений поля, ограничив тем самым выбор пользователя.

Для поля в обязательном порядке следует задать тип представления - “Выпадающий список [5]”.

Подразумевает возможность сохранять данные в базе в типе, отличном от типа данных, выводимых пользователю.

Например: Если задать в качестве полей key элементы списка выбора значения даты-времени в ISODate, а в качестве value - описание события, то предоставим пользвателю возможность выбрать событие, но внутри приложения работать с данными типа ISODate.

NB: Если у атрибута со списком выбора разрешено пустое значение: "nullable": true - в списке выбора добавляется пустое значение по умолчанию!

```
{
  "orderNumber": 50,
  "name": "sp_date",
  "caption": "Сохраняем ключ дата-время",
```

(continues on next page)

(continued from previous page)


```
  "type": 9,
  "size": null,
  "decimals": 0,
  "allowedFileTypes": null,
  "maxFileCount": 0,
  "nullable": true,
  "readonly": false,
  "indexed": false,
  "unique": false,
  "autoassigned": false,
  "hint": null,
  "defaultValue": null,
  "refClass": "",
  "itemsClass": "",
  "backRef": "",
  "backColl": "",
  "binding": "",
  "semantic": null,
  "selConditions": [],
  "selSorting": [],
  "selectionProvider": {
    "type": "SIMPLE",
    "list": [
      {
        "key": "2001-03-23T09:00:00.000Z",
        "value": "Затопление орбитальной станции «Мир» (23 марта 2001 г. 09:00 мск)"
      },
      {
        "key": "1957-10-04T19:28:00.000Z",
        "value": "Запуск первого в мире искусственного спутника (4 октября 1957 г. в 19:28 гринвич)"
      },
      {
        "key": "1970-04-17T12:07:00.000Z",
        "value": "Завершение полёта «Аполлон-13» (17 апреля 1970 г. 12:07 Хьюстон)"
      }
    ],
    "matrix": [],
    "parameters": [],
    "hq": ""
  },
  "indexSearch": false,
  "eagerLoading": false,
  "formula": null
}
```


Принцип создания:

Необходимо:

1. выбрать исходя из требований предметной области наиболее удобный тип атрибута,
2. выбрать идентификаторы данного типа ("key") с той целью, что бы при необходимости автоматизированной обработки оперировать значениями в базе максимально эффективно,
3. задать к каждому идентификатору подпись, которая будет отображаться в интерфейсе "value",
4. задать в представлениях тип представления - “Выпадающий список [5]” в обязательном порядке.

Список выбора типа “MATRIX”

В матрицах результирующий список выбора это все, что попадает под условия. Если условий нет - то система считает, что список выбора применяется всегда. Для предсказуемости работы приложения, необходимо чтобы были соблюдены два условия:

1. Вектора не должны перекрывать друг друга.
2. Массив значений опорного атрибута, как основание матрицы (массив сочетаний значений опорных атрибутов) должен полностью закрываться описанными векторами.

   Система берет значение опорного поля (полей) и последовательно применяет к нему условия описанные в векторах. Каждый вектор - это набор условий и собственный список выбора. Как только система дойдет до вектора, условиям которого удовлетворяет значение опорного поля, она берет из него список выбора и определяет к выводу в пользовательском интерфейсе. Подразумевается, что на любое значение опорного поля система найдет соответствующий вектор.

Пример 1: Матрица от двух целочисленных значений

JSON класса:


```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "selection_provider_matrix_dc",
  "version": "",
  "caption": "\"MATRIX\" от двух оснований",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": [],
  "properties": [
    {
      "orderNumber": 10,
      "name": "id",
      "caption": "Идентификатор",
      "type": 12,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": false,
      "readonly": false,
      "indexed": false,
      "unique": true,
      "autoassigned": true,
      "hint": null,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
```


(continues on next page)

(continued from previous page)


```
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 20,
    "name": "matrix_base_1",
    "caption": "Первое целое основание матрицы",
    "type": 6,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "matrix_base_2",
    "caption": "Второе целое основание матрицы",
    "type": 6,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
```


(continues on next page)

(continued from previous page)


```
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 40,
    "name": "selection_provider_matrix",
    "caption": "Список выбора типа \"MATRIX\"",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": {
      "type": "MATRIX",
      "list": [],
      "matrix": [
        {
          "comment": "Оба отрицательные",
          "conditions": [
            {
              "property": "matrix_base_1",
              "operation": 5,
              "value": "0",
              "nestedConditions": []
            },
            {
              "property": "matrix_base_2",
              "operation": 5,
              "value": "0",
              "nestedConditions": []
            }
```


(continues on next page)

(continued from previous page)


```
        ],
        "result": [
          {
            "key": "Оба отрицательные",
            "value": "Оба отрицательные"
          }
        ]
      },
      {
        "comment": "Оба неотрицательные",
        "conditions": [
          {
            "property": "matrix_base_1",
            "operation": 8,
            "value": "0",
            "nestedConditions": []
          },
          {
            "property": "matrix_base_2",
            "operation": 8,
            "value": "0",
            "nestedConditions": []
          }
        ],
        "result": [
          {
            "key": "Оба неотрицательные",
            "value": "Оба неотрицательные"
          }
        ]
      },
      {
        "comment": "Первое неотрицательное второе отрицательное",
        "conditions": [
          {
            "property": "matrix_base_1",
            "operation": 8,
            "value": "0",
            "nestedConditions": []
          },
          {
            "property": "matrix_base_2",
            "operation": 5,
            "value": "0",
            "nestedConditions": []
          }
        ],
        "result": [
          {
            "key": "Первое неотрицательное второе отрицательное",
            "value": "Первое неотрицательное второе отрицательное"
          }
        ]
      },
      {
        "comment": "Первое отрицательное, второе неотрицательное",
```

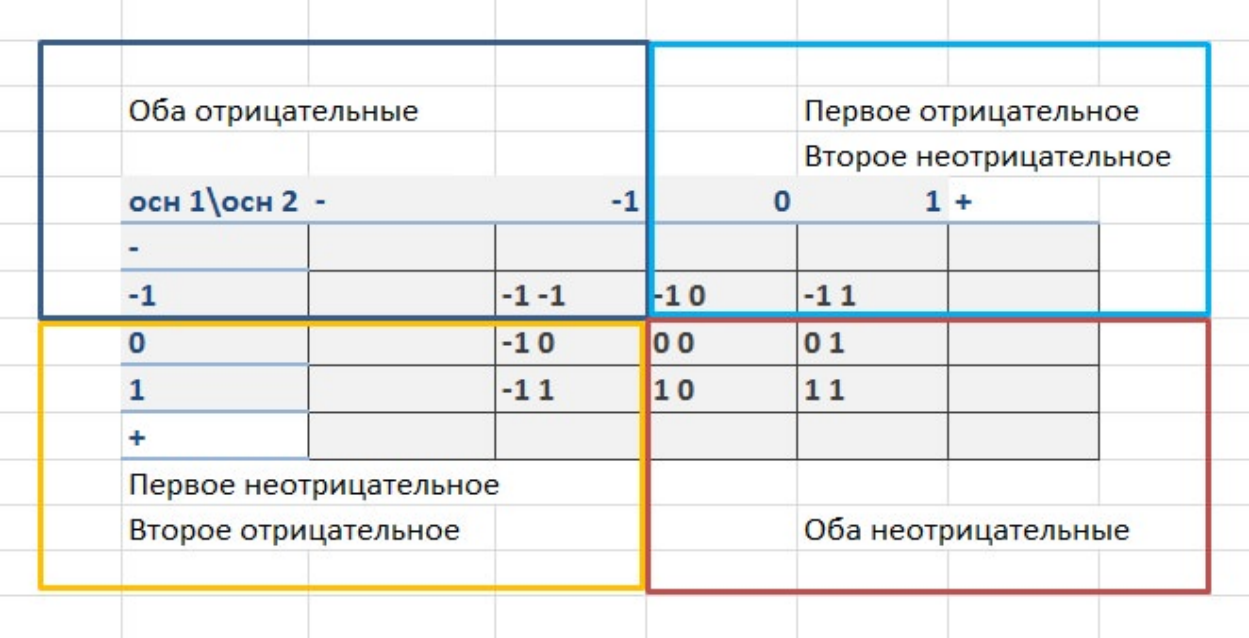

(continues on next page)

(continued from previous page)


```
          "conditions": [
            {
              "property": "matrix_base_1",
              "operation": 5,
              "value": "0",
              "nestedConditions": []
            },
            {
              "property": "matrix_base_2",
              "operation": 8,
              "value": "0",
              "nestedConditions": []
            }
          ],
          "result": [
            {
              "key": "Первое отрицательное, второе неотрицательное",
              "value": "Первое отрицательное, второе неотрицательное"
            }
          ]
        }
      ],
      "parameters": [],
      "hq": ""
    },
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  }
 ]
}
```


Порядок разработки

Необходимо разделить все возможные сочетания пар атрибутов "matrix_base_1" и "matrix_base_2" на 4 вектора. Делить необходимо относительно нуля, то есть каждое поле может быть либо отрицательным, либо неотрицательным. Ниже представлена схема:



Выписываем векторы и их условия:

1. Оба отрицательные: (matrix_base_1 < 0) && (matrix_base_2 < 0)
2. Оба неотрицательные: (matrix_base_1 >= 0) && (matrix_base_2 >= 0)
3. Первое неотрицательное второе отрицательное: (matrix_base_1 >= 0) && (matrix_base_2 < 0)
4. Первое отрицательное, второе неотрицательное: (matrix_base_1 < 0) && (matrix_base_2 >= 0)
5. Если в 3 и 4 условиях неверно указать равенство нулю, то как результат - выпадающие элементы и перекрытие векторов.

В примере выше для каждого вектора список выбора ограничен одним пунктом, но их может быть больше.

Пример 2: Матрица от свободного действительного значения со сложными условиями


```
{
  "isStruct": false,
  "metaVersion": "2.0.7",
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "selection_provider_matrix_real",
  "version": "",
  "caption": "\"MATRIX\" с векторами \u003c, \u003e, \u003c\u003d, \u003e\u003d, \u003d от
↪действительного",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
```


(continues on next page)

(continued from previous page)


```
"properties": [
 {
  "orderNumber": 10,
  "name": "id",
  "caption": "Идентификатор",
  "type": 12,
  "size": null,
  "decimals": 0,
  "allowedFileTypes": null,
  "maxFileCount": 0,
  "nullable": false,
  "readonly": false,
  "indexed": false,
  "unique": true,
  "autoassigned": true,
  "hint": null,
  "defaultValue": null,
  "refClass": "",
  "itemsClass": "",
  "backRef": "",
  "backColl": "",
  "binding": "",
  "semantic": null,
  "selConditions": [],
  "selSorting": [],
  "selectionProvider": null,
  "indexSearch": false,
  "eagerLoading": false,
  "formula": null
 },
 {
  "orderNumber": 20,
  "name": "matrix_base",
  "caption": "Действительное основание для списка выбора матричного типа",
  "type": 7,
  "size": null,
  "decimals": 0,
  "allowedFileTypes": null,
  "maxFileCount": 0,
  "nullable": true,
  "readonly": false,
  "indexed": false,
  "unique": false,
  "autoassigned": false,
  "hint": null,
  "defaultValue": null,
  "refClass": "",
  "itemsClass": "",
  "backRef": "",
  "backColl": "",
  "binding": "",
  "semantic": null,
  "selConditions": [],
  "selSorting": [],
  "selectionProvider": null,
  "indexSearch": false,
```


(continues on next page)

(continued from previous page)


```
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "selection_provider_matrix",
    "caption": "Список выбора со сложными условиями",
    "type": 6,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": false,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": {
      "type": "MATRIX",
      "list": [],
      "matrix": [
        {
          "comment": "matrix_base \u003c 3",
          "conditions": [
            {
              "property": "matrix_base",
              "operation": 5,
              "value": [
                "3"
              ],
              "nestedConditions": []
            }
          ],
          "result": [
            {
              "key": "1",
              "value": "Сохраним 1 при основании меньше 3"
            },
            {
              "key": "2",
              "value": "Сохраним 2 при основании меньше 3"
            }
          ]
        },
        {
          "comment": "matrix_base \u003d 3",
```


(continues on next page)

(continued from previous page)


```
        "conditions": [
          {
            "property": "matrix_base",
            "operation": 0,
            "value": [
              "3"
            ],
            "nestedConditions": []
          }
        ],
        "result": [
          {
            "key": "3",
            "value": "Сохраним 3 при основании 3"
          }
        ]
      },
      {
        "comment": "matrix_base \u003e 3 и matrix_base \u003c\u003d 15",
        "conditions": [
          {
            "property": "matrix_base",
            "operation": 6,
            "value": [
              "3"
            ],
            "nestedConditions": []
          },
          {
            "property": "matrix_base",
            "operation": 7,
            "value": [
              "15"
            ],
            "nestedConditions": []
          }
        ],
        "result": [
          {
            "key": "5",
            "value": "Сохраним 5 при основании \u003e 3 и \u003c\u003d 15"
          },
          {
            "key": "10",
            "value": "Сохраним 10 при основании \u003e 3 и \u003c\u003d 15"
          },
          {
            "key": "15",
            "value": "Сохраним 15 при основании \u003e 3 и \u003c\u003d 15"
          }
        ]
      },
      {
        "comment": "matrix_base \u003e\u003d16",
        "conditions": [
          {
```


(continues on next page)

(continued from previous page)


```
            "property": "matrix_base",
            "operation": 8,
            "value": [
              "16"
            ],
            "nestedConditions": []
          }
        ],
        "result": [
          {
            "key": "50",
            "value": "Сохраним 50 при основании \u003e\u003d 16"
          },
          {
            "key": "100",
            "value": "Сохраним 100 при основании \u003e\u003d16"
          },
          {
            "key": "1000",
            "value": "Сохраним 1000 при основании \u003e\u003d16"
          },
          {
            "key": "5000",
            "value": "Сохраним 5000 при основании \u003e\u003d16"
          }
        ]
      },
      {
        "comment": "matrix_base \u003e 15 и matrix_base \u003c 16",
        "conditions": [
          {
            "property": "matrix_base",
            "operation": 6,
            "value": [
              "15"
            ],
            "nestedConditions": []
          },
          {
            "property": "matrix_base",
            "operation": 5,
            "value": [
              "16"
            ],
            "nestedConditions": []
          }
        ],
        "result": [
          {
            "key": "0",
            "value": "Сохраним 0, если основание где-то между 15 и 16"
          }
        ]
      }
    ],
    "parameters": [],
```


(continues on next page)

(continued from previous page)


```
      "hq": ""
    },
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  }
 ]
}
```


Векторы и их условия:

1. matrix_base < 3
2. matrix_base = 3
3. (matrix_base > 3) && (matrix_base <= 15)
4. matrix_base >= 16
5. (matrix_base > 15) && (matrix_base < 16)

Сортировка выборки допустимых значений

Описание

Сортировка выборки допустимых значений - задается при создания сущности в поле "selSorting" и представляет собой фильтр, который задает способ сортировки объектов. Применяется для атрибутов типа "Ссылка".

Доступные типы сортировки:

- Сортировка по возрастанию
- Сортировка по убыванию

JSON


```
{
    "orderNumber": 20,
    "name": "ref",
    "caption": "Ссылка",
    "type": 13,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
```


(continues on next page)

(continued from previous page)

```
    "hint": null,
    "defaultValue": null,
    "refClass": "selSortingCatalog@develop-and-test",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [
      {
        "property": "string",
        "mode": 1
      }
    ],
```

Описание полей

Поле	Наименование	Допустимые значения	Описание
"property"	Атрибут	Строка, только латиница без пробелов	Атрибут, по которому будет производится сортировка.
"mode"	Порядок сортировки	0 - по возрастанию	Порядок сортировки
		1 - по убыванию	

Основание коллекции

Основание коллекции - атрибут в объекте, по значению которого выполняется поиск объектов в коллекции на основании сравнения с атрибутом обратной ссылки.

Цель использования

Основание коллекции может быть использовано для создания динамичных коллекций, когда объекты в коллекции могут быть загружены в зависимости от введенных данных или расчитанных по формуле значений, в отличие от обычных обратных ссылок с поиском только по ключевым атрибутам.

Пример

Для примера в проекте develop-and-test заведены два класса:

- searchRefs - в нем добавлен атрибут-строка code_binding в качестве обратной ссылки для binding.

```
{
  "orderNumber": 30,
  "name": "code_binding",
  "caption": "Код для binding",
  "type": 0,
```

(continues on next page)

(continued from previous page)


```
  "size": null,
  "decimals": 0,
  "allowedFileTypes": null,
  "maxFileCount": 0,
  "nullable": true,
  "readonly": false,
  "indexed": true,
  "unique": false,
  "autoassigned": false,
  "hint": null,
  "defaultValue": null,
  "refClass": "",
  "itemsClass": null,
  "backRef": null,
  "backColl": "",
  "binding": "",
  "semantic": "",
"selConditions": null,
  "selSorting": [],
  "selectionProvider": null,
  "indexSearch": false,
  "eagerLoading": false,
  "formula": null
}
```


- backref_searchRefs - в этом классе используются атрибуты backref_searchRefs_binding и backref_searchRefs_text. backref_searchRefs_binding является динамичной коллекцией, а backref_searchRefs_text используется в качестве фильтра для коллекции backref_searchRefs_binding. Выборка в коллекцию backref_searchRefs_binding идет по равенству значений backref_searchRefs_text из класса backref_searchRefs и code_binding из класса searchRefs. При добавлении объектов в коллекцию вручную автоматически заполняется атрибут code_binding.


```
{
  "orderNumber": 25,
  "name": "backref_searchRefs_binding",
  "caption": "Обратная ссылка (с подключенным binding) на класс searchRefs",
  "type": 14,
  "size": null,
  "decimals": 0,
  "allowedFileTypes": null,
  "maxFileCount": 0,
  "nullable": true,
  "readonly": false,
  "indexed": false,
  "unique": false,
  "autoassigned": false,
  "hint": null,
  "defaultValue": null,
  "refClass": "",
  "itemsClass": "searchRefs",
  "backRef": "code_binding",
  "backColl": "",
  "binding": "backref_searchRefs_text",
  "semantic": null,
"selConditions": null,
```


(continues on next page)

(continued from previous page)


```
  "selSorting": [],
  "selectionProvider": null,
  "indexSearch": false,
  "eagerLoading": false,
  "formula": null
},
{
  "orderNumber": 30,
  "name": "backref_searchRefs_text",
  "caption": "Значение",
  "type": 0,
  "size": null,
  "decimals": 0,
  "allowedFileTypes": null,
  "maxFileCount": 0,
  "nullable": true,
  "readonly": false,
  "indexed": true,
  "unique": false,
  "autoassigned": false,
  "hint": null,
  "defaultValue": null,
  "refClass": "",
  "itemsClass": "",
  "backRef": "",
  "backColl": "",
  "binding": "",
  "semantic": null,
"selConditions": null,
  "selSorting": [],
  "selectionProvider": null,
  "indexSearch": false,
  "eagerLoading": false,
  "formula": null
}
```


Предварительная выборка

Предварительная выборка объектов загрузки (от англ. eager loading - “жадная загрузка”) - процесс, который позволяет указать в запросе какие связанные данные, в объёме достаточном для корректного отображения семантики связанного объекта, необходимо загрузить при выполнении запроса.

Предварительная выборка помогает в отображении объектов системы в тех случаях, когда данные находятся на большом уровне вложенности, например в атрибутах типа “Ссылка” и “Коллекция”.

В мете применяется только в особых случаях и используется в основном для экономии временных ресурсов, в качестве альтернативы тонкой настройки.

Пример

Управление предварительной выборкой для атрибута определяется свойством eagerLoading принимающим значения true или false.

```
"properties": [
  {
    ...
    "eagerLoading": true,
    "formula": null
  }
```

Настройка

Рекомендуемый способ настройки предварительной выборки - через файл deploy.json проекта для атрибутов класса, навигации. Это позволит централизовано определить предварительно выбираемые атрибуты для многих классов.

Пример

```
"eagerLoading": {
        "node1": {
          "class1": {
             "list": ["attr1", "attr2.attr3"],
             "item": ["attr1", "attr2.attr3"]
          }
        }
      }
```

NB: Если вместо "node1" поставить "*", то при попадании из любой навигации на данный объект заявления можно использовать одну настройку класса для экспорта объекта.

Настройка для экспорта в списках и формах

Принцип настройки предварительной выборки для экспорта в списках и формах совпадает с настройкой в файле deploy.json, за одним исключением. Вместо list и item, указываем exportList и exportItem.

Пример

```
"eagerLoading": {
  "class1@ns": {
    "exportList": ["attr1", "attr2.attr3"]
  },
  "class1@ns": {
    "exportItem": ["attr1", "attr2.attr3.attr4"]
  }
}
```

Глубина предварительной выборки объектов загрузки

Глубина предварительной выборки задаётся свойством maxEagerDepth: 1 в файле deploy.json проекта.

Максимальная глубина предварительной выборки определяет максимально допустимый уровень вложенности объекта относительно открытого объекта на странице.

Семантика

Семантика - используется для вывода объекта класса в качестве одной строки заголовка класса.

В мете классов поле "semantic" встречается дважды:

1. в общей части меты класса, где формирует строковое представление для данного класса,
2. в мете атрибута класса, где формирует строковое представление объектов класса, на который ссылается атрибут, т.е. используется для ссылочных атрибутов.

Цель использования

Используется для корректировки отображения атрибутов и значений атрибутов в списке. В атрибутах, которые выводят табличные данные, семантика используется для ограничения вывода колонок.

Примеры использования в ссылочных атрибутах

Например, есть класс class, у которого есть атрибуты: id, name, link, date. Есть второй класс classTable, у которого есть ссылочный атрибут table на класс class. Без использования семантики в объектах класса classTable в атрибуте table будут выводится значения идентификаторов объектов класса class. Атрибуты, используемые как идентификаторы, указаны в мете класса class.

Чтобы вывести значения атрибутов name и link в атрибуте table, а не значения идентификаторов, нужно прописать "semantic": "name|link". В зависимости от типа атрибута результат будет разный:

- Если атрибут table является ссылкой, то в нем будут заполнены значения атрибутов name и link через пробел. Тут можно использовать дополнительные слова и выражения через знак |, например "semantic": "name|, |link" или "semantic": "У объекта есть 2 атрибута:|name|, |link".
- Если атрибут table является коллекцией объектов класса class, то в нем будут выведеные колонки name и link.

Формат отображения в семантике

- Можно обрезать вывод с помощью: [].

```
 "name[0,50]|..."
Указываем позицию и количество выводимых букв из семантики атрибута. Из атрибута name␣
↪выводим 50 символов семантики (значение атрибута), начиная с первого.
```

- Доступно разыменование через . т.е. доступ во вложенный объект.

```
 "semantic": "digitalTV|kachestvoCTB|analogTV.name|kachestvoAnal|period"
где ``analogTV`` - ссылочный атрибут класса, для которого задается семантика, а ``name`` -␣
↪атрибут класса по ссылке.
```

Отображение семантики на форме

1. В списках первого уровня (открываемые непосредственно по узлу навигации), в качестве заголовка выводим только значение из поля “caption” узла навигации.
2. В списках выбора в заголовок выводим только значение из поля “caption” класса объектов списка.
3. В форме редактирования в заголовок выводим только семантику объекта.
4. В форме создания в заголовок выводим только значение из поля “caption” класса.
5. В списках выбора над заголовком мелким шрифтом выводим строку “Выбор значения атрибута <...> объекта <...>”.
6. В форме создания, если создается объект в коллекции или ссылке, над заголовком мелким шрифтом выводим строку “Создание объекта в коллекции/по ссылке <...> объекта <...>”.

Атрибутивная часть меты класса описывает поля атрибута класса. Атрибутов в классе обычно минимум два - ключевое поле и поле данных. Они содержатся в виде массива в поле "properties" основной части меты классов. Каждый атрибут - это объект следующей структуры:

JSON


```
{
  "orderNumber": 20,
  "name": "integer_integer",
  "caption": "Редактор целых чисел [14]",
  "type": 6,
  "size": null,
  "decimals": 0,
  "allowedFileTypes": null,
  "maxFileCount": 0,
  "nullable": true,
  "readonly": false,
  "indexed": false,
  "unique": false,
  "autoassigned": false,
  "hint": null,
  "defaultValue": null,
  "refClass": "",
  "itemsClass": "",
  "backRef": "",
  "backColl": "",
  "binding": "",
  "semantic": null,
  "selConditions": [],
  "selSorting": [],
  "selectionProvider": null,
  "indexSearch": false,
  "eagerLoading": false,
  "formula": null,
  "cached": true
}
```

Описание полей

Код	Имя	Допустимые значения	Описание
"orderNumber"	Порядковый номер	Целое неотрицательное	Задает расположение атрибута относительно других атрибутов этого же класса.
"name"	Системное имя	Строка, только латиница без пробелов	Указывается имя атрибута, с которым будет работать система, а значит не может быть пустым, может содержитать только символовы латинского алфавита, без пробелов (задается один раз при создании атрибута). В последующем поменять имя нельзя.
"caption"	Логическое имя	Строка	Отображение имени атрибута в пользовательском интерфейсе.
"type"	Тип	Целое - идентификатор (код) типа	Тип данных атрибута. См. Типы атрибутов
"size"	Размер	Целое положительное	Максимальный размер данных атрибута, допистимые значения зависят от типа атрибута.
"decimals"	Число знаков после запятой	Целое неотрицательное	Число знаков после запятой, задается только для типа “Десятичное [8]”.
"allowedFileTypes"	Допустимые типы файлов	Массив строк	Позволяет задать допустимые расширения файлов, которые пользователь может загрузить в атрибут типа “Коллекция файлов [110]”.
"maxFilesCount"	Максимальное количество файлов	Число от 1 до 5	Задает максимальное количество файлов, которые пользователь может загрузить в атрибут типа “Коллекция файлов [110]”.
"nullable"	Допустимо пустое значение	Логический	Разрешает или запрещает пустое значение атрибута.
"readonly"	Только для чтения	Логический	Разрешает или запрещает изменять значение атрибута.
"indexed"	Индексация для поиска	Логический	Указывает, нужно ли индексировать значения данного атрибута для ускорения поиска.
"unique"	Уникальные значения	Логический	Для атрибута накладывает ограничение уникальности (Внимание: нельзя создать два объекта класса с одинаковыми значениями в уникальном атрибуте).
"autoassigned"	Автозаполнение	Логический	Разрешает или запрещает автоматическое заполнение поля приложением.
"hint"	Подсказка	Строка	Задает сообщение, которые выведется в пользовательском интерфейсе рядом с именем атрибута.
"defaultValue"	Значение по умолчанию	Зависит от типа атрибута	Указывается значение, которое будет заполнено в атрибуте в форме создания (при создании объекта).
"refClass"	Атрибут ссылки	Строка, только латиница, без пробелов	Содержит значение поля "name" (Системное имя) класса, который должен использоваться в атрибуте типа “Ссылка [13]”.
"itemsClass"	Атрибут коллекции	Строка, только латини-	Содержит значение поля "name" (Системное имя) класса, объекты которого могут привязаться к атрибуту типа “Коллекция [14]”

3.4.3 Мета навигации

Мета навигации - регулирует расположение элементов в навигационном блоке.

Мета навигации разделяется на мету узлов навигации и мету секции навигации.

Мета секций навигации

Мета секций навигации состоит из поля "name" + .section.json и находится в директории navigation.

Например: workflow.section.json.

Мета узлов навигации

Мета узлов навигации состоит из:

- Для узлов навигации первого порядка - тех узлов, которые находятся непосредственно в секции навигации: поле "code" + .json и находится в директории, имя которой совпадает с именем файла секции навигации к которому относится узел навигации.

Например: В директории navigation есть файл секции навигации simpleTypes.section.json. И есть узел навигации classString.json, который размещается в секции simpleTypes. Файл узла навигации будет иметь путь: navigation\simpleTypes\classString.json.

- Для узлов навигации второго порядка - тех узлов, которые входят в группу (особый тип узлов навигации, поле "type" в которых содержит значение 0). Разница в том, что поле "code" у таких узлов составное и состоит из поля "code" группы и личного наименования.

Например: navigation\relations\classReference.refBase.json. Это файл узла навигации refBase, который находится в группе classReferense секции навигации relations.

Условия выборки - "conditions"

Условия выборки "conditions" - это фильтры при открытии списка объектов.

Сохраненные фильтры

Перед открытием любой страницы, в реестрах происходит считывание фильтров, которые подходят для данного окна. Подходящие фильтры состоят из двух частей:

1. Общие фильтры, которые применимы для всех классов.
2. Фильтры, сохраненные конкретно для данного класса.

Общие фильтры для всех классов

Общие фильтры для всех классов - это фильтры, которые отображаются для всех открываемых классов. Их отличие в коде в том, что в атрибуте класса у общего фильтра стоит ключевое слово ALL, у персональных фильтров в этом атрибуте стоит название класса, для которого он применим.

Чтобы сделать фильтр для всех классов, при сохранении поставьте “Для всех классов”.

Фильтры для конкретных классов

Чтобы создать фильтр для конкретного класса, откройте объекты этого класса, сгенерируйте фильтр и сохраните его убедившись, что поле “Для всех классов” не отмечено.

Реализация в коде

Реализована единая спецификация выражений, как для вычисляемых выражений, так и для условий отбора и расчетов в агрегации.

В основном работаем с файлом _list-filter-ui - именно он запускает поиск нужных фильтров, а также разбирает текущие данные для создания новых фильтров. В файле _list-filter-ui описано какие атрибуты могут участвовать в создании фильтров и как именно они должны выглядеть и сохраняться (например, дата и чекбокс выглядят по разному).

В параметре cond находятся данные фильтра, которые в последствие подставляются в условие для поиска (в файле _metaCRUD.js).

Пример

```
if (cond !== undefined && cond !== '' && cond !== 'undefined') {
  var condObj = JSON.parse(cond);
  if (Array.isArray(condObj) && condObj.length > 0) {
    for (var k = 0; k < condObj.length; k++) {
      if (condObj[k].type === 6) {
        condObj[k].value = new Date(condObj[k].value);
      }
      where[condObj[k].property] = getwherebyOperation(condObj[k]);
    }
  }
}
```

NB: новые фильтры - min и max расширяют возможности создания фильтров и условий в меню.

Необходимая мета

Необходимая мета для работы - это класс фильтров ion_filter. Он находится в папке calc, которая по умолчанию является папкой с классами и метой для системы. Кроме одного класса ничего более не нужно. Пересмотреть

Поиск по ссылочным объектам

Если поиск идет по принципу равно или содержит - то ищется в семантике этого объекта. Если поиск идет по принципу - максимум/минимум - то ищем уже значение поля. Таким образом, есть возможность поиска в ссылочных атрибутах, при этом без лишних запросов к базе, что повышает производительность.

```
[{property:okatoNasPunkta_title,operation:20,value:Лес,title:Населенный пункт содержит Лес,type:2}]
```

Фильтры в меню

Есть возможность не только выдавать отсортированные данные в списке из условий меню, но и ограничивать выборки, т.е. применять фильтры.

Мета отвечающая за работу фильтров

Пример меню с фильтром

```
{
  "code": "passportObject.naselenie",
  "type": 1,
  "orderNumber": 10,
  "caption": "Население",
  "classname": "naselenie",
  "container": null,
  "collection": null,
  "conditions": [
  {"property": "god"
  ,"operation": 10}
  ],
  "sorting": [],
  "pathChains": []
}
```

Атрибут conditions содержит два объекта:

1. property - свойство, по которому происходит фильтрация
2. operation - операция фильтрации

В данном случае этот фильтр имеет такой смысл - показать все объекты класса ‘‘naselenie‘‘ с минимальным годом.

Если нужно указать значение, то третьим атрибутом пойдет value и значение для поиска.

Например:

```
{"property": "god"
  ,"operation": 0
, "value": 2015}
```

Настройка фильтра для отображения объектов класса-наследника

Страницей класса для узла навигации является родительский класс. Если при переходе по данной навигации необходимо отображать объекты класса наследника данного класса, то применяется фильтр вида:

```
{
   property: "atr1.__class",
   operation: 0,
   value: ["childClass@ns"]
}
```

где atr1.__class - атрибут родительского класса, по которому идет выборка объектов, childClass - наследник, объекты которого отображаются в навигации. То есть - показать на форме списка только те объекты, у которых атрибут ‘‘atr1‘‘ является объектом класса-наследника ‘‘childClass‘‘.

Таблица операций

Поле	Наименование	Допустимые значения	Описание
"property"	Атрибут	Строка, только латиница без пробелов	Атрибут класса, значение поля которого проверяется на соответствие данному условию данного вектора.
"operation"	Операция	Код операции	Операция, согласно которой производится определение.
		0 - равно (И)	
		1 - не равно (ИЛИ)	
		2 - пусто (НЕ)	
		3 - не пусто (МИН ИЗ)	
		4 - (МАКС ИЗ)	
		5 - < ()	
		6 - >	
		7 - <=	
		8 - >=	
		9 - IN /Похож/	
		10 - содержит	
"value"	Значение	Зависит от типа операции	Второе значение для бинарных операций
"nestedConditions"	Вложенные условия отбора	Объект, структура аналогична структуре самого объекта условий отбора.	

NB: код операции соответствует разным значениям операций, в зависимости от того, выбран атрибут или нет. Если поле "property" равно null, то кодируется логическое условие, по которому объединяются вложенные условия отбора (указаны в скобках в таблице выше).

Операции для дат

код	значение	системное имя
8	Создаем дату	DATE
9	Добавляем к дате интервал	DATEADD
10	Находим интервал между датами	DATEDIFF
12	Вычитание	
24	День месяца	

Аргументы DATEADD: дата, интервал, ед.изм интервала [ms, s, min, h, d, m, y] (по умолчанию - день(d))

Аргументы DATEDIFF: конечная дата, начальная дата, ед. изм. результата [ms, s, min, h, d, m, y] (по умолчанию - день(d)), логический флаг приведения к целому числу

Сравнение текущей даты с месяцем

Настройка выборки объектов в списке с возможностью сравнения значения даты с любым месяцем года. Например, настройка фильтра таким образом, чтобы в навигации показывались только те объекты, у которых значение атрибута “Дата окончания” - текущий месяц.

Для этого вычисляется начало текущего месяца. После этого к нему можно добавлять или вычитать произвольное колличество месяцев и сравнивать полученный результат с необходимой датой.

Вычисление конца текущего месяца:

```
{
  "property": null,
  "operation": 9,
  "value": null,
  "nestedConditions": [
    {
      "property": null,
      "operation": 9,
      "value": null,
      "nestedConditions": [
        {
          "property": null,
          "operation": 8,
          "value": ["today"],
          "nestedConditions": []
        },
        {
          "property": null,
          "operation": 12,
          "value": null,
          "nestedConditions": [
            {
              "property": null,
              "operation": null,
              "value": [0],
              "nestedConditions": []
            },
            {
              "property": null,
              "operation": 24,
              "value": null,
              "nestedConditions": [
                {
                  "property": null,
                  "operation": 8,
                  "value": ["today"],
                  "nestedConditions": []
                }
              ]
            }
          ]
        },
        {
          "property": null,
          "operation": null,
          "value": ["d"],
          "nestedConditions": []
```

(continues on next page)

(continued from previous page)

```
      }
    ]
  },
  {
    "property": null,
    "operation": null,
    "value": [1],
    "nestedConditions": []
  },
  {
    "property": null,
    "operation": null,
    "value": ["m"],
    "nestedConditions": []
  }
 ]
}
```

1. Для начала вычисляется значение дня месяца для текущей даты:

```
{
 "property": null,
 "operation": 24,
 "value": null,
 "nestedConditions": [
  {
   "property": null,
   "operation": 8,
   "value": ["today"],
   "nestedConditions": []
  }
 ]
}
```

2. Получено условное значение “d”. Далее необходимо отнять полученное значение от 0 (0-d):

```
{
 "property": null,
 "operation": 12,
 "value": null,
 "nestedConditions": [
  {
   "property": null,
   "operation": null,
   "value": [0],
   "nestedConditions": []
  },
  {
   "property": null,
   "operation": 24,
   "value": null,
   "nestedConditions": [
    {
     "property": null,
     "operation": 8,
     "value": ["today"],
```

(continues on next page)

(continued from previous page)

```
        "nestedConditions": []
      }
    ]
  }
 ]
}
```

3. Получено условное значение “-d”. Далее к текущей дате прибавляется значение “-d” дней:

```
{
 "property": null,
 "operation": 9,
 "value": null,
 "nestedConditions": [
  {
   "property": null,
   "operation": 8,
   "value": ["today"],
   "nestedConditions": []
  },
  {
   "property": null,
   "operation": 12,
   "value": null,
   "nestedConditions": [
    {
     "property": null,
     "operation": null,
     "value": [0],
     "nestedConditions": []
    },
    {
     "property": null,
     "operation": 24,
     "value": null,
     "nestedConditions": [
      {
       "property": null,
       "operation": 8,
       "value": ["today"],
       "nestedConditions": []
      }
     ]
    }
   ]
  },
  {
   "property": null,
   "operation": null,
   "value": ["d"],
   "nestedConditions": []
  }
 ]
}
```

4. Получено начало текущего месяца.

5. Для вычисления конца текущего месяца нужно прибавить к полученному значению начала ме-

сяца 1 месяц:

```
{
  "property": "date",
  "operation": 5,
  "value": null,
  "nestedConditions": [
    {
      "property": null,
      "operation": 9,
      "value": null,
      "nestedConditions": [
        {
          "property": null,
          "operation": 9,
          "value": null,
          "nestedConditions": [
            {
              "property": null,
              "operation": 8,
              "value": ["today"],
              "nestedConditions": []
            },
            {
              "property": null,
              "operation": 12,
              "value": null,
              "nestedConditions": [
                {
                  "property": null,
                  "operation": null,
                  "value": [0],
                  "nestedConditions": []
                },
                {
                  "property": null,
                  "operation": 24,
                  "value": null,
                  "nestedConditions": [
                    {
                      "property": null,
                      "operation": 8,
                      "value": ["today"],
                      "nestedConditions": []
                    }
                  ]
                }
              ]
            },
            {
              "property": null,
              "operation": null,
              "value": ["d"],
              "nestedConditions": []
            }
          ]
        },
        {
```

(continues on next page)

(continued from previous page)

```
        "property": null,
        "operation": null,
        "value": [1],
        "nestedConditions": []
      },
      {
        "property": null,
        "operation": null,
        "value": ["m"],
        "nestedConditions": []
      }
    ]
  }
 ]
}
```

Мета узлов навигации

JSON

```
{
 "code": "classDatetime",
 "orderNumber": 0,
 "type": 1,
 "caption": "Класс \"Дата/Время [9]\"",
 "classname": "classDatetime",
 "container": null,
 "collection": null,
 "url": null,
 "external": true,
 "hint": null,
 "conditions": [],
 "sorting": [],
 "eagerLoading": {
   "list": { // Здесь задается жадная загрузка для списков
      "internet": ["okato"],
      "someClass1": ["refAttr1", "refAttr2.refAttr3"],
      "someClass2": ["colAttr4"]
   },
   "item": { // Здесь задается жадная загрузка для форм редактирования
      "internet": ["okato", "standart"],
      "someClass1": ["refAttr1", "refAttr2.refAttr3", "refAttr5", "colAttr4"],
      "someClass2": ["colAttr4"]
   }
  },
 "pathChains": [],
 "searchOptions": null
 "metaVersion": "2.0.7"
}
```

Описание полей

Поле	Наименование	Допустимые значения	Описание
"code"	Системное имя	Строка латиницей, без пробелов	Системное имя узла навигации может быть составным, если принадлежит узлу навигации типа Группа [0].
"orderNumber"	Порядковый номер	Целое число	Задает порядок сортировки пунктов меню в пределах секции навигации
"type"	Тип	Целое число	Задает логику работы пункта меню, выводимые при переходе/активации значения. Накладывает ограничения на прочие поля меты узла навигации.
		0	Группа. Объединяет в себе другие узлы навигации по какому-либо общему признаку, не является страницей класса.
		1	Страница класса. Отображает структуру и объекты класса, заданного в поле “classname”.
		2	Страница контейнера. Выполняет отображение списка объектов в коллекции. Для таких узлов навигации нужно указывать класс и идентификатор объекта-контейнера, а также имя атрибута коллекции.
		3	Гиперссылка. Осуществляет переход на ссылку, заданную в поле “url”
"title"	Заголовок	Строка	Позволяет дополнительно указать заголовок страницы.
"caption"	Логическое имя	Строка	Наименование узла навигации отображаемое в интерфейсе.
"classname"	Класс	Строка латиницей, без пробелов	Если “Тип” - “Страница класса (1)”, то поле обязательно к заполнению.
"container"	ID контейнера	Строка или null	Идентификатор объекта содержащего коллекцию отображаемую на странице.
"collection"	Атрибут коллекции	Строка или null	Имя атрибута коллекции, содержимое которого надо вывести на странице.
"url"	URL	Гиперссылка (принимает любые строки)	Если “Тип” - “Гиперссылка (3)”, то поле обязательно к заполнению.
"external"	Признак внешнего ресурса	Логический	Открывает страницу по ссылке в новом окне, если присвоено значение true.
"hint"	Подсказка	Строка	Текст, заданный в этой строке, отображается при наведении на узел навигации, которому она принадлежит.
"conditions"	Условия выборки	Массив объектов	Фильтр при открытии списка объектов. Используется для узлов типа «Страница класса» и «Страница контейнера».
"sorting"	Сортировка	Массив объектов	Используется для узлов типа «Страница класса» и «Страница контейнера». Здесь задаются параметры сортировки объектов в списке

Настройка поиска в узле навигации

```
"searchOptions": {
 "person": {
  "searchBy": [ // атрибуты по которым ищем, по умолчанию то, что выводится в колонках
    "surname",
    "name",
    "patronymic"
  ],
  "splitBy": "\\s+", // разбивать поисковую фразу на регулярное выражение, части сопоставить с
↪атрибутами
  "mode": ["starts", "starts", "starts"], // режимы сопоставления - в данном случае "начинается с"
↪(доступны like, contains, starts, ends)
  "joinBy": "and" // режим объединения условий на атрибуты (по умолчанию or)
 }
}
```

Структура в mongoDB (registry)

```
{
    "_id" : ObjectId("578f07aa0ce0024ce143e71e"),
    "code" : "classDatetime",
    "orderNumber" : 0,
    "type" : 1,
    "caption" : "Класс \"Дата/Время [9]\"",
    "classname" : "classDatetime",
    "container" : null,
    "collection" : null,
    "url" : null,
    "hint" : null,
    "conditions" : [],
    "sorting" : [],
    "pathChains" : [],
    "itemType" : "node",
    "section" : "simpleTypes",
    "namespace" : ""
}
```

Мета секций навигации

JSON

```
{
  "caption": "Простые типы",
  "name": "simpleTypes",
  "mode": 0,
  "tags": null
}
```

Описание полей

Поле	Наименование	Допустимые значения	Описание
"caption"	Логическое имя	Строка	Наименование секции навигации отображаемое в интерфейсе.
"name"	Системное имя	Строка латиницей, без пробелов	Задает в том числе первую часть имени файла меты секции навигации, служебное имя.
"mode"	Режим отображения	Меню: 0	Задает режим отображения меню.
		Содержание: 1	
		Ниспадающий список: 2	
		Иерархия: 3	
"tags"	Теги	Массив строк либо null.	Теги. Могут определять дополнительные свойства секции, влиять на отображение. Не реализовано.

Структура в mongoDB (registry)

```
{
    "_id" : ObjectId("578f07aa0ce0024ce143e720"),
    "caption" : "Простые типы",
    "name" : "simpleTypes",
    "mode" : 0,
    "tags" : null,
    "itemType" : "section",
    "namespace" : ""
}
```

Заголовок страницы - "title"

Поле “title” необходимо, чтобы задать отличный от поля "caption" заголовок страницы в меню навигации.

Описание

- Для формирования заголовка страницы в первую очередь используется значение поля "title" соответствующего узла навигации. Если поле "title" не задано - пустая строка, то для формирования заголовка страницы используется поле "caption" меты узла навигации.
- Для формирования заголовка страницы на страницах списка выбора (при открытии списка класса справочника из форм) используется "caption" общей части меты класса.

JSON

```
{
  "code": "administrations.municipals",
  "orderNumber": 0,
  "type": 1,
  "title": "Администрации муниципальных районов", \\ отличается от наименования узла навигации поля `
↪"caption"`
  "caption": "Муниципальные районы",
  "classname": "Organisation",
  "container": null,
  "collection": null,
  "url": null,
  "hint": "Список администраций муниципальных районов",
  "conditions": [],
  "sorting": [],
}
```

3.4.4 Мета отчёта

Пример простого полного отчета

```
name: support
caption: Отчет по заявкам в техническую поддержку
sources:
  - name: task
    caption: Заявка
    load:
      - className: task
        results:
          - field: id
            expr: $id
          - field: date
            expr: $dateCreate
          - field: typeComunication
            expr:
              if:
                - eq:
                  - $typeComunication
                  - call
                - 'Звонок'
                - if:
                  - eq:
                    - $typeComunication
                    - metting
                  - 'Встреча'
                  - if:
                    - eq:
                      - $typeComunication
                      - letter
                    - 'Письмо'
```

(continues on next page)

(continued from previous page)

```
            - if:
              - eq:
                - $typeComunication
                - mail
              - 'E-mail'
              - ' '
      - field: typeTask
        expr:
          if:
            - eq:
              - $typeTask
              - question
            - 'Консультация'
            - if:
              - eq:
                - $typeTask
                - problem
              - 'Инцидент'
              - if:
                - eq:
                  - $typeTask
                  - offer
                - 'Предложение'
                - ' '
      - field: predmetSupport
        expr: $support.name
      - field: temaTask
        expr:
          if:
            - nempty:
              - $supportScenario0
            - $supportScenario0.name
            - ' '
      - field: nameClassification
        expr:
          if:
            - nempty:
              - $supportScenario1
            - $supportScenario1.name
            - if:
              - nempty:
                - $supportScenario2
              - $supportScenario2.name
              - if:
                - nempty:
                  - $supportScenario3
                - $supportScenario3.name
                - if:
                  - nempty:
                    - $supportScenario4
                  - $supportScenario4.name
                  - if:
                    - nempty:
                      - $supportScenario5
                    - $supportScenario5.name
                    - if:
```

(continues on next page)

(continued from previous page)

```
                    - nempty:
                      - $supportScenario6
                    - $supportScenario6.name
                    - ' '
        - field: coment
          expr: $comment
    index:
      - id
  - name: support
    caption: Заявки в техническую поддержку
    load:
      - source: task
        joins:
          - table: date
            alias: da
            left: id
            right: id
          - table: typeComunication
            alias: comun
            left: id
            right: id
          - table: typeTask
            alias: ta
            left: id
            right: id
          - table: predmetSupport
            alias: sup
            left: id
            right: id
          - table: coment
            alias: com
            left: id
            right: id
        results:
          - field: id
            expr: $id
          - field: date
            expr: $date
          - field: typeComunication
            expr: $typeComunication
          - field: typeTask
            expr: $typeTask
          - field: predmetSupport
            expr: $predmetSupport
          - field: temaTask
            expr: $temaTask
          - field: nameClassification
            expr: $nameClassification
          - field: coment
            expr: $coment
reports:
  - name: technicalSupport
    caption: Заявки ТП
    sheets:
      - name: technicalSupport
        caption: Заявки в техническую поддержку
```

(continues on next page)

(continued from previous page)

```
    type: aggregation
    source: support
    fetch:
      date: $date
      typeComunication: $typeComunication
      typeTask: $typeTask
      predmetSupport: $predmetSupport
      temaTask: $temaTask
      nameClassification: $nameClassification
      coment: $coment
    rangeFilters:
      date:
        caption: За период с|по
        format: date
        inclusive: both
    columns:
      - field: date
        caption: Дата создания
      - field: typeComunication
        caption: Тип коммуникации
      - field: typeTask
        caption: Тип заявки
      - field: predmetSupport
        caption: Предмет поддержки
      - field: temaTask
        caption: Тема заявки
      - field: nameClassification
        caption: Наименование классификации
      - field: coment
        caption: Комментарий
```

Описание

Мета отчёта - предназначена для построения шахты данных, содержащей аналитическую информацию по данным из меты системы, организованную в виде таблиц. В мете модуля отчетов указываются источники данных, на основе которых формируется информация для построения отчета, и в дальнейшем формирование колонок таблицы отчета, с указанием ресурса на данные из метаклассов системы.

Мета модуля отчетов находится в папке bi проекта в формате YML.

NB: Определение “Шахта данных”

```
Шахта данных - (смежный термин от англ. Data Mining - глубинный анализ данных)
это некое хранилище, которое содержит глубинную аналитическую информацию
обо всех источниках данных и информацию для построения отчетов,
организованную в виде таблиц.
```

Пример YML

```
name: reportTest
caption: Тестовые данные
```

(continues on next page)

(continued from previous page)

```
sources:
  - name: dataSource
    caption: Источник данных
    load:
      - className: sourceClass
        results:
          - field: id
            expr: $id
          - field: date
            expr: $dateCreate
          - field: name
            expr: $nameObject
    index:
      - id
  - name: test
    caption: Отчет тестовый
    load:
      - source: dataTest
        joins:
          - table: date
            alias: da
            left: id
            right: id
        results:
          - field: id
            expr: $id
          - field: date
            expr: $date
          - field: name
            expr: $name
reports:
  - name: reportTest
    caption: Отчет тестовый
    sheets:
      - name: reportTest
        caption: Отчет тестовый
        type: aggregation
        source: test
        fetch:
          date: $date
        rangeFilters:
          date:
            caption: За период с|по
            format: date
            inclusive: both
        columns:
          - field: date
            caption: Дата создания
          - field: name
            caption: Наименование
```

Описание примера

Отчет тестовый содержит в себе данные из класса “sourceClass”. Источник данных “dataSource” извлекает данные из меты соответствующего класса, которые указаны в свойстве results: . Далее подраздел

“test” на основе данных, полученных из источника, указанного в свойстве source: формирует и преобразовывает данные для корректного отображения в таблицах отчета. Свойство joins: задает атрибут, который является идентификатором для построения отчета (в данном случае id объекта).

Далее система формирует таблицы отчета, на основе преобразованных данных из источника, в разделе reports:. Свойство rangeFilters: содержит информацию о фильтрах, настраиваемых для отчета (в данном случае необходимо указать диапазон дат, в соответствии с данными из класса). В модуле фильтр по диапазону задается через параметры запроса: ?rangeFld[]=0&rangeFld[]=5, где rangeFld - это поле по которому ищем. Если идет поиск по датам - дату передавать в формате локали, которая передается в http-заголовке `'accept-language'`, либо в формате ISO8601. Свойство columns: позволяет формировать колонки таблицы (порядковые номера фактические).

Результат: таблица из двух колонок (Дата и Наименование), в которой будут выводится объекты класса из источника _”dataSource”_ , в соответствии с фильтром по датам, настроенном в rangeFilters:, а количество объектов в таблице будет равно количеству значений идентификатора, настроенном в свойстве joins:.

Пример простого полного отчета можно посмотреть здесь.

Настройка строгости сравнения

Настройка строгости сравнения на границах интервала rangeFilters в отчете:

```
"rangeFilters": {
        "regDate": {
          "caption": "За период с|по",
          "format": "date",
          "inclusive": "both" | "left" | "right"
        }
      }
```

both - обе границы могут быть равны искомым значениям

left - левая граница (меньшая) может быть равна искомым значениям

right - правая граница (большая) может быть равна искомым значениям

Если inclusive не указан - сравнение строгое на обоих границах.

Иерархическая сборка

Настройка иерархической сборки необходима для обработки исходных данных при сборке шахты:

- Чтобы сделать в одном источнике данных выгрузку данных по всей иерархии в базе
- Чтобы вывести данные по первому столбцу с отступами в зависимости от глубины вложенности

Настройка иерархической сборки в шахте данных:

В конфигурации источника настройка "hierarchyBy" представляет собой объект с набором свойств: id, parent, level, order.

```
hierarchyBy:
      id: guidProj
      parent: basicobj1.guidObj
      level: objLevel
      order: objOrder
```

где id - атрибут в данных, идентифицирующий элемент иерархии

parent - атрибут в данных, содержащий идентификатор родительского элемента

level - атрибут в результирующем источнике, куда будет записан уровень вложенности элемента

order - атрибут в результирующем источнике, куда будет записано значение для упорядочивания иерархии при отображении.

Поля objLevel и objOrder это поля для записи значения (их не надо считать, агрегировать и т.д.)

Пример YML

```
reports:
  - name: roadmap
    caption: Дорожная карта
    sheets:
      - name: roadmap
        caption: >-
          Дорожная карта
        type: aggregation
        needFilterSet: true
        needFilterMessage: Выберите проект
        styles:
          objLevel:
            1: text-indent-1
            2: text-indent-2
            3: text-indent-3
          nameObjIndex:
            "3": level2
            "2": level1
            "1": level0
            "0": level0
        source: roadmapSource
        fetch:
          objLevel: $objLevel
          guidObj: $guidObj
          numLevelObj: $numLevelObj
...
```

NB: Иерархическая сборка возможна только на основе источника и невозможна на основе класса.

Алгоритм сборки:

1. Создаем результирующий источник.
2. Делаем выборку корневых элементов, у которых пустое поле parent.
3. Перебираем и записываем элементы в результирующий источник (при этом в спецатрибут element_id - идентификатор (id) обьекта, в level - значение 0, в order - приведенный к строке порядковый номер элемента в выборке, дополненный до длины 6 символов лидирующими нолями).
4. Итеративно делаем выборки следующих уровней вложенности (начиная с 0), до тех пор пока на очередной итерации не будет извлечено 0 объектов. Выборки делаются путем объединения исходного источника с результирующим по связи parent = element_id и ограничению level=текущий уровень вложенности.

5. На каждой итерации перебираем и записываем элементы в результирующий сорс, при этом:
   - в спецатрибут element_id пишем идентфикатор (id) обьекта,
   - в level пишем текущий уровень вложенности,
   - в order пишем конкатенацию order родительского элемента и приведенного к строке порядкового номера элемента в выборке, дополненного до длины 6 символов лидирующими нолями.

Настройка скрытия объектов

Настройка скрытия всех объектов, если табличные фильтры не заданы. Чтобы при открытии отчета все объекты скрывались, пока не будет выбрано значение из списка в фильтре необходимо для него применить настройку "needFilterSet: true".

Отображение в заголовке отчета параметров выборки посредством паттернов

Пример YML

```
...
        byPeriod:
          sum:
            - if:
                - and:
                    - gte:
                        - $date
                        - ':since' # берем из params->since
                    - lte:
                        - $date
                        - ':till' # берем из params->till
                - $amount
                - 0
        byMonth:
          sum:
            - if:
                - and:
                    - eq:
                        - month:
                            - dateAdd:
                                - $date
                                - 10
                                - h
                        - ':month' # берем из params->month
                    - eq:
                        - year:
                            - dateAdd:
                                - $date
                                - 10
                                - h
                        - ':year' # берем из params->year
                - $amount
                - 0
        byYear:
          sum:
            - if:
                - eq:
```

(continues on next page)

(continued from previous page)

```
                - year:
                  - dateAdd:
                    - $date
                    - 10
                    - h
                - ':year' # берем из params->year
              - $amount
              - 0
...
    params:
      year:
        caption: Год
        format: int
      month:
        caption: Месяц
        format: int
        select: # выпадающий список
          '1': январь
          '2': февраль
          '3': март
          '4': апрель
          '5': май
          '6': июнь
          '7': июль
          '8': август
          '9': сентябрь
          '10': октябрь
          '11': ноябрь
          '12': декабрь
      since:
        caption: с
        format: date
      till:
        caption: по
        format: date
...
    columns:
      - field: title
        caption: Показатель
      - field: dimension
        align: center # наименование заголовка в шапке по центру ячейки
        caption: Единица измерения
      - caption: '{$year}' # наименование заголовка в шапке из параметра year
        align: center
        columns: # колонка в шапке - группа вложенных колонок
          - field: byPeriod
            # наименование заголовка в шапке из параметров since и till
            caption: 'с {$since} по {$till}'
            align: center
            format: number
          - field: byMonth
            # наименование заголовка в шапке из параметра month
            caption: 'За {$month}'
            align: center
            format: number
          - field: byYear
```

(continues on next page)

(continued from previous page)

```
        caption: За год
        align: center
        format: number
```

Стилизация строк отчета на основании данных

Пример YML

```
...
    fetch:
      category: $category
      title:
        case:
          - eq:
              - $category
              - AA4
          - 'Выдано заключений, всего в т.ч.:'
          - eq:
              - $category
              - AB5
          - '1. Государственная экспертиза, всего в т.ч.:'
          - eq:
              - $category
              - AC6
          - '- положительных'
          - eq:
              - $category
              - AD7
          - '- отрицательных'
...
      dimension:
        case:
          - eq:
              - $category
              - AA4
          - штук
          - eq:
              - $category
              - AB5
          - штук
...
    styles:
      category:
        AA4: level0
        AB5: level1
        AC6: level2
        AD7: level2
```

Возможность использования комбобоксов в параметрах и фильтрах

Пример YML

```
...
        params:
          year:
            caption: Год
            format: int
          month:
            caption: Месяц
            format: int
            select: # выпадающий список
              '1': январь
              '2': февраль
              '3': март
              '4': апрель
              '5': май
              '6': июнь
              '7': июль
              '8': август
              '9': сентябрь
              '10': октябрь
              '11': ноябрь
              '12': декабрь
          since:
            caption: с
            format: date
          till:
            caption: по
            format: date
...
```

Настройка обработки параметров в фильтре на странице отчета

Пример YML

```
reports:
  ...
  filter:
          eq:
            - $yearStart
            - year:
              - ':dateSelect'
 ...
```

Значение года в атрибуте $yearStart равно значению года из даты в атрибуте :dateSelect.

Настройка пагинатора "pageSize"

NB: Применяется для отчетов с типом type: list.

Для случаев, когда отчет содержит в себе много объектов и на страницах нужно выводить строки постранично, чтобы не нагружать браузер тяжелой обработкой данных.

Пример YML

```
reports:
 - name: test
    caption: Тестовый отчет
    sheets:
      - name: main
        caption: Тестовый отчет
        type: list
        pageSize: 100
```

Настройка вывода построчно

Настройка вывода вложенных данных в отчете построчно настраивается следующим образом:

Пример YML

```
...
reports:
  - name: testReport
  ...
      columns:
        - caption: Группирующее поле
          columns: // поля для группировки
            - field: columns1
              caption: Поле1
              format: string
            - field: columns2
              caption: Поле2
              format: string
    ...
```

Настройка инкрементальной загрузки

Для настройки инкрементальной загрузки данных в источник при сборке шахты необходимо выставить параметр:

```
append: true
```

Он необходим для подгрузки статистики за день в шахту, чтоб не пересчитывать весь объем исходных данных и иметь историю по периодам.

Особенности сортировки объектов

Учитывая функционал агрегации MongoDB - сортировка возможна только по результирующим полям. Это значит, что для обратной совместимости поля результата, по которым сортируем, необходимо называть так же, как и поля в источнике данных.

Пример сортировки (свойство sort):

```
reports:
  - name: sors
    caption: Источник
    sheets:
      ...
        rangeFilters:
          ...
        sort:
          regDateOrder: asc
        columns:
          ...
```

3.4.5 Мета безопасности

Описание

Мета безопасности - регулирует настройку прав безопасности на объекты системы. Можно разделить на статическую и динамическую безопасность:

Статическая безопасность - регулирует права доступа на объекты системы для определенной роли.

Динамическая безопасность - регулирует права доступа на объекты системы для конкретной персоны, в соответствии с какими-либо условиями, в то время как групповая динамическая безопасность - это права для группы безопасности.

Настройка динамической безопасности производится в файле deploy.json, а также в файле acl/resources-and-roles.yml. Статическая безопасность задается только в файле acl/resources-and-roles.yml.

Правила формирования идентификаторов ресурсов

- узел навигации - n:::namespace@code
- класс - c:::classname@namespace
- объект - i:::classname@namespace@id
- атрибут - a:::classname@namespace.propertyname
- геомета:
  - узел навигации: geonav:::код узла@namespace
  - слой: geolayer:::код слоя@namespace
  - данные: geodata:::код слоя@namespace@индекс запроса
- пути (модулей):
  - модуль portal: sys:::url:portal/*
  - модуль geomap: sys:::url:geomap/*

Типы прав

Чтение read

read - это право на просмотр информации по объектам класса. Задает разрешение просматривать объекты класса только для чтения и запрещает их создание/редактирование.

```
- id: Users
    name: Обычные пользователи
    permissions:
      n:::ns@navigationName:
        - read
...
```

Запись write

write - это право на создание объектов класса. Задает разрешение на создание новых объектов класса, но запрещает редактирование существующих.

```
- id: Users
    name: Обычные пользователи
    permissions:
      n:::ns@navigationName:
        - write
...
```

Использование use

use - это право на создание объектов класса. Задает разрешение на создание объектов класса, и использование объектов класса в ссылках и коллекциях.

Без use - ссылки тоже отображаются в коллекции. Если есть read, но нет use, то нельзя выбрать объект и поместить его в коллекцию.

```
- id: Users
    name: Обычные пользователи
    permissions:
      n:::ns@navigationName:
        - use
...
```

Удаление delete

delete - это право на удаление объектов класса.

Полный доступ full

full - это право на полный доступ к объектам класса.

Динамическая безопасность

```
"PROJECT_BENEFITIAR": {
"resource":

{ "id": "pm::project" }
,
"attribute": "stakeholders.id"
}
```

Если у проекта в stakeholders.id есть значение ассоциированное с текущим пользователем (настроено подтягивание организации как глобальной роли пользователя), то стоит учитывать текущего пользователя PROJECT_BENEFITIAR и проверить права на ресурс pm:project - эти права и будут правами на проект.

pm:project - это некий виртуальный ресурс безопасности. Необходимо абстрагировать настройки доступа от проверяемого объекта для разных ролей, можно разные ресурсы указать на один класс и наоборот.

Если ресурс не указать, то будут проверяться права на класс объекта. Тогда эту роль можно использовать как статическую, то есть выдавать статические права динамически.

Групповая динамическая безопасность

```
"roleMap": {
  "organization@project-management": {
    "ORGANIZATION_STAFF": {
      "caption": "Сотрудник организации",
       "resource": {
         "id": "pm::organization",
         "caption": "Организация"
       },
       "sids": [ // применять роль, если:
         "$employee", // в атрибуте employee связанное с user значение (user это сотрудник)
         // ИЛИ
         "admin", // user это admin (здесь роль, учетная запись или идентификаторы связанные с user)
         // ИЛИ
         [
            "$boss", // в атрибуте $boss связанное с user значение (user это руководитель)
            // И
            "supervisor" // user это supervisor (роль или учетная запись)
         ]
       ],
       "conditions": {"eq": ["$state", "active"]}, // применять роль только для активных организаций
       "attribute": "employee.id", // добавляется к sids
     }
    }
     }
```

При указании sids каждый уровень вложенности массивов значений меняет вид операции И/ИЛИ. На первом уровне применяется ИЛИ.

Определение ролей пользователя

1. Регистрируем пользователя с полным административным доступом - admin.

2. Под admin в registry в разделе Безопасность.Подразделения заводим иерархию подразделений (код подразделения = идентификатор безопасности).
3. Регистрируем пользователя без прав - user.
4. Под админ в registry в разделе Безопасность.Сотрудники заводим Сотрудника, указываем у него в атрибуте Пользователь пользователя без прав. Привязываем сотрудника к самому нижестоящему подразделению.
5. Заходим под user - прав нет ни на что.
6. Заходим под admin и даем роли (соответствующей самому вышестоящему подразделению) права на произвольные классы и узлы навигации.
7. Заходим под user - есть доступ к объектам, к которым есть доступ у подразделения.
8. Аналогично проверяем применение разрешений по всей иерархии подразделений.

Пример настройки в deploy.json

```
"actualAclProvider":{
      "module": "core/impl/access/aclmongo",
      "initMethod": "init",
      "initLevel": 1,
      "options":{
        "dataSource": "ion://Db"
      }
    },
    "roleAccessManager": {
      "module": "core/impl/access/amAccessManager",
      "initMethod": "init",
      "initLevel": 1,
      "options": {
        "dataSource": "ion://Db"
      }
    },
    "aclProvider": {
      "module": "core/impl/access/aclMetaMap",
      "options":{
        "dataRepo": "ion://dataRepo",
        "acl": "ion://actualAclProvider",
        "accessManager": "ion://roleAccessManager",
        "map": {
          "employee@develop-and-test": {
            "isEntry": true,
            "sidAttribute": "uid",
            "jumps": ["department"]
          },
          "department@develop-and-test": {
            "sidAttribute": "code",
            "jumps": ["superior"]
          }
        }
      }
    }
```

Модель отображения атрибутов и объектов в соответствии с заданными правами

Есть класс [Проекты], в нем содержится атрибут с типом “Коллекция” - [Мероприятия]. Если для класса [Мероприятия] нет прав на чтение, то атрибут этого класса не показывается на форме класса [Проекты].

Если для класса есть динамическая безопасность - то независимо есть или нет права на чтение класса [Мероприятия] - атрибут на форме класса [Проекты] будет показан, но сами объекты мероприятий будут показаны только если на них есть права.

NB: Для отображения атрибута и объектов необходимо задавать как статическую, так и динамическую безопасность на класс по ссылке атрибута.

Если есть статическое право на чтение на класс, то пользователь увидит все объекты этого класса вне зависимости от динамических прав, и дополнительно будет делаться выборка объектов, на которые настроена динамическая безопасность и они будут отображаться для пользователя в соответствии с настройками в ней.

3.4.6 Мета представлений - общая часть

Поиск в списках объектов “allowSearch”

Поле основной части меты представления списка "allowSearch" разрешает или запрещает отображение в форме поля поиска.

Логика работы

Архитектурой платформы и регистри накладываются следующие ограничения.

Для того, чтобы работал поиск в представлении “СПИСОК”, нужно выполнение одного из условий:

- В ключе класса должны быть поля какого-нибудь из следующих типов: Строка, Дата-Время, Целое, Действительное, Десятичное
- В классе должны быть не ключевые атрибуты этих же типов помеченные как индексируемые

Если ни одно из условий не соблюдается, поиск в представлении “СПИСОК” невозможен - соответственно поле для поиска не отображается. Если одно или оба условия выполняются, то поиск доступен и выполняется путем сопоставления каждого индексированного атрибута с поисковой фразой. Если хотя бы один из атрибутов соответствует поисковой фразе - объект считается удовлетворяющим условию поиска и добавляется в выборку.

В зависимости от поля, сопоставление выполняется по следующим правилам:

- Строка - ищется вхождение поисковой фразы в значение строкового атрибута посредством регулярного выражения.
- Дата-время - поисковая фраза приводится к дате-времени, и, если получилось это сделать, то проверяется эквивалентность значения атрибута полученной дате. Сравнение строгое - до секунд, т.е. если в поиске не указано время - будут искаться даты с временем 00:00.
- Целое, Действительное, Десятичное число - поисковая фраза приводится к числу и проверяется равенство значению атрибута.

Поиск по объектам вычислимых атрибутов

Для объектов вычисляемых атрибутов поиск будет работать только в случае наличия настройки кеширования подробнее. Соответсвенно, нет необходимости для вычисляемых атрибутов ставить indexed , если они не cached.

Действия

Действия - это доступные команды, которые можно выполнять над объектом класса.

JSON

```
{
    "id": "SAVE",
    "caption": "Сохранить",
    "visibilityCondition": null,
    "enableCondition": null,
    "needSelectedItem": false,
    "signBefore": false,
    "signAfter": false,
    "isBulk": false
}
```

Описание полей

Поле	Наименование	Допустимые значения	Описание
"id"	Код	“CREATE” - создать объект/объект ссылочного поля	Внутренний код действия с объектом.
		“EDIT” - изменить объект/привязанный объект ссылочного поля	
		“DELETE” - удалить объект	
		“CREATE-INLINE” - создать объект (не переходя на форму создания)	Создание объекта без открытия формы создания, для ускорения заведения объектов.
		“SAVEANDCLOSE” - сохранить изменения и закрыть фрейм	
		“SAVE” - сохранить изменения	
		“REMOVE” - удалить привязку к объекту по ссылке	
		“ADD” - добавить привязку к объекту по ссылке	
"caption"	Имя	Строка	Видимое наименование - подпись на кнопке действия (при наличии).
"visibilityCondition"	Условие видимости	Строка или null	Условие, при котором кнопка действия будет отображена.
"enableCondition"	Условие активности	Строка или null	Условие, при котором кнопка действия будет активна.
"needSelectedItem"	Условие видимости - наличие выбранного элемента	Логическое значение	Поле выставляется в true для действий, которым необходимо наличие выбранного элемента для активации.
"signBefore"	ЭП входящих данных	Логическое значение	
"signAfter"	ЭП исходящих данных	Логическое значение	
"isBulk"	Групповая	Логическое значение	Признак пакетной операции, для действий ссылочных полей. Выставляется в true для действий, которые выполняются со всеми привязанными объектами одновременно.

Логика действий над ссылочными объектами

Операция "Выбрать" - это установление связи между объектами, вне зависимости от типа связи должна быть возможность ее установить (и разорвать) на уровне бизнес логики.

1. Если это связь по ссылке один ко многим, т. е. ссылка на ключ - то приняв сторону “один” за A, сторону “многие” за B, реализуем:
   - на стороне объекта B (ссылка на A): Операция "Выбрать" (SELECT) - находим объект (A), устанавливаем значение ссылочного атрибута объекта B равным ключу объекта A (обнуляем атрибут для разрыва связи);
   - на стороне объекта A (коллекция или т.н. обратная ссылка): операция "Добавить" (ADD) - находим объект (B), устанавливаем значение ссылочного атрибута этого объекта (B) равное ключу объекта A, операция "Извлечь" (REMOVE) - выбираем в коллекции объект (B), обнуляем ссылку на A.
2. Если это связь по ссылке многие ко многим, т. е. ссылка на не ключевой атрибут, то для обоих концов коллекции реализуем связи:
   - Операция “Добавить” (ADD) - находим объект, устанавливаем значение его ссылки равным соответствующему атрибуту контейнера. При этом объект окажется в коллекциях всех контейнеров с таким же значением атрибута, это специфика данного типа связей)
   - Операция "Извлечь" (REMOVE) - выбираем объект в коллекции, обнуляем ссылку, при этом объект также удаляется из коллекций всех контейнеров с соответствующим значением атрибута.
3. Если это связь многие-ко-многим без ссылки (связь через системную промежуточную сущность, сюда относятся и “прямые” коллекции и “обратные”, как разные концы связи), то для обоих коллекций реализуем:
   - Операция "Добавить" (ADD) - находим объект, создаем сущность-связь с контейнером - объект появляется только в коллекции данного контейнера
   - Операция "Извлечь" (REMOVE) - выбираем объект в коллекции, удаляем сущность-связь с контейнером - объект пропадает только из коллекции данного контейнера

С точки зрения UI различий между типами связей вообще нет никаких - везде имеем дело с коллекциями, везде потенциально доступны операции добавления и извлечения. А настраивать наличие тех или иных кнопок у поля колекции можно и нужно на уровне модели представления. В бизнес логике должны быть реализованы стандартные обработчики для кнопок ADD и REMOVE в соответствии с описанной выше логикой.

Логика действий над объектами класса

Поле "commands", заданное в общей части меты представлений класса, задает список действий, допустимых над объектами данного класса.

В общей части меты представлений класса могут быть указаны команды следующих кодов "id":

1. "CREATE" - создать объект
2. "EDIT" - изменить объект
3. "DELETE" - удалить объект
4. "SAVEANDCLOSE" - сохранить изменения и закрыть
5. "SAVE" - сохранить изменения

Для представления атрибутов со свойством "type":2 применяются следующие действия:

1. "SELECT" - добавить
2. "EDIT" - править
3. "REMOVE" - удалить

Структура в mongoDB (registry)

```
{
    "id" : "SAVE",
    "caption" : "Сохранить",
    "visibilityCondition" : null,
    "enableCondition" : null,
    "needSelectedItem" : false,
    "signBefore" : false,
    "signAfter" : false,
    "isBulk" : false
}
```

Режим наложения

Поле Режим наложения - "overrideMode" позволяет задать два значения "Перекрыть" и "Переопределить".

Настройка режимов наложения в мете представления:

Тип 1 - "Переопределить" - переопределяет стандартный способ отображения соответствующих атрибутов заданных в мете представления класса. Атрибуты, которые не заданы в мете представления класса, отображаются стандартным образом на форме согласно заданной очередности по умолчанию.

Тип 2 - "Перекрыть" - выводятся только атрибуты заданные в мете представления.

Пример

```
"actions": null,
"siblingFixBy": null,
"siblingNavigateBy": null,
"historyDisplayMode": 0,
"collectionFilters": null,
"version": null,
"overrideMode": 1,
"commands": [
```

Выделение цветом строк в списке

Условия выделение цветом строк в списке задаются с помощью формулы.

Список поддерживаемых функций можно посмотреть :doc‘здесь </3_development/metadata_structure/meta_class/met
Настройка задается в общей части меты представления списка и выглядит следующим образом:

```
"styles": {
    "css-class": "формула"
  }
```

где, "css-class" - класс доступных тем оформления, а "формула" - формула с условием для применения выделения цветом строк в списке.

Доступные темы оформления "css-class"

- attention-1 - red
- attention-2 - yellow
- attention-3 - orange
- attention-4 - gray
- attention-5 - blue
- attention-6 - green

Назначение

При открытии формы представления списка объектов, в соответствии с условиями формулы и класса темы оформления - строки таблицы выделяются цветом.

Пример

```
"styles": {
    "attention-1": "lt($real, 0)"
  },
```

$real - любое целое число. Если $real - меньше 0, то столбец выделяется красным цветом.

Вкладки

Вкладки - задаются в представлении создания и редактирования и используются для разбиения атрибутов класса по отдельным вкладкам на форме.

Структура:

```
{
  "tabs": [
    {
      "caption": "1",
        "fullFields": [],
        "shortFields": []
    },
    {
      "caption": "2",
        "fullFields": [],
        "shortFields": []
    }
  ]
}
```

где, caption - наименование вкладки и fullFields - атрибуты на вкладке

Пример

```
{
  "tabs": [
    {
      "caption": "Первая вкладка",
      "fullFields": [
        {
          "caption": "Первый атрибут на первой вкладке",
          "type": 1,
          "property": "tab_1_1",
          "size": 2,
          "maskName": null,
          "mask": null,
          "mode": null,
          "fields": [],
          "hierarchyAttributes": null,
          "columns": [],
          "actions": null,
          "commands": [],
          "orderNumber": 10,
          "required": false,
          "visibility": null,
          "enablement": null,
          "obligation": null,
          "readonly": false,
          "selectionPaginated": true,
          "validators": null,
          "hint": null,
          "historyDisplayMode": 0,
          "tags": ["css:background-color:#AFFFAF"]
        },
        {
          "caption": "Второй атрибут на первой вкладке",
          "type": 1,
          "property": "tab_1_2",
          "size": 2,
          "maskName": null,
```

(continues on next page)

(continued from previous page)

```
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 20,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
      }
    ]
```

Представление Комментарий для атрибутов типа “Коллекция”

> warning Для корректной работы функционала обязательно в зависимостях проекта должен быть указан репозиторий viewlib.

Подключение:

1. В файле package.json проекта:

```
...
"ionMetaDependencies": {
   "viewlib": "0.7.1"
 }
...
```

2. В директорию application рядом с текущим проектом добавить репозиторий проекта viewlib

Инструкция по подключению функционала

Представление реализуется посредством шаблона вида templates/registry/item_footer.ejs. Обратите внимание на пояснение строк после знака // :

```
ejs
<%
let status = item.get('status'); // атрибут со статусом БП
let readOnly = state === 'conf' || status === 'approv'; //статус в котором отображаем коллекцию на
↪форме
if ((item.getMetaClass().checkAncestor('classColl@ns')) //класс, в котором содержится атрибут Комментарий
  && item.getItemId() && (status === 'onapprov' || readOnly)) { // статус, на котором атрибут
↪отображается только для чтения
  let comments = resolveTpl('comments', null, true);
```

(continues on next page)

(continued from previous page)

```
  if (comments) {
    let prop = item.property('atrClassColl'); //системное имя атрибута с представлением Комментарий
    let commId = `${form.ids.attr}_${prop.getName()}_com`;
%>

<div class="line-tabs tabs">

  <div id="item-footer-order-toggle" class="order-toggle asc" data-direction="asc"
       title="Изменить порядок в списке">
    <span class="glyphicon"></span>
  </div>

  <ul class="nav nav-tabs">
    <li class="active">
      <a href='#footer-tab-1' data-toggle="tab">
        <%- prop.getCaption() %>
      </a>
    </li>
  </ul>

  <div class="tab-content">
    <div id="footer-tab-1" class="tab-pane active">
      <div class="comments">
        <%-partial(comments, {
          item,
          id: commId,
          property: prop,
          comment: { // атрибуты для класса по ссылке из атрибута Коллекции
            text: 'descript',
            user: 'owner',
            parent: 'answlink',
            photo: 'owner_ref.foto.link'
          },
          count: 100,
          orderToggleId: '#item-footer-order-toggle',
          readOnly
        })%>
      </div>
    </div>
  </div>
</div>

<script>
  $(function () {
    $('#<%= commId %>').on('comment-added comment-deleted', function () {
      loadWorkflowState();
    });
  });
  $('#item-footer-order-toggle').click(function () {
    if ($(this).hasClass('asc')) {
      $(this).removeClass('asc').addClass('desc').data('direction', 'desc');
    } else {
      $(this).removeClass('desc').addClass('asc').data('direction', 'asc')
    }
  });
  $(document.body).on('mouseenter', '.item-comment', function () {
```

(continues on next page)

(continued from previous page)

```
    $(this).addClass('mouse-enter');
  });
  $(document.body).on('mouseleave', '.item-comment', function () {
    $(this).removeClass('mouse-enter');
  });
</script>
<% }} %>
```

Настройка "options"

Далее подключаем функционал "options" для представление Комментарий на форме представления изменения для атрибута типа “Коллекция”:

```
{
        "caption": "Коментарий",
        "type": 3,
        "property": "coment",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": 3,
        "fields": [],
        "columns": [
          {
            "sorted": true,
            "caption": "Дата",
            "type": 120,
            "property": "date",
            "size": 2,
            "maskName": null,
            "mask": null,
            "mode": null,
            "fields": [],
            "columns": [],
            "actions": null,
            "commands": null,
            "orderNumber": 2,
            "required": false,
            "visibility": null,
            "enablement": null,
            "obligation": null,
            "readonly": false,
            "selectionPaginated": true,
            "validators": null,
            "hint": "",
            "historyDisplayMode": 0,
            "tags": null,
            "selConditions": null,
            "selSorting": null
          },
          {
            "sorted": true,
            "caption": "Подтверждение (Обоснование)",
            "type": 7,
            "property": "descript",
```

(continues on next page)

(continued from previous page)


```
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": null,
        "orderNumber": 1,
        "required": true,
        "visibility": null,
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null,
        "selConditions": null,
        "selSorting": null
      },
      {
        "caption": "Ведущий",
        "type": 2,
        "property": "owner",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": 1,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": null,
        "orderNumber": 6,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
      }
    ],
    "actions": null,
    "commands": [
      {
        "id": "CREATE",
        "caption": "Создать",
        "visibilityCondition": null,
        "enableCondition": null,
```


(continues on next page)

(continued from previous page)

```
            "needSelectedItem": false,
            "signBefore": false,
            "signAfter": false,
            "isBulk": false
          },
          {
            "id": "EDIT",
            "caption": "Править",
            "visibilityCondition": null,
            "enableCondition": null,
            "needSelectedItem": true,
            "signBefore": false,
            "signAfter": false,
            "isBulk": false
          }
        ],
        "orderNumber": 80,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": "",
        "historyDisplayMode": 0,
        "tags": null,
        "options": {
          "template": "comments",
          "comments": {
            "textProperty": "descript", // атрибут "Описание" из класса по ссылке
            "userProperty": "owner", // атрибут "Ответственный" из класса по ссылке (отображается имя
↪пользователя, оставившего комментарий)
            "parentProperty": "answlink", // атрибут "Ответ" из класса по ссылке (для возможности "Ответить
↪" на комментарий пользователя)
            "photoProperty": "owner_ref.foto.link", // атрибут "Фото" из класса Персона (отображается фото
↪персоны)
            "dateProperty": "date" // атрибут "Дата" из класса по ссылке
          }
        }
      }
```

Особенности

Мета класса с атрибутом типа “Коллекция” с представлением Комментарий

1. В классе создается обычный атрибут с типом “Коллекция”.
2. В представлении формы изменения создается аналогично стандартному атрибуту с типом “Коллекция”, но с добавлением настройки "options", подробнее смотрите настройка “options”.
3. В классе создается атрибутивный состав и их системные наименования обязательно должны соответствовать наименованиям в шаблоне item_footer.ejs и в свойстве "options". Дополнительно к обязательным - класс может содержать любые атрибуты.

Мета дополнительных классов

Класс Персона должен содержать атрибут, в которых будет задаваться информация об имени пользователя (в данном случае это атрибут “user”) и фотография персоны (атрибут “Фото”), а так же ФИО персоны, которые являются семантикой данного класса.


```
{
    "namespace": "develop-and-test",
    "isStruct": false,
    "key": [
      "id"
    ],
    "semantic": "surname| |name| |patronymic",
    "name": "person",
    "version": "",
    "caption": "Персона",
    "ancestor": null,
    "container": null,
    "creationTracker": "",
    "changeTracker": "",
    "creatorTracker": "",
    "editorTracker": "",
    "history": 0,
    "journaling": true,
    "compositeIndexes": [],
    "properties": [
      {
        "orderNumber": 10,
        "name": "id",
        "caption": "Идентификатор",
        "type": 12,
        "size": 24,
        "decimals": 0,
        "allowedFileTypes": null,
        "maxFileCount": 0,
        "nullable": false,
        "readonly": true,
        "indexed": false,
        "unique": true,
        "autoassigned": true,
        "hint": null,
        "defaultValue": null,
        "refClass": "",
        "itemsClass": "",
        "backRef": "",
        "backColl": "",
        "binding": "",
        "semantic": null,
        "selConditions": [],
        "selSorting": [],
        "selectionProvider": null,
        "indexSearch": false,
        "eagerLoading": false,
        "formula": null
      },
      {
        "orderNumber": 20,
```


(continues on next page)

(continued from previous page)


```
    "name": "surname",
    "caption": "Фамилия",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": true,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "name",
    "caption": "Имя",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": true,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
```


(continues on next page)

(continued from previous page)


```
    {
      "orderNumber": 40,
      "name": "patronymic",
      "caption": "Отчество",
      "type": 0,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": true,
      "readonly": false,
      "indexed": true,
      "unique": false,
      "autoassigned": false,
      "hint": null,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    },
    {
      "orderNumber": 40,
      "name": "user",
      "caption": "Пользователь",
      "type": 18,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": true,
      "readonly": false,
      "indexed": true,
      "unique": false,
      "autoassigned": false,
      "hint": null,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
```


(continues on next page)

(continued from previous page)


```
      "formula": null
    },
    {
      "orderNumber": 70,
      "name": "foto",
      "caption": "Фотография",
      "type": 5,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": true,
      "readonly": false,
      "indexed": false,
      "unique": false,
      "autoassigned": false,
      "hint": null,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    }
  ],
  "metaVersion": "2.0.61.21119"
}
```


Ведение проектных документов

Ведение проектных документов - это настройка "fileshare-list" и "fileshare" предназначена для управления документами, как например, возможность скачать и/или получить ссылку на файл. Для настройки, в представлении атрибута типа “Файл” необходимо указать свойство:

```
"options": {
          "template": "fileshare-list"
        }
```

- fileshare-list - для типа multifile - множественные файлы
- fileshare - для типа file - один файл

В deploy.json в настройках registry подключаем кастомный файл-аплоадер (который с шарой директории и расширенными настройками):

```
"modules": {
  "registry": {
    "globals": {
      ...
      "di": {
        ...
        "fileshareController": {
          "module": "applications/viewlib/lib/controllers/api/fileshare",
          "initMethod": "init",
          "initLevel": 0,
          "options": {
            "module": "ion://module",
            "fileStorage": "ion://fileStorage"
          }
        }
```

Сохранение файлов в облаке

Путь для сохранения файла в облаке настраивается в deploy.json приложения. Для обращения к свойствам объекта используется знак $.

```
{item.названиеСвойства.свойствоСсылочногоОбъекта}
```

т.е. используем ${item.} для того чтобы обозначить что это обращение к объекту.

Пример:

```
"modules": {
    "registry": {
      "globals": {
        "storage": {
          "basicObj@project-management": {
            "cloudFile": "/${class}/pm_${attr}/${dddd}/"
          },
          "project@project-management": {
            "cloudFile": "/${item.name} (${item.code})/"
          },
          "eventControl@project-management": {
            "cloudFile": "/${class}/pm_${attr}/${dddd}/"
          },
          "eventOnly@project-management": {
            "cloudFile": "/${class}/pm_${attr}/${dddd}/"
          }
        }
      }
    }
  }
```

Настройка позволяет задавать любую структуру хранения файлов (линейно/иерархически, коллекцией/один файл)

Функционал шаринга на файлы

Подключение

В представлении необходимо задать свойство атрибута типа "Файл":

```
"tags": [
          "share"
        ]
```

Использование

При клике на значок “share” необходимо открывать окно управления шарой аналогичное как в овнклауд - в окне предусмотреть кнопки:

- применить - по апи облачного хранилища для файлика/каталога передаем все выбранные параметры:
- поделиться ссылкой - формируем шару на файлик/каталог - после применения возвращаем в поле для возможности копирования ссылки
- разрешить на редактирование
- защитить паролем- поле для ввода пароля
- установить срок действия - поле для даты
- перейти в хранилище - открываем в новой вкладке по ссылке на шару где находится файл/каталог - (к примеру файл к примеру каталог)
- закрыть - закрыть окно управления файлом

NB: Настройка шаринга доступна как для каждого файла, так и для всего каталога. Если на файле/каталоге уже есть шаринг, то при открытии управляющего окна отображаются настройки шаринга, при необходимости свойства можно изменить.

Переход по прямой ссылки до хранения файла

Возможности

- сразу скачать
- перейти на nextCloud и там увидеть/редактировать

Условия хранения ссылок

Условия хранения ссылок, созданных в процессе работы с файлами: TO DO возможность удалять все ссылки, созданные в процессе работы с файлом спустя какое-то время или же за ненадобностью.

Настройка доступа

Настройка пользователей и прав доступа к объектам хранилища Owncloud. В ряде случаев необходимо задавать пользователей и права для них на создаваемые объекты хранилища.

Настройка задается в файле deploy.json проекта.

Пример:

```
"ownCloud": {
  "module": "core/impl/resource/OwnCloudStorage",
  "options": {
   ...
     "users": [
       {
         "name": "user",
         "permissions": {
           "share": true,
           "create": false,
           "edit": true,
           "delete": false
         }
       }
     ]
  }
}
```

---

list_view

TODO

Описание

Мета представлений - позволяет задавать желаемый состав атрибутов этого класса для отображения на форме, в соответствии с видом формы представления (представление формы списка list.json, создания create.json, изменения класса item.json) и указывать для каждого отдельного атрибута свойства, переопределяемые и (или) дополняемые свойства, задаваемые в мете класса для данного атрибута.

Виды меты представлений

Мета представлений подразделяется на виды:

- Форма представления списка
- Форма представления создания и изменения

Форма представления списка

Форма представления списка - позволяет выводить объекты класса в виде списка.

JSON

```
{
  "columns": [...],
  "styles": {},
  "actions": null,
```

(continues on next page)

(continued from previous page)

```
  "commands": [...],
  "allowSearch": false,
  "pageSize": null,
  "useEditModels": true,
  "version": null,
  "overrideMode": null,
  "filterDepth": 3
}
```

Описание полей

Поле	Наименование	Допустимые значения	Описание
"columns"	Колонки	Массив объектов	Колонки или столбцы атрибутов класса, каждый из которых описывается атрибутивной частью меты представлений.
"styles"	Выделение цветом строк	Формула	В соответствии с условиями формулы столбцы таблицы окрашены в заданный цвет.
"actions"	Поведение	Целое или Null	не используется в текущей версии
"commands"	Действия	Массив объектов	Набор действий над объектами класса.
"allowSearch"	Доступен поиск	Логическое	Разрешает или запрещает отображение формы поиска.
"pageSize"	Количество записей на странице	Целое положительное	Указывает количество объектов на одной странице по умолчанию.
"useEditModels"	Использовать формы редактирования для детализации	Логическое	Разрешает или запрещает использование формы редактирования для детализации данных объекта класса.
"version"	Версия	Строка	Версия метаданных.
"overrideMode"	Режим наложения	0 - Перекрыть	Задает режим наложения представлений.
		1 - Переопределить	
"filterDepth"	Глубина запроса фильтра в списках	Целое положительное	Глубина для фильтра в списках объектов. По умолчанию 2.

Форма представления создания и изменения

Форма представления создания и изменения - позволяет создавать и изменять объекты класса.

JSON


```
{
  "tabs": [
    {
      "caption": "",
      "fullFields": [...],
      "shortFields": []
    }
  ],
  "actions": null,
  "commands": [...],
  "siblingFixBy": null,
  "siblingNavigateBy": null,
  "historyDisplayMode": 0,
  "collectionFilters": null,
  "version": null,
  "overrideMode": null
}
```

Описание полей

Поле	Наименование	Допустимые значения	Описание
"tabs"	Вкладки	Объект	Позволяет создавать несколько страниц объектов на одной форме представления.
"caption"	Имя вкладки	Строка	Поле объекта "tabs" наименование будет отображаться в строке перехода на вкладку.
"fullFields"	Поле в полном виде	Массив объектов	Поле объекта "tabs", массив содержит атрибуты которые должны отображаться в представлени с полным видом, описанные согласно атрибутивной части меты представлений.
"shortFields"	Поле в кратком виде	Массив объектов	Поле объекта "tabs", массив содержит атрибуты которые должны отображаться в представлени с кратким видом, описанные согласно атрибутивной части меты представлений.
"actions"	Поведение	Целое или Null	не используется в текущей версии
"commands"	Действия	Массив объектов	Набор действий над объектом класса.
"siblingFixedBy"	Отбор смежных объектов по	Массив строк	Перечисление атрибутов коллекции, по которым будет производится отбор смежных объектов.
"siblingNavigateBy"	Переход к смежным объектам по	Массив строк	Перечисление атрибутов коллекции, по которым будет осуществляться переход к смежным объектам.
"historyDisplayMode"	Отображение истории	Целое	Указать формат отображения истории изменения объектов.
"collectionFilters"	Фильтрация коллекций	Массив объектов	Выбор атрибутов из коллекций, по которым будет произведена фильтрация.
"version"	Версия	Строка	Версия метаданных.
"overrideMode"	Режим наложения	0 - Перекрыть	Задает режим наложения представлений.
		1 - Переопределить	

3.4.7 Мета представлений - атрибутивная часть

Действия

Действия - это доступные команды, которые можно выполнять над объектом класса.

JSON


```
{
    "id": "SAVE",
    "caption": "Сохранить",
    "visibilityCondition": null,
    "enableCondition": null,
    "needSelectedItem": false,
    "signBefore": false,
    "signAfter": false,
    "isBulk": false
  }
```

Описание полей

Поле	Наименование	Допустимые значения	Описание
"id"	Код	“CREATE” - создать объект/объект ссылочного поля	Внутренний код действия с объектом.
		“EDIT” - изменить объект/привязанный объект ссылочного поля	
		“DELETE” - удалить объект	
		“CREATE-INLINE” - создать объект (не переходя на форму создания)	Создание объекта без открытия формы создания, для ускорения заведения объектов.
		“SAVEANDCLOSE” - сохранить изменения и закрыть фрейм	
		“SAVE” - сохранить изменения	
		“REMOVE” - удалить привязку к объекту по ссылке	
		“ADD” - добавить привязку к объекту по ссылке	
"caption"	Имя	Строка	Видимое наименование - подпись на кнопке действия (при наличии).
"visibilityCondition"	Условие видимости	Строка или null	Условие, при котором кнопка действия будет отображена.
"enableCondition"	Условие активности	Строка или null	Условие, при котором кнопка действия будет активна.
"needSelectedItem"	Условие видимости - наличие выбранного элемента	Логическое значение	Поле выставляется в true для действий, которым необходимо наличие выбранного элемента для активации.
"signBefore"	ЭП входящих данных	Логическое значение	
"signAfter"	ЭП исходящих данных	Логическое значение	
"isBulk"	Групповая	Логическое значение	Признак пакетной операции, для действий ссылочных полей. Выставляется в true для действий, которые выполняются со всеми привязанными объектами одновременно.

Логика действий над ссылочными объектами

Операция "Выбрать" - это установление связи между объектами, вне зависимости от типа связи должна быть возможность ее установить (и разорвать) на уровне бизнес логики.

1. Если это связь по ссылке один ко многим, т. е. ссылка на ключ - то приняв сторону “один” за A, сторону “многие” за B, реализуем:
    - на стороне объекта B (ссылка на A): Операция "Выбрать" (SELECT) - находим объект (A), устанавливаем значение ссылочного атрибута объекта B равным ключу объекта A (обнуляем атрибут для разрыва связи);
    - на стороне объекта A (коллекция или т.н. обратная ссылка): операция "Добавить" (ADD) - находим объект (B), устанавливаем значение ссылочного атрибута этого объекта (B) равное ключу объекта A, операция "Извлечь" (REMOVE) - выбираем в коллекции объект (B), обнуляем ссылку на A.
2. Если это связь по ссылке многие ко многим, т. е. ссылка на не ключевой атрибут, то для обоих концов коллекции реализуем связи:
    - Операция “Добавить” (ADD) - находим объект, устанавливаем значение его ссылки равным соответствующему атрибуту контейнера. При этом объект окажется в коллекциях всех контейнеров с таким же значением атрибута, это специфика данного типа связей)
    - Операция "Извлечь" (REMOVE) - выбираем объект в коллекции, обнуляем ссылку, при этом объект также удаляется из коллекций всех контейнеров с соответствующим значением атрибута.
3. Если это связь многие-ко-многим без ссылки (связь через системную промежуточную сущность, сюда относятся и “прямые” коллекции и “обратные”, как разные концы связи), то для обоих коллекций реализуем:
    - Операция "Добавить" (ADD) - находим объект, создаем сущность-связь с контейнером - объект появляется только в коллекции данного контейнера
    - Операция "Извлечь" (REMOVE) - выбираем объект в коллекции, удаляем сущность-связь с контейнером - объект пропадает только из коллекции данного контейнера

С точки зрения UI различий между типами связей вообще нет никаких - везде имеем дело с коллекциями, везде потенциально доступны операции добавления и извлечения. А настраивать наличие тех или иных кнопок у поля колекции можно и нужно на уровне модели представления. В бизнес логике должны быть реализованы стандартные обработчики для кнопок ADD и REMOVE в соответствии с описанной выше логикой.

Логика действий над объектами класса

Поле "commands", заданное в общей части меты представлений класса, задает список действий, допустимых над объектами данного класса.

В общей части меты представлений класса могут быть указаны команды следующих кодов "id":

1. "CREATE" - создать объект
2. "EDIT" - изменить объект
3. "DELETE" - удалить объект
4. "SAVEANDCLOSE" - сохранить изменения и закрыть
5. "SAVE" - сохранить изменения

Для представления атрибутов со свойством "type":2 применяются следующие действия:

1. "SELECT" - добавить
2. "EDIT" - править
3. "REMOVE" - удалить

Структура в mongoDB (registry)

```
{
    "id" : "SAVE",
    "caption" : "Сохранить",
    "visibilityCondition" : null,
    "enableCondition" : null,
    "needSelectedItem" : false,
    "signBefore" : false,
    "signAfter" : false,
    "isBulk" : false
}
```

Условия активности

Описание

Условия активности - задают условия активности, то есть доступности поля для редактирования в представлении. Синтаксис условий такой же, как в условиях отображения.

Пример в JSON:

```
{
  {
        "caption": "Основание для условия активности",
        "type": 1,
        "property": "enablement_condition_base",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 20,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
```

(continues on next page)

(continued from previous page)


```
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
    },
    {
        "caption": "Поле активно, если основание заполнено",
        "type": 1,
        "property": "enablement_condition_use",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 30,
        "required": false,
        "visibility": null,
        "enablement": ".enablement_condition_base !\u003d \u0027\u0027",
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
    },
    {
        "caption": "Поле активно, если в основании \u00271\u0027",
        "type": 1,
        "property": "enablement_condition_1",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 40,
        "required": false,
        "visibility": null,
        "enablement": ".enablement_condition_base \u003d\u003d \u00271\u0027",
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
    }
}
```

Размеры полей

```
module.exports = {
  TINY: 0,
  SHORT: 1,
  MEDIUM: 2,
  LONG: 3,
  BIG: 4
};
```

Поля

Описание

Поля - содержат в себе атрибуты класса, объединенные, по какому-либо признаку, в группу (более подробное описание см. Тип “Группа [0]”).

Внимание: данное свойство применяется только для атрибута типа “Группа [0]”.

Пример

NB: На форме представления, заданные атрибуты, отображаются на нижнем уровне иерархии, на верхнем находится наименование группы.

```
{
  "tabs": [
    {
      "caption": "",
      "fullFields": [
        {
          "caption": "nameGroup",
          "type": 0,
          "property": "",
          "size": 2,
          "maskName": null,
          "mask": null,
          "mode": null,
          "fields": [
            {
              "caption": "atr1",
              "type": 1,
              ...
            },
            {
              "caption": "atr2",
              "type": 1,
              ...
            }
      ]
    }
      ],
      "shortFields": []
```

(continues on next page)

(continued from previous page)

```
    }
  ]
}
```

Маски ввода

Описание

Маски ввода - предоставляют возможность сообщить системе, какой шаблон или образец должны использовать данные. Применяются для облегчения обработки значений, имеющих фиксированный шаблон, – например, телефонных номеров. Маски ввода нужны для помощи пользователям с вводом предопределенного формата в атрибуте.

Поле "mask" для представлений

Пример

```
{
  "tabs": [
    {
      "caption": "Информационная система",
      "fullFields": [
        {
          "caption": "Уникальный идентификационный номер ОУ",
          "type": 1,
          "property": "OuId",
          "size": 2,
          "maskName": null,
          "mask": null,
```

В данный момент возможно задание маски в виде строки или объекта в поле "mask".

Маски по умолчанию

Маски по умолчанию можно переопределить в расширениях:

- 9 — число
- a — буква
- A — буква в верхнем регистре
- * – — число или буква

Виды масок

Статичные маски

Статичные маски определены заранее и не изменяется при вводе.

```
"mask": "aa-9999",
```

```
"mask": {"mask": "aa-9999"},
```

Необязательные маски

Необязательные маски допускают некоторую часть в определении маски сделать необязательной для ввода. В квадратных скобках [] задается необязательная часть ввода.

```
"mask": {"mask": "(99) 9999[9]-9999"},
```

Динамичные маски

Динамичные маски могут изменятся при вводе. В фигурных скобках {} задается динамичная часть ввода, применяется к выражению перед скобками:

- {n} - n повторений
- {n,m} - с n по m повторений
- {+} - от 1 и больше повторений
- {*} - от 0 и больше повторений

```
"mask": {"mask": "aa-9{1,4}"},
```

Генерируемые маски

Синтаксис генерируемых масок похож на выражение OR. Маска может быть одной из трех вариантов, указанных в генераторе. Для определения генератора используется |.

```
"mask": {"mask": "(aaa)|(999)"},
```

```
"mask": {"mask": "(aaa|999|9AA)"},
```

keepStatic

По умолчанию: null (~false). Используется в комбинации с синтаксисом генератора и оставляет маску статичной по мере ввода. Когда задается массив масок, keepStatic автоматически становится равен true, если явно не задан через параметры.

```
"mask": { "mask": ["+55-99-9999-9999", "+55-99-99999-9999" ], "keepStatic": true },
```

Результат: Вводим 1212345123 => должны получить +55-12-1234-5123 вводим еще 4 => переключаемся на +55-12-12345-1234

Дополнительные примеры масок:

“99 99 999999” — серия и номер паспорта

“[+]9 (999) 999-99-99” — номер мобильного телефона

“(999)|(aaa)” - позволяет ввести либо три цифры, либо три символа

Сложные маски

Задание сложной маски с определением валидации символов. Допустимые значения полей внутри определения определяются плагином inputmask версии 4.0.0. Задается массивом в field.mask:

```
field.mask: ["X{1,4}-AA №999999", {definitions: {"X": {validator: "[lLvVcCxX]", cardinality: 1, casing: "upper"}}
↪}]
```

Где индекс 0 - определение маски, а индекс 1 - дополнительные опции.

Реализованы также маски через “regex”.

```
"mask": {
   "regex": "[A-Za-z]{1,50}"
},
```

Маски по идентификатору

Для задания маски из вшитого пресета масок используется поле "maskName" атрибутивной части меты представлений. Не реализовано.

Условия обязательности

Описание

Условия обязательности - задают условие обязательности заполнения поля в представлении.

Синтаксис условий такой же, как в условиях отображения.

Условия обязательности отличаются выполнением действий. При данном условии атрибут становится обязательным для заполнения, иначе атрибут остается таким же, каким был задан в представлении до применения условия обязательности.

Пример в JSON:

```
{
    {
      "caption": "Основание для условия обязательности",
      "type": 1,
      "property": "obligation_condition_base",
      "size": 2,
      "maskName": null,
```

(continues on next page)

(continued from previous page)


```
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 20,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
    },
    {
        "caption": "Поле обязательно, если основание заполнено",
        "type": 1,
        "property": "obligation_condition_use",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 30,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": ".obligation_condition_base !\u003d \u0027\u0027",
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
    },
    {
        "caption": "Поле обязательно, если в основании \u00271\u0027",
        "type": 1,
        "property": "obligation_condition_1",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
```


(continues on next page)

(continued from previous page)


```
        "commands": [],
        "orderNumber": 40,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": ".obligation_condition_base \u003d\u003d \u00271\u0027",
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
 }
   }
```

Опции

Опции - предназначены для установки дополнительных параметров атрибута. Интерпретация настроек выполняется конкретной специфической реализацией модуля.

На данный момент в модуле registry поддерживаются следующие настройки:

```
"options": {
   "cssClasses": ["class1", "class2"],
   "cssStyles": {
      "background-color": "#FF0000"
    },
    "template": "some-custom-template"
}
```

- cssClasses - классы css, которые нужно применить к полю (в стандартном шаблоне)
- cssStyles - css стили, которые нужно применить к полю (в стандартном шаблоне)
- template - имя шаблона, который будет использован для отрисовки поля. Будет выполнен поиск ejs-шаблона с таким именем в стандартной теме, а в директориях, указанных в настройке templates модуля.

В кастомном шаблоне доступны все переменные, передаваемые в стандартные шаблоны атрибутов:

- item - объект, отображаемый формой
- prop - свойство объекта, представляемое полем (если есть)
- field- мета-объект поля
- id - идентификатор поля, сформированный по стандартному алгоритму

Сортировка - "reorderable": true

Применимо для атрибутов типа “Коллекция”. Определяет возможность сортировки элементов коллекции стрелками вверх и вниз, меняет порядковые номера между двумя элементами коллекции.

```
"options": {
  "reorderable": true`
}
```

Подключение шаблона библиотеки template

Подключение кастомного шаблона атрибута

Указываем значение пути к шаблону - располагаются в папке проекта .\templates\registry\. Пример для шаблона ниже ./templates/registry/attrs/project/stateView.ejs

```
"options": {
            "template": "attrs/project/stateView"
          }
```

Шаблону передается элемент item с основными командами ${item.getItemId()}:

- получения значения элемента item.property('stage').getValue(),
- идентификатора item.getItemId()
- имени класса item.getClassName() Элемент объекта prop
- имени (кода) свойства prop.getName()
- вычислении значения атрибута (если формула) prop.evaluate()
- значения атрибута prop.getValue()
- ссылки на класс prop.meta.refClass

Также передаются пути модуля ${module} для подключения шаблонов , например <% stylesheet(${module}/app-static/css/styles.css) -%>

Подключение для коллекции иерархии treegrid

```
"options": {
        "template": "treegrid/collection",
        "reorderable": true,
        "treegrid": {
          "width": "auto,100,100,100,100,0",
          "align": "left, center,center,center,center, left",
          "sort": "str, date, date, date, date, int",
          "enableAutoWidth": false,
          "paging": {
            "size": 20
          }
        }
      }
```

Представление в списке объектов для цветных иконок

По тому же принципу, что и в мете представлений создания/редактирования, можно задавать переопределяющие шаблоны для меты представлений списков для каждой колонки:

```
...
"options":{
  "template": "templateDir/name"
}
...
```

Пример

Подключение для цифровых полей слайдера/бегунка slider

```
"options": {
          "template": "slider",
          "slider": {
            "skin": "dhx_skyblue"
          }
        }
```

Подключение для целочисленных полей бегунка range

Задается в представлении для свойства "options":

```
...
"tags": null,
"options": {
  "template": "range"
}
...
```

Подключение для коллекций функционала создания объектов не заходя на форму inplace

```
"options": {
  "inplaceInsertion": true,
  "inplaceInsertionClass": "className@namespace"
}
```

Пример "options" атрибута “Таблица”

```
{
        "caption": "Таблица",
        "type": 3,
        "property": "table",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": 3,
        "fields": [],
        "columns": [],
        "actions": null,
        "commands": null,
```

(continues on next page)

(continued from previous page)

```
        "orderNumber": 50,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": "",
        "historyDisplayMode": 0,
        "tags": null,
        "options": {
          "inplaceInsertion": true
        },
        "selConditions": [],
        "selSorting": []
      }
```

"inplaceInsertionClass" указываем в том случае, если при создании объекта нужно выбирать класс (если есть наследники).

Настройка расположения заголовка атрибута над значением.

```
"options": {
  "cssClasses": ["top-label"]
}
```

Настройка поля атрибута на всю длину строки (без наименования).

```
"options": {
  "cssClasses": ["no-label"]
}
```

Настройка стилей применяемых к контейнеру, в котором содержится поле ввода с названием.

```
"options": {
  "cssStyles": {
   "max-width": "30%",
   "padding": "25px"
 }
}
```

Настройка параметров колонок таблицы для атрибута типа “Коллекция”

По умолчанию колонка даты имеет ширину 110 пикселей и выравнивание по центру.

Возможные опции атрибута:

```
"options": {
  "template": "treegrid/collection",
    "treegrid": {
      "width": "150,auto,200",
      "align": "center,left,center",
      "sort": "str, str, str",
      "enableAutoWidth": false,
      "paging": {
        "size": 20
  }
    }
}
```

Настройка CSS полей через tags и options

Можно настраивать CSS поля либо через tags, либо через options. В регистри есть соответствующие стандартные css-классы с нужным поведением: nolabel, toplabel, fill.

Для атрибута в мете представлений css-классы назначаются так:

В свойстве options:

```
"options": {
  "cssClasses": ["toplabel", "fill"]
}
```

В свойстве tags (обратная совместимость)

```
"tags": ["css-class:nolabel", "css-class:fill"]
```

Помимо классов можно напрямую задавать и стили (они будут применены только к контейнеру).

Задаем стили для атрибута в мете представлений:

В свойстве options:

```
"options": {
  "cssStyles": {
    "max-width": "30%",
    "padding": "25px"
  }
}
```

В свойстве tags:

```
"tags": ["css:min-width:10%", "css:background-color:green"]
```

Описание выше относится только к стандартным шаблонам полей из стандартной темы оформления.

CSS поля

CSS поля - задают стили для значений атрибутов и настраиваются посредством атрибута tags. Аналогичная настройка задается в "options" с использованием шаблонов. Подробнее см. “Опции”.

Синтаксис:

```
{
...
  tags: [
    "css-class:myCustomCssClass", // добавляем css-класс
    "css:background-color:green", // добавляем css-стиль
    "css:color:white" // добавляем css-стиль
  ]
}
```

Пример:

```
{
    "caption": "Первый атрибут на первой вкладке",
    "type": 1,
    "property": "tab_1_1",
    "size": 2,
    "maskName": null,
    "mask": null,
    "mode": null,
    "fields": [],
    "hierarchyAttributes": null,
    "columns": [],
    "actions": null,
    "commands": [],
    "orderNumber": 10,
    "required": false,
    "visibility": null,
    "enablement": null,
    "obligation": null,
    "readonly": false,
    "selectionPaginated": true,
    "validators": null,
    "hint": null,
    "historyDisplayMode": 0,
    "tags": ["css:background-color:##AFFFAF"]
  }
```

Настройка стартовой позиции на карте

Пример для атрибутов типа “Геоданные”

```
{
            "caption": "Координаты",
            "type": 100,
```

(continues on next page)

(continued from previous page)

```
            "property": "geo",
            "size": 2,
            "maskName": null,
            "mask": null,
            "mode": 0,
            "fields": [],
            "columns": [],
            "actions": null,
            "commands": null,
            "orderNumber": 34,
            "required": false,
            "visibility": null,
            "enablement": null,
            "obligation": null,
            "readonly": false,
            "selectionPaginated": true,
            "validators": null,
            "hint": "",
            "historyDisplayMode": 0,
            "tags": [
              "tryfind:Хабаровский край",
              "tryfind:$address"
            ],
            "selConditions": [],
            "selSorting": []
          }
```

Результат: при открытии формы создания координат - автоматически определятся координаты в соответствии со значением свойства "tags". Где $address - значение атрибута address из текущего класса.

Режим отображения “Коллекция” и “Ссылка”

Режимы отображения типа представления “Коллекция” и “Ссылка” - это следующие константы в платформе:

```
module.exports = {
  TEXT_SIMPLE: 0,
  TEXT_AUTOCOMPLETE: 1,

  COLLECTION_LIST: 0,
  COLLECTION_LINK: 1,
  COLLECTION_LINKS: 2,
  COLLECTION_TABLE: 3,
  COLLECTION_HASHTAGS: 4,

  REF_STRING: 0,
  REF_LINK: 1,
  REF_INFO: 2,
  REF_HIERARCHY: 3,
  REF_SPECIFY: 4,

  GEO_MAP: 0,
  GEO_LOCATOR: 1,
```

(continues on next page)

(continued from previous page)

```
  GEO_CANVAS: 2,

  GROUP_VERTICAL: 0,
  GROUP_HORIZONTAL: 1
};
```

Для атрибута с типом “Коллекция” на форме представления реализованы режимы отображения "mode":

- "mode": 0 - Список
- "mode": 1 - Ссылка
- "mode": 2 - Список ссылок
- "mode": 3 - Таблица
- "mode": 4 - Облако тегов

Для атрибута с типом “Ссылка” на форме представления реализованы режимы отображения "mode":

- "mode": 0 - Строка
- "mode": 1 - Ссылка
- "mode": 2 - Форма
- "mode": 3 - Иерархическая ссылка
- "mode": 4 - Уточняющий поиск

Подробнее “Иерархическая ссылка”

В режиме Иерархическое поле, на основании заданных вложенных полей (fields) отображаются параметры фильтра на уровни иерархии. При первичной инициализации поля, выполняется ajax-запрос к контроллеру на получение первого списка выбора (фильтр не задан). При получении ответа от сервера отображается первое поле фильтра со списком выбора. Далее при выборе значения в каждом из полей фильтра сбрасываются значения следующих за ним полей и запрашивается новый список выбора для следующего поля. Поля не имеющие списка выбора скрываются. Если в списке выбора получен один вариант, он автоматически присваивается в фильтр и выполняется определение списка выбора для следующего уровня иерархии. При получении специального значения “transit” вместо списка выбора в следующий фильтр присваивается значение текущего и выполняется процедура получения списка выбора для очередного уровня иерархии, а поле, соответствующее фильтру, скрывается. Когда заданы значения всех полей фильтрации, контроллер возвращает список выбора объекта по ссылке, который автоматически отображается в отдельном поле расположенном после полей фильтрации.

Подробнее “Уточняющий поиск”

Поля уточняющего поиска нужны для того, чтобы упростить поиск объектов в ссылке. Проектировщик меты, исходя из предметной области, может определить часть атрибутивного состава искомого объекта как “уточняющую” и, таким образом, облегчить задачу как для БД, так и для пользователя. Т.е. вместо выбора из всего множества продуктов мы сперва выбираем значение поля “производитель”, затем значение поля “тип продукта”, варианты которого уже ограничены предыдущим фильтром, и

т.д, таким образом существенно сокращаем выборку только теми продуктами, что соответствуют уточняющим атрибутам. Для такого поля становится возможным указать поля, которые будут ссылаться на атрибуты класса по ссылке и станут “уточняющими”.

Тип Группа [0]

Описание

Группа [0] - структура представлений создания и изменения, которая позволяет в рамках одного класса группировать аттрибуты из других классов в представлении создания/изменения в горизонтальном и/или вертикальном виде.

Виды отображения типа Группа [0]

- GROUP_VERTICAL "mode": 0 - поля группы располагаются друг под другом
- GROUP_HORIZONTAL "mode": 1 - поля группы располагаются горизонтально в строку (т. е. колонками, если хватает места)

Формат настройки в мете представления для типа “Группа”

```
{
  "type": 0, // группа полей
  "mode": 1, // отображается горизонтально
  "fields": [
     // поля
   ]
}
```

Настройка параметров размера колонок:

```
{
  "type": 0, // группа верхнего уровня
  "mode": 1, // колонки
  "fields": [
    {
      "type": 0, // группа-колонка 1
      "mode": 0,
      "size": 0, // очень узкая
      "fields": [
        {
          "property": "attr1",
          "type": 1,
          "caption": "Текстовое поле 1"
        },
        {
          "property": "attr2",
          "type": 1,
          "caption": "Текстовое поле 2"
```

(continues on next page)

(continued from previous page)

```
        }
      ]
    },
    {
      "type": 0, // группа-колонка 2
      "mode": 0,
      "size": 0,  // очень узкая
      "fields": [
        {
          "property": "attr3",
          "type": 1,
          "caption": "Текстовое поле 3"
        },
        {
          "property": "attr4",
          "type": 1,
          "caption": "Текстовое поле 4"
        }
      ]
    },
    {
      "type": 0, // группа-колонка 3
      "mode": 0,
      "size": 3,  // широкая
      "fields": [
        {
          "property": "attr5",
          "type": 1,
          "caption": "Текстовое поле 5"
        },
        {
          "property": "attr6",
          "type": 1,
          "caption": "Текстовое поле 6"
        }
      ]
    }
  ]
}
```

Типы представлений

Типы представлений - это следующие константы в платформе:

```
module.exports = {
  GROUP: 0,
  TEXT: 1,
  REFERENCE: 2,
  COLLECTION: 3,
  CHECKBOX: 4,
  COMBO: 5,
  DATE_PICKER: 120,
  DATETIME_PICKER: 6,
```

(continues on next page)

(continued from previous page)

```
  MULTILINE: 7,
  WYSIWYG: 8,
  RADIO: 9,
  MULTISELECT: 10,
  FILE: 11,
  PASSWORD: 12,
  IMAGE: 13,
  NUMBER_PICKER: 14,
  DECIMAL_EDITOR: 15,
  URL: 17,
  PERIOD_PICKER: 60,
  GEO: 100,
  ATTACHMENTS: 110,
  SCHEDULE: 210,
  CALENDAR: 220
};
```

NB: подробнее см. таблицу соответствий.

Код	Наименование	Описание
0	Группа	Особая структура представлений создания и изменения.
1	Строковое	Представление для текстовых данных или приведение к текстовому виду. Настроен trim - т.е. отбрасывание пробелов с начала и конца строки
2	Ссылка	Для ссылочных полей, связь 1кN. Позволяет задавать возможные операции над объектами класса, на который ссылаемся.
3	Коллекция	Для коллекций, связь Nк1. Позволяет задавать возможные операции над объектами класса, на который ссылаемся.
4	Флаг	Чекбокс для логического типа.
5	Выпадающий список	Для имеющих заданное поле selectionProvider.
7	Многострочный текст	Представление для текста. Настроен trim - т.е. отбрасывание пробелов с начала и конца строки
8	Форматированный текст	Редактор форматированного текста.
9	Альтернативный выбор	В атрибуте типа “Множество [15]” может быть один элемент множества. Не реализовано
10	Множественный выбор	В атрибуте типа “Множество [15]” может быть несколько элементов множества. Не реализовано
11	Выбор файла	Представление для выбора и загрузки файла.
12	Пароль	По идее должно обеспечивать скрытие вводимых данных, но не реализовано.
13	Выбор изображения	Представление для выбора и загрузки изображения, проверяет что загружено именно изображение, отображает превью.
14	Редактор целых чисел	Редактор для целых чисел, проверяет корректность ввода.
15	Редактор вещественных чисел	Редактор для вещественных чисел, проверяет корректность ввода, требует испольвоания . для отделения дробной части.
17	URL	Не реализовано
60	Выбор периода	Представление позволяющее вести две даты - границы периода.
100	Геоданные	Задает представление для типа “Геоданные [100]”.
110	Набор файлов	Представление для выбора и загрузки нескольких файлов. Контролирует что файлы принадлежат к одному из указанных в мете атрибута типов, общий размер файлов и количество.
210	Расписание	Представление для типа атрибута “Расписание [210]”, позволяет задать расписание, отображение в табличном виде.
220	Календарь	Представление для типа атрибута “Расписание [210]”, позволяет задать календарь, отображение в виде календаря.

Условия отображения

Описание

Условия отображения - задаёт условия отображения поля в представлении класса.

Настройка выполняется не только для логических полей, но и для строковых, строковых перечислимых (сравнивается значение по коду перечислимого).

Синтаксис

- Символ \u003d обозначает операцию =
- Символ \u0027 обозначает операцию '

```
"visibility": ".visibility_condition_base !\u003d \u0027\u0027"
```

Данные символы нужны для корректного отображения условий в формате .json

Типы условий

- Сложные условия:

```
".state == 'work' || .state == 'result' || .state == 'fin'"
```

где идет проверка по трём условиям с объединением ИЛИ (для условия И применяется знак &).

Сам синтаксис настройки похож на условия в js.

- Логический:

```
".archive == true"
```

где идет проверка по значению логического атрибута.

- Простое условие:

```
".state == 'work'"
```

где идет проверка по значению атрибута со списком выбора.

- Числовое условие:

```
".magistral == 1"
```

где идет проверка по числовому значению атрибута.

- Пусто:

```
".meeting == ''"
```

- Не пусто:

```
"!! .meeting"
```

где идет проверка - есть ли значение в указанном атрибуте.

Пример в JSON:

```
{
  {
        "caption": "Основание для условия отображения",
        "type": 1,
        "property": "visibility_condition_base",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 20,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
  },
  {
        "caption": "Поле отобразится, если основание заполнено",
        "type": 1,
        "property": "visiility_condition_use",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 30,
        "required": false,
        "visibility": ".visibility_condition_base !\u003d \u0027\u0027",
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
  },
  {
        "caption": "Поле отобразится, если в основании \u00271\u0027",
        "type": 1,
        "property": "visiility_condition_1",
        "size": 2,
        "maskName": null,
```


(continues on next page)

(continued from previous page)


```
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 40,
        "required": false,
        "visibility": ".visibility_condition_base \u003d\u003d \u00271\u0027",
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
  }
    }
```


Описание

Атрибутивная часть меты представлений - описывает представление атрибута класса на форме. Атрибуты содержатся в виде массивов в соответствующих полях основной части меты представлений. Представление каждого атрибута - это объект следующей структуры:

JSON


```
{
  "caption": "Редактор целых чисел [14]",
  "type": 14,
  "property": "integer_integer",
  "size": 2,
  "maskName": null,
  "mask": null,
  "mode": null,
  "fields": [],
  "hierarchyAttributes": null,
  "columns": [],
  "actions": null,
  "commands": [],
  "orderNumber": 20,
  "required": false,
  "visibility": null,
  "enablement": null,
  "obligation": null,
  "readonly": false,
  "selectionPaginated": true,
  "validators": null,
  "hint": "",
  "historyDisplayMode": 0,
```


(continues on next page)

(continued from previous page)

```
  "tags": null
}
```

Описание полей

Поле	Наименование	Допустимые значения	Описание
"sorted"	Разрешена сортировка	Логическое	Поле для представлений списка. Не используется в представлениях создания и изменения. Разрешает или запрещает сортировку по данному столбцу.
"caption"	Заголовок	Строка	Заголовок поля ввода/колонки атрибута отображаемый в представлениях.
"type"	Тип	Целое - идентификатор (код) типа	Тип представления атрибута.
"property"	Атрибут	Строка, только латиница без пробелов	Указывается имя атрибута, с которым будет работать система, а значит не может быть пустым (Кроме типа представления Группа [0]).
"size"	Размер	Целое - код размера поля ввода, типозависимо	Позволяет указать код размера поля для разных типов атрибута/представления. Константы в платформе FieldSizes.
"maskName"	Имя маски	Строка	При наличии в платформе предустановленных масок - можно задать маску по внутреннему имени, указав его в данном поле.
"mask"	Маска	Строка	Позволяет предопределять формат вводимого значения для атрибута.
"mode"	Режим отображения	Целое - код режима отображения	В ряде случаев бывает необходимо отображать данные атрибута по разному, как отображать - указывается в этом поле. Пример использования.
"fields"	Поля	Массив объектов	Позволяет сформировать представление создания/изменения особым образом.
"hierarchyAttributes"	Атрибуты иерархия по	объект или Null	не используется в текущей версии
"columns"	Столбцы	Массив объектов	Применяется для атрибутов типа “Коллекция [3]”. Позволяет выбрать атрибуты для отображения на форме представления в виде колонок таблицы (атрибуты берутся из класса по ссылке)
"actions"	Поведение	Целое или Null	не используется в текущей версии
"commands"	Действия	Массив объектов либо Null	Описывает допустимые действия (групповые действия) над объектами ссылочного поля. Null для дефолтного набора действий.
"orderNumber"	Порядковый номер	Целое неотрицательное	Порядковый номер атрибута задает расположение атрибута относительно других атрибутов этого же класса в пользовательском интерфейсе.
"required"	Обязательное	Логическое	Определяет обязательно ли заполнение данного атрибута при создании/изменении объекта.
"visibility"	Условия отображения	Строка	Задает условие отображения поля в представлении.
"enablement"	Условия активности	Строка	Задает условие активности (доступности для редактирования) поля в представлении с форматом, аналогичным условиям отображения.
"obligation"	Условия обязательности	Строка	Задает условие обязательности заполнения поля в представлении с форматом, аналогичным условиям отображения.
"readonly"	Только чтение	Логическое	Разрешает или запрещает изменять значение атрибута в данном представлении.
"selectionPaginated"	Постраничный выбор	Логическое	Разрешает или запрещает постраничный список выбора.

Дополнительно:

- Представление Комментарий для атрибутов типа “Коллекция” - подробнее
- Настройка "fileshare-list" и "fileshare" для управления документами - подробнее

Структура в mongoDB (registry) для представлений списка


```
{
    "sorted" : true,
    "caption" : "Редактор целых чисел [14]",
    "type" : 14,
    "property" : "integer_integer",
    "size" : 2,
    "maskName" : null,
    "mask" : null,
    "mode" : null,
    "fields" : [],
    "hierarchyAttributes" : null,
    "columns" : [],
    "actions" : null,
    "commands" : [],
    "orderNumber" : 20,
    "required" : false,
    "visibility" : null,
    "enablement" : null,
    "obligation" : null,
    "readonly" : false,
    "selectionPaginated" : true,
    "validators" : null,
    "hint" : "",
    "historyDisplayMode" : 0,
    "tags" : null
}
```


Структура атрибута для представлений создания и изменения отличается лишь отсутствием поля "sorted".

3.4.8 Мета бизнес-процесса

Безопасность бизнес-процесса

Описание

Безопасность в бизнес-процессе нужна для контроля прав над конкретным объектом одним пользователем и задается через мету класса, статусы и переходы бизнес-процесса.

Реализация

В мете заранее надо опеределить строковый атрибут, в котором будет храниться идентификатор пользователя.

Для контроля прав при переходах по БП надо в переeходе добавить присваивание текущего пользователя в атрибут, по которому будут выдаваться права на следующем статусе БП.

Затем задаем в статусе БП уровень доступа в свойстве itemPermissions:

```
"itemPermissions": [
      {
        "role": ...
        "permissions": ...
      }
    ]
```

- role - указывается атрибут, который хранит идентификатор пользователя
- permissions - задается число по битовой маске, которое соотносится с уровнем доступа role к объекту
  - 1 - чтение
  - 2 - запись
  - 4 - удаление
  - 8 - использование
  - 31 - полный доступ

Можно использовать права в любом порядке. Например:

- чтение + запись = 3
- чтение + запись + удаление = 7
- чтение + запись + удаление + использование = 15

Внимание! В бизнес-процессе динамические права могут только предоставить больше доступа. Уменьшить доступ нельзя.

Примеры

Присваивание в переходе БП атрибуту person текущего пользователя, работающим с объектом

```
"assignments": [
   {
      "key": "person",
      "value": "$$uid"
   }
]
```

Добавление itemPermissions в статус БП

```
"states": [
   {
     "itemPermissions": [
       {
         "role": "person",
         "permissions": 15
       }
     ]
   }
 ]
```

Статусы бизнес-процесса

JSON

```
"states": [
   {
     "name": "new",
     "caption": "Новое",
     "maxPeriod": null,
     "conditions": [],
     "itemPermissions": [],
     "propertyPermissions": [],
     "selectionProviders": []
   }
 ]
```

Описание полей

Поле	Описание
"name"	Системное имя статуса
"caption"	Логическое имя статуса
"maxPeriod"	нет данных
"conditions"	Условия для статуса БП. Задаются аналогично “Условиям отбора допустимых значений”.
"itemPermissions"	Разрешения для объекта
"propertyPermissions"	Разрешения для свойств
"selectionProviders"	Выборка допустимых значений

Переходы бизнес-процесса

JSON

```
"transitions": [
   {
     "name": "basic",
     "caption": "На согласование",
     "startState": "create",
     "finishState": "inAgreed",
     "signBefore": false,
     "signAfter": false,
     "roles": [],
     "assignments": [],
     "conditions": []
   }
 ]
```

Описание полей

Поле	Описание
"name"	Системное имя статуса.
"caption"	Логическое имя статуса.
"startState"	Начальный статус для осуществления перехода по БП.
"finishState"	Конечный статус по завершению перехода по БП.
"signBefore"	Логическое значение “Подписать до начала перехода”.
"signAfter"	Логическое значение “Подписать по завершению перехода”.
"roles"	Список ролей, с правами на осуществление перехода.
"assignments"	Присвоение значения атрибутам после осуществления перехода по БП.
"conditions"	Условия, выполняемые для осуществления перехода по БП. Задаются аналогично “Условиям отбора допустимых значений”.

Присвоение значения атрибутам по ссылке

Задается через свойство "assignments" в переходе БП.

Пример

```
...
    "assignments": [
      {
        "key": "resolution.stateRemPet",
        "value": "end"
      }
    ]
...
```

По ссылке атрибута типа “Ссылка” [resolution], для атрибута [stateRemPet] присвоить значение “end” - при выполнении данного перехода по БП.

Описание

Бизнес-процесс - это четкая последовательность действий, которую выполняют для получения заданного результата. Как правило, процесс многократно повторяется. Применение бизнес-процесса позволяет отображать стадии выполняемого процесса и задавать условия для его выполнения.

JSON

```
{
  "name": "basic",
  "caption": "Поручения",
  "wfClass": "basic",
  "startState": "new",
  "states": [],
  "transitions": [],
  "metaVersion": "2.0.61"
}
```

Описание полей

Поле	Наименование	Описание
"name"	Системное имя	Системное имя бизнес-процесса
"caption"	Логическое имя	логическое имя бизнес-процесса
"wfClass"	Класс БП	Класс, к которому применяется бизнес-процесс
"startState"	Статус	Статус, присвоенный началу бизнес-процесса
"states"	Список статусов	Список статусов бизнес-процесса.
"transitions"	Переходы	Переходы между статусами для бизнес-процесса.
"metaVersion"	Версионирование	Версия метаданных.

Условие “contains” (“содержит”)

Внимание! Чтобы в БП работало поле “contains”, в коллекции должна быть настроена жадная загрузка. Иначе в коллекции будет пусто, и результат будет всегда false. Условия применяются к объекту извлеченному из БД, дополнительных запросов не делается.

Пример

```
{
   "property": "charge",
   "operation": 10,
   "value": null,
   "nestedConditions": [
     {
       "property": "state",
       "operation": 1,
       "value": [
         "close"
       ],
       "nestedConditions": []
     }
   ]
 }
```

Настройка подсказок

Настройка подсказок при переходе по статусу БП - представляет собой вывод инструкции в отдельном модальном окне с кнопками - “продолжить” или “отменить”. При наведении на кнопку появляется всплывающий хинт, для более удобного использования бизнес-процессов.

Пример

```
"transitions": [
   {
     ...
     "confirm": true,
     "confirmMessage": null
   }
```

- "confirm" - подтверждение действия на переходе (+ стандартный вывод текста - вы действительно хотите выполнить действие “name”).
- "confirmMessage" - уникальный текст для вывода в подтверждении взамен стандартного.

Утилита формирования массива объектов

Утилита позволяет создавать объект в коллекцию при переходе основного объекта в заданный статус. Поля созданного объекта автоматически заполняются в соответствии с настройками, заданными для свойства "values".

Теперь чтобы прикрепить утилиту создания значений показателя к этапу БП, надо в di в свойстве options прописать опцию

state - имя этапа БП

при переходе на который, должны создаваться объекты в коллекцию.

Есть возможность использовать утилиту как “action”. При переделке нужно просто убрать команду из модели представления. Параметры задаются в файле deploy.json проекта. Синтаксис настройки:

```
"map": {
   "workflow@namespace.stage": {
     "className@namespace": { // для объекта какого класса создается объект в коллекцию
        "collection": { // наименование атрибута коллекции, в которой создается объект
           "elementClass": "className2@namespace", // класс, объекты которого создаются утилитой
           "patterns": [
             {
                "values": {
                   "attr1": "string", // строка
                   "attr2": 123, // число
                   "attr3": true,
                   "attr4": "$containerProperty1", // свойство контейнера
                   "attr5": {"add": ["$containerProperty2", 300]} // формула
                },
                "push": [
                  "workflow2@namespace.stage1", // присвоение БП созданных объектов статуса
                ]
             },
             ...
           ]
        },
        ...
     },
     ...
   },
   ....
}
```

Безопасность бизнес-процесса

Безопасность в бизнес-процессе нужна для контроля прав над конкретным объектом одним пользователем и задается через мету класса, статусы и переходы бизнес-процесса. Подробнее о безопасности БП.

3.4.9 Таблица соответствия типов атрибутов типам представлений

Код типа атрибута	Наименование типа атрибута	Код основного типа представления	Наименование основного типа представления	Предпочитаемый тип отображения для основного типа представления	Допустимые типы представлений (выборочно)
0	Строка	1	Текстовое	•	7 - Многострочный текст, 8 - Форматированный текст
1	Текст	7	Многострочный текст	•	1 - Текстовое, 8 - Форматированный текст
2	HTML	8	Форматированный текст	•	1 - Текстовое, 7 - Многострочный текст
3	URL	17	URL	•	•
4	Изображение	13	Выбор изображения	•	•
5	Файл	11	Выбор файла	•	•
6	Целое	14	Редактор целых чисел	•	•
7	Действительное	15	Редактор вещественных чисел	•	•
8	Десятичное	15	Редактор вещественных чисел	•	•
9	Дата/Время	120	Выбор даты	•	6 - Выбор даты-времени
10	Логический	4	Флаг	•	•
11	Пароль	12	Пароль	•	•
12	Глобальный идентификатор	1	Текстовое	•	•
13	Ссылка	2	Ссылка	1 - Ссылка	•
14	Коллекция	3	Коллекция	3 - Таблица	•
15	Множество	9	Альтернативный выбор	•	10 - Множественный выбор
16	Структура	0	Группа	•	•
17	Пользовательский	•	не определе-	•	•

3.4.10 Переменные

Значение атрибутов

$ - применяется вместе с системным наименованием атрибута и возвращает значение указанного атрибута, т.е. если указать $name при формировании какого-либо условия для действий над значениями атрибутов, то значение атрибута name и будет источником для заданного условия.

Пример применения для вычислямого атрибута

```
{
  "formula": {
        "count": [
            "$projects"
        ]
    }
  ...
}
```

В текущем классе есть атрибут типа “Коллекция” projects и, согласно формуле, необходимо посчитать колличество значений данного атрибута.

Результат: количество объектов атрибута projects.

Текущая дата/время, дата

$$now - возвращает текущую дату и время

Пример применения для фильтра допустимых значений в навигации:

```
{
    "property": "dateEnd",
    "operation": 5,
    "value": [
      "$$now"
    ],
    "nestedConditions": []
}
```

"dateEnd" меньше текущей даты/времени

Пример применения для вывода значения по умолчанию в мете класса:

```
...
"defaultValue": "$$now",
...
```

$$today - возвращает начало суток текущей даты принцип тот же, только дата без времени

Синтаксис форматирования дат в формате momentjs

```
'DD.MM.YYYY'
```

NB: В драйвере к монгодб поддерживаются только основные возможности формата momentjs

Текущий пользователь

$$uid - возвращает текущего пользователя

Пример применения для фильтра допустимых значений в коллекции:

```
{
  "property": "collectionAttr",
  "operation": 10,
  "nestedConditions": [
    {
      "property": "user",
      "operation": 0,
      "value": ["$$uid"]
    }
  ]
}
```

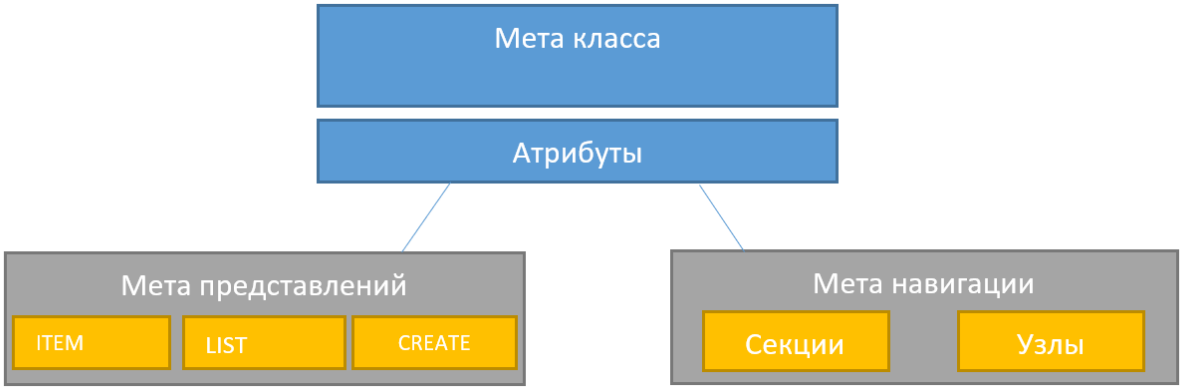
По ссылке атрибута типа “коллекция” отображаются только те объекты, у которых значение атрибута “user” совпадает со значением текущего пользователя.

Метаданные (Мета) - совокупность JSON-файлов в полной мере описывающих комплект структур, которыми оперирует приложение, способов отображения данных структур в пользовательском интерфейсе и навигации по ним, а так же файлов конфигурации приложения.

3.4.11 Типы файлов меты

1. Мета классов
2. Мета представлений
3. Мета навигации: мета секций навигации, мета узлов навигации.
4. Мета отчета
5. Мета бизнес-процессов
6. Мета безопасности
7. Мета admin
8. Геомета

3.4.12 Структура основных типов меты



Структуру основных типов меты можно представить следующим образом:

Мета классов является основным источником формирования данных в приложении. Мета классов состоит из атрибутов (атрибутивная часть) и параметров самого класса (общая часть). Атрибуты - это объекты массива “properties” общей части, которая содержит поля, имеющие отношение к самой структуре и способам оперирования данными в структуре.

На основе меты классов задается мета представлений, мета навигации, мета отчетов, мета бизнес-процессов и т.д.

Мета представления (класса) позволяет задавать желаемый состав атрибутов этого класса для отображения на форме, в соответствии с типом формы представления (представление формы списка list.json, создания create.json, изменения класса item.json) и указывать для каждого отдельного атрибута свойства, переопределяемые и (или) дополняемые свойства, задаваемые в мете класса для данного атрибута.

Мета представления + Атрибуты класса = Отображение атрибутов на форме

Мета навигации регулирует расположение элементов в навигационном блоке. Мета навигации разделяется на мету узлов навигации и мету секции навигации.

3.4.13 Наименование файлов меты:

Мета класса	Мета представлений	Мета навигации
Находится в директории meta и состоит из наименования общей части меты класса + .class.json.. Например: adress.class.json.	В наименовании директории определяется к какому классу относится представление. Мета представлений располагается в директории views, в которой содержатся директории, наименования которых совпадают с первой частью наименований файлов меты классов. Например: address@project_name, где address относится к классу address.	Мета секций навигации: состоит из поля "name" + .section.json и находится в директории navigation. Например: workflow.section.json.

3.5 Конфигурация платформы

3.5.1 Настройки авторизации и безопасности

Параметры конфигурации приложения, файл deploy.json

Параметры конфигурации приложения предназначены для определения ключевых возможностей системы при работе приложения на этапе проектирования и изменения параметров по умолчанию.

Настройка параметров авторизации при работе с паролем

Параметры и требования работы с паролем задаются в di в конфигурации компонента auth модуля. Но в основном настройки задаются глобально.

```
{
  "globals": {
    "parametrised": true,
    "plugins":{
      "auth": {
        "module": "lib/auth",
        "initMethod": "init",
        "initLevel": 2,
        "options": {
```

(continues on next page)

(continued from previous page)

```
        "app": "ion://application",
        "logger": "ion://sysLog",
        "dataSource": "ion://Db",
        "acl": "ion://aclProvider",
        "passwordLifetime": "[[auth.passwordLifeTime]]", // Максимальный срок действия пароля
        "passwordMinPeriod": "[[auth.passwordMinPeriod]]", // Минимальный срок действия пароля
        "passwordMinLength": "[[auth.passwordMinLength]]", // Минимальная длина пароля
        "passwordComplexity": { // Требования к сложности пароля
          "upperLower": true, // Обязательно верхний и нижний регистр
          "number": true, // Обязрательно использование хотя бы одного числа
          "special": true // Обязательно использование хотя бы одного специального символа
        },
        "passwordJournalSize": "[[auth.passwordJournalSize]]", // Вести журнал паролей размере паролей
        "tempBlockInterval": "[[auth.tempBlockInterval]]", // Время до сброса счетчика блокировки
        "attemptLimit": "[[auth.attemptLimit]]", // Пороговое значение количества попыток для блокировки
        "tempBlockPeriod": "[[auth.tempBlockPeriod]]" // Продолжительность блокировки учетной записи
      }
    }
```

При этом значения обозначенные [[auth.passwordLifeTime]] могут быть переконфигурированы в файле настроек приложения /config/setup.ini. Но для этого обязательно нужно проверить, что задана настройка “parametrised”: true, на уровне global.

Время жизни задается в формает [длительность][ед. изм], при этом единицы измерения:

- y - год
- d - день
- h - час
- m - минута
- s - секунда

По умолчанию значения ключевых параметров:

- passwordLifetime = 100y
- passwordMinPeriod = 0d
- passwordMinLength = 8

Все создаваемые пароли в системе, в том числе импортированные, автоматически проставляются как требуемые к смене. Чтобы при импорте пароли не требовалось менять, в свойствах пользователя в импортируемом файле acl должен быть указан параметр needPwdReset: false

Настройка минимальной длины пароля для входа в систему

Для указания минимальной длины пароля для входа в систему используем свойство "passwordMinLength"

```
"plugins":{
   "accounts": {
      "options": {
        "passwordMinLength": 8
      }
   }
}
```

Настройка прав доступа “aclProvider”

"plugins":{

```
"aclProvider": {
   "module": "core/impl/access/aclMetaMap",
   "initMethod": "init",
   "initLevel": 1,
   "options":{
     "dataRepo": "lazy://dataRepo",
     "acl": "lazy://actualAclProvider",
     "accessManager": "lazy://roleAccessManager"
   }
}
```

Параметры настроек фреймворка и приложения в файле config/setup.ini

Настройки предназначены для уточнения и изменения параметров приложения и инициализируются при запуске. Настройки имеют более высокий приоритет, чем параметры конфигурации.

Настройки приложения могут быть также заданы в переменных окружения; при этом переменные окружения имеют более высокий приоритет перед настройками.

Переопределение параметров конфигурации паролей

Параметры работы с паролями, заданные в deploy.json проекта, если включена параметризация и указан код параметр, можно переопределить через настройки платформы или через переменные окружения.

Пример файла настроек /config/setup.ini, в котором переопределяются значения, указанные в примере файла deploy.json.

```
# Максимальный срок действия пароля
auth.passwordLifeTime=90d
# Минимальный срок действия пароля
auth.passwordMinPeriod=75d
# Минимальная длина пароля
auth.passwordMinLength=8
# Вести журнал паролей размере паролей
auth.passwordJournalSize=5
# Время до сброса счетчика блокировки
auth.tempBlockInterval=30m
# Пороговое значение блокировки
auth.attemptLimit=6
# Продолжительность блокировки учетной записи
auth.tempBlockPeriod=30m
# Время жизни авторизованной сессии, при отсутствии активности
auth.sessionLifeTime=4h
```

Настройка параметров сессии в системе

Длина сессии задается в config/config.json в sessionHandler, с применением плейсхолдеров для параметра cookie.maxAge:

```
"sessionHandler": {
  "module": "lib/session",
  "initMethod": "init",
  "initLevel": 1,
  "options": {
    "app": "ion://application",
    "dataSource": "ion://Db",
    "session": {
      "secret": "ion:demo:secret",
      "resave": false,
      "saveUninitialized": true,
      "cookie": {
        "httpOnly": true,
        "secure": false,
        "maxAge": "[[auth.sessionLifeTime]]"
      }
    }
  }
}
```

Добавляем настройку в deploy.ini-файл проекта. Формат задания аналогичен настройкам периодов в auth:

```
auth.tempBlockPeriod=2s
auth.tempBlockInterval=15m
auth.blockPeriod=1d
auth.sessionLifeTime=2h
```

Также можно задавать просто числом, тогда это будет задание в миллисекундах.

Для хранения сессии не в базе данных, а в сервере кеширования redis, добавляем настройку и параметры кеширования в deploy.ini-файл проекта

```
session.type=redis
cache.redis.host=127.0.0.1
cache.redis.port=6379
```

Настройка отключения формы авторизации для перехода на страницу модуля

В конфиге ядра у поля “auth” есть настройка exclude:

```
"auth": {
  "module": "lib/auth",
  "initMethod": "init",
  "initLevel": 2,
  "options": {
    "app": "ion://application",
    "logger": "ion://sysLog",
    "dataSource": "ion://Db",
    "denyTopLevel": "[[auth.denyTop]]",
    "authCallbacks": ["[[auth.callback]]"],
    "publicRegistration": "[[auth.registration]]",
    "exclude": ["[[auth.exclude1]]", "[[auth.exclude2]]", "[[auth.exclude3]]"]
  }
}
```

То есть прописываем в ini-файле проекта:

```
auth.exclude[] = /registry/ # исключаем только запросы к корню модуля
auth.exclude[] = /registry/** # исключаем запросы ко всем страницам модуля
auth.exclude[] = \/registry\/khv-svyaz-info@naselennniePunkty\/\w+ # исключаем запросы ко всем страницам
↪модуля
внутри ноды khv-svyaz-info@naselennniePunkty
auth.exclude[] = /registry/api/naselenniyPunkt@khv-svyaz-info/** # исключаем запросы к api класса
```

При переходе на страницу указанного в настройке модуля - данные отображаются без необходимости авторизации.

Отключение авторизации для статичных путей на примере проекта develop-and-test:

```
; Исключение статичных путей ядра из проверки доступа безопасности
auth.exclude[]=/
auth.exclude[]=/vendor/**
auth.exclude[]=/css/**
auth.exclude[]=/fonts/**
auth.exclude[]=/favicon.ico

; Исключение статичных путей модулей из проверки доступа безопасности
auth.exclude[]=/registry/vendor/**
auth.exclude[]=/registry/css/**
auth.exclude[]=/registry/js/**
auth.exclude[]=/registry/app-vendor/**
auth.exclude[]=/registry/app-static/**
auth.exclude[]=/registry/common-static/**
auth.exclude[]=/registry/img/**
auth.exclude[]=/registry/fonts/**
auth.exclude[]=/dashboard/vendor/**
auth.exclude[]=/dashboard/develop-and-test/** ; для проекта develop-and-test
auth.exclude[]=/dashboard/js/**
auth.exclude[]=/registry/viewlib-ext-static/** ; для проекта viewlib-extra
auth.exclude[]=/registry/viewlib-static/js/** ; для проекта viewlib
auth.exclude[]=/gantt-chart/vendor/**
auth.exclude[]=/gantt-chart/gantt/**
auth.exclude[]=/gantt-chart/css/**
auth.exclude[]=/gantt-chart/js/**
auth.exclude[]=/gantt-chart/common-static/**
auth.exclude[]=/gantt-chart/fonts/**
auth.exclude[]=/geomap/vendor/**
auth.exclude[]=/geomap/css/**
auth.exclude[]=/geomap/js/**
auth.exclude[]=/geomap/common-static/**
auth.exclude[]=/geomap/img/**
auth.exclude[]=/geomap/fonts/**
auth.exclude[]=/report/vendor/**
auth.exclude[]=/report/css/**
auth.exclude[]=/report/js/**
auth.exclude[]=/report/common-static/**
auth.exclude[]=/report/img/**
auth.exclude[]=/report/fonts/**

; Исключение всего модуля из проверки доступа безопасности
auth.exclude[]=/portal/**
```

3.5.2 Способы конфигурации параметров:

- через ini-файлы
- через переменную окружения

NB: Приоритетной является настройка, заданная через переменные окружения, а не через ini-файлы. При этом настройки /config/setup.ini не влияют на настройки деплой приложения - они используются только для ядра и модулей. Deploy.json чаще всего параметризируеются через ini в дирректории приложения, например файл deploy.ini.

Пример конфигурации параметров ownCloud, вместе с учетной записью

Для начала необходимо задать в deploy.json параметрические настройки хранилища:

```
"ownCloud":
  "ownCloud": {
          "module": "core/impl/resource/OwnCloudStorage",
          "options": {
            "url": "[[ownCloud.url]]",
            "login": "[[ownCloud.login]]",
            "password": "[[ownCloud.pwd]]"
          }
        }
```

Конфигурация параметров deploy через ini-файлы:

В ini-файле deploy.ini рядом с deploy.json задать параметры следующего вида:

```
ownCloud.url=https://owncloud.com/
ownCloud.login=api
ownCloud.pwd=api
```

Конфигурация параметров deploy через переменную окружения:

В переменных окружения для ноды при конфигурации приложения задать параметры следующего вида:

```
ownCloud.url=https://owncloud.com/
ownCloud.login=api
ownCloud.pwd=api
```

Настройка ограничения

Настройка ограничения переключения по пунктам системного меню для анонимного пользователя. Системное меню формируется с учетом контроля доступа к страницам модулей, т.е. нет прав на страницу модуля - не отображается пункт меню, для перехода на данный модуль, в системном меню. В ini-файле приложения необходимо выставить auth.checkUrlAccess=true, чтобы задать настройку ограничения.

Изменение всех ссылок на относительные

Чтобы изменить все ссылки на относительные, в ini-файле проекта укажите:

```
app.baseUrl= '/нужный_путь/'
```

Если путь не указан то считается по умолчанию ‘/’.

Настройка в модуле админа блока управления запуском заданий по расписанию

Для отображения пункта меню необходимо добавить в ini-файл проекта настройку:

```
jobs.enabled=true
```

Она включит шедулер (англ. scheduler) в процессе веб-сервера, что даст возможность управлять джобами из модуля админа.

Шедулер — управляет таймерами запуска задач.

Джоб — конкретная задача, запускаемая по таймеру.

Настройка кеширования данных на уровне ядра

Настройка кеширования данных на уровне ядра - позволяет корректно восстановливать из кеша жадно-загружаемые ссылочные атрибуты и коллекции, а также файлы и вычисляемые атрибуты. Корректно кешируются списки. Внедрено кеширование в геомодуле. Настройка раз и навсегда решает проблему циклических ссылок при сериализации объектов.

В ini-файле прописываем:

```
cache.module=memcached
```

Настройка временных ограничений

connectTimeOut - максимальное время установления соединения.

operTimeOut - максимальное время выполнения операции.

```
db.connectTimeOut=
db.operTimeOut=
```

Настройка минимальной длины пароля

```
auth.passwordMinLength=8
```

Переопределить настройку для отдельного приложения можно в файле deploy.json

3.5.3 Конфигурационный файл - deploy.json

Сборка deploy из отдельных файлов

Разделение файла конфигурации deploy.json

Для удобства организации и повышения читаемости конфигурации приложения в deploy.json есть возможность разделить этот файл на несколько отдельных файлов конфигурации.

Варианты разделения могут быть любыми - можно вынести из него только конфигурацию модулей приложения или описывать каждый объект deploy в отдельном файле.

Для конфигурации в папке deploy приложения нужно поместить произвольные файлы с расширениями .json или .yml.

Для корректной работы в них необходимо сохранить изначальную структуру deploy.json - объекты должны быть вложены друг в друга в том же порядке, что в исходном файле. Исключение - модули приложения.

Например, если в deploy.json был определен объект globals.jobs.ticketClose.di, то для вынесения объекта di в отдельный файл в этом файле должна быть воспроизведена структура этих вложенных объектов:

```
---
globals:
  jobs:
    ticketClose:
      di:
        ticketCloser:
          executable: applications/khv-ticket-discount/lib/overnightTicketClose
          options:
            dataRepo: ion://dataRepo
            log: ion://sysLog
            workflows: ion://workflows
```

Файлы конфигурации модулей

В случае конфигов модулей, их можно также

1. Разместить в директориях deploy/modules/<имя модуля>/
2. Вынести конфиг для модуля целиком в файл deploy/modules/<имя модуля>.yml (.json)

Описание при этом начинается с корня модуля (а не приложения):

```
---
globals:
  navigation:
    namespaces:
      khv-ticket-discount: Льготные билеты
  eagerLoading:
    "*":
      applicant@khv-ticket-discount:
        item:
          - documents.vidDocument
      flight@khv-ticket-discount:
        list:
          - route.pointDeparture
```

(continues on next page)

(continued from previous page)

```
        - route.pointArrival
  listSearchOptions:
    ticketYear@khv-ticket-discount:
      "*":
        searchBy:
          - flight
          - person
          - numberTicket
          - area
        refDepth: 3
     flight@khv-ticket-discount:
      "*":
        searchBy:
          - route
          - number
        refDepth: 3
  di:
    pmListToDocx:
      module: modules/registry/export/listToDocx
       initMethod: init
      initLevel: 0
      options:
        tplDir: applications/khv-ticket-discount/export/list
        log: ion://sysLog
        injectors:
          - ion://monthTicketStatsInjector
...
```

Глобальные настройки в deploy.json

Глобальные настройки конфигурации приложения включает в себя следующие разделы:

- Пространство имен приложния "namespace"
- Параметризация "parametrised"
- Путь к шаблонам "theme"
- Время кеширования и другие параметры для статичных файлов https://expressjs.com/en/4x/api.html#express.static "staticOptions" (работает при NODE_ENV=production)
- Наименование вкладки в браузере"pageTitle"
- Модули приложения "moduleTitles" - перейти
- Настройка отображения общего системного меню для всех модулей проекта "explicitTopMenu" - перейти
- Переопределение настроек хранилища сессий "sessionHandler"
- "actualAclProvider"
- "aclProvider"
- Настройка HTML атрибутов для отображения и сохранения картинок в атрибуте "fileStorage" - перейти

- Настройки отображения имени пользователя и аватара во всех модулях проекта "customProfile" - перейти
- Настройка глубины жадной загрузки "dataRepo" - перейти
- "accounts"
- "securedDataRepo"
- "ldap"
- "auth"
- "sendmail"
- "nodemailer"
- "emailNotifier"
- "notifier"
- "eventNotifier"
- Настройки интеграции с календарем "icsMailer" - перейти
- "recache"
- "fact-creator"
- "report-builder"
- "projectReportCreator"

Структура которых строится следующим образом:

```
{
  "namespace": "...",
  "parametrised": true,
  "globals": {
    "theme": "...",
    "staticOptions": {...},
    "pageTitle": "...",
    "moduleTitles": {...},
    "explicitTopMenu": [...],
    "plugins": {
      "sessionHandler": {...},
      "actualAclProvider": {...},
      "aclProvider": {...},
      "fileStorage": {...},
      "customProfile": {...},
      "dataRepo": {...},
      "accounts": {...},
      "securedDataRepo": {...},
      "ldap": {...},
      "auth": {...},
      "sendmail": {...},
      "nodemailer": {...},
      "emailNotifier": {...},
      "notifier": {...},
      "eventNotifier": {...},
      "icsMailer": {...}
    },
    "jobs": {
      "recache": {...},
```

(continues on next page)

(continued from previous page)

```
    "fact-creator": {...},
    "report-builder": {...},
    "projectReportCreator": {...}
  }
 }
}
```

Модули приложения

Для свойства необходимо задать модули, которые будут использованы в приложении в поле “moduleTitles”. Также эти же модули будут отображаться в системном меню.

```
{
 "namespace": "crm",
 "globals": {
  "moduleTitles": {
   "registry": "Тех. поддержка",
   "report": "Отчеты"
  }
 }
}
```

Настройка скрытия модуля в системном меню

Для скрытия модуля из системного меню проекта присваиваем этому модулю, в файле deploy.json, значение null, например "ionadmin": null.

```
{
 "namespace": "project-management",
 "parametrised": true,
 "globals": {
  "moduleTitles": {
   "registry": {
    "description": "Проектное управление",
    "order": 10,
    "skipModules": true
   },
   "ionadmin": null
  }
 }
}
```

Настройка отображения общего системного меню для всех модулей проекта

Для того, чтобы в системном меню отображался одинаковый набор пунктов, не зависимо от того, на странице какого модуля находишься - необходимо в deploy.json файле проекта задать "explicitTopMenu" на глобальном уровне, с сохранением возможности переопределять "explicitTopMenu" в registry.

Пример

```
"globals": {
   "explicitTopMenu": [
    {
      "id":"mytasks",
      "url": "/registry/project-management@indicatorValue.all",
      "caption":"Мои задачи"
    },
    {
      "id":"projectmanagement",
      "url": "/registry/project-management@project",
      "caption":"Проектное управление"
    },
    {
      "type": "system",
      "name": "gantt-chart"
    },
    {
      "type": "system",
      "name": "portal"
    },
    {
      "type": "system",
      "name": "geomap"
    },
    {
      "type": "system",
      "name": "report"
    },
    {
      "id":"distionary",
      "url": "/registry/project-management@classification.okogu",
      "caption":"Справочники"
    },
    {
      "id":"mark",
      "url": "/registry/project-management@person",
      "caption":"Прогресс-индикатор"
    }
  ],
```

Описание полей

- "id" - идентификатор секции навигации
- "url" - url секции навигации
- "caption" - наименование секции навигации
- "name" - системное наименование модуля

Поле “plugins”

В данном поле задаются настройки, которые позволяют дополнительно расширить возможности приложения.

Настройка HTML атрибутов для отображения и сохранения картинок в атрибуте

"plugins":{

```
"fileStorage": {
   "module": "core/impl/resource/OwnCloudStorage",
   "options": {
     "url": "https://owncloud.iondv.ru/",
     "login": "api",
     "password": "apiapi"
   }
}
```

```
"htmlFiles": {
   "module": "core/impl/resource/FsStorage",
   "initMethod":"init",
   "initLevel": 3,
   "options": {
     "storageBase": "./htmlFiles",
     "urlBase": "/htmlFiles",
     "dataSource": "ion://Db",
     "log": "ion://sysLog",
     "app": "ion://application",
     "auth": "ion://auth"
   },
   "htmlImages": {
      "module": "core/impl/resource/ImageStorage",
      "initMethod": "init",
      "initLevel": 3,
      "options": {
        "fileStorage": "ion://htmlFiles",
        "app": "ion://application",
        "auth": "ion://auth",
        "log": "ion://sysLog",
        "urlBase": "/htmlFiles",
        "thumbnails": {
          "small": {
            "width": 100,
            "height": 100
          }
        }
      }
   }
}
```

"modules": { "registry": { "globals":

```
{
   "refShortViewDelay": 1000, // количество миллисекунд до появления окна с инфо. Если не указан или 0,␣
↪или нет shortView представления, то окно не выводится
   "defaultImageDir": "images",
   "contentImageStorage": "htmlImages"
}
```

Настройки отображения имени пользователя и аватара во всех модулях проекта

Для задания аватара через деплой прописываем связь с изображением. Аватар будет браться из соответствующего атрибута класса, объект которого привязан к текущему системному пользователю.

Пример

```
"customProfile": {
  "module": "lib/plugins/customProfile",
  "initMethod": "inject",
  "options": {
    "auth": "ion://auth",
    "metaRepo": "ion://metaRepo",
    "dataRepo": "ion://dataRepo",
    "propertyMap": {
      "person@project-management": {
        "filter": "user",
        "properties": {
          "avatar": "foto"
        }
      }
    }
  }
}
```

Настройка глубины жадной загрузки

```
"dataRepo": {
  "options": {
    "maxEagerDepth": 4
  }
}
```

Настройки интеграции с календарем

Интеграция осуществляется следующим образом: модуль по событию отправляет письмо с прикрепленным ics-файлом, в котором указано событие iCalendar. Outlook воспринимает такое письмо как приглашение на собрание. Яндекс тоже добавляет собрание в календарь.

Конфигурации модуля:

```
"icsMailer": {
  "module": "applications/extensions/lib/icsMailer",
  "initMethod": "init",
  "initLevel": 2,
  "options": {
    "dataRepo": "ion://dataRepo",
    "transport": {...}, //Настройки smtp-сервера
    "defaults": {...}, //Общий настройки всех писем
    "listeners": [
      {
        "component": //Ссылка на слушаемый компонент
```

(continues on next page)

(continued from previous page)

```
      "events": {
        "...": {// Идентификатор события
          "calendar": {...}, //Настройки календаря, несущего событие и передаваемого в ics-вложении
          "event": {...}, //Настройки VEVENT, передаваемого в ics-вложении
          "filename": "...", //Имя вложенного ics-файла
          "letter": {...} //Настройки письма, отправляемого по событию.
        }
      }
    }
  ]
 }
}
```

- Подробности настройки transport и defaults.
- Подробности настройки letter
- Подробности настройки calendar
- Подробности настройки event

Для настроек letter, event, filename и calendar предусмотрена возможность использовать данные из объекта события, указывая имена свойств через точку refAttr.stringAttr, либо обернув эту конструкцию в ${refAttr.stringAttr} когда необходимо использовать шаблон.

Полный пример файла deploy.json

Настройки модулей в deploy.json

- Модуль “registry”
- Модуль “geomap”
- Модуль “gantt-chart”
- Модуль “report”
- Модуль “rest”
- Модуль “portal”
- Модуль “ionadmin”
- Модуль “dashboard”
- Модуль “diagram”

Модуль “registry”

Модуль регистра (registry) – ключевой модуль предназначенный непосредственно для работы с данными на основе структур метаданных – в том числе по ведению проектов, программ, мероприятий и др. Подробнее о модуле регистра.

Настройка конфигурируемого сохранения файлов

Для того, что бы задать путь сохранения файла в хранилище - указываем:

```
"modules": {
   "registry": {
     "globals": {
...
       "storage": {
          "className@ns":{
              "file_attr":"/${class}/example_${attr}/${dddd}/"
          }
        },
...
```

В объекте ключом является название класса, дальше “название атрибута” : “относительный путь”.

Алиасы записываются в ${алиас} . Доступные алиасы:

- class - имя класс
- attr - имя атрибута
- также доступны обозначения дат из moment.js

Настройка для указания количества символов для поискового запроса

Для всего приложения - "listSearchMinLength".

```
"modules": {
  "registry": {
    "globals": {
      "listSearchMinLength": 1
    }
  }
}
```

Для отдельного класса "minLength".

```
"modules": {
  "registry": {
    "globals": {
      "listSearchOptions": {
        "className@ns": {
          "*": {
            "searchBy": [
              "atr1"
            ],
            "splitBy": "\\s+",
            "mode": [
              "starts"
            ],
            "joinBy": "and",
            "minLength": 3
          }
        }
    }
```

(continues on next page)

(continued from previous page)

```
  }
}
```

Настройка присвоения контейнера при создании вложенного объекта

Для случаев, когда необходимо присваивать значение для атрибут по ссылке, не при сохранении объекта, а при создании, указываем в deploy.json приложения настройку для класса, который содержит присваемое значение:

```
"registry": {
  "globals": {
    "forceMaster": {
      "name_class@ns": true
    }
  }
}
```

Пример использования генераторов последовательностей - сейчас для каждого объекта его код - это код его непосредственного контейнера плюс очередное значение счетчика последовательности привязанного к объекту-контейнеру.

Настройка жадной загрузки для печатных форм "skipEnvOptions"

Подробнее о печатных формах.

С помощью флага skipEnvOptions можно настроить/отключить жадную загрузку.

Пример

```
...
"modules": {
   "registry": {
    "globals": {
...
      "di": {
...
         "export": {
          "options": {
           "configs": {
            "class@ns": {
              "expertItemToDocx": {
                "type": "item",
                "caption": "Наименование",
                "mimeType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document",
                "extension": "docx",
                "skipEnvOptions": true,
                "preprocessor": "ion://expertItemToDocx"
              }
            }
           }
          }
```

(continues on next page)

(continued from previous page)

```
        }
...
      }
    }
  }
}
...
```

При жадной загрузке файл создается быстро, но это не всегда может быть приемлемо.

Настройка уведомления о редактировании объекта другим пользователем

В настройке уведомления о редактировании объекта другим пользователем указывается время жизни для блокировки в милисекундах:

```
"modules": {
  "registry": {
    "globals": {
      "concurencyCheck": 10000
    }
  }
}
```

Компонент ConcurencyChecker:

Компонент ConcurencyChecker в датасорсе хранит состояние блокировки для объектов. Хранит следующие параметры:

- полный id объекта (класс@id),
- датавремя блокировки (blockDate),
- заблокировавший пользователь.

Компонент создает состояния блокировки, при этом запускается таймер, по которому запись о блокировке удаляется по истечении таймаута. Если на момент срабатывания таймера запись оказывается еще актуальной (обновляли blockDate), то запись не удаляется, а таймер обновляется.

Логика в контроллере view:

Читаем из сетингов настройку registry.concurencyCheck (таймаут блокировки в секундах).

Если она больше 0, обращаемся к ConcurencyCheker - проверяем состояние блокировки.

Если не найдено (либо просрочена - blockDate < now() - registry.concurencyCheck), то через чекер записываем новую блокировку от имени текущего пользователя. Если найдена живая блокировка - передаем в шаблон информацию о блокировке, которую отображаем на форме и отображаем форму в режиме для чтения (globalReadOnly).

Дополнительный контроллер concurencyState, который принимает id объекта и проверяет его состояние блокировки. Если объект не заблокирован (нет блокировки, либо она просрочена), то блокирует объект от имени текущего пользователя. Если объект заблокирован текущим пользователем, обновляет blockDate на new Date(). Возвращает состояние блокировки.

Поведение формы объекта:

Если в шаблон передана инфа о блокировке, то добавляется скрипт, который периодически (с периодом registry.concurencyCheck/2) обращается к контроллеру concurencyState.

Если в ответ приходит информация о блокировке другим пользователем - она отображается (обновляем сообщение), если произошел перехват блокировки текущим пользователем - форма перезагружается (она при этом отображается в режиме для редактирования).

Подключение ресурсов в проекте для оформления

Это имеет отношение, например к группам в специальном стиле - чтобы не подключать ресурсы через изменения шаблонов модуля - необходимо их подключить в приложении.

```
"statics": {
  "geoicons": "applications/khv-svyaz-info/icons"
}
```

Все, что внутри директории icons доступно по ссылке registry/geoicons.

Настройка формы указания параметров экспорта (для печатных форм)

Пример с параметрами в params:

```
...
      "di": {
        "pmListToDocx": {
          "module": "modules/registry/export/listToDocx",
          "initMethod": "init",
          "initLevel": 0,
          "options": {
            "tplDir": "applications/project-management/export/item2",
            "log": "ion://sysLog"
          }
        }
...
        "export": {
          "options": {
            "configs": {
              "evaluationPerform@project-management": {
                "rating": {
                  "caption": "Оценка деятельности исполнителя и соисполнителей проекта",
                  "mimeType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document",
                  "extension": "docx",
                  "type": "list",
                  "query": {
                    "filter": {
                      "and": [
                        {
                          "eq": [
                            "$basicObjPerform",
                            ":project"
                          ]
                        },
                        {
                          "gte": [
                            "$date",
                            ":since"
                          ]
                        },
```

(continues on next page)

(continued from previous page)

```
                    {
                      "lte": [
                        "$date",
                        ":till"
                      ]
                    }
                  ]
                }
              },
              "params": {
                "project": {
                  "caption": "Проект",
                  "type": "reference",
                  "className": "project@project-management"
                },
                "since": {
                  "caption": "Период с",
                  "type": "date",
                  "default": "$monthStart"
                },
                "till": {
                  "caption": "Период по",
                  "type": "date",
                  "default": "$monthEnd"
                }
              },
              "eagerLoading": [
                "ownOrg",
                "basicObjs"
              ],
              "preprocessor": "ion://pmListToDocx"
            }
          }
...
```

Настройка параметров поиска в списке объектов

Функционал позволяет на уровне класса определять, как ищем объекты класса из представления списка: по вхождению слов или полные слова, по отдельным атрибутам или по указанным атрибутам в списке с параметрами поиска через пробел.

Формат и доступные операции:

```
"listSearchOptions": {
   "person@khv-childzem": {...} // для класса
     "khv-childzem@person": {...} // только в узле навигации person
    "*": {...} // везде по умолчанию
}
```

вместо ... подставляем атрибуты и задаем операции для поиска, например:

```
 "searchBy": [ // атрибуты по которым ищем, по умолчанию то, что выводится в колонках
  "surname",
```

(continues on next page)

(continued from previous page)

```
  "name",
  "patronymic"
],
"splitBy": "\\s+", // разбивать поисковую фразу регуляркой, части сопоставить с атрибутами
"mode": ["starts", "starts", "starts"], // режимы сопоставления - в данном случае "начинается с" (доступны
↪like, contains, starts, ends)
"joinBy": "and" // режим объединения условий на атрибуты (по умолчанию or)
```

Настройка иерархического представления для коллекций

Иерархическое представление коллекций- отображает коллекции, в которых элементы связаны друг с другом в виде иерархического справочника. В библиотеке viewlib реализован кастомный контроллер, возвращающий в формате TreeGrid очередной уровень иерархии.

Пример

```
"treegridController": {
        "module": "applications/viewlib/lib/controllers/api/treegrid",
        "initMethod": "init",
        "initLevel": 0,
        "options": {
          "module": "ion://module",
          "logger": "ion://sysLog",
          "securedDataRepo": "ion://securedDataRepo",
          "metaRepo": "ion://metaRepo",
          "auth": "ion://auth",
          "config": {
            "*": {
              "project@project-management":{
                "roots":[{
                  "property": "name",
                  "operation": 1,
                  "value": [null],
                  "nestedConditions": []
                }],
                "childs":["stakeholders", "basicObjs"]
              },
              "governmentPower@project-management": {
                "roots":[],
                "childs":null,
                "override": {
                  "descript": "url"
                }
              },
              "object@project-management": {
                "roots":[],
                "childs":null
              },
              "event@project-management": {
                "roots":[],
                "childs":null
              }
            }
```

(continues on next page)

(continued from previous page)

```
            }
          }
        }
...
```

Поле config - в нем все настройки:

- первый ключ это навигационная нода (в данном примере “*” значит распространяется на все ноды),
- потом идут классы, у классов roots - это какие объекты этого класса вытаскивать в качестве корневых (используются “conditions”),
- childs - это атрибуты класса из которых доставать иерархию.

Настройка текстового поиска в глубину по ссылочным атрибутам

searchByRefs - это массив настроек, для обозначения иерархии классов. Можно сопоставлять с несколькими классами.

Пример

```
"family@khv-childzem": {
          "*": {
            "searchByRefs":[
              {
                "class": "person@khv-childzem",
                "idProperties": ["famChilds", "famParentMale", "famParentFemale"],
                "searchBy": [
                  "surname",
                  "name",
                  "patronymic"
                ],
                "splitBy": "\\s+",
                "mode": [
                  "starts",
                  "starts",
                  "starts"
                ],
                "joinBy": "and"
              }
            ]
          }
        }
```

Модуль “geomap”

Геомодуль (geomap) – предназначен для визуализации сведений и данных, имеющих географическую привязку.

Настройка иконки приложения

Логотип для модуля описывается через стандартный механизм статичных маршрутов:

```
{
  "modules": {
    "geomap": {
      "statics":[{"path":"applications/khv-svyaz-info/icons", "name":"icons"}],
      "logo": "icons/logo.png"
    }
  }
}
```

Настройка скрытия шапки и бокового меню

Пример:

```
"geomap": {
   "globals": {
      "hidePageHead": true, //отобразить шапку
      "hidePageSidebar": false, //скрыть боковое меню
      ...
   }
 }
```

Модуль “gantt-chart”

Модуль диаграм ганта (gantt-chart) – модуль, предназначенный для вывода специфичных типов иерархических данных имеющих даты. Подробнее о модуле диаграм ганта.

Настройка шкалы времени

Шкала времени настраивается посредством настройки “Шаг” в модуле Гаанта. В преконфигурации “Шаг” задается через параметр step:

```
{
  "unit": "year",
  "step": 5
}
```

Пример

```
...
   "gantt-chart": {
      "globals": {
        "config": {
...
          "preConfigurations": {
...
```

(continues on next page)

(continued from previous page)

```
          "config3": {
            "caption": "Третья конфигурация",
            "showPlan": true,
            "units": "year",
            "step": 5,
            "days_mode": "full",
            "hours_mode": "full",
            "columnDisplay": {
              "text": true,
              "owner": true,
              "priority": true
            },
            "filters": {
              "priority": "Обычный"
            }
          }
        }
...
      }
    }
  }
```

Модуль “report”

Модуль отчетов (report) – модуль предназначенный для формирования, на основе специальных метаданных – аналитических отчетов и справочной информации. Расчеты могут выполняться по расписанию или быть инициированы оператором. Подробнее о модуле отчетов.

```
"report": {
    "globals": {
      "namespaces": {
        "project-management": "Проектное управление"
      },
      "defaultNav": {
        "namespace": "project-management",
        "mine": "projects",
        "report": "roadmap"
      },
      "mineBuilders": {
        "project-management": {
          "test": {
            "projects": "mineBuilder"
          },
          "projects": {
            "indicatorAll": "mineBuilder"
          }
        }
      },
      "di": {},
      "statics": {
        "common-static": "applications/project-management/templates/static"
      },
      "logo": "common-static/logo.png"
    },
```

(continues on next page)

(continued from previous page)

```
    "import": {
      "src": "applications/project-management/bi",
      "namespace": "project-management"
    }
  }
```

Модуль “rest”

Модуль REST-сервисов – модуль предназначенный для осуществления интеграции по формату REST. Используется для предоставления данных для модуля портала. Подробнее о модуле REST.

```
"rest": {
    "globals": {
      "di": {}
    }
  },
```

Модуль “portal”

Модуль портала (portal) – модуль, предназначенный для отображения произвольных шаблонов данных. Подробнее о модуле портала

```
"portal": {
    "import": {
      "src": "applications/project-management/portal",
      "namespace": "project-management"
    },
    "globals": {
      "portalName": "pm",
      "needAuth": true,
      "default": "index",
      "theme": "project-management/portal",
      "templates": [
        "applications/project-management/themes/portal/templates"
      ],
      "statics": {
        "pm": "applications/project-management/themes/portal/static"
      },
      "pageTemplates": {
        "navigation": {
          "index": "pages/index"
        }
      }
    }
  }
```

Модуль “ionadmin”

Модуль администрирования (ionadmin) – используется для назначения прав, управления задачами по расписанию и другими административными задачами. Подробнее о модуле администрирования.

Скрытие ролей в админе от назначения пользователю

Для ролей, которые должны быть скрыты в админе от назначения пользователю, в деплое приложения прописываем фильтры на основе регулярных выражений, по которым такие роли и будут определятся.

```
"ionadmin": {
    "globals": {
      "securityParams": {
        "hiddenRoles": [
          "^somePrefix_"
        ]
      }
    }
  }
```

Настройка скрытия модуля в системном меню

Для скрытия модуля из системного меню проекта присваиваем этому модулю, в файле deploy.json, значение null, например "ionadmin": null.

```
{
  "namespace": "project-management",
  "parametrised": true, //
  "globals": {
    "moduleTitles": {
      "registry": {
        "description": "Проектное управление",
        "order": 10,
        "skipModules": true
      }
      "ionadmin": null
    }
```

Модуль “dashboard”

Модуль дашбоарда (dashboard) – модуль предназначенный для выведения краткой информации в виде блоков. Подробнее о модуле дашбоарда.

Для того что бы данные из меты загружались в модуль “dashboard”, необходимо в файле конфигурации приложения deploy.json добавить следующую секцию, в раздел "modules":

```
"dashboard": {
  "globals": {
    "namespaces": {
      "project-management": "Проектное управление"
    },
    "root": {
      "project-management": "applications/project-management/dashboard"
    }
  }
}
```

Модуль “diagram”

```
"diagram": {
    "globals": {
     "config": {
      "org1": {
       "caption": "Организационная структура",
       "edit": true,
       "showSections": false,
       "relations": {
        "className": "organization@project-management",
        "title": "name",
        "text": "address",
        "img": "",
        "filter": [
         {
          "property": "headOrg",
          "operation": 0,
          "value": [
           null
          ],
          "nestedConditions": []
         }
        ],
        "children": [
         {
          "className": "branchOrg@project-management",
          "property": "branch",
          "title": "name",
          "text": "address",
          "children": [
           {
            "className": "branchOrg@project-management",
            "property": "branch",
            "children": []
           }
          ]
         }
        ]
       }
      }
     }
    }
  }
```

Полный пример файла deploy.json

Файл deploy.json на примере приложения “Project management system”

```
{
  "namespace": "project-management",
  "parametrised": true, //
  "globals": {
    "moduleTitles": {
```

(continues on next page)

(continued from previous page)

```
    "registry": {
      "description": "Проектное управление",
      "order": 10,
      "skipModules": true
    }
  },
  "explicitTopMenu": [
    {
      "id": "mytasks",
      "url": "/registry/project-management@indicatorValue.all",
      "caption": "Мои задачи"
    },
    {
      "type": "system",
      "name": "report"
    }
  ],
  "plugins": {
    "sessionHandler": {
      "options": {
        "storage": {
          "type": "[[session.type]]",
          "options": {
            "host": "[[cache.redis.host]]",
            "port": "[[cache.redis.port]]"
          }
        }
      }
    },
    "wfEvents": {
      "module": "applications/project-management/lib/wfEvents",
      "initMethod": "init",
      "initLevel": 1,
      "options": {
        "workflows": "ion://workflows",
        "metaRepo": "ion://metaRepo",
        "dataRepo": "ion://dataRepo",
        "log": "ion://sysLog"
      }
    },
    "actualAclProvider": {
      "module": "core/impl/access/aclmongo",
      "initMethod": "init",
      "initLevel": 1,
      "options": {
        "dataSource": "ion://Db"
      }
    },
    "aclProvider": {
      "module": "core/impl/access/aclMetaMap",
      "options": {
        "dataRepo": "ion://dataRepo",
        "acl": "ion://actualAclProvider",
        "accessManager": "ion://roleAccessManager",
        "map": {
          "person@project-management": {
```

(continues on next page)

(continued from previous page)

```
          "isEntry": true,
          "sidAttribute": "user",
          "jumps": [
            "employee"
          ]
        }
      }
    }
  },
  "fileStorage": {
    "module": "core/impl/resource/OwnCloudStorage",
    "options": {
      "url": "[[ownCloud.url]]",
      "login": "[[ownCloud.login]]",
      "password": "[[ownCloud.pwd]]"
    }
  },
  "dataRepo": {
    "options": {
      "maxEagerDepth": 4
    }
  },
  "customProfile": {
    "module": "lib/plugins/customProfile",
    "initMethod": "inject",
    "options": {
      "auth": "ion://auth",
      "metaRepo": "ion://metaRepo",
      "dataRepo": "ion://dataRepo",
      "fields": {
        "piAct": {
          "caption": "Участник прогресс-индикатора",
          "required": false,
          "readonly": true,
          "type": 4
        }
      },
      "propertyMap": {
        "person@project-management": {
          "filter": "user",
          "properties": {
            "person": "id",
            "piAct": "piAct",
            "surname": "surname"
          }
        }
      }
    }
  },
  "securedDataRepo": {
    "options": {
      "accessManager": "ion://roleAccessManager",
      "roleMap": {
        "eventBasic@project-management": {
          "PROJECT_ADMIN": {
            "caption": "Администратор проекта",
```

(continues on next page)

(continued from previous page)

```
            "resource": {
              "id": "pm::project-events"
            },
            "attribute": "project.administrator"
          },
          "PROJECT_RESPONSIBLE": {
            "caption": "Ответственный по проекту",
            "resource": {
              "id": "pm::project-events"
            },
            "sids": [
              "$project.owner"
            ]
          }
        }
      }
    },
    "indicatorWfHandler": {
      "module": "applications/project-management/lib/util/indicatorWfHandler",
      "initMethod": "init",
      "initLevel": 2,
      "options": {
        "workflows": "ion://workflows",
        "data": "ion://securedDataRepo",
        "log": "ion://sysLog"
      }
    },
    "auth": {
      "options": {
        "checkUrlAccess": [
          "/registry/project-management@project",
          "/portal"
        ]
      }
    }
  },
  "jobs": {
    "fact-creator": {
      "description": "Служба генератора фактический показателей",
      "launch": {
        "day": 1
      },
      "worker": "factCreator",
      "di": {
        "factCreator": {
          "executable": "applications/project-management/lib/fact-creator",
          "options": {
            "log": "ion://sysLog",
            "data": "ion://dataRepo",
            "workflows": "ion://workflows"
          }
        }
      }
    },
    "report-builder": {
```

(continues on next page)

(continued from previous page)

```
      "description": "Служба сборки шахт данных модуля отчетов",
      "launch": {
        "hour": 24
      },
      "worker": "rebuilder",
      "di": {
        "reportMeta": {
          "module": "modules/report/lib/impl/DsReportMetaRepository",
          "initMethod": "init",
          "initLevel": 1,
          "options": {
            "dataSource": "ion://Db",
            "calc": "ion://calculator"
          }
        },
        "stdBuilder": {
          "module": "modules/report/lib/impl/StdMineBuilder",
          "options": {
            "dataSource": "ion://Db",
            "metaRepo": "ion://metaRepo",
            "dataRepo": "ion://dataRepo"
          }
        },
        "rebuilder": {
          "executable": "modules/report/lib/rebuilder",
          "options": {
            "log": "ion://sysLog",
            "meta": "ion://reportMeta",
            "mineBuilders": {
              "project-management": {
                "projects": {
                  "indicatorAll": "ion://stdBuilder"
                }
              }
            }
          }
        }
      }
    }
  },
  "deployer": "built-in",
  "modules": {
    "registry": {
      "globals": {
        "signedClasses": [
          "indicatorBasic@project-management"
        ],
        "staticOptions": {
          "maxAge": 3600000
        },
        "explicitTopMenu": [
          "mytasks",
          {
            "type": "system",
            "name": "report"
```

(continues on next page)

(continued from previous page)


```
      }
    ],
    "eagerLoading": {
      "*": {
        "briefcase@project-management": {
          "item": [
            "projects.typeProject.name"
          ],
          "list": [
            "projects.typeProject.name"
          ],
          "exportItem": [
            "direction.name"
          ],
          "exportList": [
            "result"
          ]
        }
      }
    },
    "listSearchMinLength": 3,
    "listSearchOptions": {
      "indicatorBasic@project-management": {
        "*": {
          "searchBy": [
            "name",
            "objectBasic"
          ],
          "mode": [
            "starts",
            "starts"
          ],
          "joinBy": "and"
        }
      }
    },
    "storage": {
      "basicObj@project-management": {
        "cloudFile": "/${item.code} (${item.name})/",
        "resultCloudFile": "/${item.code} (${item.name})/"
      }
    },
    "defaultPath": "dashboard",
    "inlineForm": true,
    "navigation": {
      "namespaces": {
        "project-management": "Проектное управление"
      },
      "menus": {
        "top": [
          "project-management@mark"
        ]
      }
    },
    "templates": [
      "applications/project-management/templates/registry"
```


(continues on next page)

(continued from previous page)

```
    ],
    "customTemplates": [
      {
        "node": "project-management@eventBasic",
        "classes": [
          {
            "name": "*",
            "types": {
              "create": "task/view",
              "item": "task/view",
              "selectClass": "task/selectClass"
            }
          }
        ]
      },
      {
        "node": "*",
        "classes": [
          {
            "name": "project@project-management",
            "types": {
              "item": "to-gantt-view",
              "selectClass": "task/selectClass"
            }
          }
        ]
      }
    ],
    "statics": {
      "app-static": "applications/project-management/templates/registry/static",
      "app-vendor": "applications/project-management/themes/registry/static/vendor",
      "common-static": "applications/project-management/templates/static"
    },
    "logo": "common-static/logo.png",
    "di": {
      "pmItemToDocx": {
        "module": "modules/registry/export/itemToDocx",
        "initMethod": "init",
        "initLevel": 0,
        "options": {
          "tplDir": "applications/project-management/export/item",
          "injectors": []
        }
      },
      "pmListToDocx": {
        "module": "modules/registry/export/listToDocx",
        "initMethod": "init",
        "initLevel": 0,
        "options": {
          "tplDir": "applications/project-management/export/item2",
          "log": "ion://sysLog"
        }
      },
      "export": {
        "options": {
          "configs": {
```

(continues on next page)

(continued from previous page)

```
        "project@project-management": {
          "passport": {
            "caption": "Паспорт проекта",
            "mimeType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document",
            "extension": "docx",
            "type": "item",
            "preprocessor": "ion://pmItemToDocx",
            "isBackground": true
          },
          "markResult": {
            "caption": "Оценка проектов",
            "mimeType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document",
            "extension": "docx",
            "type": "list",
            "query": {
              "filter": {
                "and": [
                  {
                    "eq": [
                      "$guid",
                      ":project"
                    ]
                  }
                ]
              }
            },
            "params": {
              "project": {
                "caption": "Проект",
                "type": "reference",
                "className": "project@project-management"
              }
            },
            "preprocessor": "ion://pmFromListToDocx",
            "isBackground": true
          }
        }
      }
    },
    "createIndicatorValueHandler": {
      "module": "applications/project-management/lib/actions/createIndicatorValueHandler",
      "initMethod": "init",
      "initLevel": 2,
      "options": {
        "data": "ion://securedDataRepo",
        "workflows": "ion://workflows",
        "log": "ion://sysLog",
        "changelogFactory": "ion://changelogFactory",
        "state": "onapp"
      }
    },
    "actions": {
      "options": {
        "actions": [
          {
```

(continues on next page)

(continued from previous page)

```
            "code": "CREATE_INDICATOR_VALUE",
            "handler": "ion://createIndicatorValueHandler"
          }
        ]
      }
    },
    "digestData": {
      "module": "applications/project-management/lib/digest/digestData",
      "options": {
        "log": "ion://sysLog"
      }
    },
    "signManager": {
      "options": {
        "Preprocessor": "ion://digestData",
        "signaturePreprocessor": "ion://signSaver"
      }
    },
    "treegridController": {
      "module": "applications/viewlib-extra/lib/controllers/api/treegrid",
      "initMethod": "init",
      "initLevel": 0,
      "options": {
        "module": "ion://module",
        "logger": "ion://sysLog",
        "dataRepo": "ion://securedDataRepo",
        "metaRepo": "ion://metaRepo",
        "auth": "ion://auth",
        "config": {
          "*": {
            "eventBasic@project-management": {
              "roots": [
                {
                  "property": "name",
                  "operation": 1,
                  "value": [
                    null
                  ],
                  "nestedConditions": []
                }
              ],
              "childs": [
                "basicObjs"
              ]
            }
          }
        }
      }
    },
    "fileshareController": {
      "module": "applications/viewlib/lib/controllers/api/fileshare",
      "initMethod": "init",
      "initLevel": 0,
      "options": {
        "module": "ion://module",
        "fileStorage": "ion://fileStorage"
```

(continues on next page)

(continued from previous page)


```
        }
      }
    },
    "dashboard": {
      "project-management": {
        "modules": {
          "dashboard": {}
        }
      }
    }
  }
},
"geomap": {
  "globals": {
    "ymapControls": {
      "loader": {
        "position": {
          "left": 15,
          "top": 90
        }
      },
      "rulerControl": null,
      "typeSelector": {
        "float": "right"
      },
      "zoomControl": {
        "position": {
          "right": 10,
          "top": 10
        }
      }
    },
    "panels": {
      "rightInfo": {
        "type": "rightInfo"
      },
      "navFloat": {
        "type": "float",
        "cssClass": "map-nav-float nav-tree",
        "cssStyle": "left:10px; top:46px; width: 310px; max-height:calc(100% - 163px);"
      },
      "filterFloat": {
        "type": "float",
        "title": "Фильтры",
        "cssClass": "map-filter-float collapsible",
        "cssStyle": "left:10px; bottom:10px; width: 310px; max-height:calc(100% - 163px);"
      }
    },
    "hidePageHead": false,
    "hidePageSidebar": true,
    "stroke": {
      "panel": {
        "name": "filterFloat"
      },
      "path": {
        "strokeColor": "#00ff00",
```


(continues on next page)

(continued from previous page)


```
      "strokeWidth": 6,
      "opacity": 0.8
    },
    "polygon": {
      "fillColor": "#00ff00",
      "fillOpacity": 0.1,
      "strokeColor": "#00ff00",
      "strokeOpacity": 0.9,
      "strokeWidth": 3
    }
  },
  "namespaces": {
    "project-management": "Геоданные проекта"
  },
  "templates": [
    "applications/project-management/templates"
  ],
  "statics": {
    "geoicons": "applications/project-management/icons"
  },
  "start": [
    135.07,
    48.48
  ],
  "zoom": 10,
  "regions": {
    "enabled": true,
    "osmIds": [
      "151223"
    ],
    "panel": {
      "name": "filterFloat"
    },
    "button": {
      "caption": "Районы",
      "hint": "Фильтр по районам",
      "resetHint": "Сбросить фильтр"
    },
    "levels": {
      "4": {
        "strokeWidth": 3,
        "strokeColor": "#7e8dab",
        "strokeStyle": "solid",
        "strokeOpacity": 1,
        "fillColor": "#fffff",
        "fillOpacity": 0
      }
    }
  },
  "defaultNav": {
    "namespace": "project-management",
    "node": "objectBasic"
  },
  "search": {
    "panel": {
      "name": "filterFloat",
```


(continues on next page)

(continued from previous page)

```
        "orderNumber": 10
      },
      "enabled": true,
      "timeout": 2000
    },
    "formFilter": {
      "panel": {
        "name": "filterFloat"
      }
    },
    "di": {
      "dataRepo": {
        "module": "core/impl/datarepository/ionDataRepository",
        "options": {
          "dataSource": "ion://Db",
          "metaRepository": "ion://metaRepo",
          "fileStorage": "ion://fileStorage",
          "imageStorage": "ion://imageStorage",
          "log": "ion://sysLog",
          "keyProvider": {
            "name": "keyProvider",
            "module": "core/impl/meta/keyProvider",
            "options": {
              "metaRepo": "ion://metaRepo"
            }
          },
          "maxEagerDepth": 3
        }
      }
    }
  },
  "import": {
    "src": "applications/project-management/geo",
    "namespace": "project-management"
  }
},
"gantt-chart": {
  "globals": {
    "staticOptions": {
      "maxAge": 3600000
    },
    "config": {
      "columns": [
        {
          "name": "owner",
          "caption": "Владелец",
          "align": "center",
          "filter": true,
          "editor": {
            "type": "select2",
            "from": "employee@project-management"
          }
        }
      ],
      "preConfigurations": {
        "config2": {
```

(continues on next page)

(continued from previous page)

```
            "caption": "Расширенная",
            "showPlan": false,
            "units": "year",
            "days_mode": "full",
            "hours_mode": "work",
            "columnDisplay": {
              "text": true,
              "owner": true,
              "priority": true,
              "start": true,
              "progress": true
            }
          }
        },
        "roots": [
          "briefcase@project-management",
          "project@project-management"
        ],
        "initialDepth": 1,
        "createUrl": {
          "project@project-management": "registry/project-management@myprojectevent.all/new/{{parentClass}}.
↪{{parentId}}/basicObjs/event@project-management"
        },
        "searchCount": 25,
        "inplaceCreation": {
          "rootLevel": true,
          "skip": [
            "briefcase@project-management"
          ],
          "ambigiousDefault": "event@project-management",
          "force": {
            "@root": "briefcase@project-management",
            "eventObject@project-management": "eventOnly@project-management"
          }
        },
        "map": {
          "employee@project-management": {
            "eager": [
              "person",
              "organization"
            ]
          },
          "project@project-management": {
            "type": "project",
            "open": true,
            "color": "#e3fcef",
            "textColor": "#000",
            "text": "name",
            "override": {
              "owner": "head"
            },
            "parents": [
              "briefcase"
            ],
            "filter": {
              "ne": [
```

(continues on next page)

(continued from previous page)


```
            "$archive",
            true
          ]
        },
        "url": "registry/project-management@myprojectevent.all/view/:class/:id"
      }
    }
  },
  "statics": {
    "common-static": "applications/project-management/templates/static"
  },
  "logo": "common-static/logo.png",
  "rootParamNeeded": true
 }
},
"report": {
 "globals": {
  "namespaces": {
   "project-management": "Проектное управление"
  },
  "defaultNav": {
   "namespace": "project-management",
   "mine": "projects",
   "report": "roadmap"
  },
  "mineBuilders": {
   "project-management": {
    "test": {
     "projects": "mineBuilder"
    },
    "projects": {
     "indicatorAll": "mineBuilder"
    }
   }
  },
  "di": {},
  "statics": {
   "common-static": "applications/project-management/templates/static"
  },
  "logo": "common-static/logo.png"
 },
 "import": {
  "src": "applications/project-management/bi",
  "namespace": "project-management"
 }
},
"rest": {
 "globals": {
  "di": {}
 }
},
"portal": {
 "import": {
  "src": "applications/project-management/portal",
  "namespace": "project-management"
 },
```


(continues on next page)

(continued from previous page)

```
    "globals": {
      "portalName": "pm",
      "needAuth": true,
      "default": "index",
      "theme": "project-management/portal",
      "templates": [
        "applications/project-management/themes/portal/templates"
      ],
      "statics": {
        "pm": "applications/project-management/themes/portal/static"
      },
      "pageTemplates": {
        "navigation": {
          "index": "pages/index"
        }
      }
    }
  },
  "ionadmin": {
    "globals": {
      "defaultPath": "ionadmin",
      "securityParams": {
        "resourceTypes": {
          "*": {
            "title": "Общие"
          }
        },
        "hiddenRoles": [
          "^PROJ_DEPART_EMPLOYEE"
        ]
      }
    },
    "statics": {
      "common-static": "applications/project-management/templates/static"
    },
    "logo": "common-static/logo.png"
  },
  "dashboard": {
    "globals": {
      "namespaces": {
        "project-management": "Проектное управление"
      },
      "root": {
        "project-management": "applications/project-management/dashboard"
      }
    }
  },
  "diagram": {
    "globals": {
      "config": {
        "org1": {
          "caption": "Организационная структура",
          "edit": true,
          "showSections": false,
          "relations": {
            "className": "organization@project-management",
```

(continues on next page)

(continued from previous page)


```
            "title": "name",
            "text": "address",
            "img": "",
            "filter": [
              {
                "property": "headOrg",
                "operation": 0,
                "value": [
                  null
                ],
                "nestedConditions": []
              }
            ],
            "children": [
              {
                "className": "branchOrg@project-management",
                "property": "branch",
                "title": "name",
                "text": "address",
                "children": [
                  {
                    "className": "branchOrg@project-management",
                    "property": "branch",
                    "children": []
                  }
                ]
              }
            ]
          }
        }
      }
    }
  }
}
```


Конфигурационный файл deploy.json - это файл, в котором описывается структура параметров программной системы и её конкретная настройка.

Предусмотрен также способ задания конфигурации через множество отдельных файлов: сборка deploy из отдельных файлов

Структура файла deploy.json:

Поле	Имя	Описание
"namespace":	Название проекта	Пространство имен проекта.
"parametrization": true,	Параметризация	Подключение параметризации. При установленном значении “true” есть возможность задавать параметры, в которые, при сборке приложения, передаются переменные, заданные в ini-файлах или переменных окружения проекта.
"globals": {	Глобальные настройки	Глобальные параметры конфигурирования.
"deployer": "built-in"	Сборки	Параметр конфигурирования сборки, на данный момент единственный.
"modules":	Настройки модулей	Массив объектов - описание модулей.

Пример файла deploy.json

3.5.4 Зависмости в package.json

Файл package.json - определяет структуру зависимостей и детальный состав модулей системы.

```
"ionMetaDependencies": {
  "viewlib": "0.0.1"
}
```

Логика подключения при помощи скрипта

- если в названии объекта отсутствует слеш - / => “project-management”- подставляем в путь по умолчанию группу ION-APP - т.е. путь - //git.iondv.ru/ION-APP/project-management.
- если в названии есть слеш - значит задан уже с группой и просто склеиваем путь к гиту с группой и метой, пример “ION-METADATA/viewlib” - путь - //git.iondv.ru/ION-METADATA/viewlib.
- если значение версии начинается с git+http:// или git+https:// - то это полный путь к внешнему репозиторию - отбрасываем git+ и тянем гитом.
- если значение версии начинается с http:// или https:// - то это полный путь к архиву - тянем и распаковываем.

  Не реализовано, так как dapp не поддерживает работу с архивами.

Пример файла package.json

```
{
  "name": "develop-and-test",
  "description": "Метапроект для тестирования и разработки",
  "version": "1.9.2",
  "homepage": "http://docker.local:8080",

  "engines": {
    "ion": "1.24.1"
  },
  "scripts": {
    "test": "mocha ./test/e2e/**/*.js"
  },
  "ionModulesDependencies": {
    "registry": "1.27.1",
    "geomap": "1.5.0",
    "portal": "1.3.0",
    "report": "1.9.2",
    "ionadmin": "1.4.0",
    "dashboard": "1.1.0",
    "rest": "1.1.2",
    "gantt-chart": "0.8.0"
  },
  "ionMetaDependencies": {
    "viewlib": "0.9.1"
  }
}
```

Описание полей

Поле	Наименование	Описание
"name"	Имя	Имя проекта.
"description"	Описание	Описание проекта.
"version"	Версия	Номер текущей версии.
"homepage"	Домашняя страница	Ссылка на собранный проект на докере.
"bugs"	Ошибки	Указывается ссылка на проект приложения в GitLab, где принимаются заявки об ошибках.
"repository"	Репозиторий	Состоит из полей “type” и “url”. Указывается тип репозитория и ссылка на него.
"engines"	Ядро	Номер версии ядра.
"scripts"	Скрипты	Скрипт для сборки меты из разных групп и разных url./
"ionModulesDependencies"	Зависимости модулей ion	Зависимости модули и их версии, используемые в приложении. Проект включает в себя следующий состав модулей: • “ionadmin” – модуль администрирования • “registry” – модуль регистра • “report” – модуль отчетов • “rest” - модуль rest-сервисов • “dashboard” – модуль дашбоардов • “geomap” - геомодуль • “gantt-chart” – модуль диаграмм ганта • “portal” – модуль портала
"ionMetadataDependencies"	Зависимости метаданных ion	Дополнительные приложения для функционирования системы.
"dependencies"	Зависимости	Прочие зависимости проекта из репозитория npm.

3.5.5 Операции с учетными записями MongoDB через CLI

Авторизация

Для авторизации при взаимодействии с СУБД через консольный интерфейс, в mongo передаются параметры

1. --authenticationDatabase <база данных, на основе которой производится авторизация>
2. -u <имя пользователя>

3. -p <пароль>

пример:

```
mongo --authenticationDatabase admin -u admin -p 123
```

Создание пользователя

Для создания пользователя необходимо иметь роль dbOwner, либо иметь роль userAdmin в базе данных, на основании которой производится авторизация, например admin, и во всех базах данных, где добавляются роли, или привилегию на действие createUser в этой базе и привилегии на действие grantRole в базах данных, где добавляются роли.

Создание пользователя через консольный интерфейс осуществляется командой

```
mongo --eval "(new Mongo()).getDB(<имя бд, куда писать данные авторизации>').createUser( \
{ \
  user: '<пользователь>', \
  pwd: '<пароль>', \
  roles: [ \
    { role: 'readWrite', db: '<бд куда доступ на чтение-запись>' }, \
    { role: 'read', db: '<бд где только чтение>' }, \
    { role: 'write', db: '<бд где только запись>' } \
  ]})"
```

пример:

```
mongo --eval "(new Mongo()).getDB('admin').createUser( \
{ \
  user: 'admin', \
  pwd: '123', \
  roles: [ \
    { role: 'readWrite', db: 'admin' }, \
    { role: 'readWrite', db: 'config' }, \
    { role: 'readWrite', db: 'local' } \
  ]})"
```

или в одну строку

```
mongo --eval "(new Mongo()).getDB('admin').createUser({user: 'demo',pwd: '123',roles: [{ role: 'readWrite',␣
↪db: 'admin' },{ role: 'readWrite', db: 'config' },{ role: 'readWrite', db: 'local' }]})"
```

Удаление пользователя

Для удаления пользователя необходимо иметь роль dbOwner, либо иметь роль userAdmin или привилегию на действие dropUser в базе данных, на основе которой производится авторизация, например admin.

Удалить пользователя через CLI mongodb можно командой

```
mongo --eval "(new Mongo()).getDB('<бд с данными авторизации>').dropUser('<имя>')"
```

пример:

```
mongo --eval "(new Mongo()).getDB('admin').dropUser('demo')"
```

ГЛАВА 4

4. Модули

4.1 Модуль Registry

4.1.1 Логика формирования id элементов, связанных с объектами по типам

Реализовано

INPUT, SELECT, TEXTAREA

1. для редактирования атрибута объекта a_{неймспейс, класс, атрибут}

```
<input type="text" class="form-control" id="a_develop-and-test_class_integer_integer_integer" name=
↪"integer_integer" pattern="[0-9]+([\.|,][0-9]+)?" value="5120">
```

BUTTON

1. действие списка la_{неймспейс, класс, имя команды}

```
<button id="la_develop-and-test_class_integer_edit" class="edit btn btn-info command" title="Редактировать
↪" data-id="EDIT" style="display: inline-block;">Править</button>
```

2. действие формы fa_{неймспейс, класс, имя команды}

```
<button id="fa_develop-and-test_class_integer_saveandclose" data-id="SAVEANDCLOSE" type="button"␣
↪class="btn command object-control SAVEANDCLOSE" style="">
      Сохранить и Закрыть
    </button>
```

3. действие ссылочного поля ra_{неймспейс, класс, атрибут, имя команды}

```
<button id="ra_develop-and-test_ref_use_ref_use_create" type="button" class="create-btn btn btn-success"
↪data-ref-property="ref_use" title="Создать">
                <span class="glyphicon glyphicon-plus-sign"></span>
            </button>
```

4. действие поля коллекции ca_{неймспейс, класс, атрибут, имя команды} нет возможности проверить, коллекции в процессе

5. плавающие кнопки формы ffa_{неймспейс, класс, атрибут, имя команды}

```
<button id="ffa_develop-and-test_class_integer_close" type="button" class="btn btn-default CLOSE" title=
↪"Закрыть" data-cmd="CLOSE">
    <span class="glyphicon glyphicon-remove"></span>
  </button>
```

FORM, DIV, TR, LI (для представления объектов)

1. секции и узлы навигации n_{имя секции навигации}

```
<a id="n_simple_types" href="#" title="Простые типы">
  <i class="fa fa-menu-arrow pull-right toggler"></i>
  <i class="main-icon fa fa-institution" title="Простые типы"></i>
  <span>Простые типы</span>
 </a>
```

Узел навигации n_{код узла навигации}.

```
<a id="n_class_integer" class="menu-link" href="/registry/develop-and-test@class_integer" title="Класс &
↪quot;Целое [6]">Класс "Целое [6]</a>
```

Узел навигации открытого класса.

4.1.2 Настройки DI

Подключение в глобальных настройках модуля регистра

Пример в deploy.json

```
"modules": {
   "registry": {
     "globals": {
    "di": {
```

treegridController

Описание

Предназначен для создания иерархичных списков объектов в атрибуте-коллекции класса или в навигации класса.

Работает с использованием компонента dhtmlxSuite_v51_pro (https://dhtmlx.com/docs/products/dhtmlxTreeGrid/).

Подключение в DI

```
"treegridController": {
        "module": "applications/viewlib/lib/controllers/api/treegrid",
        "initMethod": "init",
        "initLevel": 0,
        "options": {
          "module": "ion://module",
          "logger": "ion://sysLog",
          "securedDataRepo": "ion://securedDataRepo",
          "metaRepo": "ion://metaRepo",
          "auth": "ion://auth",
          "config": { // основной конфиг
            "*": { // выборка объектов возможна в каждой навигации
              "eventBasic@project-management":{ // выборка объектов по указанному классу
                "roots":[{ // поиск корней
                  "property": "name",
                  "operation": 1,
                  "value": [null],
                  "nestedConditions": []
                }],
                "childs":["basicObjs"] // поиск дочерних элементов
              },
```

Виды шаблонов

1) "template": "treegrid/collection"

Для атрибута-коллекции. Подключается в представлении формы объекта:

```
"options": {
        "template": "treegrid/collection",
        "reorderable": true,
        "treegrid": {
          "width": "auto,100,100,100,100,0",
          "align": "left, center,center,center,center, left",
          "sort": "str, date, date, date, date, int",
          "enableAutoWidth": false,
          "paging": {
            "size": 20
          }
        }
      }
```

2) "template": "treegrid/list"

Для навигации класса. Подключение:

```
"options": {
   "template": "treegrid/list"
 }
```

3) Настройка skin

https://docs.dhtmlx.com/grid__skins.html

```
"options" : {
...
  "treegrid" : {
    "skin": "material" // по умолчанию
    // "skin": "skyblue"
    // "skin": "terrace"
    // "skin": "web"
  }
}
```

Дополнительные источники информации по treegridController

- Иерархическое представление для коллекций.

DHTMLX (dhtmlxSuite_v51_pro)

- https://docs.dhtmlx.com/
- https://dhtmlx.com/docs/products/dhtmlxTreeGrid/

Модуль регистра (registry) – ключевой модуль, предназначенный непосредственно для работы с данными на основе структур метаданных – в том числе для ведения проектов, программ, мероприятий и др.

4.1.3 Настройка

- DI (treegridController)

Deploy

Настройка частоты опроса в deploy.json

Настройка частоты опроса сервера, что объект не заблокирован в deploy.json:

```
"registry": {
  "globals": {
    "notificationCheckInterval": 60000 // раз в минуту
  }
}
```

Настройка createByCopy

Настройка отображения кнопки “Создать еще” в deploy.json:

```
"createByCopy": [
        "person@khv-childzem" // класс
      ],
```

Фильтры

Подробнее о фильтрах здесь.

Настройка помощь по фильтрам

Справка размещается в файле шаблона modules/registry/view/default/templates/view/list-filter-helper.ejs

4.1.4 Требования к коду

Стиль написания компонентов фронт-енда здесь.

4.2 Модуль Report

4.2.1 Замечания при проектировании шахты

Операции “NOT CONTAINS”

При использовании Монго БД операции NOT CONTAINS работать корректно не будут, так как NOT применяется не к CONTAINS, а к условиям вхождения.

На данный момент изменить это не представляется возможным.

Не настроен сборщик для источника данных

Не настроен сборщик для источника данных "summaryArea.serviceGrid". Кроме самой меты в deploy.json в modules.report.globals необходимо прописать следующее:

```
"mineBuilders": {
   "khv-svyaz-info": {
     "summaryArea": {
       "internet": "mineBuilder",
       "population": "mineBuilder",
       "internetGrid": "mineBuilder",
       "station": "mineBuilder",
       "serviceGrid": "mineBuilder"
     }
   }
 }
```

Это - привязка сборщиков к источникам данных, чтобы иметь возможность агрегировать из разных БД (и вообще источников). Сейчас пока используется стандартный mineBuilder, использующий локальный датасорс Db.

Нельзя сравнить два поля

Если сравниваем два поля между собой с использованием поисковых выражений, то работать не будет. Монго БД не умеет сравнивать два поля между собой. Например подобные выражения работать не будут:

```
{"attr1": {"$regex": "$attr2", "$options": "i"}}
```

Фильтры

Для сорсов на основе классов фильтры указываются через кондишнсы.

```
"filter": [
          {
            "property": "typePet",
            "operation": 0,
            "value": "statement",
            "nestedConditions": []
          }
        ]
```

Модуль отчетов (report) – модуль для формирования аналитических отчетов и справочной информации в виде графиков на основе специальных метаданных. Расчеты могут выполняться по расписанию или быть инициированы оператором.

4.2.2 Библиотека для построения отчетов вида Pivot

Библиотекой для построения отчетов вида Pivot служит PivotTable.js (Примеры и описание: https://pivottable.js.org) Функционал богатый, в то же время сложный для построения отчетов уровня экселя или ворда. На данный момент, при проектировании отчета, лучше посоветоваться с разработчиками и поставить таск на реализацию отчета вида Pivot, если ожидаемый функционал отчета относительно сложен.

4.2.3 Метаданные

Метаданные модуля отчетов (шахта, навигация)

Замечания при проектировании шахты

4.2.4 Настройка автоматической сборки шахты данных

Добавьте настройку jobs.enabled=true в файл config.ini чтобы настроить автоматическую сборку шахты данных.

Пример

Чтобы запускать задание при старте приложения и далее каждые 6 часов, нужно настроить джоб следующим образом:

```
"jobs": {
    "report-builder": {
      "description": "Служба сборки шахт данных модуля отчетов",
      "launch": {
        "hour": 21600000
      }
    }
```

4.3 Модуль Gantt-chart

Модуль диаграм ганта (gantt-chart) – модуль, предназначенный для вывода специфичных типов иерархических данных, имеющих даты.

4.3.1 Конфигурация в deploy

Указание длины поисковой выдачи

```
"searchCount": 25
```

Добавление фильтров на колонки

```
"gantt-chart": {
    "globals": {
      "config": {
        "columns": [
          {
            "name": "text",
            "caption": "Название"
          },
          {
            "name": "owner",
            "caption": "Владелец",
            "align": "center",
            "filter": true
          },
          {
            "name": "priority",
            "caption": "Приоритет",
            "align": "center",
            "filter": true
          },
```

Указание для разных классов разные createUrl

```
"createUrl": {
        "project@project-management": "registry/project-management@eventBasic/new/eventBasic",
        "event@project-management": "registry/project-management@eventBasic/new/eventBasic",
```

(continues on next page)

(continued from previous page)

```
        "eventObject@project-management": "registry/project-management@eventOnly/new/eventOnly",
        "eventOnly@project-management": "registry/project-management@eventOnly/new/eventOnly",
        "projectKNA704@project-management": "registry/project-management@eventKNA704/new/
↪eventKNA704",
        "eventKNA704@project-management": "registry/project-management@eventOnlyKNA704/new/
↪eventOnlyKNA704",
        "eventObjectKNA704@project-management": "registry/project-management@eventOnlyKNA704/new/
↪eventOnlyKNA704",
        "eventOnlyKNA704@project-management": "registry/project-management@eventOnlyKNA704/new/
↪eventOnlyKNA704"
      }
```

4.3.2 Настройки видов представлений

```
"preConfigurations": {
        "config1": {
          "caption": "Основная",
          "showPlan": true,
          "units": "month",
          "step": 3,
          "days_mode": "full",
          "hours_mode": "full",
          "default": true
        },
        "config2": {
          "caption": "Расширенная",
          "showPlan": false,
          "units": "year",
          "days_mode": "full",
          "hours_mode": "work",
          "columnDisplay": {
            "text": true,
            "owner": true
          },
          "filters": {
            "priority": "Высокий"
          }
        },
        "config3": {
          "caption": "Обзорная",
          "showPlan": true,
          "units": "year",
          "step": 5,
          "days_mode": "full",
          "hours_mode": "full",
          "columnDisplay": {
            "text": true,
            "owner": true,
            "priority": true
          },
          "filters": {
            "priority": "Обычный"
          }
        }
      }
```

В поле filters - задаем свойство и значения для фильтра

Настраиваемый фильтр при выборке подузлов

В формулах в общем синтаксисе выражений теперь можно обращаться к данным контекста. Пока реализовано только для списков в регистри и ганте. По мере перехода на общий синтаксис реализуем поддержку повсеместно в ядре.

Настраиваемый фильтр не применяется к корневому обьекту явно указанному через параметр урла, или выбранный в выпадающем списке. Фильтр применяется только при ВЫБОРКЕ ПОДУЗЛОВ.

Сортировка выдачи

При выводе проекта, в них мероприятия сортируются по атрибуту numEvent - на всех уровнях иерархии.

```
"sortBy": "numEvent"

// либо
"sortBy": {"numEvent": -1, "anyOtherAttr": 1}
```

Настройка списка выбора объекта для вывода информации

Применяется при условии настроенного фильтра для колонки и позволяет не отображать все объекты сразу, а выбирать из списка. Если значение "rootParamNeeded:true" - выводится пустой экран и окно для выбора проекта.

```
"gantt-chart": {
    "globals": {
      "rootParamNeeded": true
    }
  }
```

4.4 Модуль портала

Модуль портала (portal) – модуль, предназначенный для отображения произвольных шаблонов данных. Модуль портала выполняет функцию отображения дизайна различной информации с помощью языков разметки Markdown и HTML.

4.4.1 Состав

Слой представления - отвечает за отображение данных:

1. Adapter Provider - выполняет функцию связи между данными и отображением их на портале (настройка через файл ‘deploy.json’)
2. File Adapter - возвращает ресурсы типа “Файл” (остается в памяти приложения)
3. Class Adapter - предназначен для отображения данных из базы через репозиторий данных (каждый раз обновляются)

4.4.2 Логика отображения данных на портале

Стили портала

Для описания/оформления внешнего вида страниц портала применяются CSS (каскадные таблицы стилей) Пример оформления страницы портала в Dnt - /develop-and-test/portal/view/static/css/style.css

Возможные варианты для применения стилей задаются в папке portal/view/static/style.css.

```
.navbar {
  border-radius: 0;
  margin-bottom: 0;
}

.navbar .navbar-brand {
  padding: 5px;
}
...
```

Страницы портала

Шаблоны страниц портала настраиваются в папке portal/view/templates.

Расположение объектов на странице описывается в формате языка разметки HTML:

```
<html>
<head>
  <title><%= portal %></title>
  <link href="/<%= module %>/vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">
  <link href="/<%= module %>/dnt/css/style.css" rel="stylesheet">
</head>
<body>
  <%- partial('../parts/menu', { menu }) %>
  <%- body -%>
  <script src="/<%= module %>/vendor/jquery/jquery.min.js"></script>
  <script src="/<%= module %>/vendor/bootstrap/js/bootstrap.min.js"></script>
  <script src="/<%= module %>/js/scripts.js"></script>
</body>
</html>
```

- Содержание тега <head> включает в себя отображение стилей оформления внешнего вида портала.
- В тег <body> включается информация об объектах, отображаемых на странице портала.
- Тег <script> содержит в себе информацию в виде ссылки на программу или ее текст на определенном языке. Скрипты могут располагаться во внешнем файле и связываться с любым HTML-документом, что позволяет использовать одни и те же функции на нескольких веб-страницах, тем самым ускоряя их загрузку. В данном случае тег <script> применяется с атрибутом src, который указывает на адрес скрипта из внешнего файла для импорта в текущий документ.

Навигация портала

Мета навигации портала представляет собой набор узлов навигации, у каждого из которых указан тип ресурса.

Пример создания секции навигации:

```
{
  "code": "main",
  "caption": "Главное меню",
  "itemType": "section",
  "subNodes":["classes", "texts"]
}
```

- code - системной имя объекта
- caption - логическое имя объекта
- itemType - тип отображения объекта
- subNodes - массив узлов навигации, содержащихся в данной секции

Пример создания узла навигации:

```
{
  "code": "texts",
  "caption": "Публикация текстов",
  "resources": "texts",
  "PageSize": 5,
  "itemType": "node"
}
```

- code - системное имя объекта
- caption - логическое имя объекта
- resources - превращение данных в контент портала
- PageSize - размер страницы
- itemType - тип отображения объекта

Оформление данных

1. Формат разбиения информации на страницы

```
<% layout('./layout/content') %>
<%
if (Array.isArray(resources) && resources.length) {
 resources.forEach(function(resource){
%>
<div>
<h3 id="<%= node.code %>_<%= resource.getId() %>">
  <a href="/<%= module %>/<%= node.code %>/<%= resource.getId() %>">
    <%= resource.getTitle() %>
  </a>
  <%
    var formatedDate = null;
    var date = resource.getDate();
    if (date) {
       formatedDate = date.toLocaleString('ru',{year: 'numeric', month: 'numeric', day: 'numeric'});
    }
    %>
  <% if (formatedDate) { %><small><%= formatedDate %></small><% } %>
</h3>
<p><%- resource.getContent() %></p>
```

(continues on next page)

(continued from previous page)

```
</div>
<%
 })
}
%>
<%- partial('./parts/pagination', { resources }) %>
```

2. Формат корректного отображения текста ошибок

```
<% layout('./layout/layout') %>
<div class="container">
  <h1>404</h1>
  <h2>Страница не найдена</h2>
</div>
```

3. Формат преобразования данных в контент портала

```
<% layout('./layout/layout') %>

<div class="container">

  <div class="row">
    <div class="col-md-12">
      <div class="page-header">
        <h2><%= resource.getTitle() %></h2>
      </div>
      <div>
        <%
        var formatedDate = null;
        var date = resource.getDate();
        if (date) {
          formatedDate = date.toLocaleString('ru',{year: 'numeric', month: 'numeric', day: 'numeric'});
        }
        %>
        <% if (formatedDate) { %><h1><small><%= formatedDate %></small></h1><% } %>
      </div>
      <div>
        <%- resource.getContent() %>
      </div>
    </div>
  </div>

</div>
```

4. Формат отображения текста

```
<% layout('./layout/layout') %>

<div class="container">

  <div class="row">
    <div class="col-md-12">
      <div>
        <%
        var formatedDate = null;
        var date = resource.getDate();
```

(continues on next page)

(continued from previous page)

```
      if (date) {
        formatedDate = date.toLocaleString('ru',{year: 'numeric', month: 'numeric', day: 'numeric'});
      }
      %>
      <% if (formatedDate) { %><h1><small><%= formatedDate %></small></h1><% } %>
    </div>
    <div>
      <%- resource.getContent() %>
    </div>
  </div>
</div>

</div>
```

4.5 Модуль Панель управления (dashboard)

Модуль дашбоарда (dashboard) – модуль для выведения краткой информации в виде блоков. Основывается на модели виджетов.

4.5.1 Структура модуля

Панель управления состоит из трех базовых сущностей - менеджер, макет и виджет.

Менеджер (manager)

Менеджер - это основной компонент модуля, отвечающий за создание и инициализацию виджетов, макетов, подключение панели к другим модулям.

```
let manager = require('modules/dashboard/manager');
```

Макет (layout)

Макет - это EJS шаблон, в котором определяются схема размещения виджетов, параметры ддя шаблонов виджетов, плагин для управления сеткой макета на клиенте (например gridster), подключаются общие ресурсы.

Базовые макеты модуля находятся в папке /dashboard/layouts. Опубликованные из мета-данных в папке /applications/${meta-namespace}/layouts.

Каждый макет имеет уникальный ID. При публикации макета из меты к ID добавляется префикс.

```
let dashboard = require('modules/dashboard');
dashboard.getLayout('demo');
dashboard.getLayout('develop-and-test-demo');
```

При рендере макета необходимо передать объект manager.

```
res.render(dashboard.getLayout('demo'), { dashboard });
```

Виджет (widget)

Виджет - это объект, который размещается на макете и взаимодействует с сервером через ajax-запросы.

Базовые виджеты модуля находятся в папке /dashboard/widgets. Опубликованные из мета-данных в папке /applications/${meta-namespace}/widgets.

Виджет состоит из файла класса index.js и шаблона представления view.ejs. Класс должен наследоваться от базового класса /dashboard/base-widget или его потомков.

- Метод init() отвечает за начальную инициализацию виджета при старте сервера.
- Метод refresh() вызывается при получении ajax-запроса от клиента.
- Метод job() получает данные для виджета.

Каждый виджет имеет уникальный ID. При публикации виджета из меты к ID добавляется префикс.

```
dashboard.getWidget('demo');
dashboard.getWidget('develop-and-test-demo');
```

При рендере представления виджета необходимо передать объект widget.

```
<% var widget = dashboard.getWidget('develop-and-test-demo') %>
<%- partial(widget.view, {widget}) %>
```

4.5.2 Публикация из меты

Пример структуры в applications/develop-and-test

```
dashboard
    layouts
      demo-layout
    widgets
      demo-widget
        index.js
        view.ejs
    static
        layouts
        widgets
          demo-widget
```

Для того, чтобы данные из меты подгружались в модуль дашборда, необходимо в файле конфигурации приложения deploy.json добавить следующую секцию, в раздел "modules":

```
"dashboard": {
  "globals": {
    "namespaces": {
      "develop-and-test": "Мета для тестирования и разработки"
    },
    "root": {
      "develop-and-test": "applications/develop-and-test/dashboard"
    }
  }
}
```

4.6 Модуль Ionadmin

4.6.1 Безопасность

Инициализация

При первичной настройке безопасности необходимо выполнить следующее:

1) Синхронизировать права, чтобы права, заведенные через утилиту acl, появились в админке. Для этого на странице /ionadmin/security/sync жмем кнопку "Синхронизация прав доступа".

По завершении будет сообщение "Синхронизация прав доступа успешно проведена!".

2) Сделать импорт ресурсов. Если на странице /ionadmin/security/resource нет объектов или их действительно мало, тогда надо выполнить импорт ресурсов.

Выполняется на странице /ionadmin/security/sync по кнопке "Импорт ресурсов".

По завершении будет сообщение "Импорт успешно завершен!".

Управление ролями

На странице /ionadmin/security/role можно создавать, редактировать или удалять роли:

1) Создание роли. Для создания роли на странице /ionadmin/security/role нажимаем кнопку “Создать”

Происходит переход на новую страницу, где надо указать идентификатор роли на английском языке. Нажимаем на кнопку “Сохранить” для подтверждения создания указанной роли

2) Редактирование роли. Для редактирования роли на странице /ionadmin/security/role выбираем необходимую роль и нажимаем кнопку “Править”

Происходит переход на новую страницу, где на форме отображаются следующие поля:

Идентификатор - выражение на английском языке для присваивания уникального имени роли. При ее изменении могут слететь права у пользователей, которые ранее привязали данную роль со старым идентификатором. В таком случае, необходимо каждому пользователю привязать роль заново.

Название - осмысленное название роли, может содержать выражения на русском языке.

Права доступа - вкладки для раздачи прав:

- Общие - используется для раздачи роли доступа ко всем ресурсам (* - все ресурсы)
- Навигация - используется для раздачи роли доступа к меню модуля регистра. Сначала отображается системное наименование проекта, у которого есть знак плюса для отображения внутренних и импортированных ресурсов. Раздаются в данном пункте пока только права на чтение меню.
- Классы - используется для раздачи роли доступа к классам метаданных. Сначала отображается системное наименование проекта, у которого есть знак плюса для отображения внутренних и импортированных ресурсов. У этих ресурсов можно задать отдельные права по каждому ресурсу.

3) Удаление роли. Для удаления роли на странице /ionadmin/security/role выбираем необходимую роль и нажимаем кнопку “Удалить”

Подтверждаем удаление роли:

Доступ к ресурсу

При управлении ролями в правах доступа предоставляются следующие доступы к каждому ресурсу:

Доступ	Описание доступа
Полный доступ	Включает в себя все остальные доступы. Нельзя выбрать полный доступ и еще дополнительный доступ на чтение, запись, удаление или использование. Для навигации предоставляется доступ только на чтение ресурса.
Чтение	Возможность чтение объектов ресурса
Запись	Возможность редактирования объектов ресурса, не применяется для навигации
Удаление	Возможность удаления объектов ресурса, не применяется для навигации
Использование	Возможность создания объектов ресурса, не применяется для навигации

Доступ может назначаться всей группе ресурсов или отдельно каждому ресурсу и доступу в нем.

Управление пользователями

На странице /ionadmin/security/user можно создавать, редактировать или удалять пользователей:

1) Создание пользователя. Для создания пользователя на странице /ionadmin/security/user нажимаем кнопку “Создать”

Происходит переход на новую страницу, где надо указать логин пользователя на английском языке, пароль, описание в имени. Нажимаем на кнопку “Сохранить” для подтверждения создания указанного пользователя

2) Редактирование пользователя. Для редактирования пользователя на странице /ionadmin/security/user выбираем необходимого пользователя и нажимаем кнопку “Править”

Происходит переход на новую страницу, где на форме отображаются следующие поля:

Тип - тип учетной записи пользователя, пока доступны в админке локальные пользователи.

Логин - идентификатор пользователя на английском языке.

Пароль - пароль пользователя.

Имя - осмысленное название пользователя, может содержать выражения на русском языке, например, Фамилия И.О.

Роли - список ролей пользователя. Если проставлена галка у роли, то роль привязана к пользователю.

3) Удаление пользователя. Для удаления пользователя на странице /ionadmin/security/user выбираем необходимого пользователя и нажимаем кнопку “Удалить”

Подтверждаем удаление пользователя.

Настройки аутентификации пользователя. Требования к паролю.

Настройка требований к паролю задается в ini-файле приложения, после чего переменные необходимо объявить в файле настройки конфигурации приложения deploy.json.

В файле setup.ini:

```
auth.passwordLifeTime=90d # Максимальный срок действия пароля
auth.passwordMinPeriod=75d # Минимальный срок действия пароля
auth.passwordMinLength=8 # Минимальная длина пароля
auth.passwordJournalSize=5 # Число уникальных новых паролей пользователя до повторного использования
↪старого пароля
auth.tempBlockInterval=30m # Время до сброса счетчика блокировки
auth.attemptLimit=6 # Количество неудачных попыток входа в систему, приводящее к блокировке учетной
↪записи пользователя
auth.tempBlockPeriod=30m # Продолжительность блокировки учетной записи
auth.sessionLifeTime=4h # Время жизни авторизованной сессии, при отсутствии активности
```

Длительность везде задается в формате: [длительность][ед. изм]

Ед. измерения	Значение
y	Год
d	День
h	Час
m	Минута
s	Секунда

В файле deploy.json:

NB: Нужно обязательно, чтобы стояла настройка "parametrised": true, на уровне "global"

```
{
  "parametrised": true,
  "globals": {
    "plugins":{
      "auth": {
        "module": "lib/auth",
        "initMethod": "init",
        "initLevel": 2,
        "options": {
          "app": "ion://application",
          "logger": "ion://sysLog",
          "dataSource": "ion://Db",
          "acl": "ion://aclProvider",
          "passwordLifetime": "[[auth.passwordLifeTime]]", // максимальный срок действия пароля
          "passwordMinPeriod": "[[auth.passwordMinPeriod]]", // минимальный срок действия пароля
          "passwordMinLength": "[[auth.passwordMinLength]]", // минимальная длина пароля
          "passwordComplexity": { // требования к сложности пароля
            "upperLower": true, // требование использовать буквы в разном регистре
            "number": true, // требование использовать числа
            "special": true // требование использовать спецсимволы
          },
          "passwordJournalSize": "[[auth.passwordJournalSize]]", // ведение журнала паролей
          "tempBlockInterval": "[[auth.tempBlockInterval]]", // счетчик блокировки
          "attemptLimit": "[[auth.attemptLimit]]", // пороговое значение блокировки
          "tempBlockPeriod": "[[auth.tempBlockPeriod]]" // продолжительность блокировки
```

(continues on next page)

(continued from previous page)

```
      }
    }
  }
 }
}
```

Модуль администрирования (ionadmin) – используется для назначения прав, управления задачами по расписанию и другими административными задачами.

4.6.2 Настройка модуля ionadmin в файле config.json

Настройка записи медленных запросов

Настройка в виде модального окна на списке медленных запросов. В в файле config.json модуля ionadmin указывать источник:

```
"profiling": {
   "slowQuery": {
     "sources": [
        {
          "collection": "system.profile"
        }
     ]
   }
 }
```

Если свойство "sources" не задано или null, то будет браться из таблицы:

```
{
  "profiling": {
    "slowQuery": {
      "sources": null
    }
  }
}
```

Если задан пустой массив, то источников нет.

Настройка источников логов

Источники логов (может быть несколько) указываются в конфиге модуля:

```
"profiling": {
   "slowQuery": {
     "sources": [
       {
         "collection": "system.profile"
       },
       {
         "file": "D:/Temp/slow-query.txt"
       }
     ]
```

(continues on next page)

(continued from previous page)

```
  }
}
```

Сделанные выборки хранятся в отдельной таблице и не зависят от текущего состояния источников логов. Их можно дополнить редактированием. Например, комментариями или пометками, сообщающими решена проблема или нет.

Настройка резервирования БД

Настройка в ionmodule/config:

```
"backup": {
  "dir": "../ion-backups",
  "zlib": {
    "level": 1
  }
}
```

- dir содержит путь папки, в которой было запущено приложение ноды. По умолчанию "../ion-backups"
- zlib.level - уровень сжатия, также влияет на скорость создания архива. По умолчанию - значение 3
- К тому же, необходимо что бы утилита export.js с заданными параметрами корректно отрабатывала сама по себе.

4.6.3 Руководство пользователя по безопасности

Руководство пользователя по безопасности находится здесь.

4.7 Личный кабинет

Личный кабинет - используется для отображения различных "инструментальных" форм, т.е. в приложениях при заведении навигации (или форм-страниц), модуль их считывает и формирует на их основании навигацию и логику отображения страниц.

4.7.1 Настройка модуля

1. В приложении в некую директорию кладем html-страницы.
2. В deploy.json в настройках модуля ЛК прописываем эту директорию, а также опционально хеш-массив соответствия между именем файла и отображаемым названием.
3. ЛК считывает эти настройки и в мастер-шаблоне отображает меню навигации ЛК.
4. По клику на пункте меню в рабочей области окна ЛК отображается соответствующая разметка из html-файла.
5. Также в deploy.json в настройках модуля ЛК указывается страница-инструмент по умолчанию, если не указан берется первый считанный по порядку.

Также есть возможность структурирования меню посредством вложенности директорий.

4.8 Модуль Soap

NB: модуль Soap не поддерживает GET запросы к сервисам.

Отчасти это объясняется тем, что в теле запроса передаётся SOAP-запрос, а у GET-запроса тела нет (если можно так выразиться). По этой причине нужно отправлять POST запрос. Это можно сделать с помощью утилиты SOAP-UI (можно и в браузере, но в теле запроса нужно писать SOAP-запрос, который основан на WSDL и является достаточно громоздким).

4.8.1 Настройки структуры данных crud-сервиса

Опция types содержит в себе задаваемое в виде мапинга соответствие между классом (полным именем) и мапой публикуемых атрибутов этого класса. В мапе ключом является имя атрибута, а значением - либо строковый псевдоним, либо булево значение, указывающее включается атрибут в схему или нет. т.е. если указан псевдоним, то атрибут появляется в схеме под этим псевдонимом, во всех других случаях, кроме указания false, атрибут появляется под своим именем.

Настройка применяется при парсинге классов при формировании схемы данных сервиса, а также при парсинге входящих сообщений и генерации ответов. Путем замены normalize на функцию корректно приводящую данные к схеме.

Для работы с коллекциями и ссылками

Если значения коллекций и ссылок тоже надо распарсить как то иначе, объекты, которые находятся в этих свойствах можно тоже описать мапой в deploy.json вот по такому принципу:

```
"название_свойства": {
  "name": "новое_название(если нужно, поле необязательное)",
 "types": {
 //описания свойств
 }
}
```

Пример

```
"petitionExperts": {
        "module": "modules/soap/service/crud",
        "options": {
          "dataRepo": "ion://dataRepo",
          "metaRepo": "ion://metaRepo",
          "keyProvider": "ion://keyProvider",
          "namespace": "khv-gosekspertiza",
          "className": "petitionExpert",
          "types": {
            "petitionExperts@khv-gosekspertiza":{
              "property1":"new_property_name",
              "property2":true
```

(continues on next page)

(continued from previous page)

```
            }
          }
        }
      }
```

4.8.2 Настройка, убирающая из запроса системные атрибуты

```
let gosEkspRemPetNew = normalize(e.item, null, {skipSystemAttrs: true});
```

4.8.3 Авторизация по токену oauth2 в модуле SOAP

Аутентификация по парам логин-пароль и логин-токен по умолчанию применяется для всех сервисов. Для аутентификации soap-запросов добавляем в сообщение заголовок безопасности WSSecurity. Для аутентфиикации REST-сервисов добавляем стандартные заголовки HTTP-аутентифкации.

Настройка типа проверки - по паролю или токену (pwd/token) выполняется в deploy.json настройкой authMode в соответствующем модуле:

Пример

```
"soap": {
    "globals": {
      "authMode": {
        "petitionExperts": "none",
        "petitionEstimated": "none",
        "gosEkspContract": "none",
        "bankAccounts": "none",
        "resolution": "none"
      }
```

По умолчанию все сервисы аутентифицируются по паролю. Для генерации токена пользователя в админке реализована специальная форма. Настраиваем authMode для сервиса в token, переходим в админку, генерируем токен, используем его вместо пароля в заголовках.

4.9 Модуль Image-storage

Image-storage - модуль IONDV.Framework. Используется для предварительного просмотра картинок.

4.9.1 Описание и назначение модуля

Модуль представляет собой дополнительный компонент системы и используется при наличии в приложении атрибутов, позволяющих содержать в себе изображения.

Для подключения модуля необходимо обозначить его в зависимостях приложения. Для этого указываем модуль в файле package.json приложения, в разделе "dependencies":

```
"dependencies": {
   "image-storage": "git+https://github.com/iondv/image-storage.git"
}
```

с указанием ссылки на репозиторий модуля в github.com.

Далее нужно добавить на форму списка атрибут, изображение которого будет использовано в режиме предпросмотра. После чего, изображения, загруженные для объектов можно просматривать сразу на форме списка.

Назначение модуля image-storage

- Отображение превью картинок без перехода на форму объекта

4.10 Модуль REST

REST - модуль, обеспечивающий работу с данными приложения IONDV через REST API. Используется для создания веб-сервисов к различным видам данных, созданным, в том числе, только визуальным способом путем проектирования в IONDV. Studio.

4.10.1 Описание

Модуль IONDV. REST предназначен для:

- готовой “из коробки” CRUD модели для доступа к данным, управления бизнес-процессами, различными типами авторизации - в качестве бакенда для мобильных приложений, для SPA приложений (созданных на фреймворках Angular, Redux, Vue и т.д.) или приложений с разделенным фронт и бек офисом;
- быстрой разработки собственных веб-сервисов, путем их регистрации и написания кода на готовом API работы с данными для реализации микросервисной архитектуры;
- обеспечения интеграции приложений созданных на IONDV. Framework с другими системами по REST API.

Модуль IONDV. REST является оберткой для работы с данными через стандартные функции CRUD или подключает собственные сервисы приложения, в том числе использующие API ядра.

Подробнее:

Авторизация при запросах к сервисам

Получение токена

Получение токена возможно двумя способами: в консоли модуля ionadmin или через сервис token модуля rest.

Все сгенерированные токены хранятся в коллекции ion_user_tokens в базе данных приложения.

Получение постоянного токена через модуль ionadmin

Для получения токена через консоль администратора перейдите в пункт навигации “Ключи безопасности веб-сервисов” модуля ionadmin, например по адресу locahost:8888/ionadmin/token

На странице “Генератор токенов безопасности”:

- Введите имя пользоватля в поле “Имя пользователя”
- Укажите в поле “Тип учетной записи” значение “local”
- Нажмите кнопку “Сгенерировать токен”
- В поле “Токен” появится значение токена подобное 3a546090355317c287886c0e81dfd304fa5bda99, его и нужно использовать как значение заголовка auth-token.

Время жизни токена созданного по умолчанию 100 лет.

Получение временного токена через сервис rest/token

Вторым способом получения токена является использование веб-сервиса модуля rest - token. Получить токен можно через аутентифицированный запрос на адрес rest/token. Подробнее.

Прокси-клиент для доступа к функциям модуля без получения нового токена

Подключение клиента осуществляется в modules.registry.globals.di в deploy.json:

```
{
  "modules": {
    "registry": {
      "globals": {
        "di":{
          "apiGateWay": {
            "module": "modules/rest/client/GateWay",
            "options": {
              "log": "ion://sysLog",
              "base": "/registry-ajax-api",
              "clientId": "ext@system",
              "clientSecret": "ion-demo",
              "tokenPath": "/rest/token",
              "endPoint": "[[rest.endPoint]]",
              "definition": {
                "paths": {
                  "/rest/echo-token": {
                    "post": true,
                    "get": true
                  }
                }
              }
            }
          }
        }
```

Пример запроса к rest/echo-token через прокси-клиент:

```
curl -X POST --cookie-jar 1.txt -d username="demo@local" -d password="ion-demo" http://localhost:8888/auth
curl -X GET --cookie 1.txt https://dnt.iondv.com/registry-ajax-api/rest/echo-token
```

пример запроса в dnt: test/modules/rest/gateway.spec.js

```
/Checking rest-api proxy
```

Авторизация для доступа к сервисам может быть осуществлена следующими способами:

- Без авторизации
- По учетной записи
- По токену
- Oauth2

Сервисы без аутентификации

Для реализации работы сервиса без аутентификации, необходимо задать для него значение none в настройке authMode в deploy.json

```
{
  "modules": {
    "rest": {
      "globals": {
        "authMode": {
          "echo": "none"
```

Запрос к сервису не будет требовать аутентификации, пример запроса curl https://dnt.iondv.com/rest/echo

Пример запроса к сервису без аутентификации в dnt: test/modules/rest/echo.spec.js

```
/Checking echo-pwd service/# Requesting echo-pwd GET/check if the request can be made using the headers auth
```

Сервисы со стандартрным механизмом авторизации по учетной записи

Все сервисы по умолчанию используют стандартный механизм авторизации, подразумевающий передачу учетных данных в заголовке:

- путем авторизации через Basic Auth, пример

  ```
  curl -u demo@local:ion-demo https://dnt.iondv.com/rest/simple
  ```

  пример запроса с авторизацией Basic Auth в develop-and-test (dnt): test/modules/rest/echopwd.spec.js

  ```
  /Checking echo-pwd service/# Requesting echo-pwd GET/check if the request can be made using the␣
  ↪basicAuth
  ```

- путем передачи учетных данных в заголовках запроса

  ```
  curl -H "auth-user: demo" -H "auth-pwd: ion-demo" -H "auth-user-type: local" https://dnt.iondv.com/rest/
  ↪simple
  ```

  или

  ```
  curl -H "auth-user: demo@local" -H "auth-pwd: ion-demo" https://dnt.iondv.com/rest/simple
  ```

пример запроса с авторизацией учетными данными в заголовке в dnt: test/modules/rest/echopwd.spec.js

```
/Checking echo-pwd service/# Requesting echo-pwd GET/check if the request can be made using the␣
↪headers auth
```

Сервисы с аутентификацией через токен

Аутентификация по токену используется для исключения постоянной передачи учетной записи в запросах. Токены ограничены по времени жизни.

Для реализации работы сервиса с аутентификацией через токен, необходимо задать для него значение token в настройке authMode в deploy.json

```
{
  "modules": {
    "rest": {
      "globals": {
        "authMode": {
          "echo-token": "token"
```

Аутентификация через токен осуществляется путем отправки значения токена в заголовке запроса auth-token

```
curl -H "auth-token: c369a361db9742e9a9ae8e9fe55950a571493812" http://dnt.iondv.com/rest/echo-token
```

пример запроса с авторизацией через токен в dnt: test/modules/rest/token.spec.js

```
/Checking token service/# basicAuth authorization with admin rights/# check if the generated token is valid␣
↪(basicAuth) (using echo-token)
```

подробнее о получении токена: Получение токена

Прокси-клиент для доступа к функциям модуля без получения нового токена.

Сервисы с аутентификацией методом OAuth2

Для реализации работы сервиса с аутентификацией oauth2, необходимо предварительно подключить в deploy.json плагин вида

```
"oauth": {
        "module": "lib/oAuthAdapter",
        "options": {
          "auth": "ion://auth",
          "dataSource": "ion://Db"
        }
      }
```

затем можно задать для сервиса значение oauth в настройке auth_mode:

```
{
  "modules": {
    "rest": {
      "globals": {
```

(continues on next page)

(continued from previous page)

```
    "authMode": {
      "echo-oauth": "oauth"
```

спецификация oauth2 доступна по ссылке: https://oauth2-server.readthedocs.io/en/latest/index.html

Этот тип авторизации используется для предоставления третьей стороне ограниченного доступа к ресурсам пользователя без необходимости предоставлять логин и пароль. Запросы для получения доступа производятся в следующем порядке:

1. Со стороны пользователя получаем cookie с id сессии:

   ```
   curl -X POST --cookie-jar 1.txt -d username="demo@local" -d password="ion-demo" http://dnt.iondv.com/
   ↪auth
   ```

2. Используя авторизованную сессию разрешаем клиенту ext@system запросы от нашего имени:

   ```
   curl -X POST --cookie ./1.txt "http://dnt.iondv.com/oauth2/grant?client_id=ext@system&response_
   ↪type=code&state=123"
   ```

   В ответе будет содержаться параметр code.

3. Теперь используя code можно получить токен:

   ```
   curl -X POST -d grant_type="authorization_code" -d code="<code>" -H "Authorization:Basic
   ↪ZXh0QHN5c3RlbTppb24tZGVtbw==" http://dnt.iondv.com/oauth2/token
   ```

   в заголовке Authorization нужно ввести Basic <client_secret> код клиента. В ответе будет получен access_token.

4. Для запросов от лица пользователя в сервисах с авторизацией oauth2 теперь можно авторизоваться используя access_token:

   ```
   curl -X POST -H "Authorization:Bearer <access_token>" http://dnt.iondv.com/rest/echo-oauth
   ```

пример запроса к сервису с авторизацией oauth2 в dnt: test/modules/rest/echooauth.spec.js

```
/Checking echo-oauth service
```

Сервисы REST

Встроенный сервис Acceptor

Сервис acceptor предназначен для массового сохранения объектов разных классов.

Доступен по адресу <адрес сервера>/rest/acceptor.

Для работы с сервисом требуется его регистрация в файле конфигураций приложений deploy.json. При этом для сервиса обязательно должны быть указаны репозитории dataRepo и metaRepo в options. Например:

```
{
  "modules": {
    "rest": {
      "globals": {
        "di": {
          "acceptor": {
```

(continues on next page)

(continued from previous page)

```
        "module": "modules/rest/lib/impl/acceptor",
        "options": {
          "dataRepo": "ion://dataRepo",
          "metaRepo": "ion://metaRepo"
        }
      }
```

Авторизация осуществляется через все основные типы доступа.

Сервис работает по методу POST, объекты передаются в виде массива объектов в формате JSON в теле запроса с обязательным указанием в заголовке содержания json Content-Type:application/json. Автосоздаваемые поля указывать не обязательно.

В заголовке (header) в свойстве ion-converter может быть передано имя конвертора, который нужно использовать при обработке данных - как запроса, так и ответа. При этом сам конвертор данных должен быть зарегистрирован в options сервиса. Если обработчик не указан, используется обработчик по умолчанию.

В данных объекта обязательно указываются:

- _id - идентификатор объекта по ключевому полю
- _class - класс объекта с неймспейсом
- _classVer - версия класса

Остальные значения должны соответствовать свойствам класса, включая соответствие типов данных. Пример:

```
curl -X POST -u demo@local:ion-demo \
  -H "Content-Type:application/json" \
  -d '[{"_class": "class_string@develop-and-test", "__classVer": null,"id": "10101010-5583-11e6-aef7-
↪cf50314f026b",\
      "string_text": "Example10", "string_miltilinetext": "Example10", "string_formattext": "Example10"}]' \
  https://dnt.iondv.com/rest/acceptor
```

пример запроса на создание объектов к сервису acceptor в dnt: test/modules/rest/acceptor.spec.js

```
/Checking acceptor service/# basicAuth authorization with admin rights, POSTing strings/# result of creation␣
↪of objects
```

Метод возвращает код 200 и массив сохраненных объектов.

```
[
  {
    "id": "10101010-5583-11e6-aef7-cf50314f026b",
    "_class": "class_string@develop-and-test",
    "_classVer": "",
    "string_formattext": "Example10",
    "string_miltilinetext": "Example10",
    "string_text": "Example10",
    "_id": "10101010-5583-11e6-aef7-cf50314f026b"
  }
]
```

В случае ошибки код ответа будет 400, а текст ответа содержать

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>Bad Request</pre>
</body>
</html>
```

Встроенный сервис Token

Сервис token предназначен для выдачи токена пользователю, прошедшему аутентификацию, для его дальнейшего использования в сервисах, осуществляющих аутентификацию по токену.

Доступен по адресу <адрес сервера>/rest/token.

Сервис не требует регистрации в deploy.json. Сервис обеспечивает выдачу токена для авторизованного пользователя, если он имеет права use для ресурса ws:::gen-ws-token или имеет права администратора.

В ответ на запрос, возвращается токен вида e444c69894d2087696e0a6c6914788f67ebcf6ee. Время жизни токена по умолчанию 100 лет.

Пример запроса через аутентификацию типа Basic Auth

```
curl -u demo@local:ion-demo https://dnt.iondv.com/rest/token
```

Пример запроса с аутентификацией через параметры в заголовке

```
curl -H "auth-user: demo@local" -H "auth-pwd: ion-demo" -H "auth-user-type: local" https://dnt.iondv.com/rest/
↪token
```

Примеры запросов к сервису token в dnt: test/modules/rest/token.spec.js

```
/Checking token service/# basicAuth authorization with admin rights
/Checking token service/# authorization with admin rights using header parameters
```

Права на ресурс

Добавить ресурс возможности генерации токенов для роли можно из комадной строки node bin/acl.js --role restGrp --p USE --res ws:::gen-ws-token (где restGrp - название существующей группы)

Второй способ добавления прав на ресурс - использование консоли администратора модуля ionadmin, например, по адресу locahost:8888/ionadmin/:

- Выберите пункт навигации “Безопасность”,
- Выберите подпункт навигации “Роли”
- Выберите существующую роль и нажмите кнопку “редактировать” или “создать новую роль”.
- В поле “Права доступа” роли выберите вкладку “Services”
- Раскройте список прав для ресурса “Генерация токенов безопасности посредством веб-сервисов (ws:::gen-ws-token)”
- Выбрать пункт “Использование” и нажмите кнопку “Сохранить”

Встроенный сервис crud

Сервис crud реализует REST API по модели основных операций CRUD (create, read, update, delete).

Доступен по адресу <адрес сервера>/rest/crud.

Сервис требует регистрации в deploy.json приложения и обязательного указания источника данных dataRepo в options сервиса, а также источника авторизации auth для доступа к данным пользователя. Целесообразно указывать в качестве репозитория данных - репозиторий с полной обработкой безопасности, чтобы отрабатывать доступ к объектам с учетом динамической безопаности. Например:

```
{
  "modules": {
    "rest": {
      "globals": {
        "di": {
          "crud": {
            "module": "modules/rest/lib/impl/crud",
            "options": {
              "auth": "ion://auth",
              "dataRepo": "ion://securedDataRepo"
            }
          }
```

Аутентификация осуществляется через все основные типы доступа.

Пример:

```
curl -X POST -u demo@local:ion-demo https://dnt.iondv.com/rest/crud
```

Пример запроса к сервису crud без параметров в dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/# check if the response for null parameters is valid
```

По умолчанию, без правильных параметров - код ответа сервера 404 об ошибке

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>Cannot POST /rest/crud</pre>
</body>
</html>
```

Информация по взаимодействию с crud через основные методы:

Создание объекта: метод POST

Создание объекта осуществляется методом POST, при этом указывается код класса с нейспейсом, например rest/crud/class_string@develop-and-test. Сам объект передается в теле запроса в формате json с обязательным указанием в заголовке типа содержания json Content-Type:application/json. Автосоздаваемые поля указывать не обязательно.

Пример:

```
curl -X POST -u demo@local:ion-demo \
  -H "Content-Type:application/json" \
  -d '{"string_text": "Example3", "string_miltilinetext": "Example3", "string_formattext": "Example3"}' \
  https://dnt.iondv.com/rest/crud/class_string@develop-and-test/
```

Пример запроса к сервису crud для создания объекта dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/POST/# creating an object (POST)
```

В ответ будет возвращён созданный объект, в котором будут заполнены все автосозданные поля и указан код ответа 200.

```
{
   "_creator": "admin@local",
   "_id": "10c77900-b96e-11e9-a7ce-314f02bd4197",
   "__string": "10c77900-b96e-11e9-a7ce-314f02bd4197",
   "__class": "class_string@develop-and-test",
   "__classTitle": "Class \"String [0]\"",
   "id": "10c77900-b96e-11e9-a7ce-314f02bd4197",
   "string_text": "Example3",
   "string_miltilinetext": "Example3",
   "string_formattext": "Example3"
}
```

В случае ошибки код ответа будет 400, а текст ответа содержать

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>Bad Request</pre>
</body>
</html>
```

Получение объекта или списка объектов: метод GET

Получение объекта

Получение объекта осуществляется методом GET, при этом указывается код класса с нейспейсом и значение ключа объекта, например rest/crud/class_string@develop-and-test/66dbb3d0-5583-11e6-aef7-cf50314f026b

```
curl -X GET -u demo@local:ion-demo https://dnt.iondv.com/rest/crud/class_string@develop-and-test/66dbb3d0-
↪5583-11e6-aef7-cf50314f026b
```

Пример запроса к сервису crud для получения объекта в dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/GET/# getting an object (GET)
```

При этом дополнительно в query может быть задан параметр _eager содержащий список свойств класса, разделенных символом | для которых необходимо осуществить жадную загрузку данных (ссылки или коллекции). Например

```
curl -X GET -u demo@local:ion-demo https://dnt.iondv.com/rest/crud/class_string@develop-and-test/66dbb3d0-
↪5583-11e6-aef7-cf50314f026b?_eager=string_text
```

Пример запроса к сервису crud для получения объекта с жадной загрузкой свойства “table” в dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/GET/# getting an object with eager loading of the "table" property (GET)
```

Если объект существует - возвращает код ответа 200 и сам объект в формате json, если объект не найден 404, если нет прав 403.


```
{
   "_id": "66dbb3d0-5583-11e6-aef7-cf50314f026b",
   "__string": "Example of the \"String [0]\" type in the \"Text [1]\" view",
   "__class": "class_string@develop-and-test",
   "__classTitle": "Class \"String [0]\"",
   "id": "66dbb3d0-5583-11e6-aef7-cf50314f026b",
   "string_text": "Example of the \"String [0]\" type in the \"Text [1]\" view",
   "string_miltilinetext": "Example of the \"String [0]\"\r\n in the Multiline text [7] view",
   "string_formattext": "Example of the \r\n \"String [0]\" type \r\n in the \r\nFormatted text [7] view"
}
```


Получение списка объектов

Запрос списка объектов осуществляется методом GET, при этом указывается код класса и нейспейс, например rest/crud/class_string@develop-and-test/

```
curl -X GET -u demo@local:ion-demo https://dnt.iondv.com/rest/crud/class_string@develop-and-test/
```

Пример запроса к сервису crud для получения списка объектов в dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/GET/# getting a list of text objects
```

В ответ сервис выдает JSON Объект со смещением 0 и кол-вом 5ть записей и статусом 200, если такого класса нет возвращает код 404.


```
[{"_creator":"admin@local",
"_id":"4567a830-b8ea-11e9-9cdf-7bd384cbb7a5",
"__string":"example1",
"__class":"class_string@develop-and-test",
"__classTitle":"Class \"String [0]\"",
"id":"4567a830-b8ea-11e9-9cdf-7bd384cbb7a5",
"string_text":"example1",
"string_miltilinetext":"example1",
"string_formattext":"<p>example1</p>"},
{"_id":"66dbb3d0-5583-11e6-aef7-cf50314f026b",
"__string":"Example of the \"String [0]\" type in the \"Text [1]\" view",
"__class":"class_string@develop-and-test",
"__classTitle":"Class \"String [0]\"",
"id":"66dbb3d0-5583-11e6-aef7-cf50314f026b",
"string_text":"Example of the \"String [0]\" type in the \"Text [1]\" view",
"string_miltilinetext":"Example of the \"String [0]\"\r\n in the Multiline text [7] view",
"string_formattext":"Example of the \r\n \"String [0]\" type \r\n in the \r\nFormatted text [7] view"}]
```


Запрос может быть осуществлен со следущими query параметрами:

- _offset - смещение выборки, по умолчанию 0
- _count - кол-во значение в выборке, по умолчанию 5
- _eager - список свойств класса, разделенных символом | для которых необходимо осуществить жадную загрузку данных.
- [name of property] - все параметры запроса, кроме начинающихся на _ считаются именами атрибутов класса, а их значения задаются в качестве фильтров.

Примеры:

1. Запрос списка объектов класса со смещением 1 и кол-вом 2

```
curl -X GET -u demo@local:ion-demo https://dnt.iondv.com/rest/crud/class_string@develop-and-test/?_
↪offset=1&_count=2
```

2. Запрос списка объектов, у которы свойство string_text имеет значение example1

```
curl -X GET -u demo@local:ion-demo https://dnt.iondv.com/rest/crud/class_string@develop-and-test/?
↪string_text=example1
```

3. Запрос списка объектов, у которы свойство string_text имеет значение example1, со смещением 1 и кол-вом 2

```
curl -X GET -u demo@local:ion-demo https://dnt.iondv.com/rest/crud/class_string@develop-and-test/?
↪string_text=example1&_offset=1&_count=2
```

Пример запроса к сервису crud для получения списка объектов с различными параметрами сдвига и фильтрации в dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/GET/# getting a list of text objects, with an offset of 1 and a count of 2
/Checking crud service/GET/# getting a list of text objects containing a specific string
```

Получение объектов методом SEARCH

Обращение к HTTP-методу SEARCH в сервисе CRUD осуществляется аналогично методу GET. Исключением является то, что в теле сообщения можно в формате .json задать условия фильтрации, сортировки и жадной загрузки в нотации метода GetList репозитория данных:

```
{
  "filter": {
    "eq": ["attr3", "etalon"]
  },
  "forceEnrichment": [["attr1", "attr2"], ["col1"]]
  "sort": {
    "attr4": -1
  }
}
```

```
curl -X GET -u admin@local:ION-admin http://modws-26.develop-and-test.kube.local/rest/crud/class_
↪string@develop-and-test/
```

Проверка наличия объекта: метод HEAD

Проверка наличия объекта осуществляется методом HEAD, при этом указывается код класса с нейспейсом и значение ключа объекта, например rest/crud/class_string@develop-and-test/66dbb3d0-5583-11e6-aef7-cf50314f026b

```
curl -X HEAD -u demo@local:ion-demo https://dnt.iondv.com/rest/crud/class_string@develop-and-test/66dbb3d0-
↪5583-11e6-aef7-cf50314f026b
```

Пример запроса к сервису crud для проверки наличия объекта в dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/HEAD/# checking if an object is present (HEAD)
```

Если объект существует - возвращает код ответа 200, если объект не найден 404, если нет прав 403.

Обновление объекта: методы PATCH и PUT

Обновление объекта осуществляется методом PATCH или PUT, при этом указывается код класса с нейспейсом и значение ключа объекта, например rest/crud/class_string@develop-and-test/66dbb3d0-5583-11e6-aef7-cf50314f026b. Сам объект передается в теле запроса в формате json с обязательным указанием в заголовке типа содержания json Content-Type:application/json.

Пример:

```
curl -X PATCH -u demo@local:demo-ion -H "Content-Type:application/json" -d '{"string_text": "NEW Example
↪", "string_miltilinetext": "NEW Example", "string_formattext": "NEW Example"}' https://dnt.iondv.com/
↪rest/crud/class_string@develop-and-test/66dbb3d0-5583-11e6-aef7-cf50314f026b
# Или эквивалентно
curl -X PUT -u demo@local:demo-ion -H "Content-Type:application/json" -d '{"string_text": "NEW Example",
↪"string_miltilinetext": "NEW Example", "string_formattext": "NEW Example"}' https://dnt.iondv.com/rest/
↪crud/class_string@develop-and-test/66dbb3d0-5583-11e6-aef7-cf50314f026b
```

Пример запроса к сервису crud для обновления объекта в dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/PATCH/# updating an object (PATCH)
```

Если объект существует - возвращает код ответа 200 и сам объект в формате json, если объект не найден код 404, при ошибке обработки код 500, если нет прав 403.

Пример объекта.

```
{
    "_editor": "admin@local",
    "_id": "66dbb3d0-5583-11e6-aef7-cf50314f026b",
    "__string": "NEW Example",
    "__class": "class_string@develop-and-test",
    "__classTitle": "Class \"String [0]\"",
    "id": "66dbb3d0-5583-11e6-aef7-cf50314f026b",
    "string_text": "NEW Example",
    "string_miltilinetext": "NEW Example",
    "string_formattext": "NEW Example"
}
```

Удаление объекта: метод DELETE

Удаление объекта осуществляется методом DELETE, при этом указывается код класса с нейспейсом и значение ключа объекта, например rest/crud/class_string@develop-and-test/66dbb3d0-5583-11e6-aef7-cf50314f026b.

```
curl -X DELETE -u demo@local:demo-ion https://dnt.iondv.com/rest/crud/class_string@develop-and-test/
↪66dbb3d0-5583-11e6-aef7-cf50314f026b
```

Пример запроса к сервису crud на удаление объектов в dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/DELETE/# deleting an object (DELETE)
```

В случае успеха - сервис возвращает код ответа 200, в случае если объект не найден 404.

Отправка запросов с файлами в CRUD сервисе

При запросе к CRUD методами POST, PATCH и PUT, в теле запроса можно передать файлы.

Отправка и прием файлов осуществляется двумя способами:

- данные отправляются в формате json, тогда контент файла передается как строка в формате Base64 в соответствующем поле класса метаданных.
- данные отправляются как FormData (application/x-www-form-urlencoded), тогда файлы передаются как компонент multipart.

Корректный прием файловых атрибутов в случае отправки таких запросов осуществляется методами POST, PUT и PATCH в CRUD сервисе.

Так же возможна передача ссылок и коллекций по примеру, описанному для soap модуля.

Примеры POST запросов с файлами к CRUD в dnt: test/modules/rest/crud.spec.js

```
/checking crud service/# sending a file with multipart body request (POST)
/checking crud service/# sending a file with json body request (POST)
```

Сервис публикации метаданных Meta

Meta meta - встроенный сервис в rest модуле, который предоставляет доступ к интерфейсу метарепозитория в формате веб-сервиса.

Сервис требует подключения в deploy.json и обязательного указания options.dataRepo и options.metaRepo, пример:

```
"meta": {
        "module": "modules/rest/lib/impl/meta",
        "options": {
          "dataRepo": "ion://dataRepo",
          "metaRepo": "ion://metaRepo"
        }
}
```

Поддерживаются все типы авторизации, по умолчанию - авторизация учетными данными.

Сервис предоставляет доступ к следующим GET запросам:

Получение информации о классе метаданных: getMeta

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/getMeta/<имя класса>, где имя класса указывается с неймспейсом.

В запросе можно указать дополнительные параметры:

- version
- namespace

Пример запроса:

```
https://localhost:8888/rest/meta/getMeta/class_text@develop-and-test
```

будет запрошена информация по классу class_text в неймспейсе develop-and-test, пример ответа:

```
{ namespace: 'develop-and-test',
isStruct: false,
metaVersion: '2.0.7',
key: [ 'id' ],
semantic: '',
name: 'class_text',
version: '',
caption: 'Class "Text [1]"',
ancestor: null,
container: null,
creationTracker: '',
changeTracker: '',
history: 0,
journaling: false,
compositeIndexes: null,
properties:
[ { orderNumber: 10,
   name: 'id',
   caption: 'Identifier',
   type: 12,
   size: 24,
   decimals: 0,
   allowedFileTypes: null,
   maxFileCount: 0,
   nullable: false,
   readonly: false,
   indexed: true,
   unique: true,
   autoassigned: true,
   hint: null,
   defaultValue: null,
   refClass: '',
   itemsClass: '',
   backRef: '',
   backColl: '',
   binding: '',
   semantic: null,
   selConditions: null,
   selSorting: [],
```

(continues on next page)

(continued from previous page)

```
  selectionProvider: null,
  indexSearch: false,
  eagerLoading: false,
  formula: null },
{ orderNumber: 20,
  name: 'text_text',
  caption: 'Text [1]',
  type: 1,
  size: null,
  decimals: 0,
  allowedFileTypes: null,
  maxFileCount: 0,
  nullable: true,
  readonly: false,
  indexed: true,
  unique: false,
  autoassigned: false,
  hint: null,
  defaultValue: null,
  refClass: '',
  itemsClass: '',
  backRef: '',
  backColl: '',
  binding: '',
  semantic: null,
  selConditions: null,
  selSorting: [],
  selectionProvider: null,
  indexSearch: false,
  eagerLoading: false,
  formula: null },
{ orderNumber: 30,
  name: 'text_multilinetext',
  caption: 'Multiline text [7]',
  type: 1,
  size: null,
  decimals: 0,
  allowedFileTypes: null,
  maxFileCount: 0,
  nullable: true,
  readonly: false,
  indexed: true,
  unique: false,
  autoassigned: false,
  hint: null,
  defaultValue: null,
  refClass: '',
  itemsClass: '',
  backRef: '',
  backColl: '',
  binding: '',
  semantic: null,
  selConditions: null,
  selSorting: [],
  selectionProvider: null,
  indexSearch: false,
```

(continues on next page)

(continued from previous page)

```
  eagerLoading: false,
  formula: null },
{ orderNumber: 40,
  name: 'text_formattext',
  caption: 'Formatted text [8]',
  type: 1,
  size: null,
  decimals: 0,
  allowedFileTypes: null,
  maxFileCount: 0,
  nullable: true,
  readonly: false,
  indexed: true,
  unique: false,
  autoassigned: false,
  hint: null,
  defaultValue: null,
  refClass: '',
  itemsClass: '',
  backRef: '',
  backColl: '',
  binding: '',
  semantic: null,
  selConditions: null,
  selSorting: [],
  selectionProvider: null,
  indexSearch: false,
  eagerLoading: false,
  formula: null } ] }
```

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# accessing info about metadata class: getMeta
```

Получение списка всех классов метаданных: listMeta

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/listMeta.

В запросе можно указать дополнительные параметры:

- ancestor
- version
- direct
- namespace

Пример запроса:

```
https://localhost:8888/rest/meta/listMeta
```

будет запрошена информация по всем классам из метарепозитория, указанного в настройках сервиса meta в deploy.json.

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# accessing the list of metadata classes: listMeta
```

Получение информации о классе-предке: ancestor

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/ancestor/<имя класса>, где имя класса указывается с неймспейсом.

В запросе можно указать дополнительные параметры:

- version
- namespace

Пример запроса:

```
https://localhost:8888/rest/meta/getMeta/event@develop-and-test
```

будет запрошена информация по классу-предку класса event в неймспейсе develop-and-test - basicObj.

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# accessing info about class ancestor
```

Получение информации о свойствах объектов класса: propertyMetas

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/propertyMetas/<имя класса>, где имя класса указывается с неймспейсом.

В запросе можно указать дополнительные параметры:

- version
- namespace

Пример запроса:

```
https://localhost:8888/rest/meta/propertyMetas/class_text@develop-and-test
```

будет запрошена информация о свойствах объектов класса class_text в неймспейсе develop-and-test, например:

```
[ { orderNumber: 10,
name: 'id',
caption: 'Identifier',
type: 12,
size: 24,
decimals: 0,
allowedFileTypes: null,
maxFileCount: 0,
nullable: false,
readonly: false,
indexed: true,
unique: true,
autoassigned: true,
hint: null,
defaultValue: null,
```

(continues on next page)

(continued from previous page)


```
  refClass: '',
  itemsClass: '',
  backRef: '',
  backColl: '',
  binding: '',
  semantic: null,
  selConditions: null,
  selSorting: [],
  selectionProvider: null,
  indexSearch: false,
  eagerLoading: false,
  formula: null,
  definitionClass: 'class_text@develop-and-test',
  mode: 0 },
{ orderNumber: 20,
  name: 'text_text',
  caption: 'Text [1]',
  type: 1,
  size: null,
  decimals: 0,
  allowedFileTypes: null,
  maxFileCount: 0,
  nullable: true,
  readonly: false,
  indexed: true,
  unique: false,
  autoassigned: false,
  hint: null,
  defaultValue: null,
  refClass: '',
  itemsClass: '',
  backRef: '',
  backColl: '',
  binding: '',
  semantic: null,
  selConditions: null,
  selSorting: [],
  selectionProvider: null,
  indexSearch: false,
  eagerLoading: false,
  formula: null,
  definitionClass: 'class_text@develop-and-test',
  mode: 0 },
{ orderNumber: 30,
  name: 'text_multilinetext',
  caption: 'Multiline text [7]',
  type: 1,
  size: null,
  decimals: 0,
  allowedFileTypes: null,
  maxFileCount: 0,
  nullable: true,
  readonly: false,
  indexed: true,
  unique: false,
  autoassigned: false,
```


(continues on next page)

(continued from previous page)

```
  hint: null,
  defaultValue: null,
  refClass: '',
  itemsClass: '',
  backRef: '',
  backColl: '',
  binding: '',
  semantic: null,
  selConditions: null,
  selSorting: [],
  selectionProvider: null,
  indexSearch: false,
  eagerLoading: false,
  formula: null,
  definitionClass: 'class_text@develop-and-test',
  mode: 0 },
{ orderNumber: 40,
  name: 'text_formattext',
  caption: 'Formatted text [8]',
  type: 1,
  size: null,
  decimals: 0,
  allowedFileTypes: null,
  maxFileCount: 0,
  nullable: true,
  readonly: false,
  indexed: true,
  unique: false,
  autoassigned: false,
  hint: null,
  defaultValue: null,
  refClass: '',
  itemsClass: '',
  backRef: '',
  backColl: '',
  binding: '',
  semantic: null,
  selConditions: null,
  selSorting: [],
  selectionProvider: null,
  indexSearch: false,
  eagerLoading: false,
  formula: null,
  definitionClass: 'class_text@develop-and-test',
  mode: 0 } ]
```

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# accessing info about meta object properties: propertyMetas
```

Получение списка секций навигации: getNavigationSections

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/getNavigationSections.

В запросе можно указать дополнительные параметры:

- namespace

Пример запроса:

```
https://localhost:8888/rest/meta/getNavigationSections
```

будет запрошен список всех объектов секций навигации из метарепозитория, указанного в настройках сервиса meta в deploy.json.

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# accessing the list of navigation sections: getNavigationSections
```

Получение информации о секции навигации: getNavigationSection

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/getNavigationSection/<код секции>, где код секции указывается в формате namespace@name.

В запросе можно указать дополнительные параметры:

- namespace

Пример запроса:

```
https://localhost:8888/rest/meta/getNavigationSection/develop-and-test@simpleTypes
```

будет запрошен объект секции навигации simpleTypes в неймспейсе develop-and-test, пример ответа:

```
{ caption: 'Simple types',
  code: 'simpTypes',
  name: 'simpleTypes',
  orderNumber: 20,
  mode: 0,
  tags: null,
  metaVersion: '2.0.7',
  itemType: 'section',
  namespace: 'develop-and-test',
  nodes:
  { class_boolean:
    { namespace: 'develop-and-test',
      code: 'class_boolean',
      orderNumber: 0,
      type: 0,
      caption: 'Class "Boolean [10]"',
      classname: null,
      container: null,
      collection: null,
      url: null,
      hint: null,
      conditions: null,
      sorting: [],
      pathChains: [],
      title: '',
      metaVersion: '2.0.0',
      itemType: 'node',
      section: 'develop-and-test@simpleTypes',
      children: [Array] },
   class_custom:
```

(continues on next page)

(continued from previous page)

```
    { namespace: 'develop-and-test',
      code: 'class_custom',
      orderNumber: 0,
      type: 1,
      caption: 'Class "User type [17]"',
      classname: 'class_custom@develop-and-test',
      container: null,
      collection: null,
      url: null,
      hint: null,
      conditions: null,
      sorting: [],
      pathChains: [],
      title: '',
      metaVersion: '2.0.0',
      itemType: 'node',
      section: 'develop-and-test@simpleTypes',
      children: [] },
   class_datetime:
    { namespace: 'develop-and-test',
      code: 'class_datetime',
      orderNumber: 0,
      type: 1,
      caption: 'Class "Date/time [9]"',
      classname: 'class_datetime@develop-and-test',
      container: null,
      collection: null,
      url: null,
      hint: null,
      conditions: null,
      sorting: [],
      pathChains: [],
      title: '',
      metaVersion: '2.0.0',
      itemType: 'node',
      section: 'develop-and-test@simpleTypes',
      children: [] },
   class_decimal:
    { code: 'class_decimal',
      orderNumber: 0,
      type: 1,
      caption: 'Class "Decimal [8]"',
      classname: 'class_decimal@develop-and-test',
      container: null,
      collection: null,
      url: null,
      hint: null,
      conditions: null,
      sorting: [],
      pathChains: [],
      metaVersion: '2.0.7',
      itemType: 'node',
      section: 'develop-and-test@simpleTypes',
      namespace: 'develop-and-test',
      children: [] }
  }
}
```

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access info about a navigation section: getNavigationSection
```

Получение информации об узле навигации: getNode

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/getNode/<код навигации>, где код навигации указывается в формате namespace@code.

В запросе можно указать дополнительные параметры:

- namespace

Пример запроса:

```
https://localhost:8888/rest/meta/getNode/develop-and-test@semantic
```

будет запрошен объект узла навигации semantic в неймспейсе develop-and-test, пример ответа:

```
{ code: 'semantic',
orderNumber: 0,
type: 0,
caption: 'Display semantics [semantic]',
classname: null,
container: null,
collection: null,
url: null,
hint: null,
conditions: null,
sorting: [],
pathChains: [],
metaVersion: '2.0.61',
itemType: 'node',
section: 'develop-and-test@classProperties',
namespace: 'develop-and-test',
children:
[ { code: 'semantic.semErrCatalog',
   orderNumber: 0,
   type: 1,
   caption:
    'Catalog for checking semantics (string1| |string2| |string3| |date| |integer)',
   classname: 'semErrCatalog@develop-and-test',
   container: null,
   collection: null,
   url: null,
   hint: null,
   conditions: null,
   sorting: [],
   pathChains: [],
   metaVersion: '2.0.61',
   itemType: 'node',
   section: 'develop-and-test@classProperties',
   namespace: 'develop-and-test',
   children: [] },
 { code: 'semantic.semErrClass',
   orderNumber: 0,
   type: 1,
```

(continues on next page)

(continued from previous page)

```
  caption: 'Attribute semantics is taken from reference class',
  classname: 'semErrClass@develop-and-test',
  container: null,
  collection: null,
  url: null,
  hint: null,
  conditions: null,
  sorting: [],
  pathChains: [],
  metaVersion: '2.0.61',
  itemType: 'node',
  section: 'develop-and-test@classProperties',
  namespace: 'develop-and-test',
  children: [] }]
}
```

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access info about a navigation node: getNode
```

Получение списка узлов навигации в секции: getNodes

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/getNodes/<код секции>, где код секции указывается в формате namespace@name.

В запросе можно указать дополнительные параметры:

- section
- parent
- namespace

Пример запроса:

```
https://localhost:8888/rest/meta/getNodes/develop-and-test@simpleTypes
```

будет запрошен список узлов навигации в секции simpleTypes в неймспейсе develop-and-test.

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access the list of navigation nodes in a section: getNodes
```

Получение информации о списочной форме представления объектов класса: getListViewModel

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/getListViewModel/<имя класса>, где имя класса указывается с неймспейсом.

В запросе можно указать дополнительные параметры:

- node
- namespace
- version

Пример запроса:

```
https://localhost:8888/rest/meta/getListViewModel/class_text@develop-and-test
```

будет запрошен объект представления объектов класса class_text в списочной форме в неймспейсе develop-and-test, пример ответа:


```
 { columns:
[ { sorted: true,
    caption: 'Identifier',
    type: 1,
    property: 'id',
    size: 2,
    maskName: null,
    mask: null,
    mode: null,
    fields: [],
    hierarchyAttributes: null,
    columns: [],
    actions: null,
    commands: [],
    orderNumber: 10,
    required: false,
    visibility: null,
    enablement: null,
    obligation: null,
    readonly: true,
    selectionPaginated: true,
    validators: null,
    hint: '',
    historyDisplayMode: 0,
    tags: null },
  { sorted: true,
    caption: 'Text [1]',
    type: 1,
    property: 'text_text',
    size: 2,
    maskName: null,
    mask: null,
    mode: null,
    fields: [],
    hierarchyAttributes: null,
    columns: [],
    actions: null,
    commands: [],
    orderNumber: 20,
    required: false,
    visibility: null,
    enablement: null,
    obligation: null,
    readonly: false,
    selectionPaginated: true,
    validators: null,
    hint: '',
    historyDisplayMode: 0,
    tags: null },
  { sorted: true,
    caption: 'Multiline text [7]',
    type: 7,
```


(continues on next page)

(continued from previous page)


```
    property: 'text_multilinetext',
    size: 2,
    maskName: null,
    mask: null,
    mode: null,
    fields: [],
    hierarchyAttributes: null,
    columns: [],
    actions: null,
    commands: [],
    orderNumber: 30,
    required: false,
    visibility: null,
    enablement: null,
    obligation: null,
    readonly: false,
    selectionPaginated: true,
    validators: null,
    hint: '',
    historyDisplayMode: 0,
    tags: null },
  { sorted: true,
    caption: 'Formatted text [8]',
    type: 8,
    property: 'text_formattext',
    size: 2,
    maskName: null,
    mask: null,
    mode: null,
    fields: [],
    hierarchyAttributes: null,
    columns: [],
    actions: null,
    commands: [],
    orderNumber: 40,
    required: false,
    visibility: null,
    enablement: null,
    obligation: null,
    readonly: false,
    selectionPaginated: true,
    validators: null,
    hint: '',
    historyDisplayMode: 0,
    tags: null } ],
actions: null,
commands:
 [ { id: 'CREATE',
    caption: 'Create',
    visibilityCondition: null,
    enableCondition: null,
    needSelectedItem: false,
    signBefore: false,
    signAfter: false,
    isBulk: false },
  { id: 'EDIT',
```


(continues on next page)

(continued from previous page)

```
    caption: 'Edit',
    visibilityCondition: null,
    enableCondition: null,
    needSelectedItem: true,
    signBefore: false,
    signAfter: false,
    isBulk: false },
  { id: 'DELETE',
    caption: 'Delete',
    visibilityCondition: null,
    enableCondition: null,
    needSelectedItem: false,
    signBefore: false,
    signAfter: false,
    isBulk: true } ],
allowSearch: false,
pageSize: null,
useEditModels: true,
version: null,
overrideMode: null,
metaVersion: '2.0.7',
type: 'list',
className: 'class_text@develop-and-test',
path: '',
caption: '' }
```

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access meta class list view model: getListViewModel
```

Получение информации о форме представления объектов класса в виде коллекции: getCollectionViewModel

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/getCollectionViewModel/<имя класса>, где имя класса указывается с неймспейсом.

В запросе можно указать дополнительные параметры:

- collection
- node
- namespace
- version

Пример запроса:

```
https://localhost:8888/rest/meta/getCollectionViewModel/class_text@develop-and-test
```

будет запрошен объект представления объектов класса class_text в форме коллекции в неймспейсе develop-and-test.

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access meta class collection view model: getCollectionViewModel
```

Получение информации о форме представления объектов класса при создании: getCreationViewModel

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/getCreationViewModel/<имя класса>, где имя класса указывается с неймспейсом.

В запросе можно указать дополнительные параметры:

- node
- namespace
- version

Пример запроса:

```
https://localhost:8888/rest/meta/getCreationViewModel/class_text@develop-and-test
```

будет запрошен объект представления объектов класса class_text при их создании в неймспейсе develop-and-test, пример ответа:

```
  { tabs: [ { caption: '', fullFields: [Array], shortFields: [] } ],
actions: null,
commands:
 [ { id: 'SAVE',
     caption: 'Save',
     visibilityCondition: null,
     enableCondition: null,
     needSelectedItem: false,
     signBefore: false,
     signAfter: false,
     isBulk: false },
   { id: 'SAVEANDCLOSE',
     caption: 'Save and close',
     visibilityCondition: null,
     enableCondition: null,
     needSelectedItem: false,
     signBefore: false,
     signAfter: false,
     isBulk: false } ],
siblingFixBy: null,
siblingNavigateBy: null,
historyDisplayMode: 0,
collectionFilters: null,
version: null,
overrideMode: null,
metaVersion: '2.0.7',
type: 'create',
className: 'class_text@develop-and-test',
path: '',
caption: '' }
```

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access meta class creation view model: getCreationViewModel
```

Получение информации о форме представления объектов класса при редактировании: getItemViewModel

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/getItemViewModel/<имя класса>, где имя класса указывается с неймспейсом.

В запросе можно указать дополнительные параметры:

- node
- namespace
- version

Пример запроса:

```
https://localhost:8888/rest/meta/getItemViewModel/class_text@develop-and-test
```

будет запрошен объект представления объектов класса class_text при их редактировании в неймспейсе develop-and-test, пример ответа:

```
  { tabs: [ { caption: '', fullFields: [Array], shortFields: [] } ],
actions: null,
commands:
 [ { id: 'SAVE',
     caption: 'Save',
     visibilityCondition: null,
     enableCondition: null,
     needSelectedItem: false,
     signBefore: false,
     signAfter: false,
     isBulk: false },
   { id: 'SAVEANDCLOSE',
     caption: 'Save and close',
     visibilityCondition: null,
     enableCondition: null,
     needSelectedItem: false,
     signBefore: false,
     signAfter: false,
     isBulk: false } ],
siblingFixBy: null,
siblingNavigateBy: null,
historyDisplayMode: 0,
collectionFilters: null,
version: null,
overrideMode: null,
metaVersion: '2.0.7',
type: 'item',
className: 'class_text@develop-and-test',
path: '' }
```

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access meta class item view model: getItemViewModel
```

Получение информации о форме представления объектов класса при просмотре: getDetailViewModel

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/getDetailViewModel/<имя класса>, где имя класса указывается с неймспейсом.

В запросе можно указать дополнительные параметры:

- node
- namespace
- version

Пример запроса:

```
https://localhost:8888/rest/meta/getDetailViewModel/class_text@develop-and-test
```

будет запрошен объект представления объектов класса class_text при их просмотре в неймспейсе develop-and-test.

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access meta class detail view model: getDetailViewModel
```

Получение списка возможных бизнес-процессов для класса: getWorkflows

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/getWorkflows/<имя класса>, где имя класса указывается с неймспейсом.

В запросе можно указать дополнительные параметры:

- namespace
- version

Пример запроса:

```
https://localhost:8888/rest/meta/getWorkflows/workflowBase@develop-and-test
```

будет запрошен список бизнес-процессов для класса workflowBase в неймспейсе develop-and-test, пример ответа:

```
[ { name: 'simpleWorkflow',
caption: 'Simple WF',
wfClass: 'workflowBase@develop-and-test',
startState: 'canStart',
states: [ [Object], [Object], [Object], [Object], [Object] ],
transitions: [ [Object], [Object], [Object], [Object], [Object] ],
metaVersion: '2.0.61.16945',
namespace: 'develop-and-test' } ]
```

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# accessing the list of possible workflows for meta class: getWorkflows
```

Получение информации о форме представления объекта класса при некотором состоянии бизнес-процесса: getListViewModel

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/getListViewModel/<имя класса>/<имя бизнес-процесса>/<имя состояния>, где имя класса и имя бизнес-процесса указываются с неймспейсом.

В запросе можно указать дополнительные параметры:

- workflow
- state
- namespace
- version

Пример запроса:

```
https://localhost:8888/rest/meta/getWorkflowView/workflowBase@develop-and-test/simpleWorkflow@develop-and-
↪test/canStart
```

будет запрошен объект представления объектов класса workflowBase при состоянии canStart бизнес-процесса simpleWorkflow в неймспейсе develop-and-test, пример ответа:

```
  { tabs: [ { caption: '', fullFields: [Array], shortFields: [] } ],
actions: null,
siblingFixBy: null,
siblingNavigateBy: null,
historyDisplayMode: 0,
collectionFilters: null,
version: null,
overrideMode: 1,
commands:
 [ { id: 'SAVE',
     caption: 'Save',
     visibilityCondition: null,
     enableCondition: null,
     needSelectedItem: false,
     signBefore: false,
     signAfter: false,
     isBulk: false },
   { id: 'SAVEANDCLOSE',
     caption: 'Save and close',
     visibilityCondition: null,
     enableCondition: null,
     needSelectedItem: false,
     signBefore: false,
     signAfter: false,
     isBulk: false } ],
metaVersion: '2.0.61',
type: 'item',
className: 'workflowBase@develop-and-test',
path: 'workflows:simpleWorkflow@develop-and-test.canStart',
caption: '' }
```

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access meta class view model in a certain workflow state: getWorkflowView
```

Получение информации о бизнес-процессе класса: getWorkflow

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/getWorkflow/<имя класса>/<имя бизнес-процесса>, где имя класса и имя бизнес-процесса указываются с неймспейсом.

В запросе можно указать дополнительные параметры:

- workflow
- namespace
- version

Пример запроса:

```
https://localhost:8888/rest/meta/getWorkflow/workflowBase@develop-and-test/simpleWorkflow@develop-and-test
```

будет запрошен объект бизнес-процесса simpleWorkflow для класса объектов workflowBase в неймспейсе develop-and-test. Пример ответа:

```
  { name: 'simpleWorkflow',
caption: 'Simple WF',
wfClass: 'workflowBase@develop-and-test',
startState: 'canStart',
states:
 [ { name: 'canStart',
     caption: 'Ready to check',
     maxPeriod: null,
     conditions: [Object],
     propertyPermissions: [],
     itemPermissions: [],
     selectionProviders: [] },
   { name: 'inProcess',
     caption: 'In process',
     maxPeriod: null,
     conditions: null,
     itemPermissions: [Array],
     propertyPermissions: [],
     selectionProviders: [] },
   { name: 'accepted',
     caption: 'Accepted',
     maxPeriod: null,
     conditions: null,
     itemPermissions: [],
     propertyPermissions: [],
     selectionProviders: [] },
   { name: 'returned',
     caption: 'Returned',
     maxPeriod: null,
     conditions: null,
     itemPermissions: [Array],
     propertyPermissions: [],
     selectionProviders: [] },
   { name: 'rejected',
     caption: 'Rejected',
     maxPeriod: null,
     conditions: null,
     itemPermissions: [],
     propertyPermissions: [],
```

(continues on next page)

(continued from previous page)

```
    selectionProviders: [] } ],
transitions:
 [ { name: 'startCheck',
    caption: 'Start checking',
    startState: 'canStart',
    finishState: 'inProcess',
    signBefore: false,
    signAfter: false,
    roles: [],
    assignments: [Array],
    conditions: null,
    confirm: false,
    confirmMessage: null },
   { name: 'return',
    caption: 'Return',
    startState: 'inProcess',
    finishState: 'returned',
    signBefore: false,
    signAfter: false,
    roles: [],
    assignments: [Array],
    conditions: null,
    confirm: false,
    confirmMessage: null },
   { name: 'accept',
    caption: 'Accept',
    startState: 'inProcess',
    finishState: 'accepted',
    signBefore: false,
    signAfter: false,
    roles: [],
    assignments: [Array],
    conditions: null,
    confirm: false,
    confirmMessage: null },
   { name: 'reject',
    caption: 'Reject',
    startState: 'inProcess',
    finishState: 'rejected',
    signBefore: false,
    signAfter: false,
    roles: [],
    assignments: [Array],
    conditions: null,
    confirm: false,
    confirmMessage: null },
   { name: 'notify',
    caption: 'To check',
    startState: 'returned',
    finishState: 'canStart',
    signBefore: false,
    signAfter: false,
    roles: [],
    assignments: [Array],
    conditions: [Object],
    confirm: false,
```

(continues on next page)

(continued from previous page)

```
    confirmMessage: null } ],
metaVersion: '2.0.61.16945',
namespace: 'develop-and-test' }
```

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access information about workflow: getWorkflow
```

Получение информации о маске формы представления: getMask

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/getMask/<имя маски>.

Например:

```
https://localhost:8888/rest/meta/getMask/snils
```

будет запрошен объект маски snils.

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access information about view mask: getMask
```

Получение списка доступных валидаторов ввода: getValidators

Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/getValidators, например:

```
https://localhost:8888/rest/meta/getValidators
```

будет запрошен список доступных валидаторов.

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access information about input validators: getValidators
```

Дополнительные параметры запросов к сервису метаданных

В большинстве запросов к сервису метаданных (meta), помимо самого запроса, можно указать дополнительные параметры. Список дополнительных параметров, которые необходимо применить к запросу начинается символом ?, после этого пишется название и значение параметра через знак =, несколько параметров разделяются между собой с помощью &.

Пример GET запроса к meta/getList с дополнительным параметром ancestor - фильтрацией по предку:

```
https://localhost:8888/rest/meta/listMeta?ancestor=basicObj@develop-and-test
```

В ответ будет возвращен список классов метаданных, которые являются производными от basicObj.

Этот пример в dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# accessing the list of metadata classes filtering by ancestor: listMeta
```

Сервис исполнения бизнес-процессов Workflows

Workflows workflows - встроенный сервис в rest модуле, который предоставляет возможности контроля и управления бизнес-процессами.

Сервис требует подключения в deploy.json и обязательного указания options.dataRepo, options.metaRepo, options.auth и options.workflow, пример:

```
"workflows": {
      "module": "modules/rest/lib/impl/workflows",
      "options": {
        "auth": "ion://auth",
        "dataRepo": "ion://dataRepo",
        "metaRepo": "ion://metaRepo",
        "workflow": "ion://workflow"
      }
}
```

Поддерживаются все типы авторизации, по умолчанию - авторизация учетными данными.

Сервис содержит три метода:

- GET - без параметров, возвращает информацию о текущем статусе в БП (возможные переходы)
- PUT - выполняет перевод обьекта в указанные следующие этапы разных БП.
- PATCH - выполняет принудительный перевод обьекта в указанные этапы разных БП.

Для всех методов запросы принимаются по пути <URL сервера>/rest/<имя сервиса>/:class/:id для идентификации обьекта данных.

Спецификация методов:

Получение текущего положения обьекта в бизнес-процессе: GET

Методом GET осуществляется получение текущего положения обьекта в бизнес-процессах. Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/<имя класса>/<id объекта>, пример:

```
https://localhost:8888/rest/workflows/workflowBase@develop-and-test/1
```

В ответ будет получен объект с состояниями бизнес-процессов:

```
{ stages:
   { 'simpleWorkflow@develop-and-test':
     { stage: 'inProcess',
       since: '2020-05-12T06:36:06.045Z',
       next: [Object],
       workflowCaption: 'Simple WF',
       stageCaption: 'In process' } },
   itemPermissions: { read: true },
   propertyPermissions: {},
   selectionProviders: {} }
```

Пример GET запроса к workflows в dnt: test/modules/rest/workflows.spec.js

```
/checking workflows service/# accessing workflow statuses of the object: GET
```

Выполнение перехода объекта по бизнес-процессу: PUT

Методом PUT сервиса workflows осуществляется выполнение переходов объекта по БП (в том числе последовательных). Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/<имя класса>/<id объекта>.

Имя класса указывается с неймспейсом.

В теле запроса передается один из вариантов:

- объект с атрибутами - именами БП, которые содержат список из переходов по этим бизнес-процессам.
- список из строк формата <имя бизнес-процесса>.<название перехода>

имена бизнес-процессов указываются с неймспейсом.

Пример запроса:

```
PUT
https://localhost:8888/rest/workflows/workflowBase@develop-and-test/1
body: {
     'simpleWorkflow@develop-and-test': [
       'startCheck',
       'accept'
     ]
   }
```

что равносильно:

```
PUT
https://localhost:8888/rest/workflows/workflowBase@develop-and-test/1
body: [
    'simpleWorkflow@develop-and-test.startCheck',
    'simpleWorkflow@develop-and-test.accept'
]
```

Переходы выполняются последовательно. Для каждого перехода будет предпринята попытка выполнения, даже если в одном из них произошла ошибка.

В ответ будет возвращен список из ошибок по каждому переходу:

```
[ { code: 'workflow.ti',
    params: { workflow: 'Simple WF', trans: 'Start checking' },
    message:
     'Невозможно выполнение перехода \'Start checking\' рабочего процесса \'Simple WF\'.' },
{ code: 'workflow.ti',
    params: { workflow: 'Simple WF', trans: 'Accept' },
    message:
     'Невозможно выполнение перехода \'Accept\' рабочего процесса \'Simple WF\'.' } ]
```

или пустой список.

Примеры PUT запросов к workflows в dnt: test/modules/rest/workflows.spec.js

```
/checking workflows service/# move the object through workflow: PUT, list body
/checking workflows service/# move the object through workflow: PUT, object body
```

Перемещение объекта в указанное состояние бизнес-процесса: PATCH

Методом PATCH осуществляется принудительное перемещение объекта в указанные состояния бизнес-процессов. Запрос осуществляется по пути <URL сервера>/rest/<название сервиса>/<имя класса>/<id объекта>. Имя класса указывается с неймспейсом.

В теле запроса передается массив целевых состояний бизнес-процессов, которые указываются как строки в формате <имя бизнес-процесса>.<состояние>. Имя бизнес-процесса указывается с неймспейсом. Объект последовательно перемещается в каждое из состояний.

Пример запроса:

```
PATCH
https://localhost:8888/rest/workflows/workflowBase@develop-and-test/1
body: [
    'simpleWorkflow@develop-and-test.canStart'
  ]
```

В ответ будет возвращен список ошибок, возникших при перемещениях, либо пустой список.

Пример PATCH запроса к workflows в dnt: test/modules/rest/workflows.spec.js

```
/checking workflows service/# move the object to certain state in a workflow: PATCH
```

- Получение текущего положения обьекта в бизнес-процессе: GET
- Выполнение перехода объекта по бизнес-процессу: PUT
- Перемещение объекта в указанное состояние бизнес-процесса: PATCH

Разработка обработчика сервиса в приложении

Все сервисы реализуются как наследники от Service - функции модуля rest.

Каждый сервис должен экспортировать функцию обработчика, в которой реализован асинхронный метод this._route, в котором необходимо зарегистрировать обрабатываемые методы и пути через функции this.addHandler, возвращающие Promise. Функция обработки будет иметь доступ к options, через который доступ к репозиториям данных, авторизации, метаданным и классам (если они указаны в конфигурации приложения в файле deploy.json), а также получит объект с типовым названием req - являющимся объектом request библиотеки express. Данные, прошедшие парсинг в объект будут находиться в req.body.

Функция-обработчик должна вернуть Promise разрешающийся в результат обработки (для обработки в Service modules/rest/lib/interfaces/Service.js), обработчик выдаст его с кодом 200 и типом содержания Content-Type:application/json. Если в ходе обработки будет ошибка, перехваченная catch, то для ошибок связанных с контролем доступа будет возвращен ответ с текстом ошибки и с кодом 403, а для всех остальных код ответа 500 и сообщением об ошибке Internal server error.

Заголовок можно переопределить, для этого в ответе нужно отдать тип заголовка headers, а объект в атрибуте data

```
Promise.resolve({headers: ['Content-Type: image/png', 'Content-Length: 107'],
  data: Buffer.from([0x89, 0x50, 0x4E, 0x47, 0x0D, 0x0A, 0x1A, 0x0A, 0x00, 0x00, 0x00, 0x0D, 0x49,
        0x48, 0x44, 0x52, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x01, 0x08, 0x06, 0x00, 0x00, 0x00,
        0x1F, 0x15, 0xC4, 0x89, 0x00, 0x00, 0x00, 0x06, 0x62, 0x4B, 0x47, 0x44, 0x00, 0xFF, 0x00, 0xFF,
        0x00, 0xFF, 0xA0, 0xBD, 0xA7, 0x93, 0x00, 0x00, 0x00, 0x09, 0x70, 0x48, 0x59, 0x73, 0x00, 0x00,
        0x2E, 0x23, 0x00, 0x00, 0x2E, 0x23, 0x01, 0x78, 0xA5, 0x3F, 0x76, 0x00, 0x00, 0x00, 0x0B, 0x49,
```

(continues on next page)

(continued from previous page)

```
        0x44, 0x41, 0x54, 0x08, 0xD7, 0x63, 0x60, 0x00, 0x02, 0x00, 0x00, 0x05, 0x00, 0x01, 0xE2, 0x26,
        0x05, 0x9B, 0x00, 0x00, 0x00, 0x00, 0x49, 0x45, 0x4E, 0x44, 0xAE, 0x42, 0x60, 0x82])
})
```

Пример реализации сервиса, выдающего списки объектов с фильтрами для класса class_string есть в приложении develop-and-test. Также, для изучения удобно смотреть сам метод crud, находящийся по адресу modules/rest/lib/impl/crud.js

```
const Service = require('modules/rest/lib/interfaces/Service');

/**
 * @param {{dataRepo: DataRepository, echoClassName: String}} options
 * @constructor
 */
function listClassString(options) {

  /**
   * @param {Request} req
   * @returns {Promise}
   * @private
   */
  this._route = function(router) {
    this.addHandler(router, '/', 'POST', (req) => {
      return new Promise(function (resolve, reject) {
        try {
          let filter = [];
          if (req.body.string_text)
            filter.push({string_text: {$eq: req.body.string_text}});
          if (req.body.string_miltilinetext)
            filter.push({string_miltilinetext: {$eq: req.body.string_miltilinetext}});
          if (filter.length === 0)
            filter = {};
          else if (filter.length === 1)
            filter = filter[0];
           else
            filter = {$and: filter};
          options.dataRepo.getList(options.stringClassName, {filter: filter}).then(function (results) {
            let items = [];
            for (let i = 0; i < results.length; i++) {
              const props = results[i].getProperties();
              const item = {};
              for (let p in props) {
                if (props.hasOwnProperty(p))
                  item[props[p].getName()] = props[p].getValue();
              }
              items.push(item);
            }
            resolve({data: items});
          });
        } catch (err) {
          reject(err);
        }
      });
    });
  }
}
```

(continues on next page)

(continued from previous page)

```
listClassString.prototype = new Service();

module.exports = listClassString;
```

Запрос без атрибутов в теле запроса

```
curl -X POST -u demo@local:ion-demo https://dnt.iondv.com:8888/rest/string-list
```

вернет весь список:

```
[{"__class":"class_string@develop-and-test",
  "__classTitle":"Class \"String [0]\"",
  "id":"4567a830-b8ea-11e9-9cdf-7bd384cbb7a5",
  "string_text":"example1",
  "string_miltilinetext":"example1",
  "string_formattext":"<p>example1</p>"},
{"__class":"class_string@develop-and-test",
  "__classTitle":"Class \"String [0]\"",
  "id":"4a80bdc0-b8ea-11e9-9cdf-7bd384cbb7a5",
  "string_text":"example1",
  "string_miltilinetext":"example2",
  "string_formattext":"<p>example2</p>"},
{"__class":"class_string@develop-and-test",
  "__classTitle":"Class \"String [0]\"",
  "id":"66dbb3d0-5583-11e6-aef7-cf50314f026b",
  "string_text":"Example of the \"String [0]\" type in the \"Text [1]\" view",
  "string_miltilinetext":"Example of the \"String [0]\"\r\n in the Multiline text [7] view",
  "string_formattext":"Example of the \r\n \"String [0]\" type \r\n in the \r\nFormatted text [7] view"}]
```

А запрос с параметром атрибута равного значению в атрибуте string_text Example of the \"String [0]\" type in the \"Text [1]\" view

```
curl -X POST -d "string_text=Example of the \"String [0]\" type in the \"Text [1]\"" \
    -u demo@local:ion-demo https://dnt.iondv.com:8888/rest/string-list
```

вернет объекты удовлетворяющие условию:

```
[{"__class":"class_string@develop-and-test",
  "__classTitle":"Class \"String [0]\"",
  "id":"66dbb3d0-5583-11e6-aef7-cf50314f026b",
  "string_text":"Example of the \"String [0]\" type in the \"Text [1]\" view",
  "string_miltilinetext":"Example of the \"String [0]\"\r\n in the Multiline text [7] view",
  "string_formattext":"Example of the \r\n \"String [0]\" type \r\n in the \r\nFormatted text [7] view"}]
```

Пример регистрации тестового сервиса, подробнее см. Регистрация сервиса в конфигурации приложения

```
{
  "modules": {
    "rest": {
      "globals": {
        "di": {
          "string-list": {
            "module": "applications/develop-and-test/service/String-list",
            "options": {
```

(continues on next page)

(continued from previous page)

```
        "stringClassName": "class_string@develop-and-test",
        "dataRepo": "ion://dataRepo"
      }
    }
```

Для реализации обработки multipart запросов, например для запросов, содержащих файлы, можно использовать библиотеку multipart.js (rest/backend/multipart.js) модуля REST. Пример реализации есть в сервисе CRUD:

```
function reqToData(req) {
  return multipart(req).then(data => data || req.body);
}
```

Этой цели также служит библиотека util.js (rest/backend/util.js), обеспечивающая корректность действий при работе с файлами и файловым хранилищем, пример из CRUD:

```
reqToData(req)
  .then(data => <util.js.>prepareUpdates(options, data, cm, req.params.id))
```

Регистрация сервиса в конфигурации приложения

Для подключения сервисов в приложении их необходимо сконфигурировать в глобальных настройках модуля rest в файле deploy.json приложения. Пример:

```
{
  "modules": {
    "rest": {
      "globals": {
        "di": {
          "simple": {
            "module": "applications/develop-and-test/service/SimpleRest"
          },
          "string-list": {
            "module": "applications/develop-and-test/service/String-list",
            "options": {
              "stringClassName": "class_string@develop-and-test",
              "dataRepo": "ion://dataRepo"
            }
          },
          "crud": {
            "module": "modules/rest/lib/impl/crud",
            "options": {
               "auth": "ion://auth",
               "dataRepo": "ion://securedDataRepo"
            }
          }
```

Путь к регистрациям сервиса в файле deploy.json - modules.rest.globals.di. Далее указывается название сервиса, которое будет доступно по адресу https://domain.com/rest/serviceName, где serviceName - имя сервиса, указываемого в di, например в примере выше simple или string-list.

В атрибуте module указывается путь к js-файлу с обработчиком сервиса с путем относительно корня фреймворка. Обработчик может быть как в приложении, так и в любом модуле или фреймворке, в т.ч. типовые обработчики модуля rest.

В параметре options указываются специфические настройки сервиса. Например, для сервиса crud указаны:

- в поле dataRepo - репозиторий данных с контролем доступа, используемый для операций над обьектами
- в поле auth- компонент аутентификации, используемый для получения текущей учетной записи пользователя.

А для сервиса string-list указаны:

- в поле dataRepo - репозиторий данных, используемый для выборки данных
- в поле stringClassName - имя класса получаемых обьектов в данном случае класс class_string@develop-and-test будет передан в метод getList репозитория данных

```
options.dataRepo.getList(options.stringClassName, {})
```

Запросы к сервисам на примере тестов

Примеры запросов к основным сервисам REST можно найти в подготовленных для этих сервисов тестах. Тесты написаны для mocha и находятся в репозитории приложения iondv develop-and-test.

Для запуска тестов нужно установить пакеты request-promise-native и mocha, например запустив npm install в папке develop-and-test/test.

Предварительно нужно сконфигурировать параметры сервера и пользователей в develop-and-test/test/modules/rest/config.js.

Запуск осуществляется вызовом mocha/lib/cli/cli.js для тестового скрипта, например:

```
node node_modules/mocha/lib/cli/cli.js modules/rest/echo.spec.js
```

Можно запустить все тесты в папке:

```
node node_modules/mocha/lib/cli/cli.js modules/rest/*.spec.js
```

Также возможен запуск отдельных юнит-тестов:

```
node node_modules/mocha/lib/cli/cli.js -g "<ЧАСТЬ НАЗВАНИЯ ТЕСТА>" modules/rest/*.spec.js
```

например:

```
node node_modules/mocha/lib/cli/cli.js -g "creating an object" modules/rest/*.spec.js
```

Модуль REST имеет несколько встроенных сервисов, предназначенных для реализации типовых операций с модулем:

- Сервис Acceptor - обеспечивает массовое создание объектов.
- Сервис Token - обеспечивает выдачу токена для авторизованного пользователя.
- Сервис CRUD - сервис CRUD для объектов системы.
- Сервис Meta - обеспечивает доступ к интерфейсу метаданных.
- Сервис Workflows - обеспечивает возможности контролировать и управлять бизнес-процессами.

Также поддерживается разработка и подключение пользовательских сервисов.

Разработка обработчика в приложении - подробнее.

Регистрация сервиса в конфигурации приложения - подробнее.

Модули - это независимые блоки с обязательной или дополнительной функциональностью, структура которых подчиняется определенным правилам.

Наименование	Описание
Registry	Ключевой модуль предназначенный непосредственно для работы с данными на основе структур метаданных.
Report	Модуль предназначенный для формирования аналитических отчетов.
Gantt-chart	Модуль для вывода данных с датами.
Portal	Модуль для вывода произвольных шаблонов данных.
Dashboard	Модуль предназначенный для выведения краткой информации в виде блоков.
Ionadmin	Модуль администрирования для назначения прав.
Account	Модуль личного кабинета, который формирует логику отображения.
Soap	Модуль запросов к сервисам.
Image-storage	Модуль предварительного просмотра картинок.
REST	Модуль обеспечивающий работу с данными приложения IONDV через REST API.

ГЛАВА 5

5. Создание проекта модели ИОН

5.1 Настройка среды разработки ION

ГЛАВА 6

Functionality of IONDV. Framework and its modules

6.1 IONDV. Framework

IONDV. Framework provides the following functionality:

- descriptive metadata into the data storage structure in the DBMS;
- functionality to work with various DBMS (ORM technology);
- authorization in a system with different policies, by default oath2, with an open, configurable API for connecting passport library authorization modules which provides up to 500 different authorization policies;
- securing access to data - static securing to data types, to navigation, to stages of business processes, to actions on a form; dynamic securing- through the conditions in the data that the profile of the current user must correspond to (belonging to the unit or organization specified in the object, group or other conditions); through url; providing exceptions in authorization and security by url or for a special user;
- connection of modules providing additional functionality and implemented through access to the kernel interfaces (APIs);
- providing import, export of data in the system, metadata, security from files;
- providing interaction with the file system for storing data, including external file storages, such as nextcloud;
- calculating values with formulas and caching this data;
- providing eager loading and data filtering in connected collections;
- caching requests and sessions in memcached, redis;
- scheduled tasks;
- notification of users by events.

6.2 Modules

Additional functionality is implemented by standard modules.

6.2.1 Data accounting module - registry:

- hierarchical display of navigation;
- displaying lists of data objects according to navigation conditions, filters, search results;
- the ability to create objects;
- display of unified forms of objects with the ability to edit, delete, modify work-flows, implement the conditions for displaying and overloading the presentation of a form in a business process;
- display of various types of attributes, including related in the form of tables or links, geo objects (including search for coordinates by address);
- displaying data according to their semantics (the terms of changes);
- the ability to change the display and interaction with the attributes of objects through custom HTML templates that receive data by REST-API;
- preparation of printed forms in docx and xlsx format based on lists or object data;
- display of user notifications;
- the ability to implement your own action buttons with server data processing.

6.2.2 Reporting and Analytics Module - report:

- formation of calculated forms, with the ability to filter by values;
- data filtering;
- mathematical operations on data;
- pivot tables;
- REST API to report data.

6.2.3 Display of data with geo-coordinates – geomap:

- data layers implementation with filtering by conditions;
- ability to set data view icons according to data type;
- display a pop-up window with brief information on an object;
- display of a template of detailed information on an object;
- search for objects;
- arbitrary boundary filtering;
- zoning and filtering by district boundaries;
- report module data connection, including the calculated data for the region.

6.2.4 REST and SOAP integration modules with standard APIs and user security:

- various custom types of authorization: in the header, token (inclusion of the service receiving a token after authorization in the header), without authorization;
- receiving lists of objects of each type with different filtering conditions;
- CRUD service for any data type;
- work-flow transition service;
- metadata retrieval service;
- the ability to connect arbitrary custom processing services.

6.2.5 Dashboard module - dashboard:

- ensuring of the formation of information blocks with digital and graphic data;
- allows to adjust several groups of views and customize them for each user.

6.2.6 Administration module - ionadmin:

- provides user management, rights and roles control, user blocking;
- generation of security keys (tokens) for integration services;
- monitoring of key server resources (using the dashboard module);
- analysis of slow DBMS queries;
- scheduled tasks setting;
- tracking of system objects changes;
- data and metadata backup;
- recalculation of semantics and formulas caches;
- notification management.

6.2.7 Custom webpage creation module - portal:

- registration of arbitrary pages at the processing address (route);
- registration of static content;
- security access management;
- support for rendering pages from EJS templates.

6.3 The IONDV. Studio application for metadata creation:

- creation of navigation;
- creation of class structures;
- creation of views for classes;
- creation of work-flows;

- basic application setup;
- export and import of metadata;
- work with project files in standalone mode;
- online work with several projects hosted in the browser repository.

6.4 License Contact us English

Эта страница на Русском

ГЛАВА 7

IONDV. Framework

7.1 Description

IONDV. Framework - это опенсорный фреймворк на node.js для разработки учетных приложений на основе метаданных в формате JSON/YAML и отдельных функциональных модулей. Визуальный редактор Studio позволяет создавать приложения по технологии “no code” и собирать приложение с веб-сервисами REST-API (модуль rest). Ключевой модуль registry является универсальным средством представления и редактирования данных, обработки их по бизнес-процессам.

На видео технология разработки и сборки приложения.

7.2 Free Demos

For now, we have three demos to show you:

- Studio - специализированная IDE созданная как приложение iondv, для визуальной (no code) разработки приложений на IONDV. Framework. GitHub Репозиторий. Видео инструкция и текстовая по созданию приложения при помощи ION. Studio.
- Telecom - приложение по организации учета, хранения и отображения данных о наличии услуг связи (интернет, сотовая связь, телевидение, почта и др.) в населенных пукнтах региона. GitHub Repo
- DNT - приложение для разработки и тестирования функциональности фреймворка, в котором каждая учетная сущность отражает тим метаданных, например класс “строка”, или класс “коллекция”. Это позволяет изучать возможности фреймворка через приложение. GitHub Репозиторий.
- War Archive (in Russian) - is the IONDV. Framework web-application designed to store, group and demonstrate the data based on archival documents about Great Patriotic War (World War II). ‘GitHub Repo‘_.
- Project Management - приложение по организации проектной деятельности региональных ОГВ , целью которой является контроль результатов, соблюдение и сокращение сроков их достижения,

эффективное использование временных, человеческих и финансовых ресурсов, принятие своевременных и обоснованных управленческих решений. GitHub Repo

- CRM - coming soon on GitHub.

The login for access is - demo and the password is - ion-demo. No registration required.

7.3 Typical applications

Фреймворк - конструктор веб-приложений любой специфики, так как преметная область определяется структурой метаданных, описывающих поведение приложение. Например можно создать приложения

- CRM - управления отношения с клиентами;
- учет и управление ресурсов предприятия;
- автоматизация бизнес-процессов организации и документооборота;
- сбор и хранение любых данных, например метрик оборудования (IoT);
- представление данных в виде порталов;
- REST-API для SPA приложений;
- REST-API и бекграунд для мобильных приложений;

7.4 Top features

IONDV. Framework provides the following functionality:

- descriptive metadata into the data storage structure in the DBMS;
- functionality to work with various DBMS (ORM technology);
- authorization in a system with different policies, by default oath2, with an open, configurable API for connecting passport library authorization modules which provides up to 500 different authorization policies;
- securing access to data - static securing to data types, to navigation, to stages of business processes, to actions on a form; dynamic securing- through the conditions in the data that the profile of the current user must correspond to (belonging to the unit or organization specified in the object, group or other conditions); through url; providing exceptions in authorization and security by url or for a special user;
- connection of modules providing additional functionality and implemented through access to the kernel interfaces (APIs);
- providing import, export of data in the system, metadata, security from files;
- providing interaction with the file system for storing data, including external file storages, such as nextcloud;
- calculating values with formulas and caching this data;
- providing eager loading and data filtering in connected collections;
- caching requests and sessions in memcached, redis;
- scheduled tasks;
- notification of users by events.

Структура фреймворка: core + metadata + modules = application

На рискнуке: - ioncore - ядро приложения в виде IONDV. фреймворка - meta class, meta view, meta navigation, meta workflow, meta security - функциональные метаданные приложения - структуры, представления, навигации, бизнес-процессов и безопасности соответственно - registry module - подлкючаемые функциональные модули, например модуль registry для просмотра и редактирования данных Чуть ниже представлены дополнительные типы меты и модули. Они представляют собой дополнительную функциональность и применяются в соответствии с приложением. Зависимости приложения смотрим в файле package.json.

Так как приложение - это метаописание его поведения в файлах формата JSON (YAML) и функциональный код и HTML шаблоны расширяющие типовую функциональность - то с ним удобно работать через репозиторий версий git. Посмотрите примеры на Github

Подробнее о функциональных возможностях фреймворка и его модулей можно узнать в документации.

7.5 Quick start

You can get access to the already built applications deployed on Cloud servers or explore the different ways on the IONDV.Framework site, for example:

- инсталятор для операционной системы Linux
- клонирование репозитория приложения и установка всех компонентов (инструкция ниже)
- docker-контейнеры с собранными приложениями

7.6 Software requirements

Install Node.js runtime and npm package manager to run the IONDV.Framework. Version 10.x.x.

Для хранения данных необходимо установить и запустить MongoDb версии старше 3.6.

7.7 Installer

Для ускорения Вы можете использовать установщик приложений IONDV. Framework iondv-app, требующий установленных node, mongodb и git. В ходе установки будет проверены и установлены все остальные зависимости, а также собрано и запущено само приложение.

Install in one command:

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) -t git -q -i -m␣
↪localhost:27017 develop-and-test
```

Где параметры для iondv-app localhost:27017 адрес MongoDB, a develop-and-test название приложения. После запуска открыть ссылку 'http://localhost:8888', типовая учетная запись бек офиса demo, пароль ion-demo.

Дугой способ заключается в клонировании сборщика - (git clone https://github.com/iondv/iondv-app.git) и установите приложение с помощью команды bash iondv-app -m localhost:27017 develop-and-test.

You can also build the application in docker containers, then from the environment you only need docker and the mongodb DBMS in the docker container. More details on the IONDV. Framework application builder page iondv-app

7.8 Gitclone with repository

7.8.1 Global dependencies

To build all components and libraries, you need to install the following components globally:

- package node-gyp npm install -g node-gyp. For the Windows operating system, it is additionally necessary to install the windows-build-tools package npm install -g --production windows-build-tools.
- Gulp <http://gulpjs.com/>‘_installation package ‘‘npm install -g gulp@4.0‘. 4.0 - supported version of Gulp.
- для версий IONDV. Framework 3.x.x и более ранних нужен менджер пакетов фронтенд библиотек Bower npm install -g bower. Для версия 4.x.x и старше не требуется.

7.8.2 Ручная установка ядра, модулей и приложения

Рассматриваем на примере приложения develop-and-test. Находим приложение develop-and-test в репозитории. Смотрим зависимости указаные в файле package.json.

```
 "engines": {
  "ion": "1.24.1"
},
"ionModulesDependencies": {
  "registry": "1.27.1",
  "geomap": "1.5.0",
  "graph": "1.3.2",
  "portal": "1.3.0",
  "report": "1.9.2",
  "ionadmin": "1.4.0",
  "dashboard": "1.1.0",
  "lk": "1.0.1",
  "soap": "1.1.2",
  "gantt-chart": "0.8.0"
},
"ionMetaDependencies": {
  "viewlib": "0.9.1"
  "viewlib-extra": "0.1.0"
```

- Начинаем установку с ядра, версия которого указана в парметре "engines": {"ion": "1.24.1"}. Скопируйте адрес репозитория ядра и в командной строке выполните комманду git clone https://github.com/iondv/framework. Перейдите в папку ядра, переключитесь на tag номера версии git checkout tags/v1.24.1. Так как совместимость обеспечивается на уровне метаданных, а новые версии выпускались из-за изменения технологии сборки, то вы можете использовать последнии, например 4.0.0.
- После этого устанавливаются необходимые для приложения модули, указанные в параметре "ionModulesDependencies". Модули устанавливаются в папку modules ядра, для этого перейдите в неё командой cd modules. Клонируем модули из списка "ionModulesDependencies", для модуля registry это осуществляется коммандой git clone https://github.com/iondv/registry. Перейдите в папку установленного модуля, переключитесь на tag номера версии git checkout tags/v1.27.1. Повторите для каждого модуля. Для большинства приложений, можно использовать последние совместимые с ядром модули.
- Установка самого приложения осуществляется в папку applications, для этого перейдите в неё

командой cd ..\applications, если вы находитесь в папке модулей. Установку выполните клонированием репозитория коммандой git clone https://github.com/iondv/dnt_ru.

- Finally, install all necessary applications listed in the "ionMetaDependencies" parameter in the applications folder. Make sure that you're inside this folder. Clone the dependencies in ionMetaDependencies, in particularly viewlib - a additional application - library of views templates. Execute the git clone https://github.com/iondv/viewlib.git to clone to the applications folder. Go to the folder of installed application and switch to the tag of the version number git checkout tags/0.9.1. Repeat for each application.

7.8.3 Building, configuring and deploying the application

Building the application provides installation of all dependent libraries, importing data into the database and preparing the application for launch.

Создайте конфигурационный файл setup.ini в папке config куда вы клонировали фреймворк для задания основных параметров окружения приложения.

```
auth.denyTop=false
auth.registration=false
db.uri=mongodb://127.0.0.1:27017/db
server.ports[]=8888
module.default=registry
fs.storageRoot=./files
fs.urlBase=/files
```

Open the file and paste the text above. The main parameter is db.uri=mongodb://127.0.0.1:27017/iondv-dnt-db. It shows the base name that we use for the application. The DB will be created automatically.

Задайте переменную окружения NODE_PATH равной пути к ядру приложения следующей командой set NODE_PATH=c:\workspace\dnt для Windows и export NODE_PATH=/workspace/dnt для Linux, где workspace\dnt - папка куда склонирован фреймворк.

При первом запуске необходимо выполнить npm install - она поставит ключевые зависимости, в том числе локально сборщик gulp.

Further, execute the gulp assemble command to build and deploy the application.

Если вы хотите выполнить импорт данных в вашем проекте, проверьте папку data в приложении и выполните команду: node bin/import-data --src ./applications/develop-and-test/data --ns develop-and-test

Add the admin user with the 123 password executing the node bin/adduser.js --name admin --pwd 123 command.

Add admin rights to the user executing the node bin/acl.js --u admin@local --role admin --p full command.

7.8.4 Running

Запустите приложение командой в папке ядра npm start или node bin\www.

Open this link http://localhost:8888 in a browser and log in. 8888 — is a port in the server.ports parameter.

7.9 Docker

Follow these steps to deploy docker container on the example of the develop-and-test application:

1. Run mongodb DBMS: docker run --name mongodb -v mongodb_data:/data/db -p 27017:27017 -d mongo
2. Run IONDV. develop-and-test docker run -d -p 80:8888 --link mongodb iondv/dnt.
3. Откройте ссылку http://localhost в браузере через минуту (время требуется для инициализации данных). Для авторизации используйте типовой логин: demo, пароль: ion-demo

ГЛАВА 8

Documentation

The IONDV.Framework documentation is available in two languages — english and russian.

ГЛАВА 9

Reference

Some handy links to learn more information on developing applications using IONDV.Framework.

- ‘Documentations https://iondv.readthedocs.io/en/latest/index.html>‘_
- Homepage
- Feedback on Facebook
- Обучающие видеоролики на youtube

ГЛАВА 10

License Contact us English