605aad03f032762902080b4a76fb249af0f7c7ec

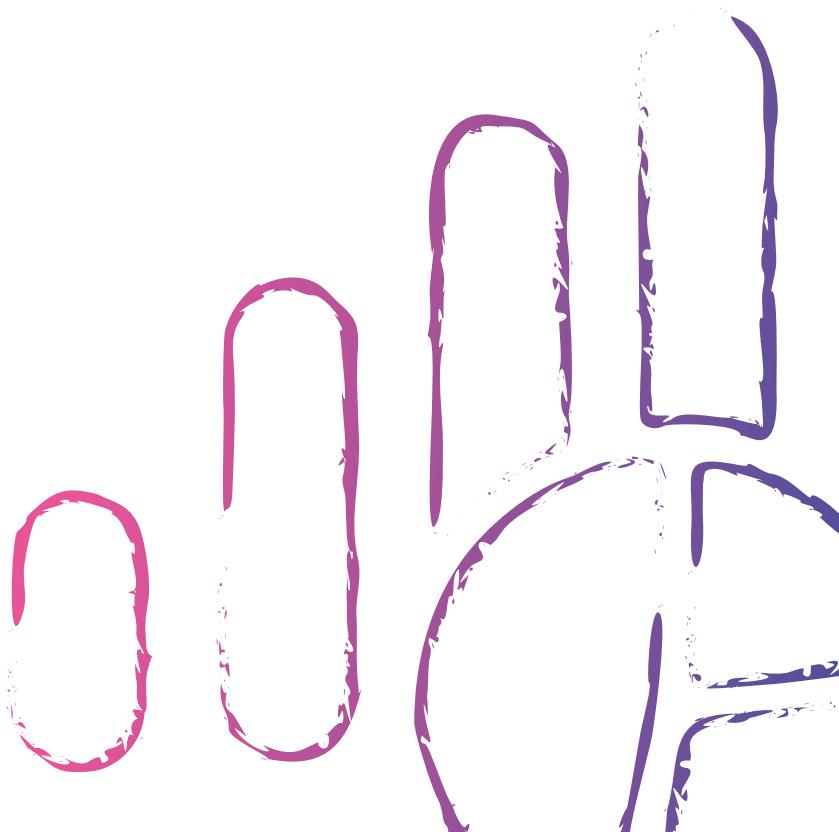
Luxms BI

Руководство разработчика

REST API

ЗАПРОСЫ ВО ВНЕШНИЕ ТАБЛИЦЫ

JAVASCRIPT API



2023-02-16

Оглавление

1 Протокол обмена данными Luxms BI DB API

1.1 Термины и определения

1.1.1 VCP

Visual Control Point, см. ТВК

1.1.2 ТВК

Точка Визуального Контроля. Координаты точки в многомерном пространстве измеряемых показателей вместе с её значением. Другими словами, это характеристики измеряемого значения показателя вместе с самим значением. Например, фраза “значение показателя `средняя температура`, измеренное `20 декабря 2015 года` в `Преображенской больнице` и равное 36 градусам” описывает одну ТВК. Эта ТВК имеет одно значение, равное 36-и градусам, и три характеристики, как представлено в таблице:

Характеристика	Значение характеристики	Описание
Метрика	Средняя Температура	Метрики отвечают на вопрос `Что?`. Каждая метрика имеет единицу измерения (размерность). Например: км/ч, рубли, штуки и т.д.
Место взятия замера	Преображенская больница	Места (объекты контроля) отвечают на вопрос `Где?`. Объекты контроля могут иметь географические координаты: широту и долготу, а также границы.
Время взятия замера	20 декабря 2015 года	Временные метки (периоды) отвечают на вопрос `Когда?` Периоды задают временной интервал, в течение которого верно значение ТВК. Периоды имеют дату начала и стандартизованную длительность, например: день, неделя, месяц и т.д.

1.2 Введение

Протокол Luxms BI DB API разработан для поддержки четырёх типов операций с данными:

- создание (Create)
- получение (Retrieve)
- изменение (Update)
- удаление (Delete)

Этот набор операций получил название CRUD и поддерживается всеми хранилищами данных в том или ином виде.

Протокол Luxms BI DB API разработан с использованием элементов архитектурного стиля REST. Для обмена данными между клиентом и сервером используется формат JSON.

Для передачи данных используется кодировка `UTF-8`.

Перед тем как подавать запросы по протоколу Luxms BI DB API, необходимо пройти аутентификацию в системе.

Для загрузки ТВК в Luxms BI, нужно предварительно подготовить/узнать значения характеристик этих Точек Визуального Контроля.

Внимание! Для корректного копирования примеров из настоящего руководства в консоль, рекомендуется использовать Acrobat Reader. Другие программы для чтения PDF файлов не поддерживают формат PDF в полном объёме и работают некорректно при копировании текста.

1.3 Перед началом работы

Предполагается, что примеры будут запускаться в терминале UNIX-подобной системы. Для запуска примеров из этой главы необходимо настроить переменную окружения Shell `luxmsbi_url`.

Например:

```
1 export luxmsbi_url='https://127.0.0.1:8080'
```

Также, нужно получить у системного администратора логин и пароль для доступа к датасетам на сервере Luxms BI. Если требуется не только читать данные, но и изменять их, то необходимо, чтобы у пользователя был соответствующий уровень доступа к датасету.

1.4 Структуры данных Luxms BI

Luxms BI хранит ТВК (значения показателей) в виде 3-х мерного OLAP куба. Другими словами, каждое значение показателя характеризуется тремя ключами (идентификаторами). Ключи позволяют ответить на вопросы “Что?”, “Где?”, “Когда?” по отношению к значению показателя. Для хранения дополнительной информации о характеристиках “Что?-Где?-Когда?” используются таблицы-справочники.

Идентификатор	Вопрос	Таблица-справочник	Ключ в таблице значений
Идентификатор метрики	Что?	metrics	metric_id
Идентификатор локации	Где?	locations	loc_id
Идентификатор периода	Когда?	periods	period_id

Упрощённая схема данных представлена на рисунке:

Рис. 1.1 **Упрощенная схема данных LuxmsBI**

1.4.1 Справочник метрик

Метрики отвечают на вопрос `Что?` по отношению к значению показателя. Другими словами, метрики характеризуют что именно измеряет то или иное значение. Подходящими названиями для метрик будут “Доходы”, “Средняя температура”, “Совокупная установленная мощность”.

Метрики в Luxms BI организованы в иерархию, при этом количество уровней вложенности технически не ограничено.

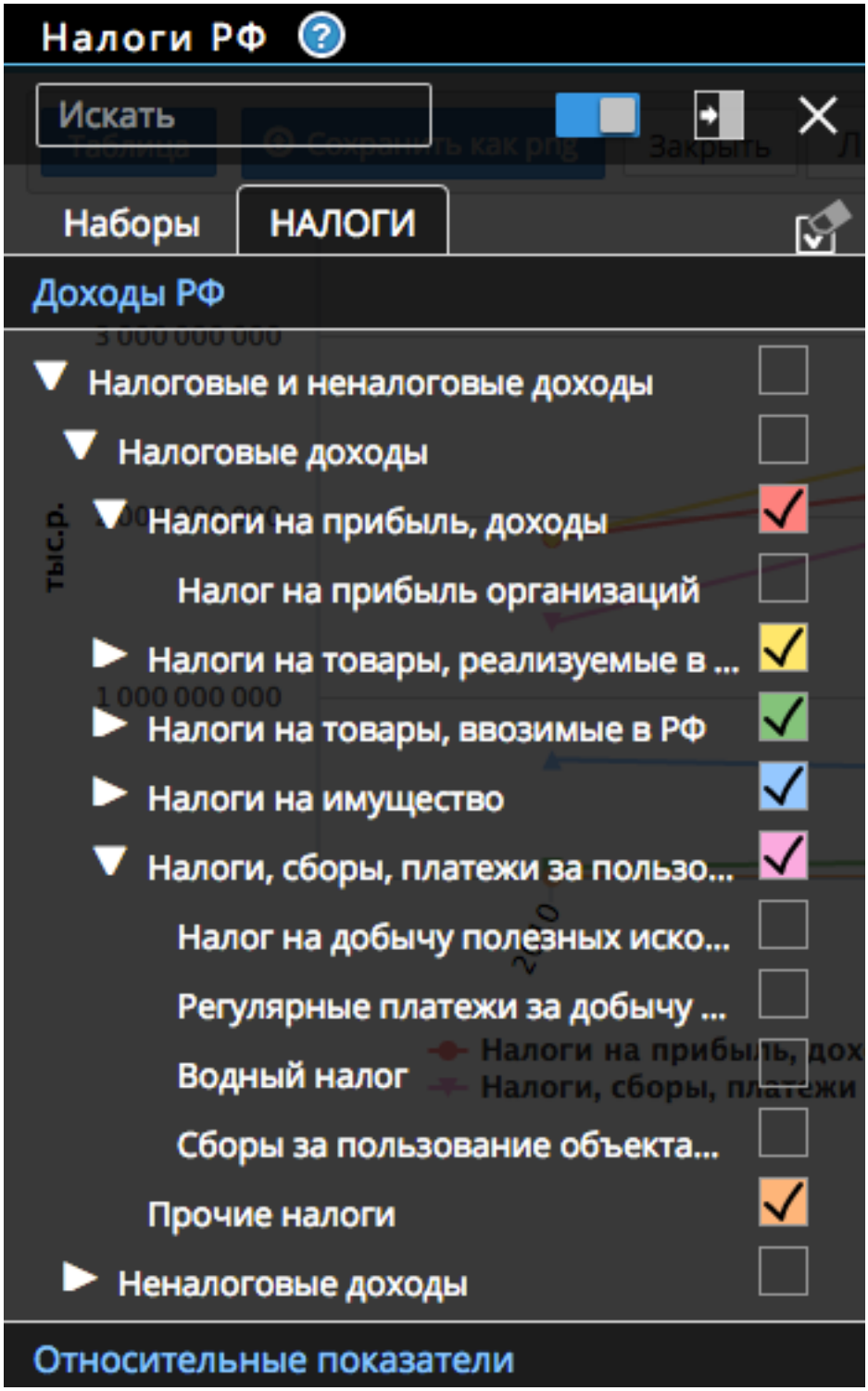


Рис. 1.2 **Пример иерархии метрик в Luxms BI**

Значения родительской метрики чаще всего являются агрегатами значений своих дочерних метрик. Например, значение для родительской метрики “Доходы” может быть суммой значений метрик “Процентные доходы”, “Операционные доходы” и “Прочие доходы”.

Основные поля, характеризующие метрики указаны в таблице:

Имя поля	Значение	Тип поля	Пример JSON
id	Идентификатор метрики	Целое	`{"id": 11}`
title	Название метрики	Строка	`{"title": "Доходы"}`
tree_level	Уровень метрики в дереве. У корня дерева `tree_level=0`	Целое	`{"tree_level": 1}`
parent_id	Ключ родительской метрики. У корня `parent_id=null`	Строка	`{"parent_id": null}`
is_hidden	Признак скрытия метрики в UI	1 или 0	`{"is_hidden": 0}`
unit_id	Ключ единицы измерения из таблицы units	Целое	`{"unit_id": 2}`
srt	Порядковый номер для сортировки в UI	Целое	`{"srt": 10}`

Примечание: Перечень полей может отличаться от описанных в документации и зависит от установленной версии Luxms BI. Недокументированные поля следует игнорировать во всех запросах Luxms BI DB API. Изменение значений недокументированных полей может привести к неожиданным побочным эффектам.

1.4.1.1 Справочник единиц измерений

В описании метрик используется поле `unit_id`, которое задаёт единицу измерения для значений. Например, это может быть килограмм, рубль или км/ч. Используемые в датасете единицы измерения хранятся в таблице `units`.

Если поле `unit_id` у метрики установлено в `null`, то пользователь не сможет выбрать эту метрику в панели метрик, а значит, не сможет вывести эту метрику на график. Метрики с неустановленным полем `unit_id` могут быть использованы для логической группировки метрик в панели метрик.

Основные поля, характеризующие единицы измерения указаны в таблице:

Имя поля	Значение	Тип поля	Пример JSON
id	Идентификатор единицы измерения	Целое	`{"id": 1}`
title	Название единицы измерения для административного интерфейса	Строка	`{"title": "килограммы"}`
value_prefix	Префикс, отображаемый в UI перед значением	Строка	`{"value_prefix": "$"}`
value_suffix	Суффикс, отображаемый в UI после значения	Строка	`{"value_suffix": "%"}`
tiny_title	Название единицы измерения, используемое в легенде	Строка	`{"tiny_title": "шт."}`
axis_title	Название единицы измерения, используемое для подписи осей графиков	Строка	`{"axis_title": "килограммы"}`

1.4.2 Справочник локаций

Локации отвечают на вопрос `Где?` по отношению к значению показателя. Другими словами, локации характеризуют где именно получено то или иное значение. Подходящими названиями для локаций будут “Южный филиал”, “Невский район”, “г. Москва”.

Локации в Luxms BI организованы в иерархию, при этом количество уровней вложенности технически не ограничено.

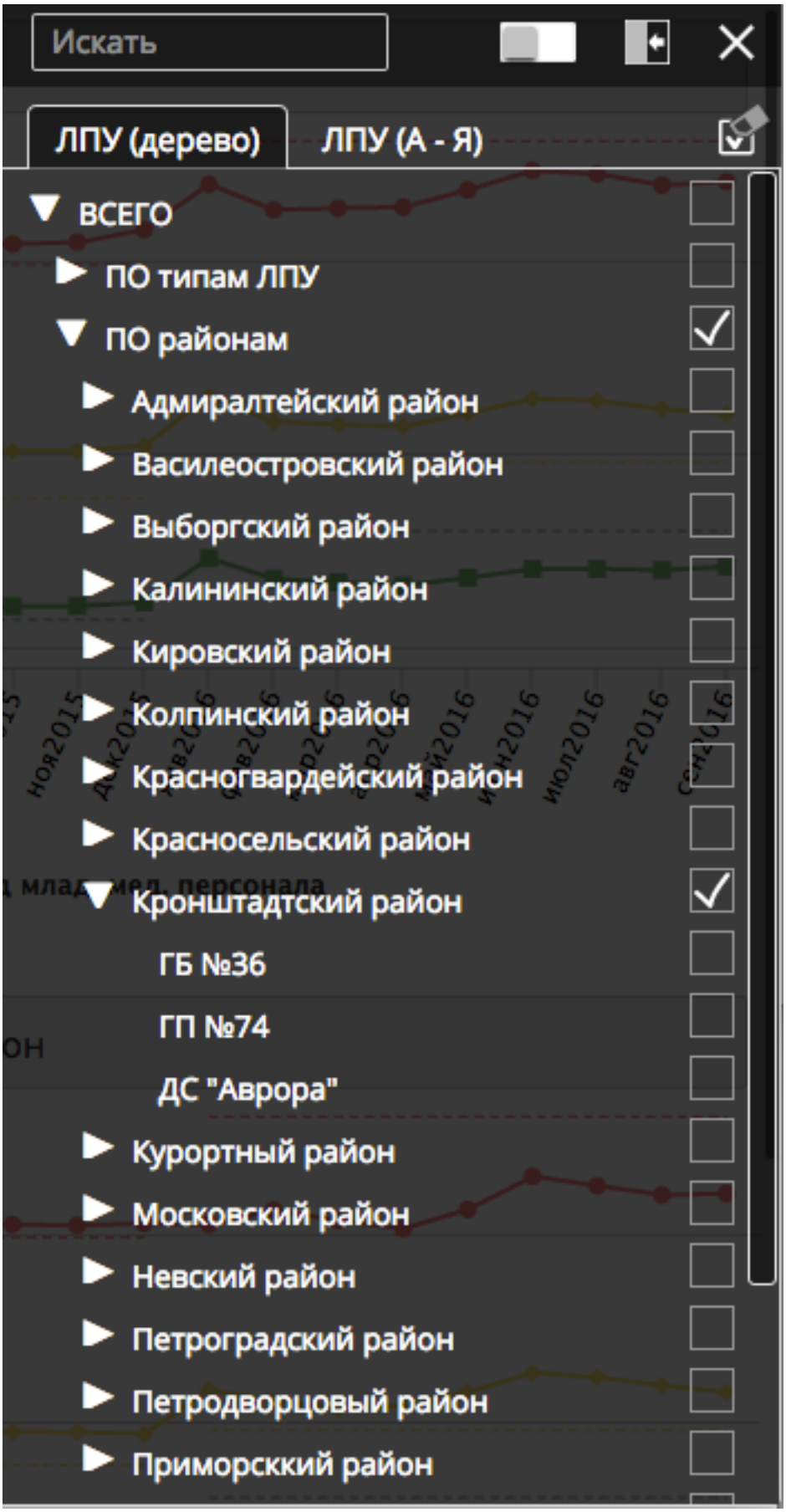


Рис. 1.3 **Пример иерархии локаций в Luxms BI**

Значения родительской локации чаще всего являются агрегатами значений своих дочерних локаций. Например, значение для родительской локации “г. Санкт-Петербург” может быть суммой значений локаций “Невский район”, “Центральный район” и “Кировский район”.

Основные поля, характеризующие локации указаны в таблице:

Имя поля	Значение	Тип поля	Пример JSON
id	Идентификатор локации	Целое	`{"id": 11}`
title	Название локации	Строка	`{"title": "офис"}`
tree_level	Уровень локации в дереве. У корня дерева `tree_level=0`	Целое	`{"tree_level": 1}`
parent_id	Ключ родительской локации. У корня `parent_id=null`	Строка	`{"parent_id": null}`
is_hidden	Признак скрытия локации в UI	1 или 0	`{"is_hidden": 0}`
latitude	Долгота в десятичных градусах	Рациональное	`{"latitude": 37.61556}`
longitude	Широта в десятичных градусах	Рациональное	`{"longitude": 55.75222}`
srt	Порядковый номер для сортировки в UI	Целое	`{"srt": 10}`

Примечание: Перечень полей может отличаться от описанных в документации и зависит от установленной версии Luxms BI. Недокументированные поля следует игнорировать во всех запросах Luxms BI DB API. Изменение значений недокументированных полей может привести к неожиданным побочным эффектам.

1.4.3 Справочник периодов

Периоды отвечают на вопрос `Когда?` по отношению к значению показателя. Другими словами, периоды характеризуют когда именно получено то или иное значение. Подходящими названиями для периодов будут “Первый квартал 2016 года”, “12 часов”, “март 2011г.”.

Основные поля, характеризующие периоды указаны в таблице:

Имя поля	Значение	Тип поля	Пример
id	Идентификатор периода	64 битное целое	`{"id": "2020010100000054"}`

Имя поля	Значение	Тип поля	Пример
title	Название периода	Строка	`{"title": "01.01.2020"}`
start_time	Дата начала периода	Строка/Дата SQL	`{"start_time" : "2020-01-01 00:00:00"}`
period_type	Тип периода, характеризующий длительность. Поддерживаются интервалы от секунды до года.	Целое	`{"period_type":4}`

Периоды характеризуются временем начала периода `start_time` и длительностью `period_type`. В Luxms BI можно использовать 8 стандартных типов периодов, от секунды до года, как описано в разделе Справочник типов периодов.

Внимание! Идентификатор периода представляет собой 64 битное целое число, в котором используется 53 бита. Такое целое число может быть корректно представлено средствами JavaScript без потери точности. По историческим причинам протокол Luxms BI DB API передаёт идентификаторы периодов в виде строки.

1.4.3.1 Справочник типов периодов

Идентификатор типа периода `period_type` используется для кодирования стандартной длительности периодов в Luxms BI.

Используются следующие типы периодов:

идентификатор типа периода	идентификатор гранулярности	длительность периода
1	81	Секунды
2	72	Минуты
3	63	Часы
4	54	Дни
5	45	Недели
6	36	Месяцы
7	27	Кварталы
8	18	Годы

Идентификатор гранулярности однозначно соответствует идентификатору типа периода и используется при кодировании идентификатора периода в виде 64-битного числа.

1.5 Аутентификация

Для начала работы с Luxms BI DB API необходимо подать `POST` запрос на аутентификацию с указанием имени пользователя и пароля.

URL: `/api/auth/login`

Параметры запроса:

Имя параметра	Тип параметра	Пример
username	строка	username=developer
password	строка	password=devpass

Пример запроса:

```
curl -k -v \--
header 'Content-type: application/x-www-form-urlencoded' \--
data 'username=admin&password=abcd' \-
X POST "${luxmsbi_url}/api/auth/login"
```

Внимание! При запуске примеров, скопированных из этого документа, убедитесь, что Вы выполнили шаги, описанные в разделе Перед началом работы.

Пример ответа при успешной аутентификации:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Set-Cookie: LuxmsBI-User-Session=97f948ed-58ae-414c-b6fe-58e8105a29f4; expires=
  Thu, 03 Nov 2016 09:55:25 GMT; path=/;

{
 "id" : "1",
 "username" : "admin",
 "email" : "blackhole@localhost.localdomain",
 "access_level" : "admin",
 "name" : "Default Admin",
 "config" : {},
 "sys_config" : {}
}
```

Для корректной пары логин/пароль возвращается HTTP статус `200` и в заголовке `Set-Cookie` возвращается идентификатор сессии (cookie). Предоставленный идентификатор сессии необходимо использовать в последующих запросах к сервису Luxms BI DB API.

Полученный при успешной аутентификации идентификатор сессии действителен в течение суток с момента получения. Время действия идентификатора сессии может быть изменено администратором Luxms BI.

В теле HTTP ответа передаётся информация о пользователе, который прошёл аутентификацию.

При ошибке аутентификации (логин и/или пароль неверны, доступ пользователю запрещен) возвращается HTTP статус `403 Forbidden`.

Пример ответа при ошибке аутентификации:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json; charset=utf-8

{"key":"WRONG_PASSWORD_OR_USER_NAME", "type":"ERR", "message":"Неверный логин
  и/или пароль."}
```

В теле HTTP ответа передаётся сообщение об ошибке аутентификации в формате JSON.

1.6 Авторизация

Для авторизации запроса необходимо передать на сервер полученный на стадии аутентификации идентификатор сессии. Это можно сделать с помощью HTTP заголовков `Cookie` или `LuxmsBI-User-Session`. Если в запросе указаны оба заголовка, то проверяется только заголовок `Cookie`, а заголовок `LuxmsBI-User-Session` игнорируется.

Для запуска примеров из этой главы необходимо присвоить переменной окружения Shell `luxmsbi_cookie` значение cookie, полученное в процессе аутентификации.

Например:

```
export luxmsbi_cookie='4350779b-5a72-4523-a87a-61b59295b18d'
```

1.6.1 Авторизация с помощью заголовка Cookie:

Для авторизации необходимо передать на сервер стандартный заголовок `Cookie`, как описано в rfc6265.

Пример заголовка:

```
Cookie: LuxmsBI-User-Session=4350779b-5a72-4523-a87a-61b59295b18d
```

1.6.2 Авторизация с помощью заголовка LuxmsBI-User-Session:

Для авторизации необходимо передать на сервер заголовок `LuxmsBI-User-Session`

Пример заголовка:

```
LuxmsBI-User-Session: 4350779b-5a72-4523-a87a-61b59295b18d
```

1.7 Структура URL

Точкой входа для запросов является URL путь `/api/db/`. Далее следует идентификатор датасета и имя таблицы, с которой будет работать запрос. Символ точки `.` используется в качестве разделителя между идентификатором датасета и именем таблицы.

В Luxms BI DB API используется два типа URL :

1. `/api/db/${dataset}.${table}`
2. `/api/db/${dataset}.${table}/${object_id}`

сегмент URL	Значения	Пример
`${dataset}`	GUID, имя схемы или id датасета	`4ebc64b0-39ea-4b62-a28d-722724daaa0a` или `ds_180` или `12`
`${table}`	имя таблицы	`locations` или `data`
`${object_id}`	идентификатор записи в таблице	`192` или `param1`

Тип операции задаётся методом HTTP запроса в соответствие с таблицей:

HTTP метод	Тип операции	Наличие HTTP Body в запросе	Тип URL
GET	Получение объектов	нет	`1` или `2`
POST	Создание объектов	да	`1`
PUT	Редактирование объектов	да	`2`
DELETE	Удаление объектов	нет	`2`

1.8 Кодировка данных

Luxms BI DB API использует формат JSON и кодировку UTF-8 в запросах и ответах.

1.9 Получение информации

1.9.1 Получение идентификаторов объектов из URL

1.9.1.1 Идентификатор набора данных

Набор данных в Luxms BI DB API идентифицируется тремя способами:

1. целочисленный id датасета, уникальный в рамках одного сервера Luxms BI.
2. текстовое имя схемы, уникальное в рамках одного сервера Luxms BI. В имени схемы используются только буквы латиницы, символ подчёркивания и цифры.
3. GUID, сохраняющийся при переносе датасета с одного сервера Luxms BI на другой.

Имя схемы датасета можно получить из URL при навигации в клиентской части Luxms BI. Например, из ссылки `http://demo.luxmsbi.com/#/ds/ds_demo20/dashboards?locations=3&=` можно получить имя схемы `ds_demo20`.

1.9.1.2 Идентификатор метрики

Это целочисленный идентификатор, который можно получить из ссылки. Например, из ссылки `http://demo.luxmsbi.com/#/ds/ds_demo12/trends?metrics=162,150&locations=3&period.end=2013120100000036` можно получить идентификаторы метрик `162` и `150`. Идентификаторы метрик кодируются с помощью параметра URL `metrics`.

1.9.1.3 Идентификатор локации

Это целочисленный идентификатор, который можно получить из ссылки. Например, из ссылки `http://demo.luxmsbi.com/#/ds/ds_demo12/trends?metrics=162,150&locations=3&period.end=2013120100000036` можно получить идентификатор локации `3`. Идентификаторы локаций кодируются с помощью параметра URL `locations`.

1.9.1.4 Идентификатор периода

Это целочисленный идентификатор, который можно получить из ссылки. Например, из ссылки `http://demo.luxmsbi.com/#/ds/ds_demo12/trends?metrics=162,150&locations=3&period.end=2013120100000036` можно получить идентификатор периода `2013120100000036`. Идентификаторы периодов кодируются с помощью параметра URL `period.start` для начального периода и `period.end` для конечного периода.

1.9.2 Получение информации с помощью запроса GET

Данные Luxms BI можно получить с помощью HTTP запроса `GET`. Для этого нужно указать датасет и имя таблицы, из которой требуется получить данные:

```
1 /api/db/${dataset}.${table}
```

Например, чтобы получить справочник единиц измерения в датасете с именем схемы `ds_tests`, нужно использовать URL:

```
1 /api/db/ds_tests.units
```

В качестве идентификатора датасета можно использовать любой идентификатор, например целочисленный:

```
1 /api/db/123.units
```

Также можно использовать и GUID датасета:

```
1 /api/db/5b3a1c32-5a91-4bdc-9036-cdb28768ccf4.units
```

Все эти способы указания датасета равнозначны и возвращают одинаковый результат.

Для авторизации запроса, необходимо указать идентификатор сессии, полученный на стадии аутентификации. Например, это можно сделать, отправив заголовок `LuxmsBI-User-Session:`

```
curl -k -v \--
header "LuxmsBI-User-Session: ${luxmsbi_cookie}" \-
X GET "${luxmsbi_url}/api/db/ds_demo12.units"
```

Пример ответа:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

[
{
"id":2,
"value_prefix":null,
"value_suffix":"тыс.руб.",
"title":"Тыс. рублей",
"tiny_title":"тыс.руб.",
"axis_title":"тыс.руб.",
"updated":"2015-02-25T16:34:06.604562+03:00",
"created":"2015-04-03T12:36:22.923085+03:00"
},
{
"id":12,
"value_prefix":null,
"value_suffix":"тыс.мин.",
"title":"Тыс. минут",
"tiny_title":"тыс.мин.",
"axis_title":"тыс.мин.",
"updated":"2015-02-25T16:34:06.604562+03:00",
"created":"2015-04-03T12:36:22.923085+03:00"
}]
```

В теле ответа можно увидеть список единиц измерения с указанием их идентификаторов.

Используя `GET` запросы можно узнать идентификаторы метрик, локаций и периодов, чтобы в дальнейшем использовать их для вставки значений в Luxms BI.

1.10 Вставка данных в Luxms BI

Для вставки данных в Luxms BI следует использовать HTTP запросы `POST`.

1.10.1 Вставка данных с помощью запроса POST

В URL запроса нужно указать датасет и имя таблицы, в которую будет происходить вставка данных:

```
/api/db/${dataset}.${table}
```

В теле одного HTTP запроса можно указать один и более объектов для добавления в датасет. Для ускорения процесса вставки данных рекомендуется в теле одного HTTP запросе

передавать сразу несколько объектов. При этом либо все переданные в одном HTTP запросе объекты будут добавлены, либо ни один из них не будет добавлен из-за ошибки.

Для авторизации запроса, необходимо указать идентификатор сессии, полученный на стадии аутентификации. Например, это можно сделать, отправив заголовок `LuxmsBI-User-Session:`

Для корректной обработки запроса на стороне сервера необходимо указывать тип и кодировку передаваемых данных:

```
Content-type: application/json; charset=utf-8
```

При вставке одного объекта в теле HTTP запроса необходимо использовать JSON нотацию объекта `{}`, при вставке нескольких объектов в теле HTTP запроса используется JSON нотация списка `[]`.

При попытке вставить объект, который нарушает ограничение уникальности ключа (например: совпадение `id`), сервер вернёт HTTP статус 409 и в теле ответа вернёт запись, которая стала причиной конфликта. Такое поведение реализовано только для вставки одиночных объектов в нотации JSON объекта `{}`.

1.10.1.1 Добавление одной ТВК

URL: `/api/db/${dataset}.data`

В URL необходимо указать идентификатор датасета. В качестве идентификатора можно использовать GUID, имя схемы или id схемы.

Данные для вставки кодируются в формате JSON, как указано в таблице:

Имя поля	Значение	Тип поля	Пример
metric_id	Идентификатор метрики	строка	`{"metric_id" : 1}`
loc_id	Идентификатор локации	целое число	`{"loc_id":123 }`
period_id	Идентификатор периода	строка (64бит целое)	`{"period_id" : "2013120100000036"}`
val	Значение показателя	число с плавающей точкой	`{"value":123.23}`

Пример запроса:

```
curl -k -v \--
data '{"metric_id": 131, "loc_id":1, "period_id":2013120100000036, "val":
  123.23}' \--
header "LuxmsBI-User-Session: ${luxmsbi_cookie}" \--
```

```
header 'Content-type: application/json; charset=utf-8' \-
X POST "${luxmsbi_url}/api/db/ds_tests.data/"
```

Пример тела ответа при успешной вставке:

```
{"loc_id":1, "period_id": 2013120100000036, "val":123.23, "metric_id":131}
```

Внимание! При добавлении ТВК не рекомендуется указывать `id` записей. Указание конкретных `id` может привести к ошибкам при последующей вставке других ТВК.

1.10.1.2 Добавление нескольких ТВК

Запрос аналогичен вставке одной ТВК, но в теле запроса передаётся массив (список) объектов для вставки.

Пример запроса:

```
curl -k -v \--
data '[
{"metric_id": 131,"loc_id":10, "period_id":"2013120100000036", "val":123.23},
{"metric_id": 131,"loc_id":11, "period_id":"2013120200000036", "val":0.87}]
' \--
header "LuxmsBI-User-Session: ${luxmsbi_cookie}" \--
header 'Content-type: application/json; charset=utf-8' \-
X POST "${luxmsbi_url}/api/db/ds_tests.data"
```

Пример тела ответа при успешной вставке:

```
[
{"loc_id":10, "period_id": 2013120100000036, "val":123.23, "metric_id":131},
{"loc_id":11, "period_id": 2013120200000036, "val":0.87, "metric_id":131}]
```

Внимание! При добавлении ТВК не рекомендуется указывать `id` записей. Указание конкретных `id` может привести к ошибкам при последующей вставке других ТВК.

1.10.1.3 Добавление одной метрики с указанием id

URL: `/api/db/${dataset}.metrics`

В URL необходимо указать идентификатор датасета. В качестве идентификатора можно использовать GUID, имя схемы или id схемы.

Данные для вставки кодируются в формате JSON, как указано в таблице:

Имя поля	Значение	Тип поля	Пример JSON
id	Идентификатор метрики	Целое	`{"id": 11}`

Имя поля	Значение	Тип поля	Пример JSON
title	Название метрики	Строка	`{"title": "Доходы"}`
tree_level	Уровень метрики в дереве. У корня дерева `tree_level=0`	Целое	`{"tree_level": 1}`
parent_id	Ключ родительской метрики. У корня `parent_id=null`	Строка	`{"parent_id": null}`
is_hidden	Признак скрытия метрики в UI	1 или 0	`{"is_hidden": 0}`
unit_id	Ключ единицы измерения из таблицы units	Целое	`{"unit_id": 2}`
srt	Порядковый номер для сортировки в UI	Целое	`{"srt": 10}`

Внимание! Система назначит идентификатор новой метрике автоматически, если в запросе не будет указано значение для ключа `id`.

Пример запроса:

```
curl -k -v \--
data '{"id": 98765, "title": "Средняя температура", "unit_id": 1}' \--
header "LuxmsBI-User-Session: ${luxmsbi_cookie}" \--
header 'Content-type: application/json; charset=utf-8' \-
X POST "${luxmsbi_url}/api/db/ds_tests.metrics"
```

Пример ответа в случае ошибки:

```
HTTP/1.1 409 Conflict
Content-Type: application/json; charset=utf-8

{
"id":98765,
"parent_id":null,
"alt_id":"98765",
"title":"Минимальная температура",
"tree_level":0,
"unit_id":1,
"srt":2147483647,
"is_hidden":0
}
```

В данном случае ошибка произошла из-за конфликта уникальности ключа метрики. Метрика с `id=98765` уже существует.

Пример ответа в случае ошибки:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json; charset=utf-8

{
"title": "Ошибка SQL webapi.route",
"message": "отношение \"ds_tests.metrics\" не существует (символ 13)"
}
```

Статус ответа 500 используется при любых внутренних ошибках сервера.

Пример тела ответа в случае успеха:

```
{
"id":98765,
"parent_id":null,
"alt_id":"98765",
"title":"Средняя температура",
"tree_level":0,
"unit_id":1,
"srt":2147483647,
"is_hidden":0
}
```

1.10.1.4 Добавление одной метрики без указания id

URL: `/api/db/${dataset}.metrics`

При вставке метрик без указания `id`, Luxms BI присвоит целочисленный `id` автоматически.

Если часть метрик добавляется с явным указанием `id`, а другая честь с использованием автоназначения, это может привести к проблеме неуникальных ключей в таблице `metrics`.

Пример запроса без указания `id`:

```
curl -k -v \--
data '{"title": "Средняя температура", "unit_id": 1}' \--
header "LuxmsBI-User-Session: ${luxmsbi_cookie}" \--
header 'Content-type: application/json; charset=utf-8' \-
X POST "${luxmsbi_url}/api/db/ds_tests.metrics"
```

Пример тела ответа в случае успеха:

```
{
"id":132,
"parent_id":null,
"alt_id":"132",
"title":"Средняя температура",
"tree_level":0,
"unit_id":1,
"srt":2147483647,
```

```
"is_hidden":0
}
```

1.10.1.5 Добавление одной локации с указанием id

URL: `/api/db/${dataset}.locations`

В URL необходимо указать идентификатор датасета. В качестве идентификатора можно использовать GUID, имя схемы или id схемы.

Данные для вставки кодируются в формате JSON, как указано в таблице:

Имя поля	Значение	Тип поля	Пример JSON
id	Идентификатор локации	Целое	`{"id": 11}`
title	Название локации	Строка	`{"title": "офис"}`
tree_level	Уровень локации в дереве. У корня дерева `tree_level=0`	Целое	`{"tree_level": 1}`
parent_id	Ключ родительской локации. У корня `parent_id=null`	Строка	`{"parent_id": null}`
is_hidden	Признак скрытия локации в UI	1 или 0	`{"is_hidden": 0}`
latitude	Долгота в десятичных градусах	Рациональное	`{"latitude": 37.61556}`
longitude	Широта в десятичных градусах	Рациональное	`{"longitude": 55.75222}`
srt	Порядковый номер для сортировки в UI	Целое	`{"srt": 10}`

Внимание! Система назначит идентификатор новой локации автоматически, если в запросе не будет указано значение для ключа `id`.

Пример запроса:

```
curl -k -v \--
data '{"id": 987650, "title": "Головной офис", "latitude": 37.61556,
  "longitude": 55.75222}' \--
header "LuxmsBI-User-Session: ${luxmsbi_cookie}" \--
header 'Content-type: application/json; charset=utf-8' \-
X POST "${luxmsbi_url}/api/db/ds_tests.locations"
```

Пример ответа в случае ошибки:

```
HTTP/1.1 409 Conflict
Content-Type: application/json; charset=utf-8

{
"id":987650,
"title":"Офис Тайга",
"latitude":37.61556,
"longitude":55.75222,
"parent_id":null,
"tree_level":0,
"srt":2147483647,
"is_hidden":0
}
```

В данном случае ошибка произошла из-за конфликта уникальности ключа локации. Локация с `id=987650` уже существует.

Пример ответа в случае ошибки:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json; charset=utf-8

{
"title": "Ошибка SQL webapi.route",
"message": "отношение \"ds_tests.locations\" не существует (символ 13)"
}
```

Статус ответа 500 используется при любых внутренних ошибках сервера.

Пример тела ответа в случае успеха:

```
{
"id":987650,
"title":"Головной офис",
"latitude":37.61556,
"longitude":55.75222,
"parent_id":null,
"tree_level":0,
"srt":2147483647,
"is_hidden":0
}
```

1.10.1.6 Добавление одного периода

URL: `/api/db/${dataset}.periods`

В URL необходимо указать идентификатор датасета. В качестве идентификатора можно использовать GUID, имя схемы или id схемы.

Данные для вставки кодируются в формате JSON, как указано в таблице:

Имя поля	Значение	Тип поля	Пример
id	Идентификатор периода	64 битное целое	`{"id": 2013120200000036}`
title	Название периода	Строка	`{"title": "1кв. 2015"}`
start_time	Дата начала периода	Строка/Дата SQL	`{"start_time" : "2015-12-01"}`
period_type	Тип периода, характеризующий длительность. Поддерживаются интервалы от секунды до года.	Целое	`{"period_type":6}`

Внимание! Алгоритмы визуализации данных используют особые свойства `id` периодов. Для корректной и эффективной работы клиентской части рекомендуется всегда пользоваться автоматической генерацией `id` периодов и никогда не указывать их при вставке данных.

Внимание! В Luxms BI используются специальные id периодов, которые могут быть корректно представлены средствами Javascript в качестве целых чисел без потери точности.

Пример запроса:

```
curl -k -v \--
data '{"title":"12-2010", "start_time": "2010-12-01", "period_type":6}' \--
header "LuxmsBI-User-Session: ${luxmsbi_cookie}" \--
header 'Content-type: application/json; charset=utf-8' \-
X POST "${luxmsbi_url}/api/db/ds_tests.periods"
```

Пример ответа в случае ошибки:

```
HTTP/1.1 409 Conflict
Content-Type: application/json; charset=utf-8

{
"id": 2010120100000036,
"period_type": 6,
"start_time": "2010-12-01T00:00:00+03:00",
"qty":1,
"title": "12-2010",
"updated": "2016-06-29T15:34:38.740998+03:00",
"created": "2016-06-29T15:34:38.740998+03:00"
}
```

В данном случае ошибка произошла из-за конфликта уникальности периода. Период с указанной датой и типом уже существует и имеет `id` 2010120100000036.

Пример ответа в случае ошибки:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json; charset=utf-8

{
"title": "Ошибка SQL webapi.route",
"message": "отношение \"ds_tests.periods\" не существует (символ 13)"
}
```

Статус ответа 500 используется при любых внутренних ошибках сервера.

Пример тела ответа в случае успеха:

```
{
 "id":2010120100000036,
 "parent_id":null,
 "tree_level":0,
 "period_type":6,
 "qty":1,
 "start_time":"2010-12-01T00:00:00",
 "title":"12-2010",
 "updated":"2020-09-30T11:59:28.629989+00:00",
 "created":"2020-09-30T11:59:28.629989+00:00"
}
```

1.11 Изменение данных в Luxms BI

Для изменения данных в Luxms BI следует использовать HTTP запросы `PUT`.

1.11.1 Изменение данных с помощью запроса PUT

1.11.1.1 Изменение значения ТВК по условию

URL: `/api/db/${dataset}.data/${LPE}`

Выборку записей для обновления можно производить не только указывая `id` записи, но и с помощью выражений `Lux Path Expressions`, которые позволяют фильтровать записи по условию. Например, можно указать значения ключей `Что?-Где?-Когда?`, которые уникальным образом идентифицируют запись в таблице `data`, и тем самым, выбрать единственную запись для обновления.

Для фильтрации записей используется функция `.filter`. В качестве аргумента функции нужно указать выражение, накладывающее условия на поля записи.

Допускается использовать логические операции `and`, `or`, `and`.

Допускается использовать операции сравнения: `>`, `<`, `!=` и т.д.

Двойной амперсанд `&&` можно использовать вместо `and` (логическое `И`).

Символы `||` можно использовать вместо `or` (логическое `ИЛИ`).

В выражениях допускается использование пробелов для удобства чтения и скобок для создания сложных запросов.

Таким образом, условие SQL:

```
WHERE
 loc_id = 4 AND
 period_id = 2010120100000036 AND
 metric_id = 17
```

можно записать с помощью Lux Path Expression:

```
.filter(loc_id=4 && period_id= 2010120100000036 and metric_id=17)
```

Для выполнения запроса, это выражение `LPE` необходимо указать в сегменте URL на месте `${LPE}`:

```
PUT /api/db/ds_tests.data/.filter(loc_id=4 && period_id= 2010120100000036 and
  metric_id=17)
Content-type: application/json; charset=utf-8

{"val":19.74}
```

Пример для выполнения в командной строке:

```
curl -k -v --globoff \--
data '{"val":20.02}' \--
header "LuxmsBI-User-Session: ${luxmsbi_cookie}" \--
header 'Content-type: application/json; charset=utf-8' \-
X PUT "${luxmsbi_url}/api/db/ds_tests.data/.filter(loc_id=4 && period_id=
  2010120100000036 and metric_id=17)"
```

В ответе всегда возвращается список записей в состоянии после внесения изменений. В данном случае возвращается список из одного элемента, так как в запросе был указан уникальный `id` записи.

Пример ответа:

```
[
{
 "loc_id":4,
 "period_id": 2010120100000036,
 "updated":"2016-06-16T12:37:24.998473+03:00",
 "created":"2015-04-03T12:36:23.491845+03:00",
 "val":20.02,
 "metric_id":17
}]
```

1.11.1.2 Изменение метрики с указанием id

URL: `/api/db/${dataset}.metrics/${id}`

Для изменения одной определённой метрики нужно знать её `id`.

`id` метрики указывается в URL запроса в сегменте `${id}`, а список изменяемых полей и их значений передаётся в теле запроса в формате JSON.

Например, чтобы изменить название метрики с `id` = `10` в датасете `ds_tests` на `ЗАМЕТНОЕ название`, нужно выполнить такой HTTP запрос:

```
PUT /api/db/ds_tests.metrics/10
Content-type: application/json; charset=utf-8

{"title": "ЗАМЕТНОЕ название"}
```

Пример для выполнения в командной строке:

```
curl -k -v \--
data '{"title": "ЗАМЕТНОЕ название"}' \--
header "LuxmsBI-User-Session: ${luxmsbi_cookie}" \--
header 'Content-type: application/json; charset=utf-8' \-
X PUT "${luxmsbi_url}/api/db/ds_tests.metrics/10"
```

В ответе всегда возвращается список метрик в состоянии после внесения изменений. В данном случае возвращается список из одного элемента, так как в запросе был указан уникальный `id` метрики.

Пример ответа:


```
[
{
 "id":10,
 "parent_id":25,
 "alt_id":"I11",
 "title":"ЗАМЕТНОЕ название",
 "tree_level":2,
 "unit_id":11,
 "srt":90,
 "is_hidden":0,
 "updated":"2016-08-09T19:55:53.239045+03:00",
 "created":"2015-04-03T12:36:23.346958+03:00"
}]
```


В одном запросе можно поменять сразу несколько полей метрики. Например, для изменения и названия, и порядка сортировки, нужно в теле сообщения передать такой JSON:

```
{"title": "ЗАМЕТНОЕ название", "srt": 1}
```

1.11.1.3 Изменение локации с указанием id

URL: `/api/db/${dataset}.locations/${id}`

Для изменения одной определённой локации нужно знать её `id`.

`id` локации указывается в URL запроса в сегменте `${id}`, а список изменяемых полей и их значений передаётся в теле запроса в формате JSON.

Например, чтобы изменить название локации с `id` = `10` в датасете `ds_tests` на `Головной офис`, нужно выполнить такой HTTP запрос:

```
PUT /api/db/ds_tests.locations/10
Content-type: application/json; charset=utf-8

{"title": "Головной офис"}
```

Пример для выполнения в командной строке:

```
curl -k -v \--
data '{"title": "Головной офис"}' \--
header "LuxmsBI-User-Session: ${luxmsbi_cookie}" \--
header 'Content-type: application/json; charset=utf-8' \-
X PUT "${luxmsbi_url}/api/db/ds_tests.locations/10"
```

В ответе всегда возвращается список локаций в состоянии после внесения изменений. В данном случае возвращается список из одного элемента, так как в запросе был указан уникальный `id` локации.

Пример ответа:


```
[
 {
  "id":10,
  "title":"Головной офис",
  "latitude":57.8136,
  "longitude":28.3496,
  "parent_id":1,
  "level":1,
  "srt":0,
  "is_hidden":0,
  "tree_level":1,
  "updated":"2016-06-16T12:37:24.998473+03:00",
  "created":"2015-04-03T12:36:23.22108+03:00"
}]
```


В одном запросе можно поменять сразу несколько полей локации. Например, для изменения названия, широты и долготы, нужно в теле сообщения передать такой JSON:

```
{
"title": "Головной офис",
"latitude": 57.8,
```

```
"longitude": 28.3
}
```

1.11.1.4 Изменение периода с указанием id

URL: `/api/db/${dataset}.periods/${id}`

Для изменения одного определённого периода нужно знать его `id`.

`id` периода указывается в URL запроса в сегменте `${id}`, а список изменяемых полей и их значений передаётся в теле запроса в формате JSON.

Например, чтобы изменить название периода с `id` = `2010120100000036` в датасете `ds_tests` на `декабрь 2010`, нужно выполнить такой HTTP запрос:

```
PUT /api/db/ds_tests.periods/2010120100000036
Content-type: application/json; charset=utf-8

{"title": "декабрь 2010"}
```

Пример для выполнения в командной строке:

```
curl -k -v \--
data '{"title": "2000 год"}' \--
header "LuxmsBI-User-Session: ${luxmsbi_cookie}" \--
header 'Content-type: application/json; charset=utf-8' \-
X PUT "${luxmsbi_url}/api/db/ds_tests.periods/2010120100000036"
```

В ответе всегда возвращается список периодов в состоянии после внесения изменений. В данном случае возвращается список из одного элемента, так как в запросе был указан уникальный `id` периода.

Пример ответа:

```
[
 {
"id": 2010120100000036,
"title":"декабрь 2010",
"period_type":6,
"start_time":"2010-12-01T00:00:00+03:00",
"qty":1,
"updated":"2016-08-09T19:44:10.147325+03:00",
"created":"2016-08-09T19:41:10.724793+03:00"
}]
```

В одном запросе можно поменять сразу несколько полей периода. Например, для изменения названия и времени начала периода, нужно в теле сообщения передать такой JSON:

```
{
"title": "январь 2000",
```

```
"start_time":"2000-01-01"
}
```

1.12 Удаление данных из Luxms BI

Для удаления данных из Luxms BI следует использовать HTTP запросы `DELETE`.

1.12.1 Удаление данных с помощью запроса DELETE

1.12.1.1 Удаление ТВК по условию

URL: `/api/db/${dataset}.data/${LPE}`

Удаление записей можно производить не только указывая `id` записи, но и с помощью выражений `Lux Path Expressions`, которые позволяют фильтровать записи по условию. Например, можно указать значения ключей `Что?-Где?-Когда?`, которые уникальным образом идентифицируют запись в таблице `data`, и тем самым, выбрать единственную запись для удаления.

Для фильтрации записей используется функция `.filter`. В качестве аргумента функции нужно указать выражение, накладывающее условия на поля записи.

Допускается использовать логические операции `and`, `or`, `and`.

Допускается использовать операции сравнения: `>`, `<`, `!=` и т.д.

Двойной амперсанд `&&` можно использовать вместо `and` (логическое `И`).

Символы `||` можно использовать вместо `or` (логическое `ИЛИ`).

В выражениях допускается использование пробелов для удобства чтения и скобок для создания сложных запросов.

Таким образом, условие SQL:

```
WHERE
 loc_id = 4 AND
 period_id = 2010120100000036 AND
 metric_id = 17
```

можно записать с помощью Lux Path Expression:

```
.filter(loc_id=4 && period_id=2010120100000036 and metric_id=17)
```

Для выполнения запроса, это выражение `LPE` необходимо указать в сегменте URL на месте `${LPE}`:

```
DELETE /api/db/ds_tests.data/.filter(loc_id=4 && period_id=2010120100000036 and
  metric_id=17
```

Пример для выполнения в командной строке:

```
curl -k -v --globoff \--
header "LuxmsBI-User-Session: ${luxmsbi_cookie}" \-
X DELETE "${luxmsbi_url}/api/db/ds_tests.data/.filter(loc_id=4 && period_id=
  2010120100000036 and metric_id=17)"
```

В случае удаления хотя бы одной записи, возвращается статус 200 и пустой список в качестве тела ответа.

Пример тела ответа в случае успешного удаления записи:

```
[]
```

Пример ответа, в случае, если по условию не найдено ни одной записи:

```
HTTP/1.1 404 Not Found
Content-Type: application/json; charset=utf-8

{"key":"OBJECT_NOT_FOUND", "type":"ERR", "message":"Невозможно найти указанный
  объект."}
```

1.12.1.2 Удаление метрики с указанием id

URL: `/api/db/${dataset}.metrics/${id}`

Внимание! При удалении метрики, автоматически удаляются все значения Точек Визуального Контроля, у которых метрика равна удаляемой метрике!

Для удаления одной определённой метрики нужно знать её `id`.

`id` метрики указывается в URL запроса в сегменте `${id}`.

Например, чтобы удалить метрику с `id` = `10` в датасете `ds_tests`, нужно выполнить такой HTTP запрос:

```
DELETE /api/db/ds_tests.metrics/10
```

Пример для выполнения в командной строке:

```
curl -k -v \--
header "LuxmsBI-User-Session: ${luxmsbi_cookie}" \-
X DELETE "${luxmsbi_url}/api/db/ds_tests.metrics/10"
```

В случае успешного удаления записи, возвращается статус 200 и пустой список в качестве тела ответа.

Пример тела ответа в случае успешного удаления метрики:

```
[]
```

Пример ответа, в случае ошибки:

```
HTTP/1.1 404 Not Found
Content-Type: application/json; charset=utf-8

{"key":"OBJECT_NOT_FOUND", "type":"ERR", "message":"Невозможно найти указанный объект."}
```

1.12.1.3 Удаление локации с уазанием id

URL: `/api/db/${dataset}.locations/${id}`

Внимание! При удалении локации, автоматически удаляются все значения Точек Визуального Контроля, у которых локация равна удаляемой локации!

Для удаления одной определённой локации нужно знать её `id`.

`id` локации указывается в URL запроса в сегменте `${id}`.

Например, чтобы удалить локацию с `id` = `123` в датасете `ds_tests`, нужно выполнить такой HTTP запрос:

```
DELETE /api/db/ds_tests.locations/123
```

Пример для выполнения в командной строке:

```
curl -k -v \--
header "LuxmsBI-User-Session: ${luxmsbi_cookie}" \-
X DELETE "${luxmsbi_url}/api/db/ds_tests.locations/123"
```

В случае удаления хотя бы одной записи, возвращается статус 200 и пустой список в качестве тела ответа.

Пример тела ответа в случае успешного удаления локации:

```
[]
```

Пример ответа, в случае ошибки:

```
HTTP/1.1 404 Not Found
Content-Type: application/json; charset=utf-8

{"key":"OBJECT_NOT_FOUND", "type":"ERR", "message":"Невозможно найти указанный объект."}
```

1.12.1.4 Удаление периода с уазанием id

URL: `/api/db/${dataset}.periods/${id}`

Внимание! При удалении периода, автоматически удаляются все значения Точек Визуального Контроля, у которых период равен удаляемому периоду!

Для удаления одного определённого периода нужно знать его `id`.

`id` периода указывается в URL запроса в сегменте `${id}`.

Например, чтобы удалить период с `id` = `2013120100000036` в датасете `ds_tests`, нужно выполнить такой HTTP запрос:

```
DELETE /api/db/ds_tests.periods/2013120100000036
```

Пример для выполнения в командной строке:

```
curl -k -v \--
header "LuxmsBI-User-Session: ${luxmsbi_cookie}" \-
X DELETE "${luxmsbi_url}/api/db/ds_tests.periods/2013120100000036"
```

В случае удаления хотя бы одной записи, возвращается статус 200 и пустой список в качестве тела ответа.

Пример тела ответа в случае успешного удаления периода:

```
[]
```

Пример ответа, в случае ошибки:

```
HTTP/1.1 404 Not Found
Content-Type: application/json; charset=utf-8

{"key":"OBJECT_NOT_FOUND", "type":"ERR", "message":"Невозможно найти указанный ↩
  объект."}
```

2 Протокол загрузки данных BIPush

2.1 Введение

Проткол BIPush позволяет добавлять, удалять и модифицировать данные в хранилище данных LuxmsBI. Операции с данными кодируются в формате JSON и передаются в виде пакетов.

LuxmsBI гарантирует атомарность выполнения всех операций в одном пакете. То есть либо все операции будут успешно выполнены, либо все операции будут проигнорированы в силу какой-либо ошибки на стороне сервера.

Один пакет может содержать несколько разных операций CRUD и несколько различных структур данных, к которым будут применены операции.

Перед тем как подавать запросы по протоколу BIPush, необходимо пройти аутентификацию в системе.

2.2 URL /api/data/${dataset}/push

В сегменте `dataset` передаётся имя датасета, в который необходимо добавить значения показателей.

сегмент URL	Значения	Пример
`${dataset}`	GUID, имя схемы или id датасета	`4ebc64b0-39ea-4b62-a28d-722724daaa0a` или `ds_180` или `12`

2.3 Структура пакета данных

Каждый пакет состоит из заголовка и тела.

Пример структуры пакета:

```
{
"version"   : "",
"packetId" : "",
"mode"      : "",
```

```
"operations" : []
}
```

2.3.1 Заголовок Пакета

Заголовок пакета состоит из трех полей:

Имя	Тип	Обязатаельность	Значение по умолчанию	Описание
version	string	✔		Версия формата данных
packetId	string	✔		Уникальный идентификатор пакета
mode	string		sync	Режим выполнения команд

Протокол BIPush поддерживает три режима обработки пакетов

- *sync* - синхронный режим.

HTTP ответ будет выдан после того, как все операции в пакете будут выполнены. Не рекомендуется использовать для передачи большого количества записей (несколько тысяч и более), так как долгое выполнение операций может привести к таймауту HTTP запроса. В этом случае клиент не имеет возможности проверить успешность выполнения операций.

- *async* - асинхронный режим.

HTTP ответ будет выдан сразу после получения пакета. В ответе будет указана ссылка, по которой можно проверять статус выполнения операций.

В текущей версии Luxms BI режим *async* не реализован.

- *queue* - асинхронный режим с постановкой в очередь.

В текущей версии Luxms BI режим *queue* не реализован.

Пример заголовка:

```

"version" : "1.0",
"packetId" : "P1202.11",
"mode" : "sync"
```

2.3.2 Тело Пакета

Телом пакета является список операций.

Пример тела пакета:

```
"operations" : []
```

2.3.2.1 Операция

Операция состоит из четырех полей:

Имя	Тип	Обязательность	Значение по умолчанию	Описание
op	string	✔		Операция, которую необходимо выполнить над записями, передаваемыми в массиве по ключу `records`
tablename	string	✔		Имя таблицы (сущности), для которой предназначены записи
format	string		raw	Формат записей
records	array	✔		Список записей

Протокол BIPush поддерживает четыре типа операций

- *insert* - вставка записей.
- *update* - замена указанных значений в записях.
- *replace* - Для каждой записи будет произведена замена записи на её новую версию, либо запись будет добавлена как новая, если в хранилище LuxmsBI нет записи с указанным id.
- *delete* - удаление записей с указанными id.

Если LuxmsBI обнаружит нарушение целостности данных в процессе выполнения операции, то операция будет прервана и произойдет откат всех предыдущих операций в пакете.

Возможные причины нарушения целостности данных:

- неверный формат даты
- неуникальный ключ записи (id)
- неверная ссылка на родительскую или связанную сущность
- пустые значения для полей, которые обязаны иметь значение

Пример операции:

```
{
"op" : "replace",
"tablename" : "norms",
"format" : "flat",
"records" : []
}
```

2.4 Нормативы

Нормативы используются для визуального контроля ключевых показателей и задают ограничения на значения показателей.

В Luxms BI поддерживаются нормативы трех типов:

- план-факт
- диапазон значений (минимум и максимум)
- цветовое кодирование диапазонов значений

Плоский формат `"format" : "flat"` позволяет передавать минимальное и/или максимальное значение для показателя и даты начала и окончания действия норматива в “плоском” формате.

2.4.1 Замена значений нормативов

Для передачи нормативов рекомендуется использовать операцию замены `"op" : "replace"` и плоский формат `"format" : "flat"`.

Пример пакета для передачи нормативов:

```
{
"version" : "1.0",
"packetId" : "<Your packet Id>",
"mode" : "sync"
"operations" : [
{
"op" : "replace",
"tablename" : "norms",
"format" : "flat",
"records" : [
{
"srcId" : 1,
"title" : "Средняя температура по больнице",
"srcParamKey" : "P23",
"minValue" : 20000,
"startDate" : "2015-01-01T00:00:00+03"
},
{
"srcId" : 2,
"title" : "Процент доступности кофе-машин",
"srcParamKey" : "P1",
"minValue" : 95,
"startDate" : "2015-03-01T00:00:00+03"
}    ]

}]
```

```
28 }
```

Запись описывающая операцию состоит из нескольких полей:

Имя	Тип	Обязательность	Значение по умолчанию	Описание
srcId	int	✔0		Уникальный код норматива во внешней системе
id	int	✔0		Уникальный код норматива в LuxmsBI
title	string	✔		Название норматива
srcParentId	int			Код родительского норматива во внешней системе
parentId	int			Код родительского норматива вLuxmsBI
srcParamKey	string	✔1		Текстовый ключ показателя во внешней системе, для которого действует норматив
srcParamId	int	✔1		Числовой код показателя во внешней системе, для которого действует норматив
metricId	int	✔1		Числовой код показателя в LuxmsBI, для которого действует норматив
paramKey	int	✔1		Текстовый код показателя в LuxmsBI, для которого действует норматив
srcValueId	int	✔2		Уникальный код значения норматива во внешней системе
valueId	int	✔2		Уникальный код значения норматива в LuxmsBI
minValue	float	✔3		Минимальное значение норматива
maxValue	float	✔3		Максимальное значение норматива
startDate	date		-infinity	Дата начала действия указанных значений норматива
endDate	date		+infinity	Дата окончания действия указанных значений норматива
srcLocIdList	array			список id объектов контроля во внешней системе, для которых действует норматив
locIdList	array			список id объектов контроля в LuxmsBI, для которых действует норматив
periodTypeList	array			список типов периодов, для которых действует норматив

Типы периодов задаются как целочисленные идентификаторы

3 Протокол загрузки данных Telemetry

3.1 Введение

Протокол `Telemetry` позволяет загружать значения показателей (телеметрию) в хранилище данных LuxmsBI. Значения показателей кодируются в формате CSV и передаются в виде пакетов.

LuxmsBI гарантирует атомарность обработки одного пакета. То есть, либо все данные в пакете будут успешно сохранены, либо все данные в пакете будут проигнорированы в силу какой-либо ошибки на стороне сервера.

Пакет данных передаётся как тело HTTP POST запроса, а дополнительные параметры запроса передаются в виде `QUERY_STRING`.

При обработке пакета, при необходимости, будут созданы и сохранены в таблице `periods` новые периоды. В случае, если значение показателя уже существует в таблице `data`, значение показателя из пакета будет использовано для перезаписи существующего значения.

Перед тем как подавать запросы по протоколу Luxms BI Sync API, необходимо пройти аутентификацию в системе.

3.2 Параметры запроса /api/data/${dataset}/telemetry

В URL передаётся имя датасета, в который необходимо добавить значения показателей.

сегмент URL	Значения	Пример
`${dataset}`	GUID, имя схемы или id датасета	`4ebc64b0-39ea-4b62-a28d-722724daaa0a` или `ds_180` или `12`

Параметры запроса, передаваемые в `QUERY_STRING`:

Имя поля	Значение	Значение по умолчанию	Пример
period_type	Код типа периода. Строка принимающая одно из значений: `second` `minute` `hour` `day` `week` `month` `quarter` `year`		`period_type=second`
date_format	Строка формата, применяемая при генерации заголовка периода. Поддерживаются форматы, используемые функцией PostgreSQL to_char	Для каждого типа периода своё значение.*	`date_format=tmmon`
return_new_-periods	Флаг, указывающий на необходимость возврата созданных периодов в ответе**	`false`	`return_new_periods=true`

* Значения поля `date_format` по умолчанию для разных типов периодов:

Значение поля `period_type`	Значение поля `date_format` по умолчанию
`second`	`dd.mm.yyyy hh24:mi:ss`
`minute`	`dd.mm.yyyy hh24:mi`
`hour`	`dd.mm.yyyy hh24`
`day`	`dd.mm.yyyy`
`week`	`dd.mm.yyyy`
`month`	`tmmon yyyy`
`quarter`	`Qкв yyyy`
`year`	`yyyy`

** Для получения в ответе информации о добавленных периодах необходимо использовать для параметра `return_new_periods` значение, равное `true` или равное `1`. Все остальные значения, включая пустую строку, равносильны отсутствию параметра `return_new_periods` в URL и не влияют на ответ сервера.

Пример URL:

```
1 /api/data/ds_test/telemetry?date_format=yyyy-mi-dd&return_new_periods=true&
    period_type=day
```

3.3 Формат пакета данных

Каждый пакет состоит из нескольких строк в формате CSV, при этом строка-заголовок с названиями столбцов не используется.

Разделителем полей является символ точка с запятой (`;`). Для квотации значений допускается использование символа двойные кавычки (").

В каждой строке передаются значения в порядке следования:

Поле	Тип значения	Пример
ID еденицы измерения метрики. В настоящий момент не интерпретируется сервером.	INT	0
Текстовый идентификатор метрики	TEXT	`Показатели:Натуральные показатели:Количество зданий`
Текстовый идентификатор локации	TEXT	`Санкт-Петербург:Невский район`
Время события	TEXT	`2018-01-01T01:00:00.000Z`
Значение метрики	FLOAT	33023.0

Пример пакета в формате CSV:

```
0;Показатели:Вибрация:Геометрическая частота вибрации;Станок 1516M #102052:Сенсор 2;2018-01-01T00:00:00.000Z;10.0
0;Показатели:Вибрация:Геометрическая частота вибрации;Станок 1516M #102052:Сенсор 2;2018-01-01T01:00:00.000Z;20.0
```

Пример ответа при наличии параметра `return_new_periods`:

```
{
"periods": {
"new": [
{
"id": 2020021500000054,
"parent_id": null,
"tree_level": 0,
"period_type": 4,
"qty": 1,
"start_time": "2020-02-15T00:00:00",
"title": "15.02.2020",
"tags": [],
"src_id": null,
```

```
"alt_id": null,
"tree_path": null,
"config": {},
"updated": "2020-10-01T13:58:57.866779+00:00",
"created": "2020-10-01T13:58:57.866779+00:00"
},
{
"id": 2020010100000054,
"parent_id": null,
"tree_level": 0,
"period_type": 4,
"qty": 1,
"start_time": "2020-01-01T00:00:00",
"title": "01.01.2020",
"tags": [],
"src_id": null,
"alt_id": null,
"tree_path": null,
"config": {},
"updated": "2020-10-01T13:58:57.866779+00:00",
"created": "2020-10-01T13:58:57.866779+00:00"
}    ]

},
"timezone": "Europe/Moscow"
}
```

3.4 Порядок обработки входных данных

3.4.1 Периоды

Для сопоставления отметок времени из входящего пакета данных с теми, которые уже хранятся в хранилище Luxms BI, производится вычисление ID периода. Для этого время переводится во временную зону датасета, затем округляется до заданного типа периода (параметр `period_type`), затем вычисляется ID периода. Если в хранилище не существует периода с таким ID, то происходит вставка новой записи в таблицу `periods`.

При вставке нового периода, заголовок периода генерируется путём перевода отметки времени в часовой пояс, установленный для датасета, последующего округления до заданного типа периода (параметр `period_type`) и, наконец, применения функции форматирования `to_char` с форматом, указанном в параметре `date_format`.

По умолчанию в Luxms BI используется часовой пояс ‘Europe/Moscow’.

3.4.2 Метрики

Для метрик во входном пакете указывается путь в дереве метрик от корня до требуемой метрики. Путь состоит из заголовков метрик (поле `title` в таблице `metrics`), разделённых символом двоеточия `:`. Использование кавычек для квотации заголовков не допускается. На основании указанных заголовков метрик сервер производит поиск ID для целевой метрики, и использует найденный ID для вставки строки в таблицу `data`.

3.4.3 Локации

Для локаций во входном пакете указывается путь в дереве локаций от корня до требуемой локации. Путь состоит из заголовков локаций (поле `title` в таблице `locations`), разделённых символом двоеточия `:`. Использование кавычек для квотации заголовков не допускается. На основании указанных заголовков локаций сервер производит поиск ID для целевой локации, и использует найденный ID для вставки строки в таблицу `data`.

3.4.4 Еденицы измерения

Еденицы измерения игнорируются, но должны присутствовать во входном пакете.

4 Библиотека bixel

4.1 Краткое описание библиотеки bixel

Библиотека `bixel` используется для взаимодействия внешних виджетов с клиентской частью LuxmsBI.

Публичный репозиторий: https://github.com/luxms/bixel

Внешний виджет представляет собой html файл, доступный по http, который загружается в LuxmsBI через iframe.

Библиотека bixel представляет собой интерфейс для обмена сообщениями с клиентской частью.

4.2 Подключение библиотеки в свой проект

Библиотека состоит из одного файла (bixel.js), который может быть подключен разными способами:

4.2.1 Подключение через github

Достаточно просто подключить

```
<script src="https://cdn.jsdelivr.net/gh/luxms/bixel@master/bixel.js"></script>
```

в свой html файл.

4.2.2 Подключение локально

Скачать bixel.js в папку с html файлом и указать

```
<script src="bixel.js"></script>
```

4.2.3 Подключение при помощи bower

Запустить команду

```
bower install git://github.com/luxms/bixel
```

После чего можно подключить файл

```
<script src="bower_components/bixel/bixel.js"></script>
```

или интегрировать в свою систему сборки (например, grunt, bower-requirejs).

4.2.4 Подключение через require.js

Библиотека bixel поддерживает загрузку через require.js

Например:

`index.html`:

```
<!DOCTYPE html>
<html>
<head>
<script data-main="app" src="https://cdnjs.cloudflare.com/ajax/libs/require.js/2.2.0/require.min.js"></script>
</head>
<body>
...
</body>
</html>
```

`app.js`:

```
requirejs.config({
paths: {
bixel: 'bower_components/bixel/bixel'
}
});

require(['bixel'], function(bixel) {
bixel.init();
bixel.on('load', function(data, axis) {
// ...
});
});
```

4.3 API библиотеки bixel

4.3.1 Методы библиотеки bixel

4.3.1.1 Метод bixel.init

Метод bixel.init используется для инициализации библиотеки. В качестве единственного необязательного параметра можно передать настройки - объект содержащий необязательные поля:

- `metricsCount` - количество метрик, в которых заинтересован виджет
- `locationsCount` - количество метрик, в которых заинтересован виджет
- `periodsCount` - количество метрик, в которых заинтересован виджет

Если эти опции не указаны, будут передаваться все, которые выбраны на панелях интерфейса LuxmsBI или в настройках дэша, а также данные будут загружаться по всему кубу.

Метод `bixel.init` возвращает объект типа Promise, который сигнализирует о результате.

Пример:

```
bixel.init({
periodsCount: 1,
metricsCount: 1
}).then(function() {
// succeeded
}, function() {
// error
});
```

4.3.1.2 Метод bixel.on

Метод bixel.on добавляет подписку на события.

```
bixel.on(eventType, callback)
```

- eventType - строка, может принимать значения `loading`, `load`, `no-data`
- callback - функция, вызываемая по событию, в зависимости от типа принимает разное количество аргументов.

4.3.2 Функции обратного вызова

4.3.2.1 Функция обратного вызова loading

Callback вызывается в момент начала загрузки данных. В качестве единственного параметра получает объект типа bixel.Axis.

```
bixel.on('loading', function(axis) {
console.log('Началась загрузка данных по кубу размера' +
axis.getMetrics().length + 'x' +
axis.getLocations().length + 'x' +
axis.getPeriods().length);
});
```

4.3.2.2 Функция обратного вызова load

Callback вызывается, когда закончена загрузка данных и получает два параметра:

- data: bixel.Data
- axis: bixel.Axis

```
bixel.on('load', function(data, axis) {
var metric = axis.getMetrics()[0];
var location = axis.getLocations()[0];
var period = axis.getPeriods()[0];
var val = data.getValue(metric, location, period);
console.log("Данные загружены:", value.toString());
});
```

4.3.2.3 Функция обратного вызова no-data

Callback вызывется, когда данных нет:

- Метрика, локация или период не выбраны на панелях интерфейса
- все данные имеют пустые значения

В качестве единственного параметра получает объект типа bixel.Axis.

4.3.2.4 Функция invoke

Для создания сырого куба с использованием библиотеки `bixel` необходимо использовать функцию `invoke`. Пример использования функции представлен ниже:

```
bixel.invoke('LOAD_RAW_DATA',
{
 dimensions:['sex','age'],
measures:['sum(v_main)'],
filters:{'sex':['=', 'Мужчины']},
limit:2
}).then((data)=>{
 console.log(data)
```

```
9 })
```

Существует возможность добавить объект `koob: '...'`, тогда запрос будет уходить в другой куб

4.3.3 Объекты

4.3.3.1 Объект Axis

Объект, содержащий описание осей (метрики-локации-периоды)

Реализует методы:

- `getMetrics()` - массив объектов Metric
- `getLocations()` - массив объектов Location
- `getPeriods()` - массив объектов Period
- `getUnits()` - массив объектов Units - только те, с которыми связаны метрики

4.3.3.2 Объект Data

Объект, хранящий полученные значения

Реализует методы:

- `getValue(z, y, x)` - получить значение в указанной точке. В качестве параметров принимает объекты Metric, Location, Period в любом порядке. Возвращает объект DataItem (Number)

4.3.3.3 Объект DataItem

В общем случае это js объект типа `Number` с переопределенным методом `toString`

Метод `toString` возвращает форматированное согласно соответствующему Unit значение (добавляет префикс, суффикс, название, разделители между разрядами числа)

Метод 'valueOf' вернет примитивный number

Если значение отсутствует в базе данных, то:

- `valueOf()` вернет `NaN` (можно проверить, вызвав `isNaN(val)`)
- `toString()` вернет строку `"-"`

В случае text-data значениями могут быть объекты типа String. В том с другом случае

```
typeof value === 'object'
```

Поэтому при необходимости проверки, ее можно осуществить через `instanceof`

```
if (val instanceof String) { ... }
if (val instanceof Number) { ... }
```

4.3.3.4 Объект Metric

Структура, содержащая описание метрики:

- `id`:string - уникальный идентификатор
- `title`:string - название метрики
- `color`:string - цвет, сопоставленный метрике
- `bgColor`:string - цвет фона (предпочтительный для закрашивания)
- `unit_id`:string - идентификатор Unit

4.3.3.5 Объект Location

Структура, содержащая описание локации:

- `id`:string - уникальный идентификатор
- `title`:string - название метрики
- `color`:string - цвет, сопоставленный метрике
- `bgColor`:string - цвет фона (предпочтительный для закрашивания)

4.3.3.6 Объект Period

Структура, содержащая описание периода:

- `id`:string - уникальный идентификатор
- `title`:string - название периода
- `color`:string - цвет, сопоставленный периоду
- `bgColor`:string - цвет фона (предпочтительный для закрашивания)

4.3.3.7 Объект Unit

Структура, содержащая описание еденицы измерения (размерности):

- `id`:string - уникальный идентификатор
- `value_prefix`:string - префикс, пишется перед значением
- `value_suffix`:string - суффикс, пишется после значения

5 Работа с библиотекой bixel. Пример 1.

5.1 Введение

Библиотека `bixel` используется для взаимодействия внешних виджетов с клиентской частью LuxmsBI

Публичный репозиторий: https://github.com/luxms/bixel

Внешний виджет представляет собой html файл, доступный по http, который загружается в LuxmsBI через iframe.

Библиотека bixel представляет собой интерфейс для обмена сообщениями с клиентской частью

5.2 Цель

Создать виджет на базе датасета `Налоги США`, который бы показывал штат с максимальным значением метрики `Taxes/Total taxes` по разным годам.

LuxmsBI - Налоги США

Дэш использует:

- Локации: все штаты
- Метрики: `T1`, “Total taxes”
- Периоды: выбранные в верхней панели

5.3 Шаг 1. index.html

Создаем файл `index.html`

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
```

```
</body>
</html>
```

5.4 Шаг 2. scripts

Будем использовать простой шаблон на базе библиотеки `knockout`. Подключаем библиотеки `knockout.js` и `bixel` внутри `<head>`:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/knockout/3.3.0/knockout-
  min.js"></script>
<script src="https://cdn.jsdelivr.net/gh/luxms/bixel@master/bixel.js"></script>
```

5.5 Шаг 3. Разработка модели

Моделью (в терминах knockout) для шаблона будет объект, содержащий поля:

- `loading`: KnockoutObservable - статус, идет ли загрузка в данный момент
- `period`: KnockoutObservable<bixel.Period> - текущий период, будет иметь тип “год”
- `metric`: KnockoutObservable<bixel.Period> - текущая метрика (всегда будет `Total taxes`)
- `location`: KnockoutObservable<bixel.Location> - штат, в котором значение метрики максимальное
- `value`: KnockoutObservable<Number> - максимальное значение метрики среди всех штатов

5.6 Шаг 4. Шаблон

В `<body>` создаем шаблон, основанный на модели

```
<div id="result">
<!-- ko if: loading -->
<span>Загрузка ...</span>
<!-- /ko -->

<!-- ko if: !loading() -->
<h2>Лучшее значение по всем штатам:</h2>
<span data-bind="text:metric().title"></span>: <span data-bind="text:value">
  </span>
<br/>
<span data-bind="text:location().title"></span>
<br/>
<span>На период: </span><span data-bind="text:period().title"></span>
<!-- /ko -->
```

```
14 </div>
```

В момент загрузки (`loading() == true`) будем показывать надпись “Загрузка…”

Когда данные загружены покажем:

- название метрики (`metric().title`)
- найденное лучшее значение (`value`) - будет автоматически приведено к строке через переопределенный метод `toString()`, который форматирует значение согласно настройкам `unit` (добавляет знак доллара и разделители у числа)
- название штата с наилучшим значением (`location().title`)
- выбранный год (`period().title`)

5.7 Шаг 5. Создаем модель к шаблону

Добавляем `<script>` и в нем описываем структуру:

```
var model = {
loading: ko.observable(true),
period: ko.observable(null),
metric: ko.observable(null),
location: ko.observable(null),
value: ko.observable(null)
};
```

Глобальная переменная `model` является описанием модели

Связываем модель с шаблоном (элемент с id=result)

```
ko.applyBindings(model, document.getElementById('result'));
```

5.8 Шаг 6. Инициализация bixel

Добавляем скрипт

```
bixel.init({
periodsCount: 1,
metricsCount: 1
});
```

Виджет заинтересован в одном периоде (выбранный год) и в одной метрике (Total taxes, указывается в настройках дэша)

5.9 Шаг 7. Обработка события loading

Добавляем скрипт

```
bixel.on('loading', function(axis) {
model.loading(true);
});
```

Будем устанавливать в модели свойство `loading` в `true`

5.10 Шаг 8. Обработка события load

Обработка события load происходит, когда загружены данные

```
bixel.on('load', function(data, axis) {
var locations = axis.getLocations();
var metric = axis.getMetrics()[0];
var period = axis.getPeriods()[0];
//...
});
```

В соответствии с настройками из `bixel.init` в этом методе будет получена одна метрика и один период (размеры соответствующих массивов будут равны единице)

В качестве локаций придут все штаты, поэтому запоминаем их для дальнейшего использования

5.11 Шаг 9. Поиск лучшей локации

Будем использовать простой цикл:

```
var bestVal = -Infinity;      // Храним лучшее значение
var bestLocation = null;      // и лучшую локацию (штат)

for (var i = 0; i < locations.length; i++) {
var value = data.getValue(locations[i], metric, period);
if (value > bestVal) {
bestLocation = locations[i];
bestVal = value;
}
}
```

На каждой итерации получаем значение для очередного штата, сравниваем с `bestVal` (Сравнение работает, поскольку это объекты типа `Number`)

После окончания работы цикла в переменных `bestVal` и `bestLocation` будут соответственно лучше значение и лучший штат

5.12 Шаг 10. Заполняем модель

```
model.location(bestLocation);
model.metric(metric);
model.period(period);
model.value(bestVal);
model.loading(false);
```

5.13 Шаг 11. Deploy

Итак, у нас получился index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<script src="https://cdnjs.cloudflare.com/ajax/libs/knockout/3.3.0/knockout-
  min.js"></script>
<script src="https://cdn.jsdelivr.net/gh/luxms/bixel@master/bixel.js"></script>
</head>
<body>
<h1>Внешний виджет</h1>

<div id="result">
<!-- ko if: loading -->
<span>Загрузка ...</span>
<!-- /ko -->

<!-- ko if: !loading() -->
<h2>Лучшее значение по всем штатам:</h2>
<span data-bind="text:metric().title"></span>: <span data-bind="text:value">
  </span>
<br/>
<span data-bind="text:location().title"></span>
<br/>
<span>На период: </span><span data-bind="text:period().title"></span>
<!-- /ko -->
</div>


<script>
// ui
var model = {
loading: ko.observable(true),
period: ko.observable(null),
metric: ko.observable(null),
location: ko.observable(null),
value: ko.observable(null)
};
ko.applyBindings(model, document.getElementById('result'));
```

```
bixel.init({
locationsCount: Infinity,
periodsCount: 1,
metricsCount: 1
});

bixel.on('loading', function(axis) {
model.loading(true);
});

bixel.on('load', function(data, axis) {
var locations = axis.getLocations();
var metric = axis.getMetrics()[0];
var period = axis.getPeriods()[0];

var bestVal = -Infinity;
var bestLocation = null;

for (var i = 0; i < locations.length; i++) {
var value = data.getValue(locations[i], metric, period);
if (value > bestVal) {
bestLocation = locations[i];
bestVal = value;
}
}

model.location(bestLocation);
model.metric(metric);
model.period(period);
model.value(bestVal);
model.loading(false);
});

</script>
</body>
</html>
```

Выкладываем `index.html` на какой-нибудь доступный web-сервер, например, по адресу `http://example.com/best-state/index.html`

5.14 Шаг 12. настройка дэша в LuxmsBI

Создаем новый дэш а датасете “Налоги США”

- Тип дэша (view_class): `external`
- Заголовок дэша: “Total taxes: лучший штат”
- config.dataSource.parameters: `["T1"]` (одна метрика Total taxes)
- config.dataSource.locations: “(level~2)” - все локации уровня 2 (штаты)
- config.url: “http://example.com/best-state/index.html”

5.15 Результат

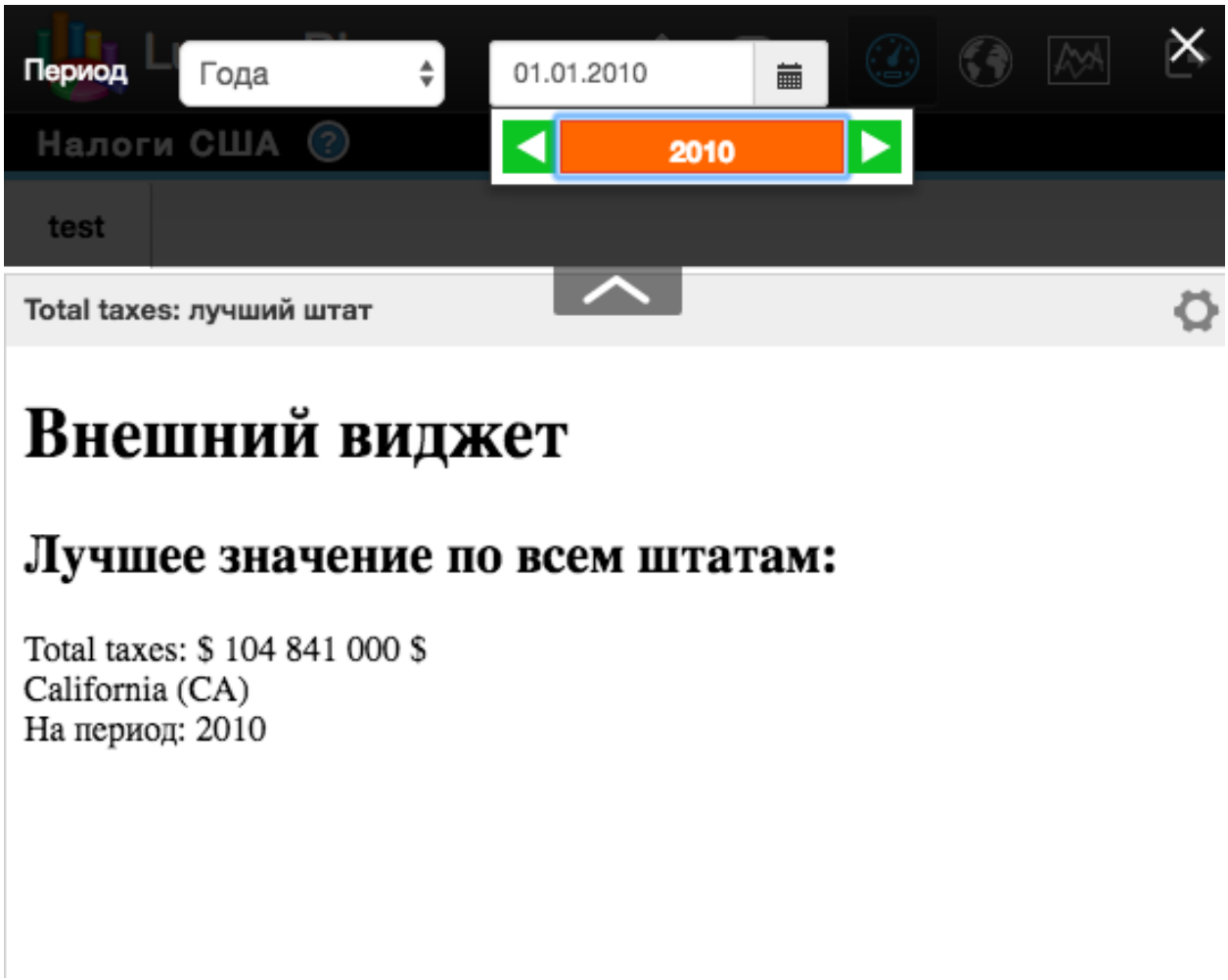


Рис. 5.1 **Результат встраивания bixel в дэшборд Luxms BI**

6 Пакет для импорта bi-internal/services

Предоставляет следующий набор сервисов:

фабрика, позволяет создать сервис для работы с моделью датасета, строковый идентификатор (schema_name) которого передан при созданнии `DatasetService.createInstance('ds_demo136')` `DatasetService`, Сервис, позволяющий получить список датасетов, которые доступны пользователю `DatasetsListService`, интерфейс модели такого сервиса:

```
interface IDatasetsListModel {
loading: boolean;
error: string;
datasets: IDatasetsListItem[];
roots: IDatasetsListItem[];
}
*/
IDatasetsListModel,
/*
Элемент такого списка датасетов
interface IDatasetsListItem extends IEntity {
id: string;
guid: string;
schema_name: string;
title: string;
description: string;
layout: string;
image: string;
lastPeriodTitle: string;
href: string;
color: string;
tiles: IDatasetsListTile[];
bookmarks: any[];              // tables.IBookmark[]
searchVisible: boolean;        // TODO: remove
children: IDatasetsListItem[];
resourcesRoot: string;
parents: IDatasetsListItem[];
uiCfg: any;
deleteBookmark(bookmark: tables.IBookmark);
}
*/
IDatasetsListItem,
DatasetsListIcon,
DatasetsListView1,
IVizelConfig, // Конфиг визеля
IVizelProps,
```

- `Vizel` - React.Component для встраивания внутрь дешлета. Может быть заменен на класс кастомного компонента
- `DsStateService` - Сервис, наблюдающий за тем, на каком датасете и его дешборде мы сейчас
- `KoobDataService` - Сервис для запросов данных в куб таблицы
- `KoobService` - загружает всю информацию о кубе, кроме данных. Все дименшны и межи, конфиги, и прочие вспомогательные ключи
- `KoobFiltersService` - Сервис, хранящий объект указанных фильтров на кубы. Отвечает за поведение управляющего дешлета.
- `PluginsManager` - позволяет подключать компоненты как отдельные разделы Bi, примеры описаны в папке ds_res/plugins проекта `luxmsbi-web-resources`
- `ISummaryModel`,
- `SummaryService`,
- `shell`,
- `getShell`,
- `useService`,
- `ISearchVM`,
- `SearchVC`,
- `service`,
- `IShellVM`,
- `IDsShellVM`,
- `DsShellVC`,
- `DsShell`,
- `createSubspaceGenerator`

Рассмотрим самые значимые сервисы для работы с таблицами OLAP кубов.

6.0.1 KoobService

Загружает информацию о таблице из куба и сохраняет информацию о ней в кеше.

```
interface IKoobModel {
id: string;
loading?: boolean;
error?: string;
dimensions: IKoobDimension[];
measures: IKoobMeasure[];
}
class KoobService extends BaseService<IKoobModel> {
public readonly id: string;
private _detailedEntities: { [entityId: string]: string[] } = {};
private _isMainLoading: boolean = true;

private constructor(koobId: string) {
super({
id: koobId,
loading: true,
error: null,
dimensions: [],
```

```
19 measures: [],
20 });
21 this.id = koobId;
22 this._init();
23 }

25 public async loadEntityDetails(entityId: string) {
26 let fullId: string = entityId;
27 if (!fullId.startsWith(this.id + '.')) {
28 fullId = this.id + '.' + fullId;
29 }
30 entityId = fullId.split('.').slice(-1)[0];

32 if (fullId in this._detailedEntities) return;                    // ↩
     marked as loading or exists
33 this._detailedEntities[fullId] = null;                                  ↩
                    // bad loading mark
34 this._detailedEntities[entityId] = null;                                ↩
                    // bad loading mark
35 const url = AppConfig.fixRequestUrl(`/api/v3/koob/${fullId}`);

37 this._updateWithLoading();

39 const request = await fetch(url, {credentials: 'include'});
40 const result = await request.json();

42 if (result.values && Array.isArray(result.values)) {
43 result.members = result.values.map(value => ({id: value, title: value}));
44 }


47 this._detailedEntities[fullId] = result;                                 ↩
                    // save detailed entity to cache
48 this._detailedEntities[entityId] = result;                               ↩
                    // save detailed entity to cache

50 let {dimensions, measures} = this._model;

52 let idx = $eidx(dimensions, entityId);                          // ↩
     update if necessary

54 if (idx !== -1) {
55 dimensions = dimensions.slice(0);
56 dimensions[idx] = result;
57 }
58 idx = $eidx(measures, entityId);
59 if (idx !== -1) {
60 measures = measures.slice(0);
61 measures[idx] = result;
62 }

64 const loading = this._isDetailsLoading() || this._isMainLoading;
65 this._updateModel({loading, dimensions, measures});
```

```
}

private _isDetailsLoading(): boolean {
for (let entityId in this._detailedEntities) {
if (this._detailedEntities[entityId] === null) {                    // ↩
  null is loading mark
return true;
}
}
return false;
}

private async _init() {
try {
const url = AppConfig.fixRequestUrl(`/api/v3/koob/${this.id}`);
const result: any = (await axios.get(url)).data;
if (!this._model) {                                                  // ↩
  disposed!
return;
}
result.dimensions.forEach((dimension, idx) => {
if (dimension.id.match(/^\w+\.\w+\.(\w+)$/)) {                       ↩
        // иногда приходят в формате x.y.ID
dimension.id = RegExp.$1;
}
dimension.axisId = dimension.id;
if (this._detailedEntities[dimension.id]) {
result.dimensions[idx] = this._detailedEntities[dimension.id];
}
});
result.measures.forEach((measure, idx) => {
if (this._detailedEntities[measure.id]) {
result.measures[idx] = this._detailedEntities[measure.id];
}
});
this._isMainLoading = false;
const loading = !!this._isDetailsLoading() || this._isMainLoading;

this._updateModel({
error: null,
loading,
dimensions: result.dimensions,
measures: result.measures,
});
} catch (err) {
console.error(err);
this._updateModel({
error: extractErrorMessage(err),
loading: false,
dimensions: [],
measures: [],
});
}
```

```
}

protected _dispose() {
KoobService._cache.remove(this.id);
super._dispose();
}

private static _cache = createObjectsCache(id => new KoobService(String(id)), ↩
  '__koobServices');

public static createInstance: (id: string | number) => KoobService = ↩
  KoobService._cache.get;
}
```

6.0.2 KoobDataService

Сервис позволяет загружать разные срезы данных по указанным фильтрам и наборам измерений и мер в рамках указанного куба.

```
interface IKoobDataRequest3 {
options?: string[];
offset?: number;
limit?: number;
sort?: string[];
subtotals?: string[];
cancelToken?: CancelToken;
}
async function koobCountRequest3(koobId: string,                                ↩

dimensions: string[],
measures: string[],
allFilters: any,
request: IKoobDataRequest3 = {}) {
const url: string = AppConfig.fixRequestUrl(`/api/v3/koob/count`);
const columns = dimensions.concat(measures);

let filters: any;
if (allFilters && typeof allFilters === 'object') {
filters = {};
Object.keys(allFilters).forEach(key => {
let value = allFilters[key];
if (value === '∀' || value === '*') {                                           ↩
                // фильтр подразумевающий 'ВСЕ'
return;                                                                         ↩
              // просто не выносим в фильтры
} else if (Array.isArray(value) && value[0] === 'IN') {                         ↩
                // много где сконфигурировано ['IN', 'a', 'b']
value = ['='].concat(value.slice(1));
}
filters[key] = value;
});
}
```

```
const body: any = {
with: koobId,
columns,
filters,
};

if (request.offset) body.offset = request.offset;
if (request.limit) body.limit = request.limit;
if (request.sort) body.sort = request.sort;
if (request.options) body.options = request.options;
if (request.subtotals?.length) body.subtotals = request.subtotals;

if (!measures.length) {
                    // если нет measures, то лучше применить distinct
body.distinct = [];
}

try {
const response = await axios({
url,
method: 'POST',
headers: {
'Content-Type': 'application/json',
'Accept': 'application/stream+json',
},
data: body,
cancelToken: request.cancelToken,
});

let data = response.data;

if (String(response.headers['content-type']).startsWith('application/stream+
  json')) {
if (typeof data === 'string') {
data = data.split('\n').filter((line: string) => !!line).map((line: string) =>
  JSON.parse(line));
} else if (data && (typeof data === 'object') && !Array.isArray(data)) {
data = [data];
}
}

return data;
} catch (e) {
AlertsVC.getInstance().pushDangerAlert(extractErrorMessage(e));
return '';
}

}
async function koobDataRequest3(koobId: string,

dimensions: string[],
measures: string[],
```

```
allFilters: any,
request: IKoobDataRequest3 = {}) {
const url: string = AppConfig.fixRequestUrl(`/api/v3/koob/data`);
const columns = dimensions.concat(measures);

let filters: any;
if (allFilters && typeof allFilters === 'object') {
filters = {};
Object.keys(allFilters).forEach(key => {
let value = allFilters[key];
if (value === '∀' || value === '*') {
                // фильтр подразумевающий 'ВСЕ'
return;
              // просто не выносим в фильтры
} else if (Array.isArray(value) && value[0] === 'IN') {
                // много где сконфигурировано ['IN', 'a', 'b']
value = ['='].concat(value.slice(1));
}
filters[key] = value;
});
}

const body: any = {
with: koobId,
columns,
filters,
};

if (request.offset) body.offset = request.offset;
if (request.limit) body.limit = request.limit;
if (request.sort) body.sort = request.sort;
if (request.options) body.options = request.options;
if (request.subtotals?.length) body.subtotals = request.subtotals;

if (!measures.length) {
                    // если нет measures, то лучше применить distinct
body.distinct = [];
}

// test
// body.limit = 2;

try {
const response = await axios({
url,
method: 'POST',
headers: {
'Content-Type': 'application/json',
'Accept': 'application/stream+json',
},
data: body,
cancelToken: request.cancelToken,
});
```

```
129 let data = response.data;

131 if (String(response.headers['content-type']).startsWith('application/stream+↩
      json')) {
132 if (typeof data === 'string') {
133 data = data.split('\n').filter((line: string) => !!line).map((line: string) => ↩
      JSON.parse(line));
134 } else if (data && (typeof data === 'object') && !Array.isArray(data)) {
135 data = [data];
136 }
137 }

139 return data;

141 } catch (e) {
142 AlertsVC.getInstance().pushDangerAlert(extractErrorMessage(e));
143 return '';
144 }

146 }
147 interface IKoobDataModel {
148 loading?: boolean;
149 error?: string;
150 dimensions: IKoobDimension[];
151 measures: IKoobMeasure[];
152 values: any[];
153 sort?: string[];
154 subtotals?: string[];
155 }
156 class KoobDataService extends BaseService<IKoobDataModel> {
157 public static koobDataRequest3 = koobDataRequest3;
158 public static koobCountRequest3 = koobCountRequest3;
159 private readonly _koobId: string;
160 private _dimensions: IKoobDimension[];
161 private _measures: IKoobMeasure[];
162 private _filters: any = undefined;
163 private _loadBy: number | null = null;
164 private _totalPages: number | null = null;
165 private _loadedPages: number | null = null;
166 private _sort: string[] | null = null;
167 private _subtotals: string[] | null = null;

169 public constructor(koobId: string,
170 dimensions: IKoobDimension[],
171 measures: IKoobMeasure[],
172 filters: any,
173 loadBy?: number,
174 sort?: any,
175 subtotals?: string[]) {
176 super({
177 loading: true,
178 error: null,
```

```
dimensions: [],
measures: [],
values: [],
sort: null,
subtotals: null,
});
this._sort = typeof sort === 'string' ? [sort] : sort;
this._koobId = koobId;
this._dimensions = dimensions;
this._measures = measures;
this._filters = filters;
this._loadBy = loadBy ?? null;
this._subtotals = Array.isArray(subtotals) ? subtotals : null;
this._load();
}

protected _dispose() {
this.abort();
super._dispose();
}

public abort() {
if (this._loadCancel) {
this._loadCancel.cancel('canceled');
this._loadCancel = null;
}
}

private async loadPage(page: number) {
if (this._totalPages !== null) return;
                  // Все загрузилось
if (page < this._loadedPages) return;
                  // уже грузятся

let loadedPages = this._loadedPages;
this._loadedPages = page + 1;
                  // выставим переменную, чтоб знали, что уже грузится

this._updateModel({loading: true});

for (let p = loadedPages; p <= page; p++) {
const newValues = await koobDataRequest3(
this._koobId,
this._dimensions.map(d => d.id),
this._measures.map(m => m.formula),
this._filters,
{offset: page * this._loadBy, limit: this._loadBy, sort: this._sort, subtotals:
  this._subtotals});
// console.log('Loaded new values with offset=', page * this._loadBy, 'limit=',
  this._loadBy, 'loaded=', newValues.length);
// сохраняем в данные
let values = this._model.values.slice(0);
for (let i = 0; i < newValues.length; i++) {
```

```
227 values[page * this._loadBy + i] = newValues[i];
228 }
229 this._updateModel({values});
230 if (newValues.length < this._loadBy) {
                // загрузилось не все - значит, кончилось
231 this._loadedPages = this._totalPages = p + 1;
232 break;
233 }
234 }
235 this._updateModel({loading: false});
236 }

238 public loadItem(n: number) {
239 if (!this._loadBy) return;
                  // данные грузятся полностью
240 if (this._totalPages !== null) return;
                  // есть переменная выставленная - значит все загружено (или
  грузится)
241 if (n < this._model.values.length) return;
                  // загружено или грузится

243 let page = Math.ceil(n / this._loadBy);
244 this.loadPage(page);
245 }

247 public setSort(sort: string | string[] | null) {
248 this._sort = typeof sort === 'string' ? [sort] : sort;
249 this._totalPages = null;
250 this._loadedPages = null;
251 this._load();
252 }

254 private _loadCancel: CancelTokenSource | null = null;

256 private async _load() {
257 try {
258 this._updateWithLoading();

260 this._loadCancel = axios.CancelToken.source();

262 const values = await koobDataRequest3(
263 this._koobId,
264 this._dimensions.map(d => d?.formula || d.id), // для сгенерированных dimension

265 this._measures.map(m => m.formula),
266 this._filters,
267 {offset: 0, limit: this._loadBy, sort: this._sort, cancelToken:
  this._loadCancel.token, });

269 this._loadCancel = null;

271 if (this._loadBy) {
272 this._loadedPages = 1;
```

```
// console.log('Loadede page 0: total=', values.length);
if (values.length < this._loadBy) {
this._totalPages = 1;
}
}

this._updateWithData({
dimensions: this._dimensions,
measures: this._measures,
values,
sort: this._sort,
});
} catch (err) {
this._updateWithError(extractErrorMessage(err));
}
}

public setFilter(filters: any) {
this._filters = filters;
this._totalPages = null;
this._loadedPages = null;
this._load();
}
}
```

6.0.3 KoobFilterService

Позволяет выставлять условия на дименшны для любых кубов, у которых названия дименшнов совпадают

```
throttleTimeout = 3000
interface IKoobFiltersModel {
loading?: boolean;
error?: string;
query?: string;
result: any;
filters: any;
pendingFilters: any;
}

class KoobFiltersService extends BaseService<IKoobFiltersModel> {
private constructor() {
super({
loading: false,
error: null,
query: undefined,
result: {},
filters: {},
pendingFilters: {},
});
UrlState.subscribeAndNotify('_koobFilters f', this._onUrlStateUpdated);
}
```

```
protected _dispose() {
UrlState.unsubscribe(this._onUrlStateUpdated);
super._dispose();
}

private _onUrlStateUpdated = (url) => {
this._updateWithData({filters: {...url._koobFilters, ...url.f}});
}
public setFilter(koob: string, dimension: string, filter?: any[]) {
let filters = this._model?.pendingFilters;
if (filter) {
let arr: string[] | undefined = filter?.slice(0);
filters = {...filters, [dimension]: arr};
} else {
filters = {...filters, [dimension]: undefined};
}
this._updateModel({pendingFilters: filters});
this._applyAllFilters();
}

public setFilters(koob: string, newFilters: any) {
let filters = this._model.pendingFilters;
for (let dimension in newFilters) {
let filter = newFilters[dimension];
if (filter) {
let arr: string[] | undefined = filter?.slice(0);
filters = {...filters, [dimension]: arr};
} else {
filters = {...filters, [dimension]: undefined};
}
}
this._updateModel({pendingFilters: filters});
this._applyAllFilters();
}

public applyDimensionFilter(dimension: string, value: string | number,
  toggleFlag: boolean, allValues: (string | number)[]) {
const filters = this._model.pendingFilters;
const current = filters[dimension];
let arr: (string | number)[];

if (toggleFlag) {
arr = current ?
current.concat(value) :
['=', value];
} else {
arr = current ?
['='].concat(current.slice(1).filter(e => e != value)) :
['='].concat(allValues.filter(e => e != value) as any);
            // when not set, consider that every was selected
}

const _filters = {...filters, [dimension]: arr};
```

```
this._updateModel({pendingFilters: _filters});
this._applyAllFilters();
}

public applyPeriodsFilter(dimension: string, lodate: string | number, hidate:
  string | number) {
const filters = this._model.pendingFilters;
const _filters = {...filters, [dimension]: ['between', lodate, hidate]};
this._updateModel({pendingFilters: _filters});
this._applyAllFilters();
}

private _applyAllFilters = throttle(() => {
const filters = {...this._model.filters, ...this._model.pendingFilters};
this._updateModel({pendingFilters: {}});

const url = UrlState.getInstance().getModel();
let publicKeys = Object.keys(url.f || {});
                  // Раскидываем ключи фильтров на две части - публичную и
  скрытую
const publicFilters = {}, privateFilters = {};
                  // в публичную попадают ключи, которые уже есть в url
for (let key in filters) {
if (publicKeys.includes(key)) {
publicFilters[key] = filters[key];
} else {
privateFilters[key] = filters[key];
}
}

UrlState.getInstance().updateModel({f: publicFilters, _koobFilters:
  privateFilters});
}, throttleTimeout);

public static getInstance = createSingleton(() => new KoobFiltersService(),
  '__koobFiltersService');
}
```

Рассмотрим пример работы с файлом экселя внутри кастомного компонента. Прочитав основные руководства по работе с данными на платформе, вы поймете как загружаются кубы на основе данных из экселя. Вы можете вывести этот куб как таблицу или иной вид визуализации в рамках коробочного решения. если вам будет нужно менять значения в этой таблице то нужно будет создать новый дешлет, где будет интерфейс, позволяющий запустить процесс редактирования данных. Для начала загрузим документ экселя как источник данных. Файл экселя это потенциально несколько листов. Каждый лист считается отдельным источником данных и имеет свой идентификатор. Загружаем файл экселя через `input[type=file]`, берем при загрузке событие и у него `event.target.files`

```
import { AppConfig } from '@luxms/bi-core';
// тут код вашего компонента
// код функции, принимающей список загруженных файлов в форму
const url = AppConfig.fixRequestUrl('/srv/datagate/source/upload');
const files = Array(event.target.files);
```

```
const formData = new FormData(); // (https://developer.mozilla.org/ru/docs/Web/API/FormData)
files.forEach(file => formData.append('source', file))
```

Далее шлем `async` запрос например через `axios`:

```
axios.post(url, formData, { headers: { 'Content-Type': 'multipart/form-data' } })
```

У ответа проверяем ключа `data.documents`, где будут храниться название файла, ряд вспомогательных ключей и список листов документа `tableContainers`. Для получения данных каждого листа нужно в масссиве `tableContainers` найти `tableName` и сделать GET запрос на `api/db/xdds.t1464/`

где `t1464` - это `tableName` нужного листа, а `xdds` - зарезервированное название пакета всех таблиц, что были загружены из экселя.

Это **REST API**, для которого **PUT, DELETE, GET** и прочие запросы работают Используйте `filter` для тонкой выборки данных:

```
api/db/xdds.t1464/.filter(a='Дорога')
```

Отфильтрует таблицу по указанному значению. Если вы уже загрузили данные в куб, залинковав через административный интерфейс источники данных в виде экселя, то вам достаточно найти нужные таблицы и работая средствами **REST API** с таблицей `xdds.{Имя_листа}` вы сможете менять данные прямо в источнике данных. Куб же автоматически подхватит данные из источника.

7 External

Наш исторически первый метод создания кастомизированных визуализаций.

Итак, вы выбрали в типе визуализаций «Внешний» (`view_class = external` в JSON editor) Суть метода в том, что вы указываете ссылку на html файл, в котором прописана логика расчетов, рендера, разметка и подключены стили и необходимые скрипты с библиотеками. Например это может быть pie-чарт на чистом svg или canvas или простая таблица, но стилизованная так, как вам нужно. Этот файл (ресурс) может лежать как в разделе ресурсов текущего датасета. так и любого другого, например того же `ds_res`. Вы можете указать ссылку на файл вручную через JSON Editor панели настройки дешборда, прописав ключ в объект конфигурации дешлета

`url: res:template_koob.html` Эта запись означает, что ресурс находится в разделе resources того же, датасета, на котором мы находимся сейчас (вы видите его строковый идентификатор в url браузера)

`url: res:ds_res:template_koob.html` Запись грузит файл уже не из текущего датасета, а из датасета `ds_res`.

Контент файла в случае `external` подгружается во фрейме, размеры которого совпадают с зарезервированным под дешлет местом (заголовок дешлета остается, если только вы его не выключили в настройках). Так как это `iframe`, то при создании такого файла `html` важно помнить, что если вы обращаетесь к глобальным переменным родительского окна или url, то вы должны обращаться к ним через `window.parent.myvariable`. Чтобы шаблон загружал данные из источников и реагировал на события извне (от родительского окна), то вам будет нужна библиотека `bixel.js` (наше внутренняя разработка), которая будет обеспечивать общение родительского окна и фрейма, а самое важное: создавать основную событийную систему, на которой вы будете строить работу в рамках данного метода. `https://github.com/rcslabs/bixel` - здесь находится дока по библиотеке

Вы будете работать с несколькими основными событиями, которые вам понадобятся: `init` - не сколько событие, сколько просто асинхронный метод инициализации библитотеки в целом. принимает на вход объект, в котором можно искуственно ограничить значения. которые приходят по осям графика. например `{xsCount: 1}` - получить только одно значение, приходящее в массиве значений по оси X графика. Возвращает объект дешлета, позволяющий увидеть его конфигурацию и любые поля, что в этой конфигурации вами были записаны.

1. `loading` - запрос на сервер отправлен, но ответ еще не получен, режим `pending` - идеален для отрисовок прелоадеров

2. `no-data` - запрос ушел, но данных в указанном месте нет или таблица пустая. Может быть полезен в случае, если изначально вы и не хотите ничего загружать, а ждете триггерного действия от пользователя. В ином случае может понадобиться для вывода заглушки типа `“нет данных”`

3. `url` - подписка на изменение урла родительского окна. например когда нужно выполнить логику только при смене определенного параметра в урле.
4. `load` - запрос ушел, данные пришли, можно работать. Принимает объект осей `axes` и объект `data`, хранящий методы получения значений сервера на основе переданных значений по соответствующим осям графика. Основной метод, который будет вызывать рендеринг.

Обратите внимание, что событие `load` или `no-data` могут вызываться много раз без перезагрузки дешлета и это нормально, ведь причин смены различных фильтров и моделей, которые слушает дешлет может быть много. Это штатная ситуация, вам лишь нужно не забывать правильно и вовремя обновлять контейнеры там, где вы манипулируете с DOM и совершаете вставки. В случае canvas вам нужно будет обновлять его context при этом событии Рассмотрим пример шаблона html файла при таком способе разработки: необходимый минимальный набор приведен по ссылке: `https://drive.google.com/drive/folders/1I90arpQRPoyorh-SQQz6UmTPFvcrmzXR?usp=sharing` Здесь представлена и файл библиотеки `bixel.js` и файл-полифил для браузеров, которые не имеют функционала `Promise` (некоторые версии **Internet Explorer**) и пример файла-шаблона html.

```
<!DOCTYPE html>
<html>
<head>
<script>window.Promise || document.write('<scr'+'ipt src="es6-promise.auto.js">
  </sc'+'ript>');</script>
<script src="bixel.js"></script>
</head>
<body>
<div id="container"></div>
<script>
let dashlet, xs, ys, z, config, data, axes, vs;
bixel.init({
zsCount: 1, // xsCount,ysCount,zsCount - позволяют искуственно установить
  ограничение на
// количество элементов соответствующей оси.
}).then(function (dashlet_) {
dashlet = dashlet_; // сохраняем общий объект инфы о дешлете
config = dashlet_.config || {}; // отдельно сохраняем конфиг дешлета для
  удобства
});
// функция  проверяет, есть ли переопределение дименшнов или меж в разделе
// dataSource.style
// пример: getStyle‘(’measures, ‘sum_’value) - отдает пустой объект или объект со
// всем, что прописано в соответствующем блоке style
function getStyle(axisId, memberId) {
const config = dashlet.config;
const dataSource = config.dataSource || {};
const style = dataSource.style || {};
const axisStyle = style[axisId] || {};
const memberStyle = axisStyle[memberId] || {};
return memberStyle;
}
bixel.on('load', function (d, a) {
```

```
data = d; // сохранили объект информации о данных с сервера, что пришли
axes = a; // информациях об осях графика
xs = axes.getXs(); // элементы оси X графика (это энтити типа дименшн или межа)
ys = axes.getYs(); // значения оси У графика аналогичного оси X типа
z = axes.getZs()[0]; // ось Z представлена в виде одного элемента и имеет
// специфическое почти декоративное значение, которое можно оставить как есть, не вдаваясь в подробности.
// Пример получения данных метрик
vs =  xs.map(x => ys.map(y => data.getValue(x, y, z).valueOf()));
возвращает значения на точках пересечений осей X и У
// основная функция для рендера графика (всего лишь для примера, функция м.б. любой)
render();
});
bixel.on('loading', function (axes) {
console.log(axes);
});
bixel.on('no-data', function (axes) {
console.log(axes);
});
function render() {
// здесь ваш код и все обработчики кликов, манипуляции с DOM и т.д.
}
</script>
</body>
</html>
```

Технически, так как это обычный html файл – тут можно использовать любую библиотеку, которая вам понадобиться, например подключить `highcharts.js` и отрисовать его график на основе осей графика дешлета и значениях, что пришли в `data`. То есть библиотека `bixel.js` задает вам событийную систему, в рамках которой вы можете писать любой код и использовать любую библиотеку, даже `react.js`

8 Internal

Итак, вы выбрали в типе визуализаций «Внутрений» (`view_class = internal` в JSON editor).

Для `internal` так же нужен ключ `url`, который строится по тем же правилам, что и в `external`, но указывается уже `js` файл.

Так же можно в случае, если компонент находится в ресурсах датасета `ds_res` прописать вместо `view_class = название_файла_собранного компонента (без расширения .js)`, например `view_class = MyComponent`, тогда блок с ключом `url` уже не нужен. В данном случае веб-клиент ожидает, что ему будет отдан `js` файл, являющийся собранной версией `tsx` или `jsx` наследника `React.Component`. Соответственно в разделе `resources.` откуда будет браться такой файл (н-р, `MyComponent.js`) должны быть два файла: `MyComponent.js` и `MyComponent.js.map`, нативно берущиеся как результат сборки `webpack`. Но собирать такие файлы вручную вам не придется, за это отвечает проект `luxmsbi-web-resources`, созданный для манипуляции всеми `resources` всех датасетов сервера, с которым вы работаете. Рассмотрим проект `luxmsbi-web-resources`. Проект создан для упрощения работы с ресурсами Luxms BI и для организации их хранения в репозитории. Для каждого проекта создается своя ветка, которая будет хранить все ресурсные файлы, их историю и скрипты для синхронизации

8.1 Сценарий использования

Требования:

- Наличие админского имени пользователя и пароля для доступа к Luxms BI

1. Перейдите в ветку проекта
2. `git checkout <project_name>`
3. Установите зависимости
4. `npm install`
5. Запустите команду
6. `npm run pull`
7. которая скачает с сервера последнюю версию ресурсов (потребуются имя пользователя и пароль). Если ресурсы на сервере отличаются от локальных, будет выведен список изменений и ожидаться подтверждение перезаписи локальных файлов
8. Запустите проект
9. `npm start`
10. Зайдите браузером по адресу `http://localhost:3000`, там будет доступен сервер проекта, но файлы ресурсов будут браться из папки `/src` проекта

11. Работайте над файлами в папке `/src` и проверяйте их работу в браузере.
12. По окончании закоммитьте код в git-репозиторий и выполните команду
13. `npm run push`
14. Для отправки изменений в ресурсы сервера. Потребуются имя пользователя и пароль. Команда выведет список измененных файлов и запросит подтверждение их отправки на сервер.

8.2 Команды

Реализованы три команды

1. `npm run pull` Все ресурсы с сервера будут скачены в папку `/src` проекта. Лишние локальные файлы будут удалены. Команда запросит имя пользователя и пароль (если не указаны к конфиге), сравнит ресурсы сервера с локальными файлами, напечатает список измененных файлов и запросит подтверждение.

2. `npm run push` Заливает все локальные файлы на сервер. Отсутствующие локально файлы будут удалены на сервере. Команда запросит имя пользователя и пароль (если не указаны в конфиге), сравнит локальные файлы с серверными, напечатает список измененных файлов и запросит подтверждение.

3. `npm start` Запускает на `http://localhost:3000` http сервер для разработки с настроенным проксированием на указанный в конфиге сервер. Проксируются все файлы, кроме `/srv/resources` которые берутся из папки `/src`

8.3 Конфигурация

Проект считывает конфигурацию последовательно из разных источников. В конфигурацию входят:

- `server` - http адрес сервера, например http://project.luxmsbi.com/
- `username` - имя пользователя для доступа к серверу. Требуются админские права
- `password` - пароль для пользователя username
- `port` - порт для запуска локального сервера для npm start
- `force` - выдавать ли предупреждение перед обновлением источника
- `noRemove` - запрещает удалять файлы, если при синхронизации они не найены
- `include` - регулярное выражение для схем, которые следует включить (`ds_\w+$` по умолчанию)
- `exclude` - регулярное выражение для схем, которые следует исключить Если значения `server`, `username` или `password` нигде не найдены, то их потребуется ввести с клавиатуры.

8.4 Источники конфигурации

8.4.1 Опции командной строки

При запуске команды можно указать любые из опций:

- `-server=...`
- `-username=...`
- `-password=...`
- `-port=...`
- `-force`
- `-no-remove`
- `-include=...`
- `-exclude=...`

Опции командной строки необходимо отделять от команды знаком `-`

```
npm run push -- --server=http://project.luxmsbi.com/ --username=admin --
  password=secret --exclude=ds_resnpm start -- --server=http://project.luxmsbi.com/
```

8.4.2 Переменные окружения

Те же имена, но должны быть написаны заглавными буквами с префиксом BI_:

- `BI_SERVER`
- `BI_USERNAME`
- `BI_PASSWORD`
- `BI_PORT`
- `BI_FORCE`
- `BI_NO_REMOVE`
- `BI_INCLUDE`
- `BI_EXCLUDE`

Например:

```
BI_SERVER=http://project.luxmsbi.com/  npm startBI_FORCE=yes BI_EXCLUDE=ds_res
  npm start
```

```
export BI_USERNAME=adminexport BI_PASSWORD=secretnpm run push
```

8.4.3 Файлы конфигурации

В корне проекта лежит файл `config.json`, который хранит значение конфигурации server

```
{    "server": "http://project.luxmsbi.com/"}
```

Этот файл должен лежать в каждой ветке проекта и указывать корректный сервер. Можно создать в корне проекта файл `authConfig.json` и в нем указать имя пользователя и пароль

```
{    "username": "admin",    "password": "secret"}
```

Этот файл занесен в `.gitignore` и не должен попасть в `git`.

Для работы с разными проектами можно в файл authConfig занести разные записи, для каждой ветки. Ключем явояется название ветки `git`, значениями - `username` и `password`

```
{    "project1": {        "username": "project1admin",        "password":
  "secret"    },    "project2": {        "username": "project2admin",
  "password": "secret2"    }}
```

Приложение будет брать соотвествующий конфиг по названию текущей ветки `git`.

Ввод с клавиатуры Если значение не найдено ни одим из предыдущих способов, команды `npm start`, `npm run pull` и `npm run push` запросят их ввод с клавиатуры

8.5 Примеры

`server` - указать удаленный сервер, с которым синхронизировать ресурсы Способы указать сервер Аргументами командной строки:

```
$ npm run push -- --server=http://project.luxmsbi.com:8000/
```

Через переменную окружения

```
$ export BI_SERVER=http://project.luxmsbi.com:8000/$ npm run push
$ BI_SERVER=http://project.luxmsbi.com:8000/ npm run push
```

Через конфиг файл

```
$ cat config.json{  "server": "http://project.luxmsbi.com:8000/"}$ npm run push
```

`port` - локальный порт для запуска `npm start` Способы запустить проект на порту 8080, вместо порта по умолчанию 3000

```
$ npm start -- --port=8080
$ BI_PORT=8080 npm start
$ cat config.json{  "server": "http://project.luxmsbi.com",  "port": 8080}$ npm
  start
```

8.6 Примечания

Команда `npm start` не требует ввода имени пользователя и пароля, для работы ей требуется только server. Однако, если перейти в браузере на `http://localhost:3000/#/ds/ds_xxx/resources` то в консоли может запроситься имя пользователя и пароль (если не указаны каким либо другим образом). Они необходимы для конвертации `resource.id` в имя файла.

Этот проект позволяет вам использовать `git`, чтобы синхронизовать работу разработчиков над ресурсами сервера и без проблем совместно разрабатывать.

Важно помнить, что тому, что вы сами коммитите и пушите в репозиторий вы должны доверять больше, чем тому, что получили средствами `npm run pull`. Эту команду следует использовать единожды при первичной инициализации проекта web-resources. Проще говоря: если `src` пустой. Итого ваша работа должна быть следующей: вы создали ряд файлов, написали код компонентов и провели какую-то иную работу, убедились, что все работает на вашем локальном сервере так, как нужно, совершили `git commit`, `git pull`, `git push` (попутно на этих этапах вы разрешили конфликты при мердже, если они будут) и уже после этого вызываете `npm run push`. Эта команда соберет все ваши `jsx-tsx` файлы и массово зальет все ваши ресурсы на сервер. Произойдет сравнение файлов на предмет различий и вам выдастся вопрос на подтверждение действий (залить новые файлы (список), удалить файлы (список)) + укажется общее число файлов, которые отправляются на сервер. В случае подтверждения - вы перезапишите все файлы на сервере вашей версией файлов. Это важно помнить, так как в случае совместной работы вы можете перезатереть файлы друг друга. Именно поэтому настоящие версии файлов хранятся в `git`, а не на сервере. Именно поэтому обязательно перед `git push` делайте `git pull` и не доверяйте `npm run pull` (эту команду использовать только в крайних случаях, в идеале единожды за все время использования проекта).

Если же кто-то зальет на сервер файл обычным способом через **drug'n'drop**, то ваш проект об этом не узнает, как он не узнает, что кто-то вручную изменил код файла, зайдя на раздел `resources` и через редактирование файла изменив его контент. Здесь необходима договоренность между собой разработчиков и всех, кто управляет файлами о едином способе взаимодействия с ними. Плюс разумеется запрещать это правами доступа, где это уместно.

Итак, мы развернли локальный сервер на проекте `luxmsbi-web-resources`. Открываем страницу дешборда датасета с нужным дешлетом (он вполне может быть единственным и занимать все доступное для дешлетов пространство). После того как вы загрузили основную картину проекта через `npm run pull` команду проекта, т.е. у вас уже есть папки с названиями существующих датасетов, внутри одной из таких папок вы можете создавать реакт-компонент в виде `tsx` или `jsx` файлов с кодом например таким:

```
import React from "react";
export default class MyComponent extends React.Component<any> {
public state: {
someState: any;
};
public constructor(props) {
super(props);
this.state = {
someState: [],
```

```
};
public render() {
return (
<>Hello Word!</>
);
}
}
```

Такой файл вы прежде всего в базовом виде должны залить на сервер через команду `npm run push`, где он будет существовать как пара `js` и `js.map` файлов с тем же именем. Теперь вы можете выбрать новый файл в режиме редактирования дешлета как источник типа визуализации «Внутренний».

Все, теперь вы должны увидеть результат рендера вашего компонента в рабочей области дешлета. и вы можете писать код внутри своего файла на компьютере, где просто обновлять страницу при изменениях (хотя в большинстве случаев вы сразу увидите изменения без нее).

В качестве props вам придут несколько переменных, в основном служебны, но значимые для разработки эти: `dp, cfg, subspace`

`cfg:` Объект типа `IVizelConfig` содержащий информацию о текущем дешлете, его конфигурации и ряд методов для получения его основных полей

интерфейс `IVizelConfig`:

```
_raw: any;
dataset: IDatasetModel;
getDataset(): IDatasetModel;
getProperty(key): any;
setProperty(key: string, value: any): void;
getLegendItem(e: IEntity, idx?: number): tables.ILegendItem;
serialize(): tables.IRawVizelConfig;
clone(): IVizelConfig;
getDisplay(vizelType?: string): IVizelConfigDisplay;
getRange(): IRange;
disableRange(): void;
getUrl(): string;
dataSource?: tables.IDataSource;
getVizelType(): string;
setVizelType(vizelType: string): void;
setTitle(title: string): void;
getSubspacePtr(): ISubspacePtr;
controller: IVizelController;
title?: string;
description?: string;
legend?: { [id: string]: tables.ILegendItem; };
badValueColor?: string;
goodValueColor?: string;
normsMainColor?: string;
onClickDataPoint?: string | any;
onClick?: string | any;
cardId?: string;
```

```
externalUrl?: IUrl;
dashboardId?: string;
dashId?: string;
normStrategy?: string;
context?: any;
titleContext?: string[];
colorResolver?: IColorResolver;
titleResolver?: ITitleResolver;
getRaw(): tables.IRawVizelConfig;
```

Самый простой случай , при котором этот объект вам понадобится - использование метода `getRaw()` чтобы получить тот конфиг, что вы видите в режиме редактирования дешлета и поле `title` - название дешлета, плюс метод `getTitle`, которому на вход можно отдать межу или дименшн и получить итоговое значение его названия, взятое с учетом того, что для него может быть прописано в блоке `style` раздела `datasource`.

`subspace` - Собирательная модель, описывающая сущности, с которыми мы работаем,. Что на осях лежит и чем придется манипулировать.

интерфейс ISubspace:

```
koob?: string;
filters?: any;
axesOrder: any;
ms: IMetric[];
ls: ILocation[];
ps: IPeriod[];
xs: IEntity[];
ys: IEntity[];
zs: IEntity[];
aas?: IEntity[];
abs?: IEntity[];
dimensions?: any[];
measures?: any[];
getZ(idx: number): IEntity;
getY(idx: number): IEntity;
getX(idx: number): IEntity;
getMLP(z: IEntity, y: IEntity, x: IEntity): any;
reduce(nx: number, ny: number, nz: number): ISubspace;
isEmpty(): boolean;
splitByZ?(): ISubspace[];
splitByY?(): ISubspace[];
splitByX?(): ISubspace[];
getZYXIndexesByMLPIds(mid: string, lid: string, pid: string): [number, number,
  number];
projectData(mlpCube: any): IValue[][][];
getArity(): number;
getRawConfig(): any;
toString(): string;
```

Вам потребуются значения `xs`, `ys` и `zs`, чтобы из них строить логику отрисовки графика, а также получения информации о каждом фигуранте на осях через методы `getY`, `getX`, `getZ`.

`dp` - Датапровайдер, обеспечивает методы для получения данных с сервера в рамках того, что указано в `subspace`.

Модуль data_engine:

```
export declare module data_engine {
export type IMLPSubscribeCallback = (m: IMetric, l: ILocation, p: IPeriod, v: number) => void;
export type ISubscribeCallback = (z: IEntity, y: IEntity, x: IEntity, v: number) => void;
export interface IRawRequest {
data?: boolean;
norms?: boolean;
aggregate?: boolean;
}

export interface IRawResponse {
data?: tables.IDataEntry[];
norms?: tables.INormDataEntry[];
aggregate?: IAggregate;
}

export interface IRawDataProvider {
getRawData(mlpSubspace: IMLPSubspace, closest?: boolean): Promise<tables.IDataEntry[]>;
getRawNorms(mlpSubspace: IMLPSubspace): Promise<tables.INormDataEntry[]>;

getRawColors(mlpSubspace: IMLPSubspace): Promise<any>;
getAggregate(mlpSubspace: IMLPSubspace): Promise<any>;
load(request: IRawRequest, mlpSubspace: IMLPSubspace, closest?: boolean): Promise<IRawResponse>;
rawSubscribe(mlpSubspace: IMLPSubspace, callback: IMLPSubscribeCallback): IDisposable;
}
export interface ICubeProvider {
getCube(subspace: ISubspace, closest?: boolean): Promise<IValue[][][]>;
}
export interface IMatrixProvider {
getMatrixYX(subspace: ISubspace, closest?: boolean): Promise<IValue[][]>;
}
export interface IVectorProvider {
getVectorX(subspace: ISubspace, closest?: boolean): Promise<IValue[]>;

getVectorY(subspace: ISubspace, closest?: boolean): Promise<IValue[]>;
}
export interface IValueProvider {
getValue(subspace: ISubspace, closest?: boolean): Promise<IValue>;
}
export interface INormsProvider {
getNorms(subspace: ISubspace): Promise<INormsResponse>;
}
export interface IColorsProvider {
// getColors
```

```
43 }
44 export interface IDataProvider extends IRawDataProvider, ICubeProvider,
     IMatrixProvider, IVectorProvider, IValueProvider, INormsProvider,
     IColorsProvider {
45 subscribe(subspace: ISubspace, callback: ISubscribeCallback): IDisposable;
46 }

48 }
```

Для получения декартового произведения фигурантов на оси X и У - нужно вызвать метод

```
1 dp.getMatrixYX(subspace).then(data => {
2 // тут обработка данных в виде массива массивов значений (строка или число)
3 })
```

Если же вся картина не нужна, можно вызвать лишь одну строку или один столбец такой матрицы через методы `getVectorX`, `getVectorY`, аналогично методу выше, но массив `data` будет одномерный.

9 KOOB API

9.1 Получение мета-информации

9.1.1 GET /api/v3/koob/

9.1.2 GET /api/v3/koob/cube

9.1.3 GET /api/v3/koob/cube.column

9.2 Получение данных

9.2.1 POST /api/v3/koob/data

Пример запроса:

```
{
"with":"luxmsbi.public_hr_indicators1",
"columns":["count(distinct(degree))"],
"options":["!MemberAll","!ParallelHierarchyFilters"]
}
```

Модельный пример:

```
{ "with":"czt.fot",

"sort": ["-dor1","val1",["-","val2"],"-czt.fot.dor2", "summa",

"filters": {
"dor1": ["=", "ГОРЬК"],
"dor2": ["=", "ПОДГОРЬК"],
"dor4": ["=", null],
"dor5": ["=", null],
"Пол":  ["or", ["!="], ["ilike", "Муж"]],
"dt": ["between", "2020-01", "2020-12"],
"sex_name": ["=", "Мужской"],
"": [">", ["+",["col1", "col2"]], 100]
},

"having": {
"dt": [">","2020-08"],
```

```
18 },

20 "subtotals": ["dor3","-dor4"],

22 "return": "count", // возможно такое!!!

24 "columns": ["dor3", "czt.fot.dor4", "fot.dor5", 'sum((val3+val1)/100):summa', ↩
     {"new":"old"}, ["sum", ["column","val2"]], {"new": ["avg", ["+",["column",↩
     "val2"],["column","val3"]]]} ]]
25 }
```

9.2.2 POST /api/v3/koob/data?count

9.2.3 POST /api/v3/koob/data?meta

10 Основные Observable сервисы и объекты

Сервисы, которые мы используем в веб-клиенте унаследованы от базового `Observable` сервиса `BaseService`.

```
export interface IBaseModel {
error?: string | null;
loading?: boolean | number;
}
export declare class BaseService<M extends IBaseModel> extends Observable {
protected _model: M;
protected _subscriptions: IDisposable[];
private _depsWatcher;
private _hDependenciesNames;
private _hDepsModels;
private _initializedInstance;
constructor(initialModel?: M);
protected _createInitialModel(): M;
protected _addDependency(depId: string, dep: BaseService<any>, ↩
  doImmediateNotify?: boolean): void;
protected _addDependencies(deps: {
[id: string]: BaseService<any>;
}, doImmediateNotify?: boolean): void;
protected _removeDependency(depId: string, doImmediateNotify?: boolean): void;  ↩

protected _onDepsUpdated(newModels: IDepsModels, prevModels: IDepsModels): ↩
  boolean;
protected _onDepsReadyAndUpdated(newModels: any, prevModels: any): void;
protected _getDependencyModel<T>(depId: string): T;
protected _updateWithLoading(): void;
protected _updateWithError(error: string): void;
protected _updateWithData(partialModel: Partial<M>): void;
protected _smartUpdate(newModel: M, prevModel: M): M;
protected _isModelChanged(nextModel: M, currentModel: M): boolean;
protected _isUpdatesFrozen: boolean;
protected _freezeUpdates(fn: () => any): any;
private _notifyUpdate;
protected _setModel(model: M): void;
protected _updateModel(partialModel: Partial<M>): void;
/**
* Returns if service is ready and has data
*
* @returns {boolean}
*/
isReady(): boolean;
/**
```

```
39  * Wait until service in state 'ready' (which means, no error and no loading)
40  * return Promise<MODEL>
41  *
42  * if model signal error, it rejects resulting promise
43  *
44  * @returns {Promise<MODEL>}
45  */
46  whenReady(): Promise<M>;
47  subscribeAndNotify(event: string, listener: (model: M) => void): IDisposable;
48  subscribeUpdates(listener: (model: M) => void): IDisposable;
49  subscribeUpdatesAndNotify(listener: (model: M) => void): IDisposable;
50  getModel(): M;
51  protected _releaseChild(name: string): void;
52  protected _dispose(): void;
53  static dependency(target: BaseService<any>, key: string, descriptor?:
      TypedPropertyDescriptor<any>): any;
```

Его особенность состоит в реализации подхода классического `Observable`: Есть поле хранящее информацию об ошибке инициализации (`error`), статус окончания процесса инициализации (`loading`), методы работы с моделью (полная установка и частичная, геттер), методы подписки на изменения модели сервиса и уведомления подписчиков (весь блок `subscribes`...). Вы можете у класса-наследника такого сервиса вызвать метод `subscribeUpdatesAndNotify` передав этому методу аргументом функцию-колбек одного аргумента (этот аргумент есть будущая модель соответствующего сервиса) и тогда при любом изменении модели сервиса вы попадете в функцию-колбек, выполняющую ваше целевое дейтсвие. Таким образом, мы можем использовать там, где это обусловлено, один и тот же экземаляр такого сервиса и уже меняя его модель - триггерить реакцию компонентов-подписчиков на это изменение (ререндер, перезапрос данных и т.д.). Часть таких сервисов и моделей мы выносим в набор специальных пакетов на экспорт для использования внутри проекта `luxmsbi-web-resources`.

Пакет для импорта `bi-internal/core` ***AppConfig***

```
1  interface IAppConfig {
2  loading: boolean;
3  error: string;
4  requestUrls: any;
5  projectTitle: string;
6  locale: string;
7  language: 'en' | 'ru';
8  region: string;
9  features: string[];
10 plugins: string[];
11 entryPoint: any;
12 dataset: any;
13 map: any;
14 themes?: any;
15 }
16 export declare class AppConfig extends BaseService<IAppConfig> {
17 constructor();
18 protected _createInitialModel(): any;
```

```
private _loadSettingsJs;
hasFeature(featureName: string): boolean;
/**
* Replaces parts of URL according to settings
*
* @param url URL to proceed
* @returns {string} resulting URL
*/
fixRequestUrl(url: string): string;
static getInstance: () => AppConfig;
static getModel(): IAppConfig;
static fixRequestUrl(url: string): string;
static hasFeature(featureName: string): boolean;
static getProjectTitle(): string;
static getLocale(): string;
static getLanguage(): 'en' | 'ru';
static getPlugins(): string[];
}
```

Представляет собой сервис для получения объекта настроек веб-клиента из файла `settings/settings.js` на сервере. Пример: `AppConfig.getInstance().getModel()` - отдаст модель настроек в соответствии с интерфейсом `IAppConfig`.

AuthenticationService

```
interface IAuthCheckData2 {
sessionId: string;
user: IRawUser;
authType?: string;
}
interface IAuthCheckData3 {
access_level: string;
id: string;
name: string;
authType?: string;
config: IRawUserConfig;
}
export interface IAuthentication extends IBaseModel {
error: string | null;
loading: boolean;
authenticating: boolean;
authenticated: boolean;
userId?: number;
access_level?: string;
username?: string;
name?: string;
userConfig?: {
[key: string]: string;
};
isNeed2FACode?: boolean;
isBlocked?: boolean;
errorKey?: string;
errorMessage?: string;
}
```

```
export declare class AuthenticationService extends BaseService<IAuthentication> {
private constructor();
private _init;
protected _setModel(m: IAuthentication): void;
isAuthenticated(): boolean;
private _check;
check(): Promise<IAuthCheckData2 | IAuthCheckData3>;
private _notifyLoggedOut;
signIn(username: string, password: string): Promise<IAuthentication>;
signOut(): Promise<any>;
resendCode(): Promise<any>;
signInWithCode(code: string): Promise<any>;
static readonly NOT_AUTHENTICATED: string;
static readonly FORCE_LOGIN_KEY = "loginme";
static getInstance: () => AuthenticationService;
static getModel(): IAuthentication;
static subscribeUpdatesAndNotify(listener: (model: IAuthentication) => void): IDisposable;
static subscribeUpdates(listener: (model: IAuthentication) => void): IDisposable;
static unsubscribe(listener: (...args: any[]) => any): boolean;
static signOut(): Promise<any>;
static signInWithCode(code: string): Promise<any>;
static signIn(username: string, password: string): Promise<IAuthentication>;
static resendCode(): Promise<any>;
}
```

Сервис и его интерфейсы нужны для авторизации пользователя и получения информации о нем `AuthenticationService.getInstance().getModel()` - отдаст модель пользователя, который авторизовался.

`AuthenticationService.signIn('adm', 'luxmsbi')` - авторизует пользователя с указанным логином и паролем в системе.

`repo` - объект, хранящий три типа наборов интерфейсов

Adm

```

export declare type IRawUserConfig = {
[key: string]: string;
};
export interface IRawUser {
id: number;
name: string;
email: string;
username: string;
config?: IRawUserConfig;
}
export declare class UsersRepository extends BaseRepository<IRawUser> {
constructor();
protected _filter(e: IRawUser): boolean;
}
```

```
export interface IRawDataset {
id: number;
guid: string;
parent_guid: string | null;
version: string;
schema_name: string;
server_id: number;
owner_user_id: number | null;
head_dataset_id: number | null;
title: string;
description: string;
src_image_type: string;
images: {
source: string;
'15x15': string;
'130x130': string;
};
srt: number;
config: any;
serial: string;
sqlite_serial: null;
sqlite_snapshot_id: null;
logged_since: null;
sync_cfg: any;
is_archive: 0 | 1;
is_visible: 0 | 1;
is_db_ready: 0 | 1;
period_spans: any;
ui_cfg: {
[key: string]: string;
};
depends_on: [];
postprocess_sql: null;
}
export declare class DatasetsRepository extends BaseRepository<IRawDataset> {

constructor();
}
export interface IRawTopic {
id: number;
parent_id: number | null;
tree_level: number;
title: string;
srt: number;
dataset_id: number | null;
src_image_type: null;
images: {};
config: any;
}
export declare class TopicsRepository extends BaseRepository<IRawTopic> {
constructor();
}
export interface IRawTopicDatasetMap {
```

```
id: number;
topic_id: number;
dataset_id: number;
config: any;
srt: number;
}
export declare class TopicDatasetMapsRepository extends BaseRepository<IRawTopicDatasetMap> {
constructor();
}
export interface IRawUserGroup {
id: number;
title: string;
config?: IRawUserConfig;
}
export declare class UserGroupsRepository extends BaseRepository<IRawUserGroup> {
constructor();
}
export interface IRawDataSource {
id: number;
ident: string;
title?: string;
url?: string;
login?: string;
pass?: string;
config?: any;
}
export declare class DataSourceRepository extends BaseRepository<IRawDataSource> {
constructor();
}
```

Определяет интерфейсы для работы административного сервиса платформы

ds

```

export interface IRawConfig {
id: number;
cfg_key: string;
cfg_val: string;
cfg_type: string | null;
updated: string;
created: string;
}
export declare class ConfigsRepository extends BaseRepository<IRawConfig> {
constructor(schemaName: string);
}
export interface IRawUnit {
id: number;
parent_id: number | null;
scale: number;
scale_op: '*';
```

```
value_prefix: string | null;
value_suffix: string | null;
title: string;
tiny_title: string;
axis_title: string;
config: any;
updated: string;
created: string;
}
export declare class UnitsRepository extends BaseRepository<IRawUnit> {
constructor(schemaName: string);
}
export interface IRawMetric {
id: number;
parent_id: number | null;
tree_level: number;
tree_path: string | null;
title: string;
unit_id: number | null;
is_text_val: 0 | 1;
is_norm: 0 | 1;
is_calc: 0 | 1;
formula: null;
is_hidden: 0 | 1;
srt: number;
tags: string[];
src_id: string | null;
alt_id: string | null;
config: any;
updated: string;
created: string;
}
export declare class MetricsRepository extends BaseRepository<IRawMetric> {
constructor(schemaName: string);
}
export interface IRawMetricSet {
id: number;
title: string;
can_be_pie: number;
config: any;
updated: string;
created: string;
}
export declare class MetricSetsRepository extends BaseRepository<IRawMetricSet> ↩
  {
constructor(schemaName: string);
}
export interface IRawLocation {
id: number;
parent_id: number | null;
tree_level: number;
tree_path: string | null;
title: string;
```

```
latitude: number;
longitude: number;
is_hidden: 0 | 1;
srt: number;
tags: string[];
src_id: null;
alt_id: string | null;
config: any;
updated: string;
created: string;
}
export declare class LocationsRepository extends BaseRepository<IRawLocation> {

constructor(schemaName: string);
}
export interface IRawPeriod {
id: number;
parent_id: number | null;
tree_level: number;
period_type: number;
qty: number;
start_time: string;
title: string;
tags: string[];
src_id: null;
alt_id: string | null;
tree_path: string | null;
config: any;
updated: string;
created: string;
}
export declare class PeriodsRepository extends BaseRepository<IRawPeriod> {
constructor(schemaName: string);
protected _processTextResponse: (response: string) => string;
protected _sort(es: IRawPeriod[]): void;
}
export interface IRawDashboardTopic {
id: number;
parent_id: number | null;
icon_id: number;
title: string;
tree_level: number;
srt: number;
config: any;
updated: string;
created: string;
}
export declare class DashboardTopicsRepository extends
  BaseRepository<IRawDashboardTopic> {
constructor(schemaName: string);
}
export interface IRawDashboard {
id: number;
```

```
topic_id: number;
icon_id: number;
title: string;
srt: number;
config: any;
updated: string;
created: string;
}
export declare class DashboardsRepository extends BaseRepository<IRawDashboard> {
constructor(schemaName: string);
}
export interface IRawDashlet {
id: number;
parent_id: number | null;
dashboard_id: number;
view_class: string;
title: string;
description: string | null;
layout: '' | 'V' | 'H';
length: '';
idx: number;
config: any;
updated: string;
created: string;
}
export declare class DashletsRepository extends BaseRepository<IRawDashlet> {

constructor(schemaName: string);
protected _sort(es: IRawDashlet[]): void;
}
export interface IRawAttachment {
id: number;
config: any;
title: string;
attachment_type: 'vcp-lookup-table' | 'metric-values' | 'external-metric';
during?: string;
m_where?: string;
l_where?: string;
p_where?: string;
pt_where?: string;
u_where?: string;
data_source?: string;
norm_id?: number;
payload?: string;
alt_payload?: string;
}
export declare class AttachmentsRepository extends
BaseRepository<IRawAttachment> {
constructor(schemaName: string);
}
export interface IRawResource {
id: number;
```

```
alt_id: string;
content_type: string;
content_length: number;
created: string;
updated: string;
config: any;
}
export declare class ResourcesRepository extends BaseRepository<IRawResource> {

constructor(schemaName: string);
}
export interface IRawTextData {
id: number;
metric_id: number;
loc_id: number;
period_id: number;
val: string;
}
export declare class TextDataRepository extends BaseRepository<IRawTextData> {

constructor(schemaName: string);
protected _processTextResponse: (response: string) => string;
}
```

описывает интерфейсы основных энтити датасета

koob

```

export interface IRawCube {
id: string;
source_ident: string;
name: string;
title: string;
sql_query: string;
config: any;
}
export declare class CubesRepository extends BaseRepository<IRawCube> {
constructor();
}
export interface IRawDimension {
id: string;
source_ident: string;
cube_name: string;
name: string;
type: 'STRING' | 'NUMBER' | 'PERIOD' | 'SUM' | 'AGGFN';
title: string;
sql_query: string;
config: any;
}
export declare class DimensionsRepository extends BaseRepository<IRawDimension>
  {
source_ident?: string;
cube_name?: string;
```

```
26 constructor(source_ident?: string, cube_name?: string);
27 }
```

Описывает интерфейсы и классы для работы с koob (энтити OLAP кубов)

srv

Объект, описывающий в декларативном стиле основные сервисы тех же разделов `adm, koob, ds`. Каждый из них наследник сервиса `BaseEntitiesService`, который наследуется от `BaseService` и предоставляет методы получения объектов разного рода энтити платформы (датасеты, источники данных, дименшны, межи, юниты (ед.изм.) и т.д.).

adm

```
1
2  export declare class UsersService extends BaseEntitiesService<IRawUser> {
3  protected _repo: UsersRepository;
4  static readonly MODEL: IBaseEntities<IRawUser>;
5  protected constructor();
6  static getInstance: () => UsersService;
7  }
8  export declare class DatasetsService extends BaseEntitiesService<IRawDataset> { ↩

9  protected _repo: DatasetsRepository;
10 static readonly MODEL: IBaseEntities<IRawDataset>;
11 protected constructor();
12 static getInstance: () => DatasetsService;
13 }
14 export declare class TopicsService extends BaseEntitiesService<IRawTopic> {
15 protected _repo: TopicsRepository;
16 static readonly MODEL: IBaseEntities<IRawTopic>;
17 protected constructor();
18 static getInstance: () => TopicsService;
19 }
20 export declare class TopicDatasetMapsService extends ↩
     BaseEntitiesService<IRawTopicDatasetMap> {
21 protected _repo: TopicDatasetMapsRepository;
22 static readonly MODEL: IBaseEntities<IRawTopicDatasetMap>;
23 protected constructor();
24 static getInstance: () => TopicDatasetMapsService;
25 }
26 export declare class UserGroupsService extends ↩
     BaseEntitiesService<IRawUserGroup> {
27 protected _repo: UserGroupsRepository;
28 static readonly MODEL: IBaseEntities<IRawUserGroup>;
29 protected constructor();
30 static getInstance: () => UserGroupsService;
31 }
32 export declare class DataSourcesService extends ↩
     BaseEntitiesService<IRawDataSource> {
33 protected _repo: DataSourceRepository;
34 static readonly MODEL: IBaseEntities<IRawDataSource>;
35 protected constructor();
```

```
static getInstance: () => DataSourcesService;
}
```

ds

```

export declare class ConfigsService extends BaseEntitiesService<IRawConfig> {

protected _repo: ConfigsRepository;
static readonly MODEL: IBaseEntities<IRawConfig>;
protected constructor(schemaName: string);
protected _dispose(): void;
private static _cache;
static createInstance: (id: number | string) => ConfigsService;
}
export declare class UnitsService extends BaseEntitiesService<IRawUnit> {
protected _repo: UnitsRepository;
static readonly MODEL: IBaseEntities<IRawUnit>;
protected constructor(schemaName: string);
protected _dispose(): void;
private static _cache;
static createInstance: (id: number | string) => UnitsService;
}
export declare class MetricsService extends BaseEntitiesService<IRawMetric> {

protected _repo: MetricsRepository;
static readonly MODEL: IBaseEntities<IRawMetric>;
protected constructor(schemaName: string);
protected _dispose(): void;
private static _cache;
static createInstance: (id: number | string) => MetricsService;
}
export declare class MetricSetsService extends
  BaseEntitiesService<IRawMetricSet> {
protected _repo: MetricSetsRepository;
static readonly MODEL: IBaseEntities<IRawMetricSet>;
protected constructor(schemaName: string);
protected _dispose(): void;
private static _cache;
static createInstance: (id: number | string) => MetricSetsService;
}
export declare class LocationsService extends BaseEntitiesService<IRawLocation>
  {
protected _repo: LocationsRepository;
static readonly MODEL: IBaseEntities<IRawLocation>;
protected constructor(schemaName: string);
protected _dispose(): void;
private static _cache;
static createInstance: (id: number | string) => LocationsService;
}
export declare class PeriodsService extends BaseEntitiesService<IRawPeriod> {

protected _repo: PeriodsRepository;
```

```
44 static readonly MODEL: IBaseEntities<IRawPeriod>;
45 protected constructor(schemaName: string);
46 protected _dispose(): void;
47 private static _cache;
48 static createInstance: (id: number | string) => PeriodsService;
49 }
50 export declare class DashboardTopicsService extends ↩
     BaseEntitiesService<IRawDashboardTopic> {
51 protected _repo: DashboardTopicsRepository;
52 static readonly MODEL: IBaseEntities<IRawDashboardTopic>;
53 protected constructor(schemaName: string);
54 protected _dispose(): void;
55 private static _cache;
56 static createInstance: (id: number | string) => DashboardTopicsService;
57 }
58 export declare class DashboardsService extends ↩
     BaseEntitiesService<IRawDashboard> {
59 protected _repo: DashboardsRepository;
60 static readonly MODEL: IBaseEntities<IRawDashboard>;
61 protected constructor(schemaName: string);
62 protected _dispose(): void;
63 private static _cache;
64 static createInstance: (id: number | string) => DashboardsService;
65 }
66 export declare class DashletsService extends BaseEntitiesService<IRawDashlet> { ↩

67 protected _repo: DashletsRepository;
68 static readonly MODEL: IBaseEntities<IRawDashlet>;
69 protected constructor(schemaName: string);
70 protected _dispose(): void;
71 private static _cache;
72 static createInstance: (id: number | string) => DashletsService;
73 }
74 export declare class ResourcesService extends BaseEntitiesService<IRawResource> ↩
     {
75 protected _schemaName: string;
76 protected _repo: ResourcesRepository;
77 private _resources;
78 static readonly MODEL: IBaseEntities<IRawResource>;
79 static readonly RECYCLE_BIN: string;
80 protected constructor(schemaName: string);
81 updateContentType(id: number, content_type: string): Promise<IRawResource>;
82 updateAltId(id: number, alt_id: string): Promise<IRawResource>;
83 private _uploadFile;
84 updateContent(id: number, file: any): Promise<boolean>;
85 createContent(file: any): Promise<IRawResource>;
86 protected _isEntitiesEquals(newEntities: IRawResource[], prevEntities: ↩
     IRawResource[]): boolean;
87 protected _dispose(): void;
88 private static _cache;
89 static createInstance: (id: number | string) => ResourcesService;
90 }
91 export declare class TextDataService extends BaseEntitiesService<IRawTextData> ↩
```

```
  {
protected _repo: TextDataRepository;
static readonly MODEL: IBaseEntities<IRawTextData>;
protected constructor(schemaName: string);
protected _dispose(): void;
private static _cache;
static createInstance: (id: number | string) => TextDataService;
}
```

koob

```

export declare class CubesService extends BaseEntitiesService<IRawCube> {
static readonly MODEL: IBaseEntities<IRawCube>;
protected constructor();
static getInstance: () => CubesService;
}
export declare class DimensionsService extends ↩
  BaseEntitiesService<IRawDimension> {
private source_ident?;
private cube_name?;
static readonly MODEL: IBaseEntities<IRawDimension>;
protected constructor(source_ident?: string, cube_name?: string);
protected _dispose(): void;
private static _cache;
static createInstance: (source_ident: string, cube_name: string) => ↩
  DimensionsService;
}
```