244eb785b5e53da7f14df3e54c8b246886290ef6

Luxms BI

Руководство по кастомизации фронтенда Luxms BI

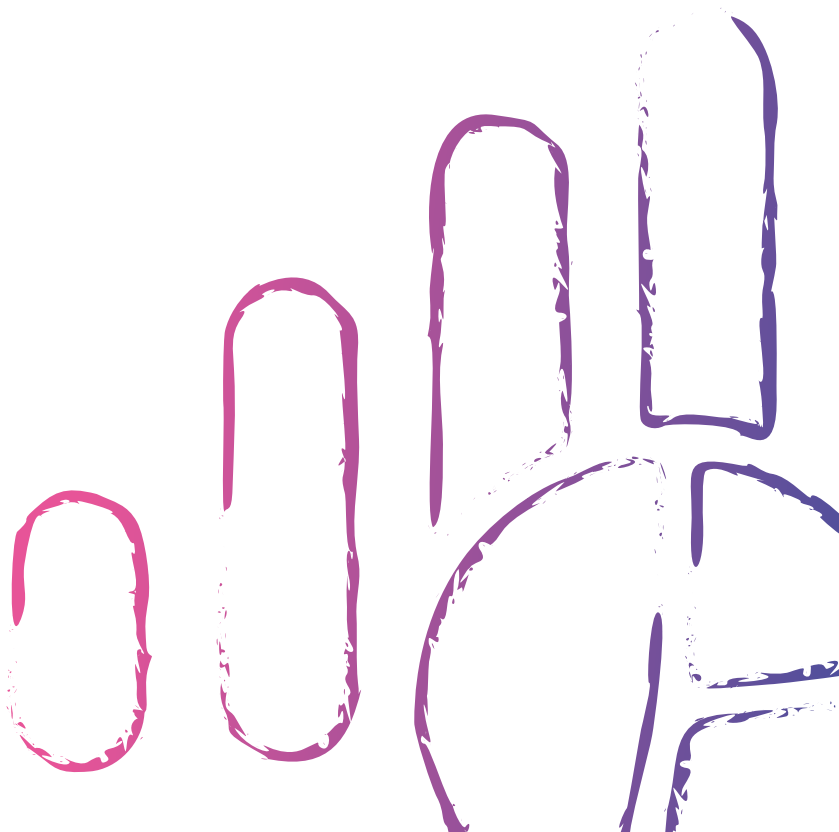
ИСПОЛЬЗУЕМЫЕ ПОДХОДЫ

РАЗРАБОТКА СОБСТВЕННЫХ КОМПОНЕНТОВ

НАСТРОЙКА ТЕМ, ЛОКАЛИЗАЦИИ, СТИЛЕЙ

СЕРВИСЫ

КНИГА РЕЦЕПТОВ



2024-01-24

Оглавление

1 Термины

Данная информация приведена для версии веб-клиента начиная с `9.0.0`. (Подход сохраняется, различия могут быть лишь в эскпортируемых по умолчанию модулях и компонентах)

Веб-клиент (Front-end, Клиент, шаблон, далее “обвязка, веб-клиент”) – веб-приложение Luxms BI для пользователей и администраторов, реализованное в виде HTML5/Javascript приложения для браузеров (поставщик: проект `luxmsbi-web-client`).

Коробка (коробочный, “из коробки”) - общая характеристика элементов, поставки, поведения и настройки компонентов, идущих в базовом виде веб-клиента и не требующих дополнительных действий, кроме тех, что указаны в основных руководствах пользователя продукта.

Атлас (Датасет, Набор данных) – логическая единица демонстрации агрегированных данных (метаданных), готовых дэшбордов и их настроек, полностью подготовленных для показа на **веб-клиенте**.

Имя схемы (“схема_нейм”) - это поле `schema_name` атласа, которое является строковым идентификатором, используемым в ряде запросов. В рамках инстанса должно быть уникальным. В идеале содержать логику именования ,выбранную вами и привязывающую атлас к инстансу, чтобы в случае переноса атласов не получить конфликт имен. По умолчанию, при создании равен `ds_${id атласа}`, например `ds_51`.

Дешборд (Dashboard) – аналитическая панель, на которой отображаются один или несколько дешлетов, объединенных какой-либо логикой.

Дешлет, деш (Dashlet, Dash) – аналитический блок, в первую очередь резервирующий место на дэшборде. По умолчанию соответствующий одному из встроенных типов визуализаций, но может хранить сразу несколько (типы визуализации “доска” и “табы”).

Визель (Vizel) - компонент React, по умолчанию рисующий график на основе входных данных и характеристик и особенностей самого типа графика. В одном деше может быть отрисовано более одного визеля.

Тип дэша, тип визуализации (`view_class`) - это строка, указывающая один из коробочных визелей, который нужно отобразить в данном дэшлете. Например, визель, позволяющий рендерить кастомный React-компонент, зовется **internal**

Глобальный - доступный для всех на всех атласах

Локальный - доступный в рамках одного атласа.

Источник данных - любое хранилище данных, в том числе файл `Excel` или `CSV`. Бывает локальным и глобальным.

Куб данных (далее “куб”) - абстракция OLAP-технологии, представляющая структуру данных, состоящую их Размерностей (Dimensions) и Фактов (Measures). Это плоская таблица с предрасчетами аггрегаций. Бывает глобальным и локальным. Локальный куб может иметь и локальный, и глобальный источник данных, но глобальный куб может иметь только глобальный источник данных.

Идентификатор куба - это `${имя источника}.${имя таблицы}`, например `"postgres.fortests_fortests_Postgres"`.

Размерности (Dimensions, Измерения, Разрезы, далее “дименшны”) – характеристики показателей в кубе данных. Обычно имеют иерархическую структуру.

Факты (Measures, далее “межи”) – числовые значения показателей в кубе данных.

Декартово произведение - каждое значение одного дименшна соединяется с каждым значением другого дименшна (кратно количеству выбранных дименшнов), формируя все возможные их комбинации, где для каждого из таких вариантов комбинаций посчитаны значения выбранных меж. Именно в таком виде вам будут приходить данные от куба (массив объектов, где каждый ключ - значение дименшна, межи или подытога в данной точке)

Подытог (subtotal) - данные, полученные без учета каких либо дименшнов. То есть, если мы укажем дименшнами `sex`, `name`, `age` и межами `sum(value)`, `avg(value)`, то в случае, если нас интересуют подытоги по дименшну `sex` - мы получим в итоговом ответе сервера декартово произведение без поля `sex`, но с двумя дополнительными полями, отражающих подытог по каждому из двух существующих меж (т.е. в аггрегации не будет разбивки по указанным в подытогах дименшнам).

Observable service (OC, далее “сервис”) - инстанс класа, являющегося наследником базового класса BaseService, который реализует паттерн Observer. Наблюдаемым объектом выступает модель такого сервиса, являющаяся аналогом простого js-объекта. Методы базового класса позволяют осуществлять подписку разных React-компонентов на изменения модели сервиса и менять ее встроенными методами (влияя на state React-компонента-подписчика через `callback`-функцию). Выступает как архитектурное решение, позволяющее синхронизировать множество компонентов, создавать кастомные события, их эммитеры и обработчики, экономить запросы на сервер за данными, хранить фильтры и многое другое.

Ресурс - в общем случае произвольный файл, находящийся в разделе resources атласа (то есть запись в таблицах вида `схема_нейм_атласа.resources`). Чаще всего в данном руководстве под этим будем подразумевать компонент React (уже собранный `webpack` в виде пары `.js`, `.js.map`) или модули и стили, которые он импортирует.

bi-magic-resources (luxmsbi-web-resources) (BMR, “веб-ресорсес”) – специализированный публичный проект на Github (https://github.com/luxms/bi-magic-resources), созданный для более комфортной разработки и кастомизации фронтенда Luxms BI командами наших клиентов. Способен управлять ресурсами и конфигурациями всех атласов, дешбордов и дешлетов

Ресурсы - раздел `resources` текущего атласа(ов) со всеми его файлами. Или даже все ресурсы во всех атласах разом (тогда речь идет о компонентах и файлах, которыми вы управляете в проекте `BMR`)

2 Введение

Основным поставщиком функционала и внешнего вида фронтенд-приложения Luxms BI по умолчанию является внутренний проект `luxmsbi-web-client` (далее **обвязка** или **веб-клиент**).

Веб-интерфейс, задаваемый этим приложением можно разделить на несколько структурных компонентов (строительных блоков):

1. Окно авторизации (страница с окном для ввода логина и пароля). Не видна при авторизации через SSO (за ненадобностью).

2. Стартовая (разводящая, Root) страница - страница, на которую вы автоматически попадаете при успешной авторизации в системе. В ней представлен список основных разделов (сегментов), управляющих той или иной областью BI (визуализации, данные, управление отчетами, коммуникации, администрирование и прочее).

3. Страница раздела (сегмента). Отражает стартовую страницу раздела с дочерними подразделами.

4. Страница подраздела. Например, страница с дешбордами конкретного атласа из раздела Визуализации.вводим

В данном руководстве нас будет большего всего интересовать раздел **Визуализации** и страница с дешбордами атласа.

Атлас имеет два важнейших раздела:

- **Дешборды** (`/dashboards`) В данном разделе доступны следующие действия:
  - Навигация на соседние атласы и их дешборды.
  - Управление дешбордами: создание, редактирование, удаление (необходимы соответствующие права)
  - Управление дашлетами: добавление, редактирование (в т.ч. расширенное редактирование с использованием **JSON Editor**-а) и удаление. (необходимы соответствующие права). Можно указать в качестве визуализации ваш кастомный React-компонент или **.html**.
  - Сохранение текущего состояния дешборда в презентацию.
  - Фильтрация данных и прочее.

- **Ресурсы** (`/resources`)
  - Загрузка, удаление файлов через drag’n’drop
  - Редактирование контента файлов, замена их MIME-типов

Ресурс имеет один из указанных MIME-типов:

```
'image/png',
'image/jpeg',
'image/svg+xml',
'text/xml',
'text/plain',
'text/html',
'text/css',
'text/javascript',
'application/javascript',
'application/x-javascript',
'application/sql',
'application/json',
'text/markdown'
```

2.1 Служебный атлас “Luxms BI Resources” (“ds_res”)

Атлас не имеет других разделов, кроме `resources`. Его ключевая особенность - ресурсы в данном атласе доступны всем пользователям.

Но это не все. Атлас используется как хранилище того, к чему хотим иметь доступ везде и всегда, без оглядки на права пользователя и, как следствие, компонентов, переопределяющих структурные блоки приложения, логотип, favicon и различные картинки для разделов.

Однако, в версиях, старше 9.0.0, появилась возможность указывать родительские атласы для атласов. Таким образом, родительский атлас выступает в качестве хранилища доступных везде ресурсов в рамках тех атласов, что доступны текущему пользователю. При этом доступ к `ds_res` сохраняется, просто число хранимых там ресурсов сокращается и локализуются в родительских атласах, позволяя достигать ситуаций, когда разные пользователи с разными ролями видят кардинально разные внешне и по функционалу типы фронтенда Luxms BI.

2.2 Файл с общими настройками приложения “settings.js”

Данный файл располагается в сборке модуля `luxmsbi-web-client`, расположенной на вашем сервере. Обычно она располагается по адресу `/opt/luxmsbi/` Внутри данной папки есть файл `settings/settings.js` с примерно такой структурой (полей может быть больше)

```
BISettings = {
  /* Название текущего проекта - по умолчанию отображается рядом с логотипом в интерфейсе
  и в постфиксе в названии вкладки браузера */
  projectTitle : 'Luxms BI',

  /* используется ли keycloak для авторизации */
  keycloak: null,
  /* Отвечает за стартовую точку, куда вы попадаете после авторизации
```

```
  Представляет собой частичную модель сервиса UrlState (о нем в разделе
  'Observable сервисы' данного руководства), которая принудительно выстроится по
  сле того, как пользователь авторизуется и попадет на корневую страницу
  `${sitename.ru}/#/`

 Обратите внимание, что пока этот функционал включен, то он не позволяет вернуть
  ся на прежний вид стартовой страницы и индифферентен к правам доступа и сайтов
  ой роли пользователя. Этот редирект происходит принудительно для всех */

  /* в данном случае мы переходим в раздел `dashboards` датасета со schema_name =
   ds_1661 */

  entryPoint: {
    route: "#dashboards",
    segment: "ds",
    segmentId: "ds_1661"
  },

  /* Текущий ключ локализации, по умолчанию "ru-RU" или "en-US",
  но может быть иной, если вы загрузите дополнительные файлы с локализацией
  в раздел ресурсов родительского атласа или ds_res */
  language: "ru-RU",

  /* Список используемых плагинов. По умолчанию включен плагин Презентации. Оста
  льные плагины, например chats (Обсуждения), tasks (Поручения) и другие требуют
   расположения в ресурсах специфических структур файлов. */
  "plugins": [
    "presentations"
  ],

  /* Список используемых редиректов для запросов */
  requestUrls: {
    serverUrl: ''
  },

  /* Используемые опции (фичи). Считаются только те, что не содержат префикса _ п
  еред именем.
  В данном случае отключен импортер и отключена технология MLP-кубов */
  features: ['_Importer', 'DisableMLP'],

  /* функция, которая будет вызвана при аутентификации чтоб получить аутентифика
  ционный bearer токен. Возвращает промис или само значение, можно вернуть токен
   как строку или структуру, где есть token и expresIn. Например, если токен для
   keycloak нам выдают через window.postMessage, то вот такая функция его подхва
  тит.
  */
 getAuthToken: function() {
    if (window.parent === window) return;
    return new Promise(function (resolve) {
      var onMessage = function (event) {
        if (event.data.type === 'jwtToken') {
          window.removeEventListener('message', onMessage)
```

```
            resolve(event.data);
          }
        };
        window.addEventListener('message', onMessage);
        window.parent.postMessage({"type":"getJwtToken","forceUpdate":false}, 'https://graph-demo-ui.datacloud.t1-cloud.ru/');
      });
    },
    /*
    В данном объекте можно переопределить ключи существующих тем обвязки, добавить свою тему, отключить существующую. Текущая запись означает, что темы берутся по умолчанию, коробочные.
    Не рекомендуется использовать без необходимости изолировать темы для редактирования пользователями. По умолчанию для кастомизации тем используется специальный файл themes.json, помещаемый как в ds_res атлас (общие темы), так и в текущий атлас (уникальный набор тем только для него)
    */
    themes: null,

    /* Общие настройки карт для всех дешлетов типа Карта */
    "map": {
      /* "osm" - OpenStreetMap */
      type: 'osm',
      /*  Рекомендуем принудительно включать векторный тип тайлов */
      osmVectorEnable: true,
      /*  Базовые настройки стилей карты, могут быть переопределены для каждой из тем через themes.json */
      osmUrlTemplate: 'https://vectortiles.luxmsbi.com/styles/basic-preview/style.json',
      osmSubdomains: '',
      osmAttribution: 'textcopyright OpenStreetMap contributors',
      mapCircleRadius: 35,
      minZoom: 2,
      maxZoom: 19
    }
};
```

Вы можете помещать сюда какие-то свои опции, структуры, которые имеют отношение только к текущему инстансу Luxms BI и влияют на поведение фронтенда, который вы реализуете на платформе.

Получить такой объект с настройками вы сможете в ваших кастомных компонентах (используя проект `BMR`), импортируя модуль `AppConfig` из пакета `bi-internal/core` и взяв его модель: `AppConfig.getModel()`. Или же в ознакомительных целях набрав в консоли браузера `__appConfig.getModel()`.

3 Темы

Тема в Luxms BI - это именованный список свойств в формате JSON, каждое из которых описывает в общем случае css-свойства (цвет, тень или значение бекграунда), используемые в **обвязке**, а также в нескольких специфических случаях может являться объектом, который необходим для управлением стилями целых модулей, вроде карты или графиков коробочных дешлетов, но конечные свойства такого вложенного объекта по итогу также есть css-свойства.

По итогу эти свойства превращаются в список css-переменных, которые используются **обвязкой** по умолчанию.

Итоговый список свойств для темы содержит как обязательные свойства, которые используются **обвязкой**, так и опциональные, которые вы добавили сами и используете при написании своих компонентов. Но тогда такие свойства должны встречаться во всех темах, или вы рискуете получить некорректное css-свойство при переключении на остальные темы.

Характер хранимых значений в свойствах тем поэтому совпадает с характером css-переменных: стандартные значения, которые можно присвоить, обычным свойствам CSS, например, цвет фона, цвет шрифта, высоту шрифта, ширину и высоту элементов и так далее.

Эти значения можно получить и при разработке в проекте `BMR` в файлах `.scss` ваших React-компонентов, чего мы коснемся чуть позже в данном руководстве.

Темы переключаются иконкой в тулбаре хидера в правом верхнем углу:

Светлая (на темную)

Рис. 3.1 **light.png**

Темная (на светлую)

Рис. 3.2 **dark.png**

В случае, если тем более двух - вы увидите иконку с дропдауном по клику и выбором нужной темы.

Рис. 3.3 **multi.png**

Темы в нашей концепции играют чисто декоративную роль, они не занимаются управлением расположением блоков на странице разделов, позиционированием и любой стилизацией более сложной, нежели цвета, шрифты, тени и бекграунд. То есть технически запретить это мы не можем, но настоятельно рекомендуем так НЕ делать. Такое рано или поздно вносит неоднозначности и сложно поддерживается.

4 Локализация

В Luxms BI локализация есть перевод пользовательского интерфейса, включая весь строковый контент обвязки на один из используемых в инстансе языков. Язык локализации по умолчанию определяется своим ключом из файла `settings.js` и наличием файла `.json` с кратким ключом выбранного языка в обвязке (по умолчанию, в обвязке есть только `en.json` и `ru.json` для английского и русского языков соответственно). Пока что встроенного переключателя языков в интерфейсе Luxms BI нет, но появится в одной из минорных версий v9 клиента. Расширение языкового набора возможно, если вы разместите в ресурсах атласа `ds_res` папку `luxury_store/locales/` и поместите туда переведенную версию одного из существующих файлов локализации, взяв его например по адресу `имя_вашего_сайта/assets/locales/ru.json`.

Пример такого файла:

```
{
"LuxmsBI": "Luxms BI",
"home": "Главная",
"back": "Назад",
"close": "Закрыть",
"collapse": "Скрыть",
"add": "Добавить",
"Between": "Между",
"More": "Больше",
"Less": "Меньше",
"contacts": "Контакты",
"header": "Заголовок",
 // много-много ключей
}
```

То есть для французского языка вы должны поместить в папку файл `fr.json` и загрузить его на сервер или средствами проекта `BMR` или загрузить этот файл в раздел ресурсов вручную, через drag'n'drop, и изменить его название на полный путь до папки выше: `luxury_store/locales/fr.json`.

Для существующих языков типа `ru` и `en` вы можете создать аналогичным образом файл `.json`, но уже размесить туда не полный перевод, а только те ключи и их значения ,которые хотите поменять. Тогда клиент, найдя одноименные файлы в ресурсах, просто смержит их между собой.

5 API для работы с OLAP-кубами

5.1 Доступные эндпойнты:

`GET` `api/db/${schema_name}.cubes/`

Получить конфигурации всех кубов (локальных и глобальных (`schema_name` = `'koob'`))

Ответ:


```
[
  {
    "id": "mysource.custom_dm_itog_reestr",
    "source_ident": "mysource",
    "is_source_global": 0,
    "name": "custom_dm_itog_reestr",
    "is_global": 0,
    "title": "custom_dm_itog_reestr",
    "sql_query": "select dm_itog_reestr_values.class_code1_name as
  class_code1_name,\n       dm_itog_reestr_values.class_code3_name as
  class_code3_name,\n       dm_itog_reestr_values.class_code4_name as
  class_code4_name,\n       dm_itog_reestr_values.cost_categories_name as
  cost_categories_name,\n       dm_itog_reestr_values.cost_sub_categories_name
  as cost_sub_categories_name,\n       dm_itog_reestr_values.do_name as do_name,
  \n       dm_itog_reestr_values.year_period as dt,\n
  dm_itog_reestr_values.year_period as year_period,\n
  dm_itog_reestr_values.fact as fact,\n       dm_itog_reestr_values.mnt_period
  as mnt_period,\n       dm_itog_reestr_values.mr_name as mr_name,\n
  dm_itog_reestr_values.param_name as param_name,\n
  dm_itog_reestr_values.plan as plan,\n       dm_itog_reestr_values.process as
  process,\n       do_code, mr_code, process_code, cost_categories_id,
  cost_sub_categories_id, class_code1, class_code3, class_code4 \nfrom
  dm_itog_reestr_values\nwhere ${filters()}  and mnt_period <> 'БП'\nunion all \
  nselect dm_itog_reestr_drivers.class_code1_name as class_code1_name,\n
  dm_itog_reestr_drivers.class_code3_name as class_code3_name,\n
  dm_itog_reestr_drivers.class_code4_name as class_code4_name,\n
  dm_itog_reestr_drivers.cost_categories_name as cost_categories_name,\n
  dm_itog_reestr_drivers.cost_sub_categories_name as cost_sub_categories_name,
  \n       dm_itog_reestr_drivers.do_name as do_name,\n
  dm_itog_reestr_drivers.year_period as dt,\n
  dm_itog_reestr_drivers.year_period as year_period,\n
  dm_itog_reestr_drivers.fact as fact,\n
  dm_itog_reestr_drivers.mnt_period as mnt_period,\n
  dm_itog_reestr_drivers.mr_name as mr_name,\n
  dm_itog_reestr_drivers.param_name as param_name,\n
```

```
  dm_itog_reestr_drivers.plan as plan,\n       dm_itog_reestr_drivers.process as process,\n       do_code, mr_code, process_code, cost_categories_id, cost_sub_categories_id, class_code1, class_code3, class_code4 \nfrom dm_itog_reestr_drivers\nwhere ${filters(do_code,mr_code,dt,mnt_period,mnt_dt)} and mnt_period <> 'БП'",
    "config": {
      "skip_where": 1,
      "is_template": 1
    }
  },
  {
    "id": "mysource.itog_table",
    "source_ident": "mysource",
    "is_source_global": 0,
    "name": "itog_table",
    "is_global": 0,
    "title": "itog_table",
    "sql_query": "/*SELECT * FROM dm_itog_table_t where  ${filters()}*/\nselect * from dm_itog_table_t \nwhere ord not in (202,203,204,205,206,207,208,209)\nand ${filters()}\nunion all\n select distinct mnt_period,do_code,mr_code,process_code,cost_categories_id,cost_sub_categories_id,0,0,0,grp_name,metric_name,value,ord,unit,norm,unit_id,dt\n from dm_itog_table_t \n where ord in (202,203,204,205,206,207,208,209) \n and ${filters(except(class_code1,class_code3,class_code4))}",
    "config": {
      "skip_where": 1,
      "is_template": 1
    }
  },
  {
    "id": "mysource.transport_e_m",
    "source_ident": "mysource",
    "is_source_global": 0,
    "name": "transport_e_m",
    "is_global": 0,
    "title": "transport_e_m",
    "sql_query": "select dm_itog_final_all.class_code1 as class_code1, dm_itog_final_all.class_code1_name as class_code1_name, dm_itog_final_all.class_code3 as class_code3, dm_itog_final_all.class_code3_name as class_code3_name, dm_itog_final_all.class_code4 as class_code4, dm_itog_final_all.class_code4_name as class_code4_name, dm_itog_final_all.cost_categories_id as cost_categories_id, dm_itog_final_all.cost_categories_name as cost_categories_name, dm_itog_final_all.cost_sub_categories_id as cost_sub_categories_id, dm_itog_final_all.cost_sub_categories_name as cost_sub_categories_name, dm_itog_final_all.do_code as do_code, dm_itog_final_all.do_name as do_name, dm_itog_final_all.driver_code as driver_code, dm_itog_final_all.driver_id as driver_id, dm_itog_final_all.dt as dt, dm_itog_final_all.etalon as etalon, dm_itog_final_all.mnt_period as mnt_period, dm_itog_final_all.mr_code as mr_code, dm_itog_final_all.mr_name as mr_name, dm_itog_final_all.norm as norm, dm_itog_final_all.process as process, dm_itog_final_all.process_code as process_code, dm_itog_final_all.unit as unit, dm_itog_final_all.unit_id as
```

```
  unit_id, dm_itog_final_all.value as value,  dm_itog_final_all.etalon_do_name as etalontitle,  dm_itog_final_all.marka as marka\nfrom dm_itog_final_all",
    "config": {}
  }]
```

Для эндпойнтов типа `api/db/${schema}.${table}` доступна фильтрация через блок `./filter(${boolean_expression})`, например такая

`.filter(is_source_global=0&&is_global=0)` - вернет конфиги только локальных кубов с локальными источниками

`GET api/db/${schema_name}.dimensions/`

Получить конфигурации всех дименшнов всех кубов (локальных и глобальных (`schema_name` = `'koob'`))

Ответ:

```
[
  {
    "id": "mysource.custom_dm_itog_reestr.class_code1",
    "source_ident": "mysource",
    "cube_id": "mysource.custom_dm_itog_reestr",
    "cube_name": "custom_dm_itog_reestr",
    "is_cube_global": 0,
    "name": "class_code1",
    "is_global": 0,
    "type": "NUMBER",
    "title": "class_code1",
    "sql_query": "class_code1",
    "config": {
      "possible_aggregations": [
        "sum",
        "avg",
        "min",
        "max",
        "count"
      ]
    }
  },
  {
    "id": "mysource.custom_dm_itog_reestr.class_code1_name",
    "source_ident": "mysource",
    "cube_id": "mysource.custom_dm_itog_reestr",
    "cube_name": "custom_dm_itog_reestr",
    "is_cube_global": 0,
    "name": "class_code1_name",
    "is_global": 0,
    "type": "STRING",
    "title": "class_code1_name",
    "sql_query": "class_code1_name",
    "config": {
     "possible_aggregations": []
```

```
    }
  },
  {
    "id": "mysource.custom_dm_itog_reestr.class_code3",
    "source_ident": "mysource",
    "cube_id": "mysource.custom_dm_itog_reestr",
    "cube_name": "custom_dm_itog_reestr",
    "is_cube_global": 0,
    "name": "class_code3",
    "is_global": 0,
    "type": "NUMBER",
    "title": "class_code3",
    "sql_query": "class_code3",
    "config": {
      "possible_aggregations": [
        "sum",
        "avg",
        "min",
        "max",
        "count"
      ]
    }
  },]
```

Доступна фильтрация через `.filter(${boolean_expr})`:

`.filter(source_ident='${source_ident}'&&cube_name='${cube_name}'&&is_global=0)` - вернет конфиги дименшнов локальных кубов

`POST /api/v3/${schema_name}/data`

Получить данные для указанного куба. Возвращается декартово произведение дименшнов и меж.

Тело запроса:

`Content-type: application/json`

```
{
  /* REQUIRED идентификатор куба, STRING */
  with: "luxmsbi.custom_country_currency",

  /* REQUIRED массив строк, которые есть или идентификаторы дименшнов, или аггрегационные функции над полями, STRING[] */
  columns: ["country_code", "sum(cnt):sum_cnt", "count(cnt):ccc"],

  /* REQUIRED В общем виде объект с фильтрами на данные при запросе, аналогично тому, как это задается в конфиге дешлетов (кроме записей вида `country_code: true`) */
  filters: {
    country_code: ["=", "BLR", "CHN", "KAZ", "RUS", "USA"]
  },
```

```
  /* отступ от начала таблицы, NUMBER */
  offset: 0,

  /* количество строк, NUMBER */
  limit: 128, // по умолчанию 128 строк, для отключения органичений укажите 0

  /* массив строк типа +id, -date, где + и - это ASC и DESC,а остальное - идентификаторы дименшнов */
  sort: ["+sum_cnt"],

  /* массив опций для управления иерархиями и блоками в БД типа "Все" */
  /* MemberAll - */
  /* ParallelHierarchyFilters - */
  options: ['!MemberAll', '!ParallelHierarchyFilters'],

  /* массив идентификаторов дименшнов, по которым будут считаться подытоги */
  subtotals : ["country_code"]

  /* массив идентификаторов дименшнов, по которым нужны distinct запросы, не имеет смысла, если есть хоть одна межа в запросе */
  distinct: ["country_code"]
  }
```

Ответ:

`Accept: application/stream+json` `Content-type: application/json`

```
{"country_code":"BLR","sum_cnt":1,"ccc":1,"Σcountry_code":0}
{"country_code":"CHN","sum_cnt":1,"ccc":1,"Σcountry_code":0}
{"country_code":"KAZ","sum_cnt":1,"ccc":1,"Σcountry_code":0}
{"country_code":"RUS","sum_cnt":1,"ccc":1,"Σcountry_code":0}
{"country_code":"USA","sum_cnt":1,"ccc":1,"Σcountry_code":0}
```

`POST /api/v3/${schema_name}/data?meta`

Получить вспомогательную информацию по запросу за данными. Например итоговый запрос `SQL`, выполняющийся в БД

Тело запроса:

Аналогично телу для `/data`

Ответ:

`Accept: application/stream+json` `Content-type: application/json`

```
{
  "id": "luxmsbi",
  "query": "SELECT DISTINCT country_code as country_code, sum(cnt) as sum_cnt, count(cnt) as ccc, GROUPING(country_code) AS \"Σcountry_code\"\nFROM (select custom.country_currency.country_code as country_code,\n       custom.country_currency.currency_code as currency_code,\n       1 cnt\nfrom custom.country_currency) AS custom_country_currency\nWHERE (country_code IN
```

```
  ('BLR', 'CHN', 'KAZ', 'RUS', 'USA'))\nGROUP BY GROUPING SETS ((country_code),↩
  \n                        (country_code)\n                        )\nORDER BY ↩
  sum_cnt LIMIT 128 OFFSET 0",
  "datagate_url": "http://datagate",
  "preserveCase": true
}
```

`POST` `/api/v3/${schema_name}/count`

Получить число строк для запроса в указанный куб (запрос аналогичен `/data`, но отдается `COUNT(*)`).

Тело запроса:

Аналогично телу для `/data`

Ответ:

`Accept`: `application/stream+json` `Content-type`: `application/json`

```
{"count":5}
```

6 Кастомизация

6.1 Слоистая структура фронтенда Luxms BI

Конструктор итогового вида фронтенда Luxms BI собирается из следующих элементов:

1. Коробочный вид компонентов текущей версии веб-клиента. Определяется пакетом `luxmsbi-web-client`.
2. Текущая тема, определяющая цветовую палитру, тени, бекграунд и набор цветов по умолчанию для графиков.
3. Текущая локализация.
4. Переопределение компонентов-строительных блоков из ресурсов (набор таких компонентов ограничен и будет приведен позже).
5. Обусловленные разделом уровни стилизации. Например переопределение компонентов, тем и стилей только для конкретного атласа. В случае стилей - и для конкретного дешборда. В случае дешей - указание кастомного компонента только у конкретного(ных) деша(ей).

Помимо этого, для раздела `dashboards` атласа возможно подключить глобальные файлы стилей, поместив их в раздел `resources` текущего атласа, `ds_res` или родительских атласов (подключаются на страницу при помощи тега `<link>`): `_global.css` или (`styles.↩ css`). Такой файл будет на всех дешбордах атласа. В случае с `ds_res` - на всех атласах инстанса вообще.

Так же, учитывая ролевые особенности родительских атласов и `ds_res`, вы можете разместить файлы стилей `_global.css` в каждом из них и по итогу получить несколько подключений файлов стилей, по приоритету:

- сначала стили `ds_res` (если есть - будут подключаться к каждому атласу и его дешбордам)
- затем цепочка стилей родительских атласов (только для дочерних атласов и их дешбордов)
- стили текущего атласа (для всех дешбордов текущего атласа)
- Стили для текущего дешборда текущего атласа будут дополнительными стилями. Такой файл для дешборда с `id = 1` будет называться `styles.dboard=1.css`

Под **кастомизацией фронтенда Luxms BI** мы будем подразумевать ситуацию, когда требования вашей бизнес-логики выходят за рамки коробочного функционала, предоставляемого фронтендом, и достичь ваших целей вы можете только работая с пунктами 2 - 4 выше.

6.2 Как достигается сама кастомизация?

Основной механизм, используемый в проекте `luxmsbi-web-client` для кастомизации строительных компонентов, дешлетов, тем и стилей: запрос на наличие соответствующих файлов в разделе `resources` в иерархии атласов перед тем как `webpack` займется построением дерева компонентов для билдования. Иерархия таких файлов работает так же, как в случае с глобальными стилями выше. По итогу используется файл, ближайший в иерархии к текущему разделу (атласу, например).

Рассмотрим принцип кастомизации строительного блока страницы:

В проекте `luxmsbi-web-client` есть компонент-обертка `LoadFromResources`, который, перед тем как отдать `webpack` компонент `MyComponent.tsx`, проверяет наличие файла `MyComponent.js` (это должен быть собранный React-компонент) в разделе `resources` текущего атласа и если он там присутствует, то использует его, если нет - идет в родительский атлас за одноименным ресурсом (поднимаясь выше по уровню вложенности атласа), если файла нет ни в одном из родительских атласов, то в `ds_res`, а если и в нем нет - вставляет коробочную версию компонента (из папки `src` проекта).

По похожему принципу работает и переопределение фавиконок, основного лого, анимированного лого при загрузке, файла с настройками тем, локализации, плагинов, картинок разделов и загрузки компонентов для подключения в качестве кастомной визуализации в дешлете.

7 Вспомогательный проект bi-magic-resources (BMR)

Данный проект является публичной версией закрытого проекта `luxmsbi-web-resources`. Оба проекта по функционалу идентичны и решают одни и те же задачи:

- Управлять ресурсами всех атласов, а в общем случае умеет получать в виде файлов `.json` настройки атласа, дешбордов и дешлетов и изменять их прямо в проекте, с последующей выгрузкой на сервер.
- Использовать git и версионировать разработку.
- Настроить CI/CD для управления ресурсами на деве, тесте, проде и т.д.
- Работать над ресурсами проекта командам разработчиков.
- Позволяет вносить изменения в поведение и внешний вид итогового веб-клиента без привлечения devops и backend а лишь силами встроенных методов и скриптов проекта `BMR` и провайдера в лице `luxmsbi-web-client`
- Можно массово загружать все ресурсы на сервер (или же только специфических атласов, если требуется), проверяя перед этим, новый ли это файл (т.е. его нужно на сервере создать), файл которого больше нет (его надо удалить на сервере), или существующий (будет перезаписан).

Первичная настройка проекта выглядит так:

- Зайдите на https://github.com/luxms/bi-magic-resources
- Ознакомьтесь с документацией в https://github.com/luxms/bi-magic-resources#readme
- Сделайте форк данного проекта в свой github аккаунт и сделайте `git clone` себе на компьютер. Если у вас есть команда - то пусть это лучше сначала форкает тимлид, а вы потом клоните его версию, но это опционально. Идея только в том, чтобы вы использовали единый репозиторий по понятным причинам.
- Этот проект создаст необходимое окружение.
- Запустите `npm install` (или `yarn`, лично мы больше любим его).
- В корне проекта найдите файл `config.json` и добавьте в поле `server` адрес инстанса Luxms BI (`"server": "https//mysite.ru/"`).

Т.е. тот урл адрес, под которым у вас открываются дашборды, но только до символа “#”. Вот полный список параметров `config.json` из документации (для ленивых): 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 найдены
include - регулярное выражение для схем, которые следует включить в сборку перед запуском проекта или отправкой на сервер (^ds_\w+$ по умолчанию)
exclude - регулярное выражение для схем, которые следует исключить из сборки (если например компоненты в указанных атласах не готовы для работы)
kerberos - http адрес сервера для аутентификации kerberos, например HTTP@sso.luxms.com
```

Если значения `server`, `username` или `password` нигде не найдены, то их потребуется ввести с клавиатуры.

Создайте в папке src папку `ds_res` (проект по умолчанию пустой и без файлов и папок, потому хотя бы одну папку нужно создать для запуска)

Создайте в корне проекта файл `authConfig.json` с содержимым

```
{
  "master": {
     "username": "ваш логин от BI",
     "password": "ваш пароль"
   }
}
```

Этот файл находится в `.gitignore` и нужен для того, чтобы вам не приходилось вводить данные пользователя каждый раз, когда используете один из скриптов проекта для массовой загрузки/выгрузки ресурсов с сервера или старта приложения.

P.S. Вы можете, вообще говоря, и указать просто

```
{
      "username": "ваш логин",
      "password": "ваш пароль"
}
```

Но первый способ вам понадобится, когда и если вы будете разрабатывать компоненты, находясь в другой ветке проекта. Тогда для каждой ветки нужно указать свою пару логина и пароля. Так можно делать, например? когда у вас будут отдельные ветки для дева, теста, прода, каждая из которых имеет свой адрес сервера и доступы к нему.

7.1 Концепция

Каждый ваш проект на `BMR` в идеале должен быть отдельным репозиторием (для учебных целей или личного пользования сойдет и простой форк проекта на `Github`). Где в качестве первого коммита можете использовать выбранную вами копию проекта `BMR`.

Если же вы хотите развернуть такой проект где-то в вашем аккаунте компании на `Gitlab`, то последовательность действий примерно такая:

Создаем у себя полностью пустой проект в гитлабе через веб-интерфейс (даже без `Readme.md`, отожмите соответствующую галочку), назовем его, например, так же `bi-magic-resources`.

`git clone gitgithub.com:luxms/bi-magic-resources.git -origin luxms cd bi-magic-resources git remote add origin gitmy.git.server.com:user/bi-magic-resources.git` - вымышленный адрес вашего репозитория `git push origin master`

Если будут изменения в нашем проекте на `Github` - забираем себе изменения:

`git checkout master` `git pull luxms master` `git push origin master`

И мержим ветку `master` в свои рабочие ветки

Так вы сохраните актуальность функционала данного проекта в базовом виде, и не потеряете ваши файлы.

Каждая ветка данного проекта - это потенциально отдельный подпроект, который может содержать свой набор модулей в `package.json`, свой адрес сервера в `config.json` и свой набор ресурсов и логин-пароль в `authConfig.json`.

Однако мы настоятельно рекомендуем не использовать ветки как проекты, а лишь делать из них этапы некоего `CI/CD` и временные ветки для разработки фич. Для проектов делать-таки отдельный репозиторий.

Если вы ведете разработку в **закрытом контуре** , то для установки `node_modules` вам потребуется доп.настройка от ваших `DevOps` репозитория с `npm` пакетами, к которому вы будете подключаться при установке пакетов.

Каждая папка внутри `src` данного проекта, которая начинается с префикса `ds_` - это папка, отображающая разделы ресурсов одноименного атласа. Вы можете создавать и простые папки типа `components`, `utils`, `services` и прочие, но есть разница:

В папках произвольного именования файлы хранятся как есть и с ними ничего не происходит, если только они не являются частью импорта внутри компонентов, расположенных в папках `ds_`. Однако в папках с `ds_` и их подпапках при старте приложения и предварительном билдовании перед отправкой на сервер все компоненты, если для них это имеет место будут билдиться вебпаком. Т.е. ваши `MyComponent.tsx` превратятся в `MyComponent.js` и `MyComponent.js.map` и именно эта пара уйдет на сервер при соотв. команде скрипта.

7.2 Процесс разработки

Рассмотрим ваш типичный рабочий процесс в данном проекте на примере.

Допустим, вам нужно создать совсем новый компонент визуализации, который вы хотите встроить в конкретный дешлет конкретного дешборда у некоего атласа (назовем его `ds_51`).

У вас есть два способа: через типы визуализации `Внешний` и `Внутренний`. Их классы отображения соответсвенно `external` и `internal`.

В первом случае вы пишете хардкорный html с подключением стилей, скриптов, вспомогательной библиотеки `bixel.js` (подробнее в следующем разделе) или иными библиотеками для визуализации (`D3.js` например) и разрабатываете с учетом событийной системы, создаваемой данной библиотекой или пишете React компоненты и полностью используете только React-подход.

Лично мы пропагандируем использование `internal` как более гибкого способа разработки, плюс таким образом вы получаете по итогу цельное React-приложение + вам доступны ряд методов, которых нет и не может быть в `external`.

Допустим, что у вас есть ваши собственные UI-компоненты ,которые будут использоваться в более чем одном компоненте. Вы можете хранить их например в папке `src/components` с любым деревом папок, которое вам удобно и импортировать в ваш React-компонент, который вы только что создали в папке `ds_51`.

Итак:

1. Укажите адрес сервера в `config.json`, логин пароль в `authConfig.json`
2. Создайте файлы `MyComponent.tsx`, `MyComponent.scss` и ряд еще каких-то файлов, которые вам понадобятся для разработки на сегодня. Вы выбрали разработку на `internal`, ибо `таков Путь`
3. Перезапустите проект, если он был запущен на момент создания новых файлов , чтобы `webpack` проекта подхватил ваши файлы и следил за изменениями в дальнейшем.
4. Откройте `localhost` с нужным портом, который можно поменять в `config.json` в ключе `port`.
5. Отправляйтесь в соответствующий атлас, где находится дешлет, в который вы хотите встроить ваш новый компонент.

Простейший код компонента React `MyComponent.tsx`, например

```
import React from 'react';
import './MyComponents.scss';

class Test extends React.Component<any> {
  constructor(props) {
    super(props);
  }
  public render() {
    return (
      <div>Hello world</div>
    );
  }
}

export default Test; // обязательно должен содержать export default!
```

Или функциональный его вариант, без разницы

```
import React from 'react';
import './MyComponents.scss';

const Test = (props) {
  return (
      <div>Hello world</div>
    );
}

export default Test;
```

6. В режиме редактирования дешборда выберите или создайте нужный дешлет, укажите тип визуализации `Внутренний` и в блоке `Файл` выберите `MyComponent.js` ибо именно в таком виде вы его увидите в списке файлов. Сохраните работу и выйдите из режима редактирования. Уже к этому моменту вы увидите результат рендера вашего компонента. В примере выше это `Hello world`. Если вы посмотрите на дешлет через инспектор браузера, то никаких фреймов вы не увидите, потому что обвязка посчитает этот компонент нативной частью себя (и будет технически права).

7. Вы разрабатываете компонент в соответствии с бизнес-логикой для него, пишете стили и т.д. Закончили, увидели удовлетворяющий вас результат на локальном сервере и готовитесь залить это в ресурсы на дев стенд и пойти домой под песню Патрика Суэйзи “She’s like the wind”.

8. Все как обычно при работе с git:

```
git commit -am "Never gonna give you up"
git pull
git push
```

Я бы на всякий случай напомнил вам, что ветку `master` лучше использовать как финальный результат сборки и мержа иных временных веток (как продакшн). Или новыми ветками задать нужный вам уровень этапов деплоя, который вас устроит, например `dev`, `test`, `production`. Тогда каждый этап последовательно, без пропусков мержится в соответствующую ветку. `dev`-`test`, `test`-`prod`. Но не `dev`-`prod` (только в самых исключительных случаях и понимая, что делаете). Все зависит от того, как устроены процессы вашей разработки в компании. Не бойтесь мержа и ветвления. Просто договоритесь в команде, кто выступает в роли ~~dungeon~~ merge-мастера.

9. Все, вы сохранили свой прогресс в `git`. Теперь будем отправлять ресурсы на сервер. Убедитесь, что перед этим, вы сделали `git pull` и у вас актуальная версия проекта (ветки) без неразрешенных мерж-конфликтов.
10. Наберите команду `yarn push` (`npm run push`) и увидите, что в консоли идет подключение к серверу, который вы указали в `config.json` с логином и паролем из `authConfig.json`. Затем идет прогрессбар, отражающий этап сравнения вашей версии файлов и того, что на сервере.
11. Затем вы увидите итоговый диалог, где проект подскажет вам статус файлов и запросит подтверждения на продолжение. Вводите `Y` и увидите прогрессбар окончательной загрузки ваших ресурсов на сервер.

12. Прогрессбар завершился, ошибок нет. Идите на сервер по “боевому” адресу и проверьте ,что по ссылке на дешборд данного атласа не с локали вы видите тот же функционал, что и на локали. Это значит, что ресурсы корректно залились.

Перед отправкой ресурсов на сервер через `yarn push` (`npm run push`) обязательно делайте `git pull`. Помните о том, что отправка ресурсов перепишет их контент на тот, что конкретно у вас сейчас в проекте. Потому. работая в команде, вы должны доверять только тому, что есть в `git`, а не на сервере. Избегайте заливки на сервер файлов, которых нет в `git`, иначе вы получите расхождение версий файлов и при ближайшем пуше ресурсов просто все удалите. Если вы таки не можете за этим следить, то хотя бы используйте ключ `"noRemove": true` внутри `config.json`

Ибо в скриптах `BMR` есть такой хитрый скрипт `yarn pull` (`npm run pull`) который стоит использовать очень ограниченно и осторожно, в идеале разово при первичном наполнении ветки проекта или тогда, когда вы указываете в `config.json` ключ `"dashboards": true`, который, при использовании `npm run pull` позволит вам получить в проекте конфиги дешлетов, дешбордов и датасетов, которые будет храниться в специфических папках, которые у вас автоматически заведутся для каждой папки типа `ds_` . Тогда вы сможете хранить конфиги в `git` и править конфиги этих сущностей так же массово, как вы будете управлять вашими компонентами.

Этот скрипт `pull` сам создаст все папки атласов, которые в данный момент присутствуют на сервере и скачает все файлы и рассортирует их по папкам, включая вложенные. Единственный его минус: он не умеет собирать из `.js` и `.js.map` пары итоговый `.tsx` например. Простим ему) он и не должен, ибо ваши файлы в первую очередь хранятся в `git`, а не на сервере. Однако вам придется после его работы вручную удалять сбилженные версии ваших компонентов, которые прилетели с сервера, благодаря работе этого скрипта. Благо, что такая операция, как я уже говорил, делается очень редко если не сказать единожды.

Вам не обязательно в `BMR` хранить полный перечень папок с атласами ,которые есть на сервере. Это не нужно. Храните только те, с которыми работает ваша команда сейчас. Те атласы, которые в `BMR` не присутствуют явно, просто не будут участвовать в итоговом массовой сборке и отправке при команде `yarn push`. То есть вы можете например создать папку `ds_res` и в ней хранить ровно тот набор компонентов, который хотите сейчас и не смотреть на ресурсы других атласов. Они останутся нетронутыми.

8 Добавление кастомных дэшей в конструктор

Для отображения кастомного дэша в конструкторе, необходимо открыть атлас, где необходимо использовать кастомный дэш, в режиме редактирования и нажать кнопку “Настроить”:

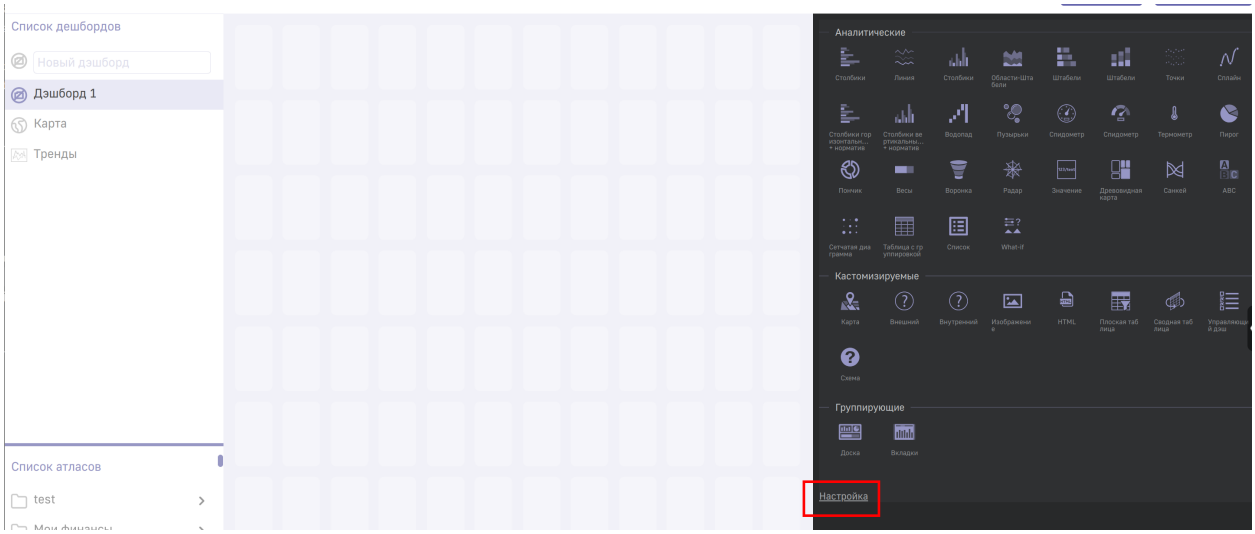
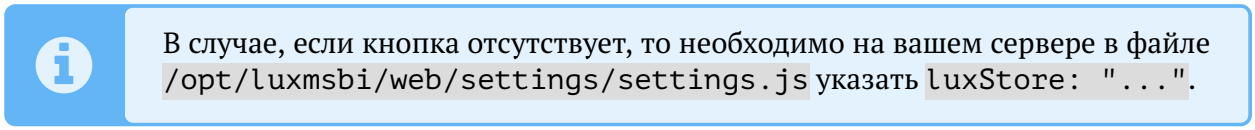


Рис. 8.1 **Открытие настройки конструктора**



В случае, если кнопка отсутствует, то необходимо на вашем сервере в файле `/opt/luxmsbi/web/settings/settings.js` указать `luxStore: "..."`.

В открывшееся окно перенести архив в формате *.zip с файлами кастомного дэша (ниже описан перечень необходимых файлов для корректного импорта):

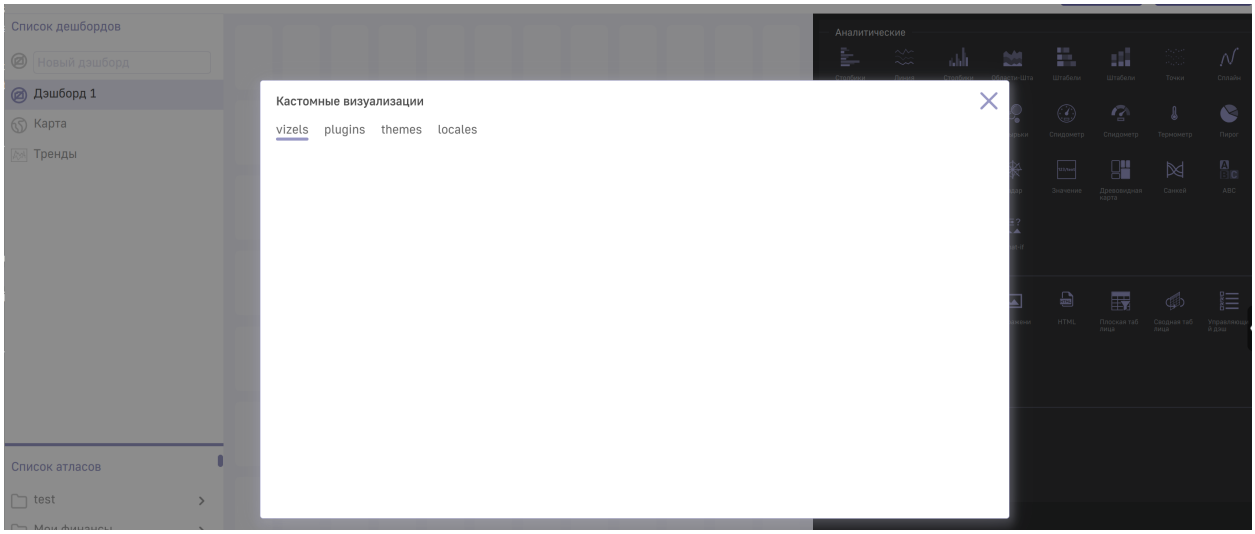


Рис. 8.2 **Добавление кастомных визуализаций**

После переноса - дэш автоматически отобразится в списке:

Рис. 8.3 **Добавление дэша в ресурсы атласа**

Архив должен содержать следующие элемены:

1. {dash_name}.js/{dash_name}.js.map - Скомпилированный Typescript/Javacripts файл с логикой дэша;
2. {dash_name}.css - CSS-файл для стилизации дэша (если необходимо);
3. manifest.json - JSON-файл, содержащий следующие поля: category - в какой группе дэшей в конструкторе будет содержаться дэш (например “analytic”), group - уникальный номер группы дэшей для переключения типа визуализации (например, “111”), title - название дэша, icon - ссылка на иконку дэша.
4. icon.svg - иконка для отображения в списке визуализаций Пример:

```
{
  "category": "analytic",
  "group": "111",
  "title": "BI LABEL",
  "icon": "icon.svg"
}
```

5. schema.json - JSON-файл, в котором указаны поля, необходимые для корректной настройки дэша в JSON-конфигурации. Пример:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "id": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "display": {
      "type": "object",
      "properties": {
        "chartColorCategoryA": {
          "type": "object",
```

```
            "title": "",
            "properties": {
              "color": {
                "type": "string",
                "title": "Color category A"
              }
            }
          }
        }
      }
    }
}
```

6. ui-schema.json - JSON-файл, в котором прописаны поля, отображаемые для данного дэша в конструкторе (Editor). Пример:

```
{
  "display": {
    "chartColorCategoryA": {
      "ui:widget": "color"
  },
  "dataSource": {
    "style": {
      "additionalProperties": {
        "additionalProperties": {
          "color": {
            "ui:widget": "color"
          }
        }
      }
    }
  }
}
```

9 Класс отображения “Внешний”

`External` был нашей первой реализацией спецкласса для отрисовки кастомных визуализаций, еще до того, как веб-клиент стала полностью React-приложением, и основное назначение класса сводится к тому, чтобы вставить `iframe`, занимающий все доступное пространство дешлета (за вычетом высоты заголовка с названием дешлета по умолчанию, если тот не был выключен в настройках) и в качестве `src` атрибута указывается адрес, который вы прописали конфиге дешлета в блоке `Файл` (`url`), который по итогу парсится веб-клиентом в полноценный урл до файла.

В режиме `JSON config` вы даже можете сделать ссылку на внешний ресурс в интернете, если необходимо, однако такой внешний дешлет будет декоративным и не будет реагировать на события веб-клиента (т.е. на любые ваши действия с данными или в интерфейсе).

В режиме `Editor` вам такая возможность по умолчанию недоступна (потому что слишком специфична для обывателя) и вы, при попытке указать файл для указанных выше типов визуализаций, обозреваете ресурсы всех атласов, которые доступны вашему пользователю. Это выглядит как навигация по папкам с названиями атласов, где список файлов - это все ,что хранится в соответствующем разделе `resources` выбранного атласа. В результате выбора файла вы получаете в конфиге дешлета ключ `url` например в таком виде: `"url": "res:MyExternalComponent.html"` Такая запись означает, что веб-клиент должен пойти в ресурсы текущего атласа (пусть это будет тот же `ds_51`), на котором мы находимся и асинхронно (via `Promise`) загрузить файл (или его контент, если это `Internal`) с именем, указанным после `res:`. По итогу файл будет запрашиваться по адресу `/srv/resources/ds_51/MyExternalComponent.html` и эта ссылка вставиться в `src` у `iframe`. То, что отрендерится в результате такого подключения зависит от скриптов внутри `MyExternalComponent.html` Вы можете писать там любой код, как при разметке обычной веб-страницы. Можете даже подключать любые библиотеки и фреймворки, если сочтете нужным. Однако, у этого типа дешлета есть ряд ограничений:

- Вам будет необходимо подключить нашу библиотеку `bixel.js`, предоставляющую вам событийную систему и синхронизирующую работу фрейма и родительского окна.

Страница этой библиотеки тут https://github.com/luxms/bixel. На всякий случай fallback (https://drive.google.com/file/d/1KQfrQQewEkfrk0zhm6TsUL2M9_0lTEoK/view?usp=sharing)

Так вот, сделав поведение вашего компонента зависящим от событий, которые генерирует эта библиотека вы получите кастомный компонент, который умеет взаимодействовать с веб-клиентом и делать широкий круг вещей. Однако ему будут недоступны многие модули и сервисы веб-клиента, либо доступны опосредованно через их сохраненные в объекте `window` версии.

Например `window.parent.__koobFiltersService` - сохраненный инстанс сервиса, который хранит в себе информацию о выбранных вами фильтрах для кубов.

Пример:

```
const service = window.parent.__koobFiltersService;
service.subscribeUpdatesAndNotify(myCallback);

const myCallback = (model) => {
/* тут логика работы с объектом текущей модели сервиса, например с его ключом filters,
  где указаны выбранные фильтры для кубов */
}
const onClick = (e) => {
service.setFilters("", {
  "age", [">=", 50],
  "name": ["=", "Rick Astley"]
}) /* При клике обратится к фильтру и выставит фильтры
на дименшн age и name (фиксированные или из e.target например) */
}
```

Плюс остаются стандартные ограничения, существующие между фреймом и родительским окном, налагаемые браузерами и javascript.

Причины для использования этого:

- Для него требуется сравнительно невысокий входной порог по навыкам
- Не нужно билдить, при особом желании можно править код прямо в разделе ресурсы на сервере, без проекта `BMR`.
- Поскольку это фрейм, то вы можете подгружать любые библиотеки. То есть добавить компонент, рендерящий `Angular` или `Vue` компонент внутри `React`-приложения. Сомнительная инициатива, но запретить ее мы не можем.
- Вставить визуал с другого инстанса BI пользуясь логикой токенов для безпарольного входа (для этого есть отдельная дока по запросу).

Пример такого файла на `external`:

```
<!DOCTYPE html>
<html>
<head>
  <!--Для старых браузеров, не умеющих в Promise, подключаем полифилл es6-promise.auto.js из интернета-->
  <script>window.Promise || document.write('<scr'+'ipt src="es6-promise.auto.js"></sc'+'ript>');</script>
  <script src="bixel.js"></script>

</head>
<style>
  html, body {
    margin: 0;
    font-size: 13px;
    min-height: 100%;
  }
  #wrapper {
    overflow-y: auto;
    overflow-x: hidden;
    width: 100%;
```

```
    min-height: 100%;
    padding: 1rem;
    box-sizing: border-box;
  }
  .elements {
    width: 100%;
    display: flex;
    align-items: center;
    flex-direction: column;
    padding: 0.25rem;
  }
  .element {
    width: 100%;
    display: flex;
    align-items: center;
    justify-content: center;
    margin-bottom: 0.5rem;
  }
  .element:last-child{
    margin-bottom: 0;
  }
  .element .title {
    flex: 0 0 auto;
    max-width: 70%;
    width: 100%;
  }
  .element .value {
    flex: 0 0 auto;
    max-width: 30%;
    padding: 0.5rem 0.25rem;
    width: 100%;
    border: 1px solid #7ba7db;
    text-align:center;
  }
</style>

<body>
<main id="wrapper">
  <div class="elements" id="target"></div>
</main>

<script>
  let dashlet, config, ms, ls, ps, M, L, P, data, axes, xs, ys, zs;
// Первоначальный инит библиотеки. Обычно нужен для разовой предварительной настройки компонента и чтении конфига дешлета
  bixel.init({
    zsCount: 1,
  }).then(function(d) {dashlet = d; config = dashlet.config || {}});

// Данных нет, или не указаны важные ключи в конфиге дешлета. Нужен для заглушки типа Нет данных
  bixel.on('no-data', function(axes) {
    // no data
```

```
  });
// Событие триггерится каждый раз, когда вы выставляете фильтр вручную или через управляющий деш. И когда данные есть
  bixel.on('load', function (d, a) {
    data = d;
    axes = a;
    xs = axes.getXs();
    ys = axes.getYs();
    zs = axes.getZs();
// Просто рандомная функция, которая отвечает за основной рендер. Учитывайте, что событие load может происходить часто, потому не забывайте очищать контейнер перед вставкой в DOM элементов
    _render();
  });

  bixel.on('url', function (url) {
// данный блок будет выполняться каждый раз, когда вы поменяете url
    _render();
  });


  function _render() {
    console.log(xs, ys, zs, data);

    console.log(data.getValue(xs[0], ys[0], zs[0])); //покажет первый элемент таблицы
  }

  document.addEventListener('DOMContentLoaded', function(){ // Аналог $(document).ready(function(){
    document.getElementById('wrapper').addEventListener('click', function(event) {
        console.log(config);
        if (config.hasOwnProperty('onClickDataPoint')) {
          // Прокидываем обработчик встроенной логики клика по элементу графика onClickDataPoint
          bixel.invoke('CLICK_DATA_POINT', {
            x_id: xs[0].id,
            y_id: ys[0].id,
            z_id: zs[0].id,
            event: { pageX: event.pageX, pageY: event.pageY },
          });
        }
    });
  });
</script>
</body>
</html>
```

```
import React from "react";
import './DatePickers.scss';
```

```
// Подключили сервис
import {KoobFiltersService, useService, useServiceItself} from "bi-internal/services";

const MyComponent = (props) => {
  const {cfg, subspace, dp} = props;
 const koob

  public render() {
    const {data} = this.state;
    return (
      <div className="DatePickers">
        {/* что-то делаем с data */}
          <div className="DatePickers__SelectButtons">
            <div className="DatePickers__SelectButton active" onClick={this.onSubmitClick}>Применить</div>
          </div>
        </div>
      </div>
    );
  }
}
```

10 Класс отображения “Внутренний”

Класс `Internal`, изначально реализующий возможность написания кастомных реакт-компонентов, хранящихся в ресурсах и позволяющих использовать экспортирумые обвязкой модули и методы как нативные. Ряд экспортируемых модулей описан и типизирован тут https://github.com/luxms/bi-internal Но этот проект может и иногда отстает в описании того, что доступно. Мы пополняем информацию по мере того, как у нас на это есть время.

10.1 Концепция

Если речь идет о банальной кастомной визуализации на `React` - то тут последовательность действий схожа с `external`:

- Выбрать нужный дешлет
- Указать тип визуализации `Внутренний`
- Указать файл сбилженного компонента `MyComponent.tsx` (ищем `MyComponent.js`)
- Сохраняем и видим результат рендера данного компонента.

Почему это работает? Потому что обвязкой совершаются следующие шаги, когда она видит у дешлета

`"viewClass": "internal"`

1. Специальный контроллер обвязки `InternalComponentVC` загружает указанный контент файла-ресурса, содержащего реакт компонент с `export default`
2. Он формирует из этого функциональный компонент с генерируемыми дополнительными аргументами и прокидывает туда определяемые на лету модули:

```
react // ту версию, которая есть в обвязке
react-dom // ту версию, которая есть в обвязке
classnames // https://www.npmjs.com/package/classnames
jquery
axios
three // Three.js
@react-three/fiber // Модуль Three.js
@react-three/drei // Модуль Three.js
echarts  // Apache Echarts, разово определяем и экспортируем, иначе очень тяжела↩
  я библиотека
```

```
/* Далее ряд модулей из проекта bi-internal. содержащие модули, сервисы, элементы UI и вспомогательные функции */

bi-internal/utils
bi-internal/core
bi-internal/face
bi-internal/root
bi-internal/types
bi-internal/ui
bi-internal/services
bi-internal/ds-helpers
```

Если вы заглянете в `webpack.config.js` то увидите там

```
externals: {
   'react': 'react',
   'react-dom': 'react-dom',
   'classnames': 'classnames',
   'jquery': 'jquery',
   'axios': 'axios',
   'three': 'three',
   '@react-three/fiber': '@react-three/fiber',
   '@react-three/drei': '@react-three/drei',
   'echarts': 'echarts',
   'bi-internal': 'bi-internal',
   'bi-internal/font': 'bi-internal/font',
   'bi-internal/core': 'bi-internal/core',
   'bi-internal/face': 'bi-internal/face',
   'bi-internal/root': 'bi-internal/root',
   'bi-internal/types': 'bi-internal/types',
   'bi-internal/services': 'bi-internal/services',
   'bi-internal/utils': 'bi-internal/utils',
   'bi-internal/ds-helpers': 'bi-internal/ds-helpers',
   'bi-internal/ui': 'bi-internal/ui'
},
```

Эта связка и позволяет прокидывать в произвольные компоненты указанные модули вдобавок к тем, что у вас есть в `package.json`. Так что нигде на сервере они не хранятся, это происходит “на лету” средствами обвязки.

Сам пакет `bi-internal` будет описан в конце руководства.

10.2 Пример кастомного компонента

Он просто ищет в конфиге дешлета ключ `formattedString` и заменяет в ней указание дименшна на его текущее значение (например `"Моей собаке Жучке $age$ лет"`). При условии, что age содержит фильтр, то он вставится в эту строку на место `$age$`

```
import React from "react";
import {KoobFiltersService, KoobService} from 'bi-internal/services'; // вот пример того, как можно получить компоненты из на лету создаваемого экспортируемого модуля (в package.json вы его не увидите!)
import {$eid} from "bi-internal/utils"; // то же)

export default class MyComponent extends React.Component<any> {
  private _koobFiltersService: KoobFiltersService;
  private _koobService: KoobService;
      public state: {
        data: any;
        parsedString: string;
      };

      public constructor(props) {
        super(props);
        this.state = {
          data: [],
          parsedString: ""
        };
      }
      public componentDidMount(): void {
          const {cfg, dp, subspace} = this.props;
          this._koobFiltersService = KoobFiltersService.getInstance();
          this._koobService = KoobService.createInstance(cfg.getRaw().dataSource.koob);
          this._koobFiltersService.subscribeUpdatesAndNotify(this._onFiltersUpdated);
          this._koobService.subscribeUpdatesAndNotify(this._onFiltersUpdated);
          // Получение данных
          dp.getMatrixYX(subspace).then(data => {
            this.setState({data});
          })
      }

      public componentWillUnmount(): void {
        this._koobFiltersService.unsubscribe(this._onFiltersUpdated);
        this._koobService.unsubscribe(this._onFiltersUpdated);
      }
      // Колбек, чтобудет вызван автоматически каждый раз, когда выставлен фильтр в упр.деше (или программно в другом месте)
      private _onFiltersUpdated = () => {
        const {cfg} = this.props;
        let formattedString = cfg.getRaw()?.formattedString || "";
        const koobModel = this._koobService.getModel();
        const koobFiltersModel = this._koobFiltersService.getModel();
        // Проверяем, что сервисы загрузили модели, нет ошибок и они готовы к работе
        if (koobModel.loading || koobModel.error || koobFiltersModel.loading || koobFiltersModel.error) return;
        if (Object.keys(koobFiltersModel.filters).length) {
          Object.keys(koobFiltersModel.filters).map(dimensionId => {
```

```
            formattedString = formattedString.replaceAll(`$${dimensionId}$`, koobFiltersModel.filters[dimensionId] ? koobFiltersModel.filters[dimensionId][1] : "")
          });
        } else {
          // Тут логика выставления дефолта должна быть
          const possibleDimensionIds = formattedString.match(/\$\w+\$/gi).map(el => String(el).slice(1, String(el).length - 1));
          if (possibleDimensionIds.length) {
            possibleDimensionIds.map(dimId => {
              // Достаем по идентификатору дименшна его конфиг из куба и
              // проверяем наличие специального поля defaultValue, где хранится значение по умолчанию

              const currentDimensionColumn = $eid(koobModel.dimensions, dimId);
              const currentDimensionDefaultValue = currentDimensionColumn?.config?.defaultValue || ""; // Можно указать или прямо в конфиге дименшна или любую произвольную строку
              formattedString = formattedString.replaceAll(`$${dimId}$`, currentDimensionDefaultValue);
            })

          }
          console.log(possibleDimensionIds, formattedString)
        }
        this.setState({parsedString: formattedString});
      }
      public render() {
        const {data, parsedString} = this.state;
        const {cfg, dp, subspace} = this.props;
        return (
          <>
            <div>{parsedString}</div>
          </>
        );
      }
}
```

Такой компонент должен обязательно содержать `export default`. Это нужно, чтобы обвязка понимала, какой именно из экспортируемых компонентов маунтить. Компонент может принимать `props`, а может и нет. Но когда принимает, то ему от обвязки прилетает объект со свойствами, ключевые пропсы из которых это

```
const {cfg, subspace, dp} = props;
```

Описание полей, типизация этих пропсов будет в готовящемся разделе по кастомной разработке в документации. Понять и простить)

`cfg`

Объект, содержащий информацию о конфиге дешлета. Его опции, источник данных, блок отображения и информация о текущем датасете, вплоть до списка дешбордов и дешлетов, рассортированных по дешбордам.

```
// отдаст тот объект конфига дешлета который вы видите в режиме редактирования в блоке JSON config.
// Включая то, чего вы внесли туда вручную
cfg.getRaw()
// отдаст только блок dataSource
cfg.dataSource
// вернет ссылку на файл, указанный в конфиге, в блоке url
cfg.getUrl()
// вернет объект датасета
cfg.getDataset()
```

`subspace`

Уникальный объект, хранящий информацию об организации данных, расположении элементов на осях, их порядок и сами массивы элементов соотв. оси, какие межи, какие дименшны, лимиты, оффсеты, подытоги используются/запрашиваются. Физического смысла не имеет, однако имеет определяющую роль в том, что в итоге мы увидим на графике, ибо аггрегирует в себе множество вспомогательной информации.

Чаще всего вам потребуется проходиться по его X (`subspace.xs`) и Y (`subspace.ys`) элементам осей, дабы строить внутреннюю структуру организации и вывода данных на ваших собственных графиках и согласно используемым вами движкам для отрисовки визуала.

В большинстве случаев вам достаточно прокинуть этот объект как есть, чтобы получить данные через провайдер данных dp

`dp`

Провайдер данных, который содержит методы для обращения к данным на основе имеющегося subspace

Пример из кода компонента выше

```
dp.getMatrixYX(subspace).then(data => {
  this.setState({data});
})
```

Вернет для каждого элемента оси Y массив значений, принимаемых осью X (декартово произведение меж и дименшнов (фактов и измерений, если угодно)) Итого в data в таком случае получим массив массивов всегда

Для одномерного случая типа пай чарта нам не нужно столько данных, достаточно указать

```
dp.getVectorY(subspace).then(data => {
  this.setState({data});
})
```

Получит массив значений, обрезанных по оси Y.

Для получения raw ответа сервера за данными используйте

```
dp.getKoobData(subspace).then(data => {
  this.setState({data});
})
```

Получит массив объектов, совпадающий с тем, что идет на `api/v3/${schema_name}/data`.

В качестве примера рабочего кастомного компонента типа internal приведем код следующих файлов:

Конфиг дешлета, где я это использовал:

```
{
  url: 'res:ds_res:MyComponent.js',
  frame: {
    h: 8,
    w: 12,
    x: 0,
    y: 0,
  },
  dataSource: {
    koob: 'BeerDataSource.beerKoob',
    style: {
      measures: {
        count_units: {
          color: 'red',
        },
        count_quantity: {
          color: 'green',
        },
      },
    },
    xAxis: 'category',
    yAxis: 'measures',
    measures: [
      'count(units):count_units',
      'count(quantity):count_quantity',
    ],
    dimensions: [
      'category',
    ],
  },
  onClickDataPoint: "lpe:setKoobFilter('','category',['=',category])",
  view_class: 'internal',
  title: 'Test',
}
```

Файл с переменными темы и прочими переменными, которые вам могут понадобиться `vars.scss` (создайте это файл где угодно внутри src проекта `BMR`). Он позволит вам использовать цвета той темы, что сейчас выбрана автоматически

```
// переменные из темы

$color1: var(--color1);
$color2: var(--color2);
$color3: var(--color3);
$color4: var(--color4);
$color5: var(--color5);
$color6: var(--color6);
$color7: var(--color7);
$color8: var(--color8);
$color9: var(--color9);
$color10: var(--color10);
$color11: var(--color11);
$color12: var(--color12);
$color13: var(--color13);
$color14: var(--color14);
$color15: var(--color15);
$color16: var(--color16);
$color17: var(--color17);
$color18: var(--color18);
$color19: var(--color19);
$color20: var(--color20);
$color_conditions: var(--color_conditions);
$shadow_button: var(--shadow_button);
$shadow_dash: var(--shadow_dash);
$shadow_panel: var(--shadow_panel);
$panel_push-up: var(--panel_push-up);
$panel_push-down: var(--panel_push-down);
$info: var(--info);
$system_panel:  var(--system_panel);
$system_background:  var(--system_background);
$system_effect:  var(--system_effect);
$system_selection: var(--system_selection);
$system_element:  var(--system_element);
$neutral_300:  var(--neutral_300);
$neutral_400:  var(--neutral_400);
$neutral_500:  var(--neutral_500);
$neutral_600:  var(--neutral_600);
$neutral_700:  var(--neutral_700);
$neutral_800:  var(--neutral_800);
$neutral_900:  var(--neutral_900);
$active_firstdefault:  var(--active_firstdefault);
$active_firsthover:  var(--active_firsthover);
$active_firstcontrast: var(--active_firstcontrast);
$active_secondcontrast:  var(--active_secondcontrast);
$active_seconddefault:  var(--active_seconddefault);
$active_secondhover:  var(--active_secondhover);
$state_success:  var(--state_success);
$state_warning: var(--state_warning);
$state_error:  var(--state_error);
```

Можете посмотреть текущий список css-переменных в инспекторе браузера у тега `body`.

Файл стилей `MyComponent.scss` (у меня все файлы внутри `ds_res`, потому будьте внимательны с адресами при импортах)

```
// Подключили переменные темы
@import "./vars.scss";

.MyComponent {
    width: 100%;
    height: 100%;
    position: relative;

    &__graphic {
        position:absolute;
        z-index: 0;
        width: 100%;
        height: 100%;
    }
    &__onClickText {
        position:absolute;
        z-index: 1;
        left: 0;
        right: 0;
        top: 1.5rem;
        text-align: center;
        color: $color1;
    }
}
```

И собственно файл компонента, который берет то, что ему автоматически прилетает в props от обвязки и практически как есть подает их в виде опций для графика `Echarts` типа `bar`. Умеет ловить клик по точке на графике, умеет выполнять стандартный `onClickDataPoint` (описание в руководстве разработчика) и использует один из стандартных observable сервисов `KoobFiltersService` для хранения и манипуляции фильтрами на данные для куба(ов). Таким образом данный кастомный дешлет может влиять на своих соседей при помощи фильтров, налагаемых на дименшны (при условии, что коробочный дешлет содержит блок `filters` с указанием у соотв. дименшна значения `true`)

```
// react и echarts на самом деле тянутся из обвязки "на лету"
import React, { useEffect, useState } from "react";
import  * as echarts from 'echarts';
// KoobFiltersService - Observable сервис, управляющий фильтрами для кубов (по у
  молчанию его использует упр.дешлет)
// useService, useServiceItself - специальные хуки для получения только модели и
  ли всего инстанса какого-либо
// Observable сервиса. Принимает по умолчанию класс нужного сервиса и если это н
  е singleton то еще и идентификатор (через запятую)
import {KoobFiltersService, useService, useServiceItself} from 'bi-
  internal/services';

import './MyComponent.scss';
```

```
12 const MyComponent = (props) => {
13   const { cfg, subspace, dp } = props;
14   const [data, setData] = useState<any>([]);
15   const [clickedPointName, setClickedPointName] = useState<string>(""); // для н
     аглядности покажем на какую точку (Y,X) кликнули
16   const koobFiltersService =
     useServiceItself<KoobFiltersService>(KoobFiltersService); // Получили инстанс
     сервиса фильтров
17   // через метод koobFiltersService.getModel() можем получить его модель,
18   // а через метод koobFiltersService.setFilter(koobId, dimensionId,
     valueArray) // ("", "category", ["=", "Beer"])
19   // можем фильтровать данные дешлетов, которые подписаны в своих блоках
     filters на изменение этого дименшна (содержат "category": true)

21   // Храним реф-ссылку на контейнер для графика, сам инстанс Echarts и опции, ко
     торые ему передаем
22   let containerRef = null;
23   let chart = null;
24   let options = {};

26   // Обрабатываем клик по точке графика. Проверяем, есть ли в конфиге дешлета св
     ойство onClickDataPoint, отвечающая
27   // в коробке за логику клика на точку по умолчанию
28   // если нет - просто реализована произвольная логика выставления текущего знач
     ения дименшна в фильтр + для наглядности
29   // показываем в интерфейсе строчку с "координатами" кликнутой точки

31   const onChartClick = (params): void => {
32 // о том, что входит в params можно подглядеть тут https:
     //echarts.apache.org/en/api.html#events.Mouse%20events
33     if (cfg.getRaw().hasOwnProperty('onClickDataPoint')) {
34       // Формируем объект информации о точке для встроенного контроллера обработ
     ки клика по точке
35       const vcpv = {m: undefined, l: undefined, p: undefined, z: undefined, y:
     params.data.y, x: params.data.x, v: params.value};
36       cfg.controller.handleVCPClick(params.event, vcpv)
37     } else {
38       const koobFiltersModel = koobFiltersService.getModel();
39       if (koobFiltersModel.loading || koobFiltersModel.error) return;
40       koobFiltersService.setFilter('', params.data.x.axisIds[0], ["=",
     params.name]);
41     }
42     setClickedPointName(`${params.data.y.title} ${params.data.x.title}`);
43   }
44   // На инит рефа создаем с нуля или обновляем существующий инстанс Echarts и по
     даем ему опции на вход
45   // конфигурацию графиков Echarts смотрите тут https:
     //echarts.apache.org/en/option.html#title
46   const onChartCreated = (container) => {
47     if (container && data.length) {
48       if (!containerRef) {
49         containerRef = container;
50         chart = echarts.init(containerRef, null, {renderer: 'svg'});
```

```
51        }
52        options = {
53          title: {
54            show: false
55          },
56          tooltip: {
57            trigger: 'item',
58            appendToBody: true,
59            show: true
60          },
61          xAxis: {
62            type: 'category',
63            data: subspace.xs.map(x => x.title)
64          },
65          yAxis: {
66            type: 'value'
67          },
68          series: subspace.ys.map((y, yIndex) => ({
69            data: subspace.xs.map((x, xIndex) => ({
70              name: x.title,
71              itemStyle: {
72                // контроллер, который получает информацию о цвете автоматически, исходя из контекста
73                color: cfg.getColor(y, null, yIndex),
74              },
75              x,
76              y,
77              value:  data[yIndex][xIndex] // мы получали матрицу YX
78            })),
79            name: y.title,
80            type: 'bar', // я задал этот тип явно, но это можно прочитать из конфига дешлета
81                         //как переменную cfg.getRaw().chartType например
82            showBackground: true,
83          })),
84          legend: {
85            show: true,
86            data: subspace.ys.map((y, yIndex) => ({
87              name: y.title,
88              icon: 'circle',
89              itemStyle: {
90                // контроллер, который получает информацию о цвете автоматически, исходя из контекста
91                color: cfg.getColor(y, null, yIndex),
92              },
93            }))
94          },
95        };
96        chart.setOption(options);
97        chart.resize(); // принудительно заставляем расшириться на весь контейнер
98        chart.on('click', 'series', onChartClick); // Обрабатываем клик по серии, если нужно
99      }
```

```
100   }
102   useEffect(() => {
103     // Получаем полное декартово произведение для указанного конфига в дешлете
104     // ожидаем матрицу [subspace.ys.length][subspace.xs.length]
106     dp.getMatrixYX(subspace).then(dataArr => {
107       setData(dataArr || []);
108     });
109   },[]);
111   return (
112     <div className="MyComponent">
113       {clickedPointName != "" && <div className="MyComponent__onClickText">Вы кл↩
      икнули на {clickedPointName}</div>}
114       <div ref={onChartCreated} className="MyComponent__graphic"></div>
115     </div>
116   );
117 }
118 export default MyComponent;
```

Все, что я здесь импорчу - прилетает мне на лету из обвязки (кроме стилей, конечно). То есть, например, если вы установите в свой `package.json` версию ниже той, в которой разрешены хуки - то к своему удивлению вы обнаружите, что компонент все равно работает. потому что для своего выполнения он использует ту версию реакта, что приходит из обвязки. Если что это `"react": "î6.5.0"` Однако в обратную сторону это не работает.

Фактически, здесь нам не важно, как зовется сам файл и какое имя компонента вы выберете. Вам только следует помнить о концепции самого React об именовании файлов компонентов (с большой буквы, одноименно с названием основного класса по умолчанию).

Однако, если речь заходит о том, чтобы переопределить поведение компонентов, являющихся основными строительными блоками каждой из страниц соотв. раздела, то тут важно называть их и хранить в определенных местах и определенным образом.

Тут все зависит от самого компонента. Сейчас из коробки ожидают потенциального переопределения следующие компоненты (строительные блоки страницы):

`DlgAuth.tsx` - отвечает за рендер и поведение диалогового окошка логина `Root.tsx` - стартовая страница, которую вы видите после ввода-логина и пароля `RootHeader.tsx` - хедер стартовой страницы выше `DsShellHeader.tsx` - на странице с дешбордами отвечает за хедер `DsShellLeftPane.tsx` - на странице с дешбордами отвечает за левое меню (включая логотип и название проекта) `DsShellLayout.tsx` - на странице с дешбордами отвечает за контейнер хидера, левой панели и списка дешлетов. Может быть полезен как для полного переписывания раздела с дешбордами, так и использования общих стилей или оборачивания данного компонента в компонент высшего порядка (HOC).

Все компоненты выше, кроме `DsShellHeader` и `DsShellLeftPane` не зависят от атласа просто по своему определению. Потому они могут находиться только в ресурсах `ds_res`, так как напоминаю, что здесь мы храним то, что нам будет нужно везде или что должно изменить встроенное поведение обвязки. То есть на сервере разумеется должны быть только их сбилженные версии в виде `.js` файлов (и возможно `.js.map`).

В `bi-magic-resources` - поместите в `ds_res` файлы с таким именем и при запуске локали увидите, что логика этих компонентов берется уже из ваших файлов. Советую помечать их спец классом, чтобы отличать ваши и коробочные компоненты в инспекторе.

Если рассмотреть `DsShellHeader` и `DsShellLeftPane` то, они могут быть размещены как в разделе ресурсов `ds_res` и тогда эти компоненты будут переопределять поведение одноименных компонентов на каждом из атласов, но только до тех пор, пока не встретят в разделе ресурсов текущего атласа свои копии, тогда уже берутся эти локальные компоненты атласа.

Еще раз: веб-клиент сначала проверяет для таких файлов раздел ресурсов сначала текущего атласа, затем `ds_res` и если их не находит, то вставляет дефолтную коробочную версию.

Таким образом, если вы достигли такой кастомизации, которая значительно отличается от коробочной Luxms BI, а после некоторых действий с nginx или БД вдруг видите стандартный внешний вид обвязки - проверьте, не удалили ли вы ресурсы атласов и не отдает ли их сервер в виде base64 в разделе `Network` браузера. Такая ошибка может случиться, если вы по какой-топ причине поставили на сервер версии веб-клиента и БД разных версий (например `8.10` клиента против `8.11` базы).

Если вы собираетесь настраивать какие-то разделы, которые не присутствуют в списке выше - скажите вашему менеджеру от Luxms BI о такой необходимости и мы обернем этот компонент в декоратор, который будет проверять наличие данного компонента в ресурсах атласов. Более того, мы оказываем партнерам консультационную помощь в написании кода кастомного компонента и в исключительных случаях - подготовим копию коробочного компонента, готового для использования и правки в ресурсах. Это обсуждается с менеджерами.

Мы настоятельно рекомендуем использовать именно `internal` подход в своей разработке как самый универсальный.

Причины использования:

- Нативная вставка в дерево реакта
- Возможность пользоваться широким кругом доступных из обвязки модулей, функций, сервисов и утилит
- Возможность создавать и использовать свои собственные и существующие observable сервисы (об этом позже)
- Все прелести React как такового)

11 Модули bi-internal

Это набор именованных модулей, в которые собираются различные сервисы, контроллеры и компоненты, которые экспортирует веб-клиент, чтобы ваш кастомный компонент мог использовать нативные методы для реализации какого-либо функционала.

Существует публичный проект на Github: https://github.com/luxms/bi-internal

В нем содержится типизация и базовые комментарии к каждому модулю семейства `bi-internal`.

Данный проект периодически актуализируется по мере наших сил, но к сожалению, он всегда будет отставать от того, что есть в наличии в текущей версии веб-клиента.

Перечень экспортируемых модулей семейства `bi-internal`:

```
bi-internal/core
bi-internal/types
bi-internal/face
bi-internal/root
bi-internal/services
bi-internal/ds-helpers
bi-internal/ui
bi-internal/utils
```

Рассмотрим каждый подробнее:

11.1 bi-internal/core

Модуль содержит в себе базовые классы и компоненты, ключевые для веб-клиента.

Модуль частично описан на Github

В клиенте этот модуль выглядит так:

```
const core = require('@luxms/bi-core');

module.exports = core;
```

сам bi-core индекс-файл

```
export * from './singleton';
```

```
export * from './Retainable';
export * from './Observable';
export * from './BaseService';
export * from './UrlState/UrlState';
export * from './extractErrorMessage';
export { AppConfig } from './AppConfig';
export { AuthenticationService, IAuthentication } from
  './AuthenticationService';
export { BaseEntitiesService, IBaseEntities } from
  './services/BaseEntitiesService';
export * from './RtService';
import * as repo from './repositories';
import * as srv from './services';
export { repo, srv };
```

Модуль описан на Github, но дам ряд комментариев:

`Observable, Retainable, BaseService` - вводит описание Observable и базового класса, работающего на нем. Все Observable-сервисы (OC) есть наследники `BaseService`.

`AppConfig` - сервис OC, синглтон, который хранит конфигурацию инстанса, пришедшую из файла настроек `settings.js` (`/opt/luxmsbi/web/settings/settings.js` на сервере по умолчанию). Есть информация о текущей локализации, фичах, темах (одно из мест, где можно хранить информацию о них), опции и базовые настройки для карт на `leaflet` (сервер для тайлов, например).

`UrlState` - сервис OC, синглтон, хранящий модель и методы работы с урлом приложения. Позволяет делать навигацию, хранить в модели отображаемые в урле и скрытые параметры или данные. Критичен для работы над презентациями. Каждая модель такого инстанса, чтобы вы в ней не сохранили перед добавлением страницы в перезентацию восстановит состояние, которое было на момент добавления, на сервере и отрисует ее в виде картинки для `.pdf` или `.pptx`

`AuthenticationService` - сервис OC, синглтон, который в модели хранит информацию о текущем залогиненном пользователе. Его id, роль, юзернейм, почту, конфигурацию и статус аутентификации. Содержит методы для авторизации и деавторизации. А также методы работы с двухфакторной авторизацией

`repo` - хранилище репозиториев. Это паттерны, реализующие способы выполнить запросы за различными сущностями, в зависимости от контекста и без оглядки на то, как такой запрос реализуется в клиенте.

`srv` - хранилище сервисов, позволяющих загрузить сущности (энтити), в зависимости от контекста, без знания API. Кубы, дименшны, локации, метрики, периоды, дешлеты, дешборды и другое.

Пример подключения элементов модуля `core` в конкретном компоненте:

```
import {UrlState, BaseService, AppConfig, AuthenticationService, repo, srv}
  from 'bi-internal/core';
```

11.2 bi-internal/types

Модуль, хранящий описание интерфейсов различных энтити, но не связанных с кубами.

Модуль частично описан на Github

в клиенте этот модуль выглядит так:

```
const types = require('../../services/ds/types');

module.exports = types;
```

Сам файл `types`:

```
import { data_engine } from '../../data-manip/data-manip';
import { IDisposable, IObservable, IUrl } from '@luxms/bi-core';
import {
  IColorResolver,
  IEntity,
  IGeo,
  ILocation,
  ILocationArea,
  ILocationCard,
  ILocationsHelper,
  IMapFill,
  IMetric,
  IMetricsHelper,
  IOptionsProvider,
  IPeriod,
  IPeriodInfo,
  IPeriodsHelper,
  IPreset,
  IRange,
  IStoplightsProvider,
  ISubspace,
  ISubspacePtr,
  ITitleResolver,
  IUnit,
  IVizel,
  IVizelConfigDisplay,
  IVizelController, IVizelProperties,
  responses,
  tables
} from '../../defs/bi';

export interface IVizelConfig extends IColorResolver, ITitleResolver,
  IOptionsProvider, IStoplightsProvider {
  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;
  getBgImage(): string;
  dataSource?: tables.IDataSource;
  getVizelType(): string;
  setVizelType(vizelType: string): void;
  setTitle(title: string): void;
  getSubspacePtr(): ISubspacePtr;
  controller: IVizelController;

  title?: string;
  description?: string;
  display?: tables.IVizelConfigDisplay;
  legend?: { [id: string]: tables.ILegendItem; };
  badValueColor?: string;
  goodValueColor?: string;
  normsMainColor?: string;
  onClickDataPoint?: string | any;
  onClick?: string | any;
  cardId?: string;
  externalUrl?: IUrl;
  dashboardId?: number | string;
  dashId?: number | string;
  normStrategy?: string;
  context?: any;

  titleContext?: string[];
  colorResolver?: IColorResolver;
  titleResolver?: ITitleResolver;

  // deprecated
  chartStyle: string;
  showLegend: boolean;
  visualMap?: any;

  getRaw(): tables.IRawVizelConfig;
}

export interface IDashlet extends IEntity {
  id: string;
  title: string;
  layout: string;           // V|H|''
  children: IDashlet[];
  getDataset(): IDatasetModel;
  getDashboard(): IDashboard;
  getFrame(): tables.IConfigFrame;
  getDescription(): string;
  getRawVizelConfig(): tables.IRawVizelConfig;
  isContainer(): boolean;
  isRoot(): boolean;
```

```
}


export interface IDashletsHelper {
  dashboards: IDashboard[];
  dashboardTopics: tables.IDashboardTopic[];
  getDashboard(id: string): IDashboard;
  getDash(id: string): IDashlet;
  getDashes(): IDashlet[];
}

export interface IDashboard extends IEntity {
  id: string;
  title: string;
  topic_id: number;
  stateColor: string;
  getRootDashes(): IDashlet[];
  getDashes(): IDashlet[];
}

export interface IConfigHelper {
  hasValue(key: string): boolean;
  getValue(key: string, defaultValue?: string): string;
  getStringValue(key: string, defaultValue?: string): string;
  getIntValue(key: string, defaultValue?: number): number;
  getFloatValue(key: string, defaultValue?: number): number;
  getBoolValue(key: string, defaultValue?: any): boolean;
  getEnumValue(key: string, values: string[], defaultValue?: string): string;
  getStringArray(key: string, defaultValue?: string[]): string[];
  getIntArray(key: string, defaultValue?: number[]): number[];
  getEnterUrl(datasetKey: string): IUrl;
}

export interface IDatasetModel {
  id: number;
  guid: string;
  // deprecated
  // schemaName: string;
  schema_name: string;
  title: string;
  description: string;
  units: IUnit[];
  metrics: IMetric[];
  rootMetrics: IMetric[];
  presets: IPreset[];
  locationCards: ILocationCard[];
  locationAreas: ILocationArea[];
  locations: ILocation[];
  rootLocations: ILocation[];
  periods: IPeriod[];
  dashlets: tables.IDashletsItem[];
  defaultMetrics: IMetric[];
  defaultLocations: ILocation[];
```

```
145   defaultPeriods: IPeriod[];
146   metricsHelper: IMetricsHelper;
147   locationsHelper: ILocationsHelper;
148   periodsHelper: IPeriodsHelper;
149   dashletsHelper: IDashletsHelper;
150   getDataProvider(): data_engine.IDataProvider;
151   getConfigHelper(): IConfigHelper;
152   getEnterUrl(): IUrl;
153   createVizelConfig(d: tables.IRawVizelConfig, view_class?: string): ↩
      IVizelConfig;
154   getDatasetTitleTemplate(route: string): string;
155   getBiQuery(): any;

157   // makes lookup in table config and in lang
158   getConfigParameter(key: string, defaultValue?: string): string;

160   update(datasetDescription: responses.IDatasetDescription, storage: ↩
      responses.ITables): void;
161   getSerial(): moment.Moment;
162   M: (id: string) => IMetric;
163   L: (id: string | number) => ILocation;
164   P: (id: string) => IPeriod;

166   getPeriodInfoByRange(startId: string, endId: string, type?: number): ↩
      IPeriodInfo;
167 }

169 export interface IDatasetServiceModel {
170   loading: boolean;
171   error: string;
172   dataset: IDatasetModel;
173 }

175 export interface IAxesOrder extends Array<string> {
176   xs?: string;
177   ys?: string;
178   zs?: string;
179   aas?: string;
180   abs?: string;
181   // etc...
182 }

184 export interface IDsState {
185   loading?: boolean | number;
186   error?: string;
187   //
188   autoscale: boolean;
189   chartType: string;
190   dash: IDashlet;
191   dboard: IDashboard;
192   geo: IGeo;
193   locations: ILocation[];
194   formulaLocations: ILocation[];
```

```
  axesOrder: IAxesOrder;
  mapfill: IMapFill;
  metrics: IMetric[];
  periodInfo: IPeriodInfo;
  periods: IPeriod[];
  preset: IPreset;
  route: string;
  mapMetricsPanelVisible: boolean;
  datasetTitle: string;
  datasetDescriptionHTML: string;
  dataset: IDatasetModel;
  customConfig: any;
}

export interface IDsStateService {
  getModel(): IDsState;
  getDataset(): IDatasetModel;
  getMaxParametersNumber(): number;
  getMaxLocationsNumber(): number;
  setMetrics(ms: IMetric[]): void;
  setPreset(p: IPreset): void;
  setPeriods(start: IPeriod, end: IPeriod, type: number);
  setFormulaLocations(ls: ILocation[]): void;
  setGeo(g: IGeo): void;
  setDboard(dboard: IDashboard): void;
  setDash(dash: IDashlet): void;
  setAxesOrder(ao: IAxesOrder): void;
  setAutoscale(autoscale: boolean): void;
  setMapfill(mf: IMapFill): void;
  setChartType(chartType: string): void;
  setCustomConfig(config: any): void;
  toggleParameter(p: IMetric): void;
  removeMetric(p: IMetric): void;
  toggleFormulaLocation(l: ILocation): void;
  removeFormulaLocation(l: ILocation): void;
  goToPlots(): void;
  setMapMetricsPanelVisible(mapMetricsPanelVisible: boolean): void;
  subscribe(items: string, callback: any): IDisposable;
  subscribeUpdates(listener: (model: IDsState) => void): IDisposable;
  subscribeUpdatesAndNotify(listener: (model: IDsState) => void): IDisposable;
  unsubscribe(listener: (model: IDsState) => void): void;
  retain(): IDsStateService;
  release(): boolean;
}
// constructor props
export interface IVizelProps {
  readonly dp: data_engine.IDataProvider;
  readonly cfg: IVizelConfig;
  readonly subspace: ISubspace;
  readonly schema_name: string;
  listener?: any;
  showLegend?: boolean;
  onVizelPropertiesChanged?: (properties: IVizelProperties, vizel: any) => void;
```

```
  properties?: IVizelProperties;
  renderError?: (error: string) => any;
  renderLoading?: (loading: boolean) => any;
}
export interface IVizelClass {
  new(props: IVizelProps): IVizel;
}
export enum LoadingStatus {
  UNDEFINED,
  LOADING_FIRST_TIME,
  LOADING,
  NO_DATA,
  HAS_DATA,
}
export interface IModuleOptions {
  useSinglePeriod?: boolean;
  metricsPanel?: boolean;
  locationsPanel?: boolean;
  periodsPanel?: boolean;
}
```

Пример использования в компоненте:

```
import {IVizelConfig, IVizelProps, IDashboard} from 'bi-internal/types';
```

11.3 bi-internal/face

Это модуль хранит библиотеку готовых UI-компонентов из `storybook` вспомогательного модуля `luxms/bi-face`

В клиенте этот модуль выглядит так:

```
const face = require('@luxms/bi-face');

module.exports = face;
```

Индексный файл таков

```
import Calendar from "./Calendar";
export { Calendar };
import DatePicker from "./DatePicker";
export { DatePicker };
import * as icons from "./icons";
export { icons };
import TreeComponent from "./TreeComponent";
export { TreeComponent };
import AccountTool from "./AccountTool";
export { AccountTool };
import AppLogo from "./AppLogo";
export { AppLogo };
```

```
import Button from "./Button";
export { Button };
import Confirm from "./Confirm";
export { Confirm };
import DataGrid from "./DataGrid";
export { DataGrid };
import Dropdown from "./Dropdown";
export { Dropdown };
import DropLabelList from "./DataGrid/DropLabelList";
export { DropLabelList };
import GeometryObserver from "./util/GeometryObserver";
export { GeometryObserver };
import Header from "./Header";
export { Header };
import Login from "./Login";
import LoginDemo from "./Login/LoginDemo";
export { Login, LoginDemo };
import Form from "./Form";
export { Form };
import themes from "./store/themes";
export { themes };
import Menu from "./Menu";
export { Menu };
import Strap from "./Strap";
export { Strap };
import TextEditor from "./TextEditor";
export { TextEditor };
import Tag from './Tag';
export { Tag };
import Breadcrumb from './Breadcrumb';
export { Breadcrumb };
```

Модуль частично описан на Github, несколько комментариев:

`icons` - дает полный перечень иконок, используемых в данном проекте

`GeometryObserver` - модуль позволяет назначить обработчик события ресайза какого-либо элемента на странице.

Пример использования в компоненте

```

import React from 'react';
import {GeometryObserver} from 'bi-internal/face';

class MyComponent extends React.Component {


  private _setupContainerRef = (el: any) => {
    if (this._container) return;
    this._container = el;
    GeometryObserver.getInstance().addSubscription(this._container, this.resize); // подписка на ресайз
  }
```

```
  public resize = () => {
    // действия на resize
  }

  public componentWillUnmount() {
    GeometryObserver.getInstance().removeSubscription(this._container, this.resize);
  }

 public render() {
   return (<div ref={this._setupContainerRef}>test</div>)
  }
}

export default MyComponent;
```

Пример создания логики дропдауна

```
import React from 'react';
import {Dropdown} from 'bi-internal/face';

class MyComponent extends React.Component {
  public state = {
   menuVisible: false,
    userIcon: ''
  }
  public render() {
    const {menuVisible, userIcon} = this.state;

    return (
      <Dropdown.Trigger offset={[2, 0]}
             menu={<SomeComponent/>}
                        trigger="mouseenter click"
                        placement="bottom-end"
                        visible={menuVisible}
                        onClick={() => this.setState({menuVisible: !menuVisible})}
                        onClickOutside={() => this.setState({menuVisible: false})}>
        <div data-test-id="profileMenu"
             className={cn('ProfileTrigger', {defined: !!userIcon})}>{userIcon ? AvatarIcons[userIcon] :
          <div>click me to dropdown</div>
      </Dropdown.Trigger>
    );
}

export default MyComponent;
```

При клике или наведении на `div` с текстом появится компонент `<SomeComponent/>` в нижнем правом углу элемента. При клике вне дропдауна - компонент `<SomeComponent/>` скроется

11.4 bi-internal/root

Модуль, который хранит компоненты, необходимые для отрисовки стартовой страницы (та, на которую вы по умолчанию попадаете при входе и видите список разделов BI)

Модуль частично описан на Github

В клиенте этот модуль выглядит так:

```
import {Root} from '../../views/Root/Root';
import {RootContent} from '../../views/Root/RootContent';
import {RootMenu} from '../../views/Root/RootMenu';
import {RootHeader} from '../../views/Root/RootHeader';
import {RootSearch} from '../../views/Root/RootSearch';
import Provider from '../../views/Root/Provider';
import {BreadcrumbControl} from '../../views/Root/BreadcrumbControl';
import {DatasetsListView} from '../../views/Root/DatasetsListView';
import {DatasetsListView1} from '../../views/Root/DatasetsListView1';

export {Root, RootContent, RootMenu, RootHeader, RootSearch, Provider,
  BreadcrumbControl, DatasetsListView, DatasetsListView1};
```

Модуль описан на Github, несколько комментариев:

`Root` - общий компонент, полностью рендерящий внешний вид и поведение страницы разделов. состоит из меню слева (`RootMenu`), шапки `RootHeader` (с хлебными крошками `BreadcrumbControl`) и основного компонента с контентом текущего раздела `RootContent`.

Пример использования в компоненте `Root.tsx`:

```
import React from 'react';
import './Root.scss';
import {Strap} from 'bi-internal/face';
import {RootContent, RootHeader, RootMenu} from 'bi-internal/root';

export class Root extends React.Component<any> {
  public render() {
    const {activeTabIndex, activeChildIndex, tabs} = this.props;
    return (
      <section className="Root view roots custom">
        <Strap>
          <RootMenu activeTabIndex={activeTabIndex} activeChildIndex=
  {activeChildIndex} tabs={tabs}/>
          <Strap.Body>
            <RootHeader/>
            <RootContent {...this.props}/>
          </Strap.Body>
        </Strap>
      </section>
    );
  }
```

```
}

export default Root;
```

11.5 bi-internal/services

Модуль, хранящий самые важные сервисы, необходимые в кастомной разработке.

Модуль частично описан на Github.

В клиенте описывается так:

```
import DatasetService from "../../services/ds/DatasetService";
import CurrentDsStateService from "../../services/ds/CurrentDsStateService";
import DsStateService from "../../services/ds/DsStateService";
import {ProviderService} from "../../services/ProviderService";
import {DashboardByTopicsService, IRawDashboardTopicMode} from
  "../../services/ds/DashboardByTopicsService";
import {IDatasetsListModel, IDatasetsListItem} from
  "../../services/DatasetsListService";
import {DatasetsListService} from "../../services/DatasetsListService";
import { DatasetsListIcon } from '../../views/Root/DatasetsListView';
import { DatasetsListView1 } from '../../views/Root/DatasetsListView1';
import {getShell, shell} from "../../views/Shell";
import {IVizelConfig, IVizelProps} from '../../services/ds/types';
import {useService, useServiceItself} from "../../views/useService";
import {OpenModalVC} from "../../view-controllers/OpenModalVC";
import {PluginsManager} from "../../plugins/plugins-manager";
import {KoobDataService} from "../../services/koob/KoobDataService";
import { ISummaryModel, SummaryService } from '../../services/SummaryService';
import {KoobService} from "../../services/koob/KoobService";
import {KoobFiltersService} from "../../services/koob/KoobFiltersService";
import {ISearchVM, SearchVC} from '../../view-controllers/SearchVC';
import {Vizel} from "../../views/components/Vizel";
import * as service from "../../services/service";
import {IShellVM} from '../../view-controllers/ShellVC';
import {IDsShellVM, DsShellVC} from '../../view-controllers/DsShellVC';
import {DsShell} from "../../views/DsShell/DsShell";
import {createSubspaceGenerator} from
  "../../services/ds/createSubspaceGenerator";
import {ThemeVC} from "../../view-controllers/ThemeVC";
import {DatasetsByTopicsService, ITopic} from
  "../../../srx/services/DatasetsByTopicsService";
import {TransactionEntitiesService} from
  "../../services/TransactionEntitiesService";
import ResourceByNameService from '../../services/ResourceByNameService';
import {ResourcesOfDatasetsTreeService} from
  '../../services/ResourceLocatorService';
import {KoobDimensionsMemberService} from
  '../../services/koob/KoobDimensionsMemberService'
```

```
import CanIService from "../../services/CanIService";

export {
  CanIService, /* проверяет наличие прав на те или иные действия для текущей роли */
  CurrentDsStateService,
  DatasetService, // загружает модель указанного датасета
  DatasetsListService, // загружает модели доступных датасетов с рядом методов
  IDatasetsListModel,
  IDatasetsListItem,
  DashboardByTopicsService,  // загружает группы дешбордов с дешбордами в них
  IRawDashboardTopicMode,
  DatasetsListIcon,
  DatasetsListView1,
  IVizelConfig,
  IVizelProps,
  Vizel,
  DsStateService, /* сервис для загрузки информации о текущем состоянии атласа. Какие дешборды есть, какие дешлеты есть, конфиг текущего дешборда */
  KoobDataService, // сервис для загрузки данных из куба. реализует методы koobDataRequest3, koobCountRequest3
  KoobFiltersService, // сервис, управляющий текущими фильтрами на кубы
  ProviderService,
  PluginsManager, // Загрузит все доступные плагины (разделы на главной странице)
  ISummaryModel, SummaryService,
  getShell // вернет корневой компонент, родительский для всех компонентов на странице,
  useService,  // хук реакта, загружает модель указанного сервиса
  useServiceItself, // хук реакта, загружает инстанс указанного сервиса (модель и методы)
  ISearchVM,
  SearchVC, // сервис для реализации поиска по чему угодно
  IShellVM,
  IDsShellVM,
  DsShellVC,
  ThemeVC, // сервис работы с темами
  DsShell, // основной компонент хранящий все, что есть на странице с дешбордами
  createSubspaceGenerator, // функция, которая создает объект subspace
  DatasetsByTopicsService, // загружает группы датасетов и датасеты в них
  ITopic,
  TransactionEntitiesService,
  OpenModalVC,
  ResourceByNameService, // сервис ищущий указанный файл по иерархии атласов
  ResourcesOfDatasetsTreeService, // сервис ищущий указанный файл по иерархии атласов
  KoobDimensionsMemberService // загружает значения дименшна из словаря
};
```

Пример реального компонента с использованием модуля

```
import React, {useEffect, useState} from 'react'
import './MainPageRangeRow.scss'
import Select from '../elements/Select/Select'
```

```
import Option from '../elements/Select/Option'
import MoreButton from "../elements/ui/MoreButton"
import Range from '../elements/ui/Range'
import {UrlState} from "bi-internal/core";
import {KoobDataService, KoobService,
        useService, useServiceItself, KoobFiltersService} from "bi-internal/services";
import {$eid} from "bi-internal/utils";
const { koobDataRequest3 }:any = KoobDataService;
/**
 Принимает:
 customClass - доп.классы для MainPageRangeRow,
 name - название строки,
 selectItems - список селекта,
 selectVal - селект по умолчанию,
 url - url по которому нужно перейти при клике на Подробнее
 */

const MainPageRangeRow = (props) => {
  const url = UrlState.getInstance();
  const [unitId, SetSelectedUnitId] = useState(url.getModel()?.f?.unit_id ? url.getModel()?.f?.unit_id[1] : props.units?.[0].id || 0);
  const [data, setData] = useState([]);
  const koob = useService<KoobService>(KoobService, props.koob);
  const koobFiltersService:any = useServiceItself<KoobFiltersService>(KoobFiltersService);
  const koobFilters = koobFiltersService.getModel();

  useEffect(() => {
    let filters = {...url.getModel().f};
    filters.do_code = ['!=', 0];
    filters[props.nameDim] = [props.filtersId == 0 ? '!=' : '=', props.filtersId];
    filters[props.nameChildDim] = ['=', 0];
    filters.unit_id = ['=', unitId];
    filters.mr_code = ['=', 0];
    filters.process_code = ['=', 0];
    for(let key in filters) {
      if(key != 'process_code' && key != 'do_code' && key != 'dt' && key != 'mr_code' && key != 'mnt_period' && key != 'unit_id' && key != props.nameDim && key != props.nameChildDim) delete filters[key]
    }
    const dimensions = props.filtersId == 0 ? ['do_name:name', 'do_code:id'] : ['do_name:name', 'do_code:id', 'value'];
    const measures = props.filtersId == 0 ? ['sum(value):value'] : [];

    koobDataRequest3(props.koob, dimensions, measures, filters,
      {schema_name: UrlState.getModel().segmentId},
      'getData').then(data => {
      setData(data);
    })
  }, [url.getModel(), unitId])
```

```
  const column = $eid(koob.dimensions, 'do_code');
  const activeItemId = koobFilters.pendingFilters.do_code?.[1] ?? koobFilters.filters.do_code?.[1] ?? (column ? column.config?.defaultValue : 5);


if (koob.error || koobFilters.error || koob.loading || koobFilters.loading)
  return null;

  return (
  <div className={'MainPageRangeRow' + " " + props.customClass}>
    <div className="MainPageRangeRow__Name">
      {props.name}
      <div className="MainPageRangeRow__WrapperMore">
        <MoreButton classes={'MainPageRangeRow__More'} onClick={e => props.moreButton(props.nav, {[props.nameDim]: ['=', props.filtersId]})} />
      </div>
    </div>
    <div className="MainPageRangeRow__WrapperSelect">
      <Select
        className="MeasureSelect__Select MainPageRangeRow__Select"
        optionListClassName="MeasureSelect__List AppScroll"
        optionListOffset={[0, 10]}
        width={120}
        value={unitId}
        onChange={SetSelectedUnitId}
      >
        {props.units.length > 0 && props.units?.map(item =>
          <Option key={item.id} value={item.id}>{item.name}</Option>
        )}
      </Select>
    </div>
    <div className="MainPageRangeRow__WrapperRange">
      <Range data={data} activeId={activeItemId}/>
    </div>
  </div>
)
}

export default MainPageRangeRow;
```

11.6 bi-internal/ds-helpers

Дополнительный модуль со вспомогательными интерфейсами, методами и переменными для работы с МЛП сущностями, дешлетами, дешбордами.

Модуль описан на Github.

Пример импорта в компоненте

```
import {VizelConfig, VizelConfigDisplay, Dashlet, Dashboard} from 'bi-internal/ds-helpers';
```

11.7 bi-internal/ui

Модуль, хранит как явные UI компоненты, так и вспомогательные модули, нужные для их создания.

Модуль частично описан на Github

в клиенте этот модуль выглядит так:

```
import {BIIcon} from "../../views/components/BIIcon/BIIcon";
import {VizelFromCfg} from '../../views/components/Vizel/VizelFromCfg'
import {ExpandableSearch} from "../../views/components/ExpandableSearch/ExpandableSearch";
import {WpLoadingIcon} from "../../views/components";
import {DlgShareWithUser} from "../../views/dialogs/DlgShareWithUser/DlgShareWithUser";
import {AlertsVC} from "../../view-controllers/AlertsVC";
import {EditMenuItem} from "../../views/dd-menu";
import {PopupVC} from "../../view-controllers/dialogs/PopupVC";
import {ModalContainer, modalContainer} from '../../views/modal-container';
import {openModal} from '../../view-controllers/OpenModalVC';
import VirtualList from "../../views/components/VirtualList/VirtualList";
import {DrilldownMenu} from '../../views/dd-menu';
import ContextMenu from "../../views/components/ContextMenu/ContextMenu";
import {MainToolbarVC, IMainToolbarVM} from "../../view-controllers/panels/MainToolbarVC";
import {MainToolbar, MainToolbar__EditModeButton, HrefButton, MainToolbar__DataBoringButton, MainToolbar__ThemeSwitchButton} from '../../views/panels/MainToolbar/MainToolbar';
import ActionList from "../../../srx/components/action/ActionList";
import DlgAbout from "../../views/Root/ProfileComponents/DlgAbout";
import {OptionsProvider} from "../../config/OptionsProvider";
import EditModeVC, {IEditModeModel} from "../../view-controllers/EditModeVC";
import SVGIcon from "../../views/components/SVGIcon";
import {DASHBOARD_ICONS} from "../../const/DashboardIcons";
import BaseVizelEcharts from "../../views/vizels/BaseVizelEcharts";
import EPlot from "../../views/vizels/eplot";
import EditModeEnabler from "../../views/helpers/EditModeEnabler";
import L10n from "../../views/components/L10n/L10n";
import L10nVC from "../../view-controllers/L10nVC";
import AppPane, {Menu, MenuState} from "../../views/components/AppPane/AppPane";
import JsonSchemaForm from "../../views/components/JSONSchema/JsonSchemaForm";
import SimpleTable from "../../views/components/SimpleTable/SimpleTable";
import {VizelYVC} from '../../view-controllers/vizels/VizelYVC';
import {VizelXVC} from '../../view-controllers/vizels/VizelXVC';
import {VizelXYVC} from '../../view-controllers/vizels/VizelXYVC';
import {VizelGaugeVC} from '../../view-controllers/vizels/VizelGaugeVC';
```

```
34 import {VizelPivotMLP} from '../../view-controllers/vizels/VizelPivotMLP';
35 import {VizelControllerVector} from '../../view-controllers/vizels/VizelControllerVector';
36 import {BaseVizelVC} from '../../view-controllers/vizels/BaseVizelVC';
37 import * as VizelKoobControl from "../../views/vizels/VizelKoobControl";
38 import IfICan from '../../views/helpers/IfICan';
39 import Search from '../../views/components/Form/Search/Search';
40 import SelectSortBy from '../../views/components/Form/SelectSortBy/SelectSortBy';
41 import OnlyWhenVisible from  '../../views/helpers/OnlyWhenVisible';
42 import ConfirmWindow from '../../views/components/ConfirmWindow/ConfirmWindow';
43 import RootLogo from '../../views/Root/RootLogo';
44 import {DsShellHeader} from "../../views/DsShell/DsShellHeader";
45 import Select from '../../views/components/Form/Select/Select'
46 import {DatasetsListIcon} from '../../views/Root/DatasetsListView1'

48 export {
49   VizelFromCfg,
50   IfICan,
51   BIIcon,
52   ExpandableSearch,
53   WpLoadingIcon,
54   DlgShareWithUser,
55   ModalContainer,
56   modalContainer,
57   AlertsVC,
58   EditMenuItem,
59   PopupVC,
60   openModal,
61   VirtualList,
62   DrilldownMenu,
63   ContextMenu,
64   IMainToolbarVM,
65   MainToolbarVC,
66   MainToolbar,
67   MainToolbar__EditModeButton, HrefButton, MainToolbar__DataBoringButton, MainToolbar__ThemeSwitchButton,
68   ActionList,
69   DlgAbout,
70   OptionsProvider,
71   EditModeVC,
72   IEditModeModel,
73   SVGIcon,
74   SimpleTable,
75   DASHBOARD_ICONS,
76   BaseVizelEcharts,
77   EPlot,
78   EditModeEnabler,
79   L10n,
80   L10nVC,
81   AppPane,
82   Menu,
83   MenuState,
```

```
  JsonSchemaForm,
  VizelYVC,
  VizelXVC,
  VizelXYVC,
  VizelGaugeVC,
  VizelPivotMLP,
  VizelControllerVector,
  BaseVizelVC,
  VizelKoobControl,
  Search,
  SelectSortBy,
  OnlyWhenVisible,
  ConfirmWindow,
  RootLogo,
  DsShellHeader,
  Select
};
```

Частично описан на Github, ряд комментариев:

`IfICan` - React-компонент для оборачивания функционала по правам. Проверит один или более claim и покажет дочерние элементы. Клеймы могут быть как “все сразу” (eachOf), “один из” (oneOf) и “указанный” (one)

```
<IfICan eachOf={['U adm.topics', 'U adm.dataset_topics_maps']}>
  {/* Мы можем редактировать топики */}
 <SomeComponent/>
</IfICan>
```

`SomeComponent` появится только если оба указанных клейма вернут `true`

`VizelFromCfg` - позволит создать визель на основе `schema_name` атласа и конфига дешлета `rawCfg`.

`BIIcon` - компонент, принимающий `icon`,`onPress`, который отрисует иконку из BI, лежащую по адресу `assets/icons/${icon}.svg`.

`modalContainer` - инстанс класса ModalContainer, который может открыть вам модальное окно в стандартном режиме: то есть как если бы вы открыли встроенный `drillDown` Это можно сделать через

```
modalContainer.push({
  rawCfg: {/* Тут конфиг дешлета с обязательным указанием ключа view_class */},
  schema_name: "ds_51"
}, "Название моего окна или html строкой")
```

`openModal` - `async` метод, который может отобразить попап с любым реакт-компонентом, который вы ему передадите. Принимает сам реакт компонент и объект опций

```
interface IOpenModalVMOpt {
  readonly cancelWrapper?: boolean;   // default = true,  закрытие модального ок
```

```
  на по клику на wrapper
  readonly cancelEsc?: boolean;       // default = true,  кнопка esc не работает на закрытие
  readonly hiddenWrapper?: boolean;   // default = false, наличие wrapper'a
  readonly style?: { [id: string]: string | number };
  readonly className?: string;
  // ... todo дальнейшее расширение
}
```

```
openModal(
  <DeleteCube item={item}/>, {
      cancelWrapper: true,
      className: 'Action active',
      style: {
        'justify-content': 'center',
        'width': '25rem',
        'height': 'min-content',
        'min-height': 'auto',
        'padding': '1rem',
      }
    })
.then(() => {/* do smth */})
```

`AlertsVC` - сервис ОС, синглтон, отвечающий за пуш уведомления о статусе действия. Отображается всплывающим коротким сообщением разных цветов (от контекста уведомления зависит) `pushNewsAlert, pushInfoAlert, pushSuccessAlert, pushDangerAlert, pushWarningAlert` - методы этого сервиса, отображающие контекстные уведомления. Принимают описание, название, таймаут

```
AlertsVC.getInstance().pushSuccessAlert('Расписание успешно создано');
```

`VirtualList` - Компонент для отрисовки списка, который хранит только элементы, укладывающиеся в доступный вьюпорт. При скролле догружает список, выгружая ранний. Мастхев для больших данных чтобы рендер реакта не убил страницу.

```
const members = [{/*...*/}, {/*...*/}];

private _renderMember = (item: repo.adm.IRawUser, idx: number): JSX.Element => {
  return (
    <div key={item.id} className="MyClass--Item">
     <div>{item.name}</div>
   </div>
 );
}

public render() {
  return (
    <VirtualList
        className="MyClass"
        items={members}
    renderItem={(item) => this._renderMember(item)}
    />
```

```
  )
}
```

`VizelYVC,VizelXVC,VizelXYVC,VizelGaugeVC, VizelPivotMLP,VizelControllerVector` - классы-сервисы ОС, наследующиеся от `BaseVizelVC` и позводяющие инициализировать встроенные визуализации для дешлетов, при их переопределении.

`BaseVizelEcharts` - компонент, являющийся базовым родителем для всех коробочных визуализаций, которые работают на ECharts. Описывает абстрактные методы и их логику использования для получения итогового компонента с использованием Echarts и реакций на основные события.

`EPlot` - более продвинутый наследник `BaseVizelEcharts`, который полностью описывает типы `line`, `scatter`, `area` и является стартовым конфигом для ряда других встроенных визуализаций.

Достаточно создать React компонент, унаследовавшийся от данного компонента и переопределить функцию `_getEchartsConfig(vm)`, которая принимает объект с информацией о сериях и категориях, взятых из конфига дешлета, в котором данный компонент работает. Вернуть функция должна объект конфига ECharts по аналогии с тем, как это выглядит в объекте `option` тут

11.8 bi-internal/utils

Модуль содержит в себе ряд вспомогательных функций или утилит, библиотек, которые используются чаще других и могут пригодиться в кастомной разработке.

Модуль частично описан на Github

В клиенте этот модуль выглядит так:

```
const utils = require('../../utils/utils');
const utilsEcharts = require('../../utils/utilsEchars');
const {$eid, $eidx, $esid} = require('../imdas/list');
const cUtils = require('../../views/vizels/utility/c-utils');
const formatNumberWithString = require('@luxms/format-number-with-string');
import {mouseWatcher} from '../../libs/MouseWatcher';
import LoadFromResources from "../../views/components/LoadFromResources";

const skin = require('../../skins/skin.json');
import {parse as wktParse} from "wellknown";
import {lpeRun} from '../../utils/lpeRun';
import L from 'leaflet';

const getBuildVersion = () => process.env.VERSION;
const getBuildDate = () => process.env.DATE;

module.exports = {
  ...utils,
  ...utilsEcharts,
```

```
  ...cUtils,
  L, wktParse,
  $eid, $eidx, $esid,
  formatNumberWithString,
  mouseWatcher,
  skin,
  lpeRun,
  getBuildVersion,
  getBuildDate,
  LoadFromResources
};
```

Пакеты `c-utils`, `utils`, `MouseWatcher` описаны в проекте на Github, с той разницей ,что `utilsEcharts` - это ряд функций из `utils`, посвященных работе с цветом и градиентом в синтаксисе библиотеке Apache Echarts.

`formatNumberWithString` - функция, способная гибко форматировать числа, пришедшие как строка по указанному формату. Страница одноименной библиотеки: https://github.com/luxms/format-number-with-string

`$eid, $eidx, $esid` - функции из пакета `list` для получения энтити по id (в зависимости от контекста это м.б. дименшн, межа, метрика, локация, период), его индекса в списке и списка энтити по списку id.

`LoadFromResources` - `React`-компонент, который является оберткой над компонентом `MyComponent` из веб-клиента и проверяет наличие одноименного файла `MyComponent.js` в ресурсах согласно иерархии атласов (т.е. в `ds_res`, родительском атласе и текущем) И принимает те же `props`, что и сам компонент-ребенок, но с полем `path="MyComponent.js"`. Если такой компонент найден (и даже не в единственном экземпляре) - будет использоваться тот, кто ниже всех в иерархии атласов. Именно этот компонент оборачивает ключевые компоненты, доступные к переопределению в ресурсах в коде веб-клиента

`skin` - JSON-файл с настройками, влияющими на отображение (устаревший способ кастомизации функционала)

`L, parse` - Библиотека `leaflet` и ее парсер `wkt` для создания компонентов, реализующих слои на карте

`lpeRun` - модуль, а также одноименная функция, позволяющий парсить строки с lpe-выражением. Подробнее тут LPE. Удобен для задавания выражения, которое будет парсить параметр из конфига дешлета, исходя из переданного контекста, и влияние таким образом на поведение компонента. Первым аргументом функции является строка с `lpe:` внутри, вторым - объект контекста, где по факту должны быть описаны все переменные или функции, которые используются в данном выражении из первого аргумента.

`getBuildVersion, getBuildDate` - отдают версию клиента и дату сборки.

Пример подключения элементов модуля в конкретном компоненте:

```
import {formatNumberWithString, coloring, bi} from 'bi-internal/utils';
```

12 Observable сервисы

Это инстансы класса, являющегося наследником базового класса `BaseService`, который вы получаете из обвязки (из модуля `"bi-internal/core"` если точнее).

Он сам по себе реализует паттерн `Observable`, при котором вы формируете наблюдаемую модель (некий объект сродни `state` в `React`), изменения которого происходят публичными методами этого сервиса (или из-за того, что сервис подписан на изменения другого сервиса, который драматически влияет на поведение текущего) и который реализует (наследует от `BaseService`) механизм уведомления компонентов-подписчиков данного сервиса.

Основная цель `Observable` сервиса - предоставить вам механизмы взаимосвязи между как `React` компонентами в разных контекстах (в разных атласах, дешбордах, дешлетах), так и экономного запрашивания данных, которые “дорого” запрашивать часто, не кешируя.

Так же он позволит вам выступать в качестве портала, через который можно передавать данные, реализовывать собственную событийную систему. Или в качестве некой смеси `Model-Controller` из `MVC` архитектуры для фронтенда.

Чаще всего обязательными ключами такой модели являются `loading` и `error` (по ним вы можете отслеживать, готова ли модель для работы или еще в процессе загрузки, а также не содержит ли она ошибок). Это наследие интерфейса `IBaseModel`

```
export interface IBaseModel {
  error?: string | null;
  loading?: boolean | number;
}
```

```
loading: true,
error: null
```

То есть мы изначально ожидаем, что ошибок нет, а модель грузится. Когда целевое действие в инициализации сделано (например вы загрузили с сервера важные структуры данных и как-то организовали их хранение), то вы выставляете `loading: false` и, если имеет место - указываете текст ошибки. Работа с сервисом начнется именно с момента `loading: false`.

Чтобы подписаться на изменения модели такого сервиса нужно заинициализировать его и указать `callback`-функцию, которая вызывается на события изменения модели сервиса, полной или частичной. По умолчанию такой `callback` автоматически принимает на вход текущий объект модели сервиса после каждого события изменения. Подписка по-разному выглядит для классовых и функциональных компонентов реакта, но об этом позднее.

Сами по себе сервисы могут быть:

- `Singleton` конструктор такого класса пустой, а сам инстанс получаем статическим методом `MyService.getInstance()`.

- `Factory` конструктор зависит от неких входных параметров (например идентификатора куба, имени схемы атласа и чего угодно, что для вас имеет смысл) и сохраняет каждый инстанс по соответствующему идентификатору `MyService.createInstance(koobId)`. Это уместно, когда для каждого из кубов (например), которые вы встречаете на странице вы делаете ресурсоемкую задачу и не хотите дубляжа - подобный сервис вам в помощь

Инстансы сервисов, какие бы они ни были, принято хранить в глобальных переменных внутри `window` и предварять его название символом `__`. Например, если в консоли браузера, в котором открыта страница инстанса Luxms BI набрать `__`, то мы увидим ряд сервисов, которые были сохранены системой после своей инициализации. Там всегда хранится актуальная его версия. Очень полезно так можно посмотреть на текущие настройки:

Если вы хотите создавать инстансы для каждого из переданных идентификаторов, то вот пример метода `createInstance`:

```
export class MyService extends BaseService<IMyServiceModel> {
  private readonly id: string | number;
  private constructor(koobId: string) {
      super({
         loading: false,
         error: null,
         data: [],
       });
      this.id = koobId;
  }

 // Тут какие-то публичные методы, которые вам понадобятся
  public setSomething = (smth: string[]) => {
    this._updateWithData({
       data: smth
      })
  }

  public static createInstance (id: string | number) : MyService {
    if (!(window.__myService)) {
      window.__myService = {};
    }
    if (!window.__myService.hasOwnProperty(String(id))) {
      window.__myService[String(id)] = new MyService(String(id));
    }

    return window.__myService[String(id)];
  };
}
```

Аналогичный пример для `getInstance`:

```
export class MyService extends BaseService<IMyServiceModel> {
  private constructor() {
      super({
         loading: false,
         error: null,
         data: [],
       });
  }

 // Тут какие-то публичные методы, которые вам понадобятся
  public setSomething = (smth: string[]) => {
    this._updateWithData({
       data: smth
      })
  }

  public static getInstance = () => {
    if (!(window.__myService)) {
      window.__myService = new MyService();
    }
    return window.__myService;
  };
}

MyService.getInstance();
```

Итого: Вы можете использовать пример с сервисами выше как шаблон написания собственных сервисов. К чему все сводится:

- Создаете папку под сервисы, например `src/services`
- Создать в нем файл `MyService.ts`, название файла одноименно с названием класса.
- Класс должен быть наследником `BaseService`
- В конструкторе прописываете логику инициализации (или вызываете функцию инита, которую в классе и пропишете)
- Пишете метод (`getInstance`, `createInstance`), который создает инстанс и прихранивает его в переменной где-то в `window`
- Создаете публичные методы, которые будете дергать из компонентов-подписчиков, и которые меняют модель сервиса одним из методов типа `_updateWithData` или `_updateModel` (о них ниже)

На этапе работы с сервисами вам может пригодиться обращение к сохраненному инстансу сервиса, чтобы проверить верность его модели или работы методов.

- Открыть консоль браузера
- Набрать `__appConfig`
- Увидим инстанс класс `AppConfig` с настройками фронтенд-приложения из `settings.js`
- `__appConfig.getModel()` или `__appConfig._model` отдаст текущий объект модели данного сервиса.

- Для сервиса типа `Factory` вы увидите не просто объект класса сервиса, а объект с ключами, являющимися идентификаторами из конструктора или метода `createInstance`. И чтобы получить модель, вам нужно будет дополнительно выбрать идентификатор того инстанса, чью модель вы хотите получить.

12.1 Методы изменения модели

От предка сервисы наследуют стандартные гетеры и сетеры в виде `_setModel(model: M)` и `getModel()` методы. Которые устанавливают полную модель или возвращают ее соответственно. Так же есть методы:

```
// Изменить несколько полей в модели и уведомить слушателей
protected _updateModel(partialModel: Partial<M>): void {
  this._setModel({
    ...(this._model as any),
      ...(partialModel as any),
  });
}

// Уведомить слушателей что модель загружается (принудительно)

protected _updateWithLoading(): void {
  this._updateModel({ loading: true, error: null } as any);
}

// Уведомить слушателей что модель содержит ошибку (принудительно)

protected _updateWithError(error: string): void {
  this._updateModel({ loading: false, error } as any);
}
// Уведомить слушателей что модель загружена с новыми значениями полей
  (принудительно)

protected _updateWithData(partialModel: Partial<M>): void {
  this._updateModel({ loading: false, error: null, ...(partialModel as any) });
}
```

Пример:

```
const publicFilters = {sex: ['=','Мужской'] };
const privateFilters = {age: ['>=',50'] };

/* Выставит в модели UrlState видимые (publicFilters) в урле фильтры и скрытые
  (privateFilters), которые пропадут после перезагрузки страницы */
UrlState.getInstance().updateModel({f: publicFilters, _koobFilters:
  privateFilters});
```

12.2 Методы уведомления подписчиков

От предка сервисы наследуют методы:

```
/* Можно вызвать событие и вызвать колбеки слушателей, передав им на вход список аргументов */
protected _notify(eventDescription: IEventDescription, ...args: any[]) {
 const listeners = this._listeners.filter(listener => shouldNotifyListener(eventDescription, listener));

    listeners.forEach(listener => {
      if (!this._listeners) {                                  // suddenly we were disposed
        debugger;
        return;
      }

      if (this._listeners.includes(listener)) {            // check if listener was removed during update
        try {
          listener.callback(...args);
        } catch (err) {
          console.log(`[Observer] Error notifying listener of event "${eventDescription}"`);
          console.log(err);
        }
      }
    });
}
```

12.3 Методы реализации подписки на изменение модели

От предка сервисы наследуют методы:

```
/* Принимает идентификатор события или поле модели и колбек, вызываемый при наступлении события без первичного уведомления подписчиков */
subscribe(event: string | string[], (model: M) => void)

/* Принимает колбек, вызываемый при при наступлении события и с первичным уведомлением подписчиков  */
subscribeAndNotify(event: string | string[], (model: M) => void)

/* Принимает колбек, вызываемый при любом изменении полей модели без первичного уведомления подписчиков  */
subscribeUpdates((model: M) => void)
```

```
/* Принимает колбек, вызываемый при любом изменении полей модели и с первичным уведомлением подписчиков */
subscribeUpdatesAndNotify((model: M) => void)
```

Эти методы вы будете использовать только в классовых компонентах, ибо в функциональных методы работы с ней иные.

Колбек есть функция написанная в `React` компоненте и которая совершает какие-то действия с его `state` (вызывает `setState`) (или другом сервисе и тогда речь о его модели). При указании нее вы автоматически передаете `this` как ссылку на компонент сервису. Благодаря ей он и вызывает нужный колбек у нужного компонента, дирижируя состояниями компонентов на страницах.

12.4 Методы реализации отписки от изменений модели

`unsubscribe` - метод, который очищает память и удаляет инстанс. На тот случай. если хотите очистить память или если вы не хотите доверять тому, что есть в кеше и хотите инициировать его заново при определенных условиях. Чаще всего вызывается на unmount реакт-компонента, если это необходимо

```
unsubscribe((model: M) => void);
```

12.5 Метод whenReady()

Метод для случаев, когда явного метода для инициализации инстанса нет. Проверяет ,что сервис загружен, ошибок нет и автоматически реализует подписку на обновление полей модели, без первичного уведомления подписчиков по умолчанию.

```
/**
   * Wait until service in state 'ready' (which means, no error and no loading)
   * return Promise<MODEL>
   *
   * if model signal error, it rejects resulting promise
   *
   * @returns {Promise<MODEL>}
   */
  public whenReady(): Promise<M> {
    return this.lock<M>(() => {
      if (this._model.error) {
        return Promise.reject(this._model.error);
      }
      if (this.isReady()) {
        return Promise.resolve(this._model);
      }
      return new Promise<M>((resolve, reject) => {
```

```
        if (this._model.error) {
          reject(this._model.error);
          return;
        }
        if (this.isReady()) {
          resolve(this._model);
          return;
        }

        const subscription = this.subscribe('update', model => {
          if (model.error) {
            subscription.dispose();
            reject(model.error);
          } else if (this.isReady()) {
            subscription.dispose();
            resolve(model);
          }
        });

        // return this.happens('ready');
      });
    });
  }
```

12.6 Сервис AppConfig для работы с настройками из settings.js

Пакет `"bi-internal/core"`, `Singleton` Инстанс получаем через `AppConfig.getInstance()`

Сервис, который хранит своей модели информацию о настройках из файла `settings.js`.

Из него вы можете подчерпнуть информацию о текущей языковой локализации, темах (подразумевается, что тут вы переопределяете базовые темы для текущего инстанса), название проекта и используемые опции в блоке `features`.

Чтобы получить информацию о настройках, вам нужен следующий код:

```
import React from 'react';
import {AppConfig} from 'bi-internal/core';

class MyClassComponent extends React.Component {
  constructor(props) {
   super(props);
    this.state = {
     appConfigModel: AppConfig.getInstance().getModel()
    }
  }

  public render() {
```

```
    const {appConfigModel} = this.state;
   return (
      {/* Отобразит Luxms BI по умолчанию*/}
     <div>{appConfigModel.projectTitle}</div>
    );
  }

}

const MyComponent = (props) => {
 const appConfigModel = useService(AppConfig);
 return (
    {/* Отобразит Luxms BI по умолчанию*/}
   <div>{appConfigModel.projectTitle}</div>
  );
};

export default MyComponent;
```

У данного сервиса есть полезный публичный метод, который вам понадобится при запросах к API приложения или файлам: `AppConfig.fixRequestUrl(url: string): string`. Формирует корректный урл к указанному в аргументе пути на основе настроек приложения.

```
const url = AppConfig.fixRequestUrl('/srv/datagate/source/upload');
```

Перечень статичных методов сервиса, позволяющих быстро получить значение соответствующего поля модели:

```
public static fixRequestUrl(url: string): string {
  return this.getInstance().fixRequestUrl(url);
}

public static hasFeature(featureName: string): boolean {
  return this.getInstance().hasFeature(featureName);
}

public static getProjectTitle(): string {
  return this.getModel().projectTitle;
}

public static getLocale(): string {
  return this.getModel().locale;
}

public static getLanguage(): 'en' | 'ru' {
  return this.getModel().language;
}

public static getPlugins(): string[] {
  return this.getModel().plugins;
}
```

12.7 AuthenticationService и информация о пользователе

Пакет `"bi-internal/core"`, `Singleton` Инстанс получаем через `AuthenticationService.getInstance()`

Сервис реализует фронтенд часть механизма аутентификации и хранит конфиг пользователя.

Подразумевается, что вы хотите переопределить окно логина и написать свой способ аутентификации пользователя. Для этого вы создали компонент `DlgAuth.js` в ресурсах атласа `ds_res` и в нем реализуете интерфейс для новой формы входа пользователя в BI.

Используемые интерфейсы:

```
// Интерфейс модели сервиса

export interface IAuthentication extends IBaseModel {
  error: string | null; // есть ли ошибка в процессе формирования модели
  loading: boolean; // загружено ли все необходимое для модели
  authenticating: boolean; // в процессе ли аутентификации
  authenticated: boolean; // аутентифицирован (есть кука)
  userId?: number; // id пользователя
  access_level?: string; // старый режим прав доступа (с выключенным rbac9). admin или usual
  name?: string;// имя пользователя
  site_role: string, // (автоматически подставляется БД) сайтовая роль
  license_role: string,  // (автоматически подставляется БД) лицензионная роль
  userConfig?: { [key: string]: string }; /* объект с доп.полями пользователя. Что угодно, например идентификатор выбранной ранее в профиле темы themeId */
  isNeed2FACode?: boolean; /* ждем ли ввод смс при двухфакторке (нужно указать в таблице adm.configs authType = '2fa') */
  isBlocked?: boolean; // заблокирован ли
  errorKey?: string; // ключ ошибки из БД
  errorMessage?: string; // описание ошибки из БД
  violationMessages?: string[]; // список нарушений парольной политики (при смене пароля)
  redirectUri?: string; // адрес редиректа при статусе запроса 303
}

// Интерфейс пользователя системы

export interface User {
  id: number; // ID
  name: string; // имя пользователя (ФИО например)
  email: string; // почта
  username: string; // логин
  phone: string; // телефон
  access_level: 'admin' | 'usual'; // старый режим прав доступа (с выключенным rbac9). admin или usual
  config: any; // объект с доп. информацией о пользователе
  is_blocked: 0 | 1; // заблокирован?
```

```
  is_local: boolean; // локальный ли пользователь или доменный
  license_role_ident: string; // идентфикатор лицензионной роли
  site_role_ident: string; // идентфикатор сайтовой роли
  password_policy: number; // парольная политика
}

interface IAuthCheckData2 {
  sessionId: string;
  user: User;
  authType?: string;
}

interface IAuthCheckData3 {
  access_level: string; // 'admin' или 'usual'
  id: string; // user id
  name: string; // имя пользователя
  authType?: string; // тип авторизации (двухфакторная (2fa) или иная)
  config: any; /* конфиг пользователя, например ключи
  avatarId: "g1x3y3" (идентификатор иконки пользователя)
 favoriteAtlases: [112,1560, 1553,70] (Атласы в избранном)
 themeId: "light" (сохраненная ранее тема)
  */
}
```

Пример модели сервиса и ответа методов `check`, `login`:

```
{
    "id": "1",
    "username": "adm",
    "email": "blackhole@localhost.localdomain",
    "access_level": "admin",
    "name": "Default Admin",
    "config": {
        "themeId": "light"
    },
    "sys_config": {
        "ext_groups": []
    },
    "site_role": "Super",
    "license_role": "Admin",
    "error": null,
    "loading": false,
    "authenticating": false,
    "authenticated": true,
    "userId": 1,
    "userConfig": {
        "themeId": "light"
    },
    "redirectUri": null
}
```

```
import {AuthenticationService, IAuthentication} from 'bi-internal/core';

const MyComponent = (props) => {
  const authService = AuthenticationService.getInstance();
  const authModel: IAuthentication = authService.getModel();

  {/* "Default Admin" */}
  return (<div>{authModel.name}</div>)
}

export default MyComponent;
```

12.7.1 Базовый метод `init` сервиса `AuthenticationService`:

Запускается в конструкторе

```
private async _init(): Promise<any> {
  /*
   Запускает процесс аутентификации, делает метод `check`, ловит ошибки, проверяет количество попыток входа и статусы ответов
  */
}
```

12.7.2 API для нужд аутентификации/авторизации

12.7.2.1 Запрос check:

`GET /api/auth/check`

По умолчанию сам факт обращения по данному урлу запускает механизм аутентификации через обращение к таблице `adm.users` (с проверкой прав доступа при `rbac9`). Так же возможно включить `SSO Kerberos / Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO)` (Подробнее о настройке сервера в `Руководстве системного администратора. Приложение D`). Возвращает объект информации о пользователе интерфейса `User` с `IAuthCheckData2` или `IAuthCheckData3`.

Соответствующий метод сервиса:

```
public async check(): Promise<IAuthCheckData2 | IAuthCheckData3> {
 /*
   считывает настройки сервиса AppConfig, UrlState
  проверяет режимы loginme, no-sso и OIDC
  Если есть функция getAuthToken, использует ее для получения токена OIDC
  */
}
```

12.7.2.2 Запрос `check-no-sso`:

`GET /api/auth/check-no-sso`

Используется автоматически, если в урле `?no-sso`. Возвращает то же, что и обычный `check`, но пользователя берем из `adm.users`, а не внешнего источника. Необходимо для случаев, когда в системе включен режим `OpenID Connect`, но нам по какой-либо причине нужно принудительно попасть в систему и что-то там настроить.

Соответствующий метод сервиса: `check()`, разница только в итоговом урле `GET` запроса.

12.7.2.3 Запрос `login`:

`POST /api/auth/login`

Осуществляет процесс передачи логина и пароля на сервер для аутентификации пользователя. При успешном входе вернется `authenticated: true` и возвращается объект информации о пользователе, аналогичный запросу `check`.

body:

```
{
  `username=${encodeURIComponent('admin')}&password=${encodeURIComponent('12345')}`
}
```

```
axios({
  method: 'post',
  url: AppConfig.fixRequestUrl(`/api/auth/login`),
  data: `username=${encodeURIComponent(username)}&password=
  ${encodeURIComponent(password)}`,
  headers: { 'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8'
  },
}
```

Соответствующий метод сервиса:

```
public async signIn(username: string, password: string, new_password?: string):
  Promise<IAuthentication> {
    /*
     выполняет запрос `login` и `2fa` (при необходимости)
    */
}
```

12.7.2.4 Запросы `2fa`:

В запросе `login` есть ситуация, когда включен режим `authType: '2fa'`. Тогда после отправки логина и пароля через `login` вернется частичная модель со статусом `449`, требующая ввести код подтверждения и `isNeed2FACode: true`.

Сервис, отсылающий смс с кодом настраивается на сервере, однако, допустим, что вам уже завели учетную запись, указав телефон, вы ввели логин/пароль и вот код уже у вас на руках. Получив частичную модель вы увидели (или отрисовали в своем компоненте `DlgAuth.js`) форму ввода кода подтверждения.

Запрос `2fa/login2`:

`POST /api/auth/2fa/login2`

Отправляет код подтверждения на сервер.

body:

```
{
  factor2: parseInt(code, 10)
}
```

Соответствующий метод сервиса:

```
public async signInWithCode(code: string): Promise<any> {
  /*
     выполняет запрос `2fa/login2`
  */
}
```

Запрос `2fa/resend-factor2`:

`GET /api/auth/2fa/resend-factor2`

Приводит к попытке повторной отправки кода подтверждения на телефон (сервис приема кодов).

Соответствующий метод сервиса:

```
public async resendCode(): Promise<any> {
  /*
     выполняет запрос `2fa/resend-factor2`
  */
}
```

12.7.2.5 Запрос `logout`:

`GET /api/auth/logout`

Осуществляет выход из учетной записи пользователя.

Соответствующий метод сервиса:

```
public async signOut(): Promise<any> {
  /*
     выполняет выход из BI и при `OIDC` перенаправляет на урл провайдера `OIDC` для выхода
  */
}
```

12.7.3 Процесс аутентификации:

В таблице `adm.configs` есть поле `authorization.mode`, отвечающее за режим авторизации. Если указан `rbac`, то используются спец. алгоритмы по проверке домена и групп LDAP и правила из таблиц в схеме `rbac`. Если указан `rbac9`, то используется новая гранулярная система прав доступа, появившаяся в версии `v9`

Есть `authentication.mode`, отвечающий за режим аутентификации. Он или пустой или `OIDC` - аутентификация через протокол `OpenID Connect` через `Keycloak`,`Blitz` и иные провайдеры данного протокола. Если включен `OIDC`, для подключения используются настройки с префиксом `authentication.OIDC`

Есть поле `luxmsbi.authentication`. Оно или пустое или `2fa` (2-факторная авторизация с кодом через смс или бот).

Как `AuthenticationService` проводит аутентификацию на уровне фронтенда:

1. Проверяем, нет ли `?loginme` в урле.
2. Если есть - не делаем `check`, принудительно показываем окно входа в систему, далее после ввода логин-пароля шлем запрос `login` (опционально запросы типа ‘2fa’), проверяем пользователя по таблице `adm.users`, получаем куку `LuxmsBI-User-Session` и мы в системе. Необходимо для тестировщиков
3. Если нет - проверяем, нет ли `?no-sso` в урле.
4. Если есть - Делается запрос `check-no-sso`. идем в `LDAP` и проверяем пользователя в `adm.users`.
5. Если нет - Делается запрос `check`, что запускает процесс аутентификации через `SPNEGO` (v9) или `LDAP` (default)
6. Если в конфиге `authentication.mode = null` - идем в `LDAP` и проверяем пользователя в `adm.users`
7. Если в конфиге `authentication.mode = 'OIDC'` (`OpenID Connect`), то

- используя настройки в конфиге с префиксом `authentication.OIDC`, выполняем `check`
- получаем в ответе статус 303 и `redirect_uri`, на который мы редиректимся на страницу провайдера `OpenID Connect` (`Blitz`, `Keycloak`)
- Авторизуемся в системе провайдера
- Возвращаемся обратно в BI.

8. Выход из учетки осуществляется аналогично. В каждый запрос после такой авторизации мы подмешиваем заголовок

```
'Authorization': `Bearer ${token}`
```

9. Если нам не нужно, чтобы нас редиректило со статусом 303 при включенном 'OIDC', то мы можем указать в `opt/luxmsbi/web/settings/settings.js` в блоке `features` опцию `!OIDCRedirect`, тогда, получив 303 мы принудительно остаемся на странице с окном авторизации и в кастомном компоненте `DlgAuth.js` реализовать логику обработки 303, храня `redirect_uri` до востребования. Таким образом возможно сделать страницу с выбором: авторизоваться через `LDAP` или через `OpenID Client` протокол, ибо урл с редиректом уже у нас, его можно просто навесить как обработчик клика по какой-либо кастомной кнопке.

10. Допустим есть система, которая ходит в провайдер `OIDC`, например `Keycloak`. В ней есть блок, где вставлен BI в `iframe` и в нем мы хотим авторизоваться по токену из родительского окна. Мы можем прописать в конфиге функцию `AppConfig.getModel().getAuthToken`, которая будет реализовывать логику кастомного получения токена внутри фрейма. Возвращает соответствующий токен .

Родительский фрейм может реализвовать API через механизм `window.postMessage`: когда родительскому фрейму приходит сообщение от `iframe` с BI:

```
{
  "type": "jwtToken",
  "forceUpdate": true | false
}
```

родительский фрейм должен отправить ответное сообщение:

```
{
  "type" :"jwtToken",
  "token": "сам токен",
  "expires": "timestamp когда надо будет его перезапросить"
}
```

Пример функции `getAuthToken`:

```
getAuthToken: function() {
  if (window.parent === window) return;
  return new Promise(function (resolve) {
    var onMessage = function (event) {
      if (event.data.type === 'jwtToken') {
        window.removeEventListener('message', onMessage)
        resolve(event.data);
      }
    };
    window.addEventListener('message', onMessage);
    window.parent.postMessage({"type":"getJwtToken","forceUpdate":false}, 'https://graph-demo-ui.datacloud.t1-cloud.ru/');
  });
},
```

12.8 UrlState и работа с url приложения

Пакет `"bi-internal/core"`, `Singleton` Инстанс получаем через `UrlState.getInstance()`

Сервис следит за состоянием `url` SPA-приложения `luxmsbi-web-client` (Обвязки).

```
import {UrlState} from 'bi-internal/core';
```

Его модель выглядит так: Ряд ключей уже устарели и являются необходимыми только для `MLP` кубов (нашей более старой версии хранения данных в атласе (тогда это звалось датасет)), потому я позволил себе их убрать из списка:

```
dash: null // строка, хранящая идентификатор текущего выделенного дешлета (раскрытого на весь экран)
dboard: "4" // идентификатор текущего дешборда
path: ( ['ds', 'ds_5142', 'dashboards'] // полный путь к текущему разделу
route: "#dashboards" // идентификатор роута текущего раздела. В данном случае мы на странице с дешбордами
segment: "ds" // идентификатор плагина (сегмента) в общем случае или одного из нескольких разделов ,
              //  которые вы видите на стандартной разводящей странице после авторизации
segmentId: "ds_5142" // идентификатор элемента раздела (сегмента)
slide: null // идентификатор слайда (актуально при предпросмотре презентации)
f: {} // опциональный ключ, который по синтаксису идентичен тому, что вы пишете в блоке filters конфига дешлета
    // с той разницей, что туда нельзя писать значение true, только идентификатор дименшна и массив
    // с оператором и операндами. Если там появится что-то вроде sex: ["=", "Мужской"] то вы таким образом
    // наложите сохраняющийся при перезагрузке страницы фильтр на дименшн sex. Потому этот способ
    // используется, чтобы передать кому-то ссылку на страницу с предустановленными фильтрами (будет get-параметр &f.sex=Мужской)
_koobFilters: {} // хранит фильтры из сервиса фильтров, которые вы не хотите показывать в урле (например потому, что урл не резиновый, а строка фильтра может быть огромной)
```

У данной модели есть особенности: все ключи здесь являются частью стандартного интерфейса `IUrlState`. И не все из них вы видите в итоговом урле приложения, потому что не все из них на данный момент времени могут иметь значение или оправданы разделом. Не являются частью стандартного интерфейса `"f"` и `"koobFilters"` они добавляются программно сервисом по фильтрации данных например.

Так вот постулируется следующее: все ключи в модели `UrlState`, которые содержат `"_"` в начале являются скрытыми и явно в урле не видны. Например вы хотите сохранить в модели `UrlState` объект, который хранит какую-то нужную вам информацию на другом дешборде или на другом атласе. Вы можете сохранить его как раз в ключе с `"_"`. А если хотите сделать новый `GET`-параметр - укажите обычный ключ, без префикса.

Этот сервис предоставляет ряд методов, которые можно использовать:

```
UrlState.navigate({segment: 'ds', segmentId: 'ds_230', dboard: "3"})
```

метод принимает объект который частично или полностью содержит ключи, которые вы хотите переопределить в модели урла и как следствие совершить переход на другой раздел, дешборд, атлас или слайд. В данном примере совершится переход на дешборд с идентификатором 3 атласа с `schema_name = 'ds_230'`.

```
UrlState.updateModel({_myData: {key1: 4546, data: [124, 5757, 575468]}})
```

И последующее получение

```
UrlState.getModel()._myData
```

Вы сохраните в модели данного сервиса данные, которые могут использовать ваши компоненты React по всему инстансу Luxms BI. Не злоупотребляйте! придерживайтесь принципов грамотного разбиения сервисов и компонентов по выполняемым ими функциям, а не использовать один как швейцарский нож.

Подобная возможность хранения в данной модели произвольных данных связана с функционалом презентаций, когда вы путешествуете по разделам BI и добавляете интересующую вас страницу (например ту, где вы выбрали какой-то фильтр или сделали действие ,которое достигается вашим кастомным сервисом). Добавление страницы в конкретный шаблон презентации под капотом есть ни что иное как добавление в список ее полной модели `UrlState`.

Ибо на сервере при генерации презентаций работает `headless chrome`, который получает объект модели `UrlState`, восстанавливает страницу, исходя из модели и результат рендерит и сохраняет картинку в `pdf` или `pptx` (и так для каждого слайда).

Это должно говорить вам о том, что если ваш кастомный сервис качественно влияет на внешний вид и функционал страницы - озаботтесь тем ,чтобы придумать и сохранить ключевую информацию в модели `UrlState`. Это означает, что вы не храните там без нужды результат тяжелого запроса или огромный массив, а храните только то, без чего например запрос за этим массивом невозможен. Тогда, при попадани такой модели в презентацию - сервер сможет восстановить именно ту ситуацию, какую вы ожидаете увидеть в презентации.

12.9 Примеры сервисов

Давайте рассмотрим пример `observable` сервиса, который вы можете использовать в своих компонентах. Предлагаю хранить все ваши сервисы в папке `src/services` проекта `BMR` и иметь в имени класса слово `"Service"`. Данный сервис есть ни что иное как локальная версия коробочного сервиса по управлению фильтрами `KoobFiltersService` (назовем файл `KoobFiltersService.ts`). И допустим мы хотим его как-то поменять и в дальнейшем в компонентах использовать эту версию сервиса, а не ту, что приходит нам из обвязки

```
import { BaseService, UrlState } from 'bi-internal/core';
import { throttle } from 'lodash';

const throttleTimeout = 0;  // можно ставить достаточно большим
                            // повторные фильтры будут срабатывать в течение это
  го времени
export interface IKoobFiltersModel {
  loading?: boolean;
  error?: string;
  query?: string;
  result: any;
  filters: any;
  pendingFilters: any;
}

export class KoobFiltersService extends BaseService<IKoobFiltersModel> {
  private constructor() {
    super({
      loading: false,
      error: null,
      query: undefined,
      result: {},
      filters: {},
      pendingFilters: {},
    });
   // Хотим, чтобы каждый раз ,когда в модели
    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 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) {
                // Может быть стоит добавить какое-то более остроумное условие
      if (publicKeys.includes(key)) {
        publicFilters[key] = filters[key];
      } else {
        privateFilters[key] = filters[key];
      }
    }

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

  public static getInstance = () => {
    if (!(window.__koobFiltersService)) {
      window.__koobFiltersService = new KoobFiltersService();
    }
    return window.__koobFiltersService;
  };
}

KoobFiltersService.getInstance();
```

`KoobFiltersService` - `singleton`, который аггрегирует в себе информацию обо всех фильтрах, которые были сделаны пользователем на текущий момент. По умолчанию, на фильтр, добавляемый в модель такого сервиса реагируют не все деши, а лишь те, для которых в блоке `filters` прописано `true` на него

```
filters: {
    category: ["=", "Clothes", "Shoes", "Scarfs", "Bags"],
    example: true
}
```

то есть если я каким-то образом (через упр.деш или программно) выставлю фильтр на `exampleDimension` - дешлет обновится и перезапросит данные.

В затроттленном методе `applyAllFilters` мы видим как раз ту самую ситуацию, что мы хотим выставить важные для нас параметры в урл (явно и неявно)

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

12.10 Подписка на сервисы через React хуки

Мы уже видели в примеров функционального компонента `MyComponent` подписку на сервис `KoobFiltersService`

Она достигалась методами `useService`, `useServiceItself`.

Так вот, такая запись

```
const koobFiltersService = ↩
  useServiceItself<KoobFiltersService>(KoobFiltersService);
```

или такая

```
const koobFiltersServiceModel = ↩
  useService<KoobFiltersService>(KoobFiltersService);
```

В обоих случаях приведет к тому, что модель сервиса, хранящаяся в переменной `koobFiltersService` (через `.getModel()`) или `koobFiltersServiceModel` будет всегда актуальной. и когда бы вы не обратились к этим переменной - они, при условии, что `loading: false` дадут вам свою актуальную модель

Достаточно в ключевых местах (но только ниже всех `useEffect`, иначе ошибка минифицированного React будет) вызвать блок

```
if (koobFiltersServiceModel.loading || koobFiltersServiceModel.error) return;
```

Или иным способом проверять что сервис загружен. И далее работать в обычных рамках функционального реакт-компонента.

Примеры для сиглтон и не-сингltон сервисов

```
const koobFiltersService = useServiceItself<KoobFiltersService>(KoobFiltersService)
const datePickerService = useServiceItself<DatePickerService
  >(DatePickerService, "luxmsbi.myKoob")
Классовые же компоненты требуют от вас немного большей организованности:

Рассмотрим пример

import React from "react";
import './DatePickers.scss';

// Подключили кастомный сервис
import {DatePickerService} from "../services/ds/DatePickerService";

export default class DatePickers extends React.Component<any> {
  private _datePickerService: DatePickerService = null; // подготовили переменну
  ю для хранения инстанса сервиса внутри текущего компонента.

  public state: {
    koob: string;
    data: any;
    error: string,
  };

  public constructor(props) {
    super(props);
    this.state = {
      koob: "",
      data: [],
      error: ""
    };
  }
  public componentDidMount(): void {
    const { cfg } = this.props;
    const koob = cfg.getRaw().koob;

   // Пусть сервис зависит от идентификатора куба
    this._datePickerService = DatePickerService.createInstance(koob);
    this._datePickerService.subscribeUpdatesAndNotify(this._onSvcUpdated); // по
  дписка на все изменения модели
  }


  private _onSvcUpdated = (model) => {
    if (model.loading || model.error) return; // проверяем, готов ли к работе се
```

```
  рвис и устанавливаем стейт из данных модели
    this.setState({
      data: model.data,
    });
  }

  public onSubmitClick = () => {
 // прим какого-то целевого действия, вызывающего метод, меняющий модель сервиса
      this._datePickerService?.setFilter(here_some_arguments);
  }

  public render() {
    const {data} = this.state;
    return (
      <div className="DatePickers">
        {/* что-то делаем с data */}
          <div className="DatePickers__SelectButtons">
            <div className="DatePickers__SelectButton active" onClick={this.onSubmitClick}>Применить</div>
          </div>
        </div>
      </div>
    );
  }
}
```

Таким образом компоненты на сервисах можно легко свести к банальному `View`, который только отображает данные, но почти ничего сам не считает и зависит только от настроек конфига дешлета. Всю работу и бизнес-логику на себя возьмет сервис.

12.11 CanIService и проверка прав

Пакет `"bi-internal/services"`, `Singleton` Инстанс получаем через `CanIService.getInstance()`

Сервис позволяющий накапливать, сохранять и получать claim'ы с сервера.

Модель реализует интерфейс:

```
import {
  IBaseModel
} from 'bi-internal/core';
type CanIModel = IBaseModel & Record<string, boolean>;
```

Используемый API:

`POST /api/can/i`

```

{
```

```
  url: AppConfig.fixRequestUrl(`/api/can/i`),
  method: 'POST',
  headers: {
   'Content-Type': 'application/json',
   'Accept': 'application/json',
  },
  data: claims, // массив строк claim типа ['L koob.cubes', 'C adm.topics']
}
```

Основные методы (у всех есть `static` версии):

```
/**
   * @method
   * @param {string} claim - единичный запрос типа "R adm.datasets/ds_xx".
   * @return {Promise<boolean>} - возвращает з-ние клейма.
   */
public one(claim: string): Promise<boolean> {
  return this.ensure([claim])
    .then(model => model[claim])
      .catch(err => false);
}

/**
   * @method
   * @param {string[]} claims - массив запрос типа ["R adm.datasets/ds_x1", "R adm.datasets/ds_x1"...]
   * @return {Promise<CanIModel>} - возвращает млодель типа {[claim:string]:boolean}.
   */
public ensure(claims: string[]): Promise<CanIModel> {
  const queueds: string[] = claims.filter(claim => this._queued[claim]);
 const pendings: string[] = claims.filter(claim => this._pending[claim]);

  claims = claims.filter(claim => this._model[claim] === undefined && !this._pending[claim] && !this._queued[claim]);

    if (!claims.length) {
                    // все клеймы уже были у нас
      if (queueds.length) {
                    // но среди них есть стоящие в очереди
        return this._queuedPromise;
      } else if (pendings.length) {
                    // есть те, которые сейчас загружаются
        return this._pendingPromise;
      } else {
                    // все готовы
        return Promise.resolve(this._model);
      }
    }

    // Надо какие-то загрузить
    claims.forEach(claim => this._queued[claim] = true);
```

```
                    // ставим их в очередь

    if (!this._queuedPromise) {
                    // если еще не было очереди, то создаем ее
      this._queuedPromise = new Promise<CanIModel>(resolve => {
  this._queueResolve = resolve; });   // и сохраняем функцию-резолвер
    }

    if (!this._currentRequest) {
                    // Ничего не запущено
      this._currentRequest = new Promise(resolve => resolve()).then(() =>
  this._run());             // запускаем в следующем тике
  }

 return this._queuedPromise;
}

/**
   * @method
   * @param {string} claim - проверяет зн-ие в модели (<CanIModel>) этот клейм т
  ипа "R adm.datasets/ds_x1"
   * @return {boolean}
   */
  public can = (claim: string): boolean => {
    return this.getModel()[claim];
  }
```

12.12 KoobDataService и работа с данными из кубов

Пакет `"bi-internal/services"`, `Factory` Инстанс получаем через `(new KoobDataService(koob, [], measures, filters)).whenReady()`, специального метода под него нет.

Сервис позволяющий запрашивать данные из куба.

Модель реализует интерфейс `IKoobDataModel`:

```
interface IKoobDimension {
  id: string;
  axisId?: string;
  formula?: string;
  source_ident?: string;
  cube_name?: string;
  name?: string;
  title: string;
  type: string | 'STRING';
  values?: any[];
  members?: any;
  config?: any;
  min?: number;
```

```
  max?: number;
  sql: string;
  key?: string;
  subtotal?: boolean;  // table
  upSubtotal?: boolean;  // table delete
  count?: number;
}

export interface IKoobMeasure {
  id: string;
                    // id для measure, например, x
  formula: string;
                    // формула, например, sum(x) или count(z):x
  axisId?: string;
  source_ident?: string;
  cube_name?: string;
  name?: string;
  title: string;
  format?: string;
  members?: any;
  type: string | 'SUM';
                    // SUM, COUNT, AVG, FN
  min?: number;
  max?: number;
  sql: string;
  key?: string;
  config?: any;
  unit?: any;
}

export interface IKoobDataModel {
  loading?: boolean;
  error?: string;
  dimensions: IKoobDimension[];
  measures: IKoobMeasure[];
  values: any[];
  sort?: string[];
  subtotals?: string[];
}
```

Конструктор содержит следующие аргументы:

```
{
  koobId: string, // полный идентификатор куба 'источник_данных.название_таблицы'
  dimensions: IKoobDimension[],
  measures: IKoobMeasure[],
  filters: any,
  loadBy?: number,
  sort?: any,
  subtotals?: string[] // подытоги по дименшнам
}
```

Может вам пригодиться, когда нужно получить все значения, объекты меж, дименшнов кон-

кретного куба.

Чаще всего будут использоваться методы данного сервиса и описанные там вспомогательные функции для запросов за данными:

Методы класса:

```
public setSort(sort: string | string[] | null) {
  /* установит сортировку по одному или нескольким полям ['+sex','-age'] - сортировка ASC по sex, DESC - по age */
}

public setFilter(filters: any) {
  /* установит фильтры на дименшны куба */
}
```

Вспомогательные функции:

```
export interface CancelToken {
  promise: Promise<Cancel>;
  reason?: Cancel;
  throwIfRequested(): void;
}

export interface IKoobDataRequest3 {
  options?: string[];
  offset?: number;
  limit?: number;
  sort?: string[];
  subtotals?: string[];
  cancelToken?: CancelToken;
  schema_name?: string;
}

/* С помощью данного метода можно получить данные так же, как их обычно получают дешлеты (только фильтры вам придется прокидывать самим через KoobFiltersService) */

export async function koobDataRequest3(koobId: string,
                                       dimensions: string[],
                                       measures: string[],
                                       allFilters: any,
                                       request: IKoobDataRequest3 = {},
                                       comment?: string) {
  const schema_name = request.schema_name || 'koob'; // запрос за данными локальными куба или глобального соответственно
  const url: string = AppConfig.fixRequestUrl(`/api/v3/${schema_name}/data` + (comment ? '?' + comment : ''));
  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];
      }
```

```
79      }
81      return data;
83    } catch (e) {
84      AlertsVC.getInstance().pushDangerAlert(extractErrorMessage(e));
85      return '';
86    }
88  }
90  /* С помощью данного метода можно получить данные так же, как их получает запрос
       /data, но у результирующей выборки берется COUNT и возвращается количество ст
      рок, например {count: 12} */
92  export async function koobCountRequest3(koobId: string,
93                                          dimensions: string[],
94                                          measures: string[],
95                                          allFilters: any,
96                                          request: IKoobDataRequest3 = {},
97                                          comment?: string) {
98    const schema_name = request.schema_name || 'koob';
99    const url: string = AppConfig.fixRequestUrl(`/api/v3/${schema_name}/count` +
      (comment ? '?' + comment : ''));
100   const columns = dimensions.concat(measures);
102   let filters: any;
103   if (allFilters && typeof allFilters === 'object') {
104     filters = {};
105     Object.keys(allFilters).forEach(key => {
106       let value = allFilters[key];
107       if (value === '∀' || value === '*') {
                        // фильтр подразумевающий 'ВСЕ'
108         return;
                        // просто не выносим в фильтры
109       } else if (Array.isArray(value) && value[0] === 'IN') {
                        // много где сконфигурировано ['IN', 'a', 'b']
110         value = ['='].concat(value.slice(1));
111       }
112       filters[key] = value;
113     });
114   }
116   const body: any = {
117     with: koobId,
118     columns,
119     filters,
120   };
122   if (request.offset) body.offset = request.offset;
123   if (request.limit) body.limit = request.limit;
124   if (request.sort) body.sort = request.sort;
125   if (request.options) body.options = request.options;
```

```
126   if (request.subtotals?.length) body.subtotals = request.subtotals;
128   if (!measures.length) {
                        // если нет measures, то лучше применить distinct
129     body.distinct = [];
130   }
132   try {
133     const response = await axios({
134       url,
135       method: 'POST',
136       headers: {
137         'Content-Type': 'application/json',
138         'Accept': 'application/stream+json',
139       },
140       data: body,
141       cancelToken: request.cancelToken,
142     });
144     let data = response.data;
146     if (String(response.headers['content-type']).startsWith('application/stream+
    json')) {
147       if (typeof data === 'string') {
148         data = data.split('\n').filter((line: string) => !!line).map((line:
    string) => JSON.parse(line));
149       } else if (data && (typeof data === 'object') && !Array.isArray(data)) {
150         data = [data];
151       }
152     }
154     return data;
155   } catch (error) {
156     console.error(error);
157     AlertsVC.getInstance().pushDangerAlert(extractErrorMessage(error));
158     return '';
159   }
161 }
```

13 Книга рецептов по кастомизации фронтенда Luxms BI

13.1 Разделение фронтенда Luxms BI на независимые секторы

Подразумевается, что вы решаете одну из следующих общих задач:

- Хотите разместить на одном инстансе Luxms BI несколько разных стендов, различающихся и по функционалу и по внешнему виду. Чтобы заходя на конкретный атлас вы получали поведение, определяемое его атласом-родителем
- Хотите объединить атласы так, чтобы разные пользователи с разными ролями видели разную итоговую картинку и функционал атласов. Причем значительно кастомизированный.

Веб-клиент версии 9 по умолчанию позволяет делать следующее:

- Ограничивать доступ к атласам в зависимости от прав пользователя и его сайтовой роли.
- Атласы способны содержать в ресурсах файлы, хранящие React-компоненты, переопределяющие строительные блоки приложения (например, окно логина, стартовая страница, заголовок раздела с дешбордами, левую панель на том же разделе и другое). Так же в его ресурсах могут храниться React-компоненты для тех кастомных визуализаций ,которые используются только в дешлетах данного атласа. То же самое касается темы и локализации

Однако, если речь заходит об обособлении набора некоторых атласов, т.е. о том, чтобы группа атласов вела себя и выглядела отлично от другой подобной группы - нужно использовать родительские атласы.

Почему так? Потому что по умолчанию, доступным для всех атласов сразу источником ресурсов, не требующим проверки на права доступа, являются ресурсы служебного атласа `ds_res`. Однако, кастомазация, достигаемая через `ds_res` будет распространяться на все атласы по умолчанию. А некоторые вещи разделить будет вообще нельзя. Потому нам нужно будет достигнуть наших задач через иерархию атласов и `ds_res` использовать более аккуратно, только для действительно общих файлов для атласов.

Родительский атлас - обычный атлас, который выступает лишь в качестве хранилища ресурсов для своих атласов-детей. Он даже может быть (и обычно так и делается) помечен как “выключенный” в списке атласов (индикатор в плашке горит серым), так как скорее всего даже не будет содержать ни одного дешборда.

При таком разделении лучше соблюдать правила именования имени схемы у атласов при создании, чтобы гарантировать непересечение `schema_name` атласов как в рамках текущего инстанса, так и в случае переносов в другие инстансы.

Правила формирования `schema_name` вариативны, пример неплохого стиля это `ds_кластер_наименованиепроекта_номератласа`. По умолчанию на платформе атлас создается с именем схемы вида `ds_14`, где `14` - идентификатор атласа в таблице атласов

Обычно родительский атлас имеет `schema_name`. содержащий индекс 0, например `ds_myinstance_0`. Соответственно, его "дети" будут иметь `schema_name` равный `ds_myinstance_1`, `ds_myinstance_2`, и т.д. Так видно, что атласы разделяют некое единое пространство имен.

Чтобы сделать какой-либо атлас **родительским** достаточно указать его значение поля `guid` в поле `parent_guid` у атласа, который соответсвенно должен выступать **дочерним**.

Пример `guid` : `18d935b4-06e9-4e59-b13e-f0cb3c2f35f0`.

Теперь мы можем хранить в ресурсах родительского атласа все необходимые компоненты, влияющие на отображение веб-клиента дочерних атласов в общем и кастомных дешлетов в частности. При этом детализировать поведение каждого атласа уже его локальными ресурсами.

13.2 Замена favicon

Для замены фавиконки загрузите в раздел ресурсов атласа файлы:

```
favicon.ico //достаточно и этого в базовом виде
apple-touch-icon.png
favicon-16x16.png
favicon-32x32.png
safari-pinned-tab.svg
```

Если вы загрузите этот набор в `ds_res` - это коснется фавиконки для всего инстанса. Если при этом еще и загрузите похожий набор в родительский атлас, то это изменит фавиконки атласов-детей. Если загрузите еще и в конкретный атлас - то применится только на нем.

13.3 Установка логотипа приложения

Блок с лого состоит из самого логотипа и названия приложения.

Рис. 13.1 **logo.png**

Для замены лого загрузите в раздел ресурсов аналогично фавиконке файлы (только один из них):

```
/* Эти файлы оба вставят лого, но разного формата */
thumbnail.png
thumbnail.svg //или logo.svg

/* Если есть этот файл, то в блоке с логотипом будет только данный логотип, а на
  звание проекта скроется */
thumbnailFull.svg
```

13.4 Установка прелоадера

Для замены лого прелоадера (анимированная иконка при первичной загрузке приложения) загрузите в раздел ресурсов атласа `ds_res` файл:

```
logo-animated.svg
```

13.5 Загружаем в атлас свои стили, шрифты

Для раздела `dashboards` атласа возможно подключить глобальные файлы стилей, поместив их в раздел `resources` текущего атласа, `ds_res` или родительских атласов.

Они подключаются на страницу при помощи тега `<link>`.

Общие для всех дешбордов текущего атласа стили зовутся `_global.css` или `styles.css` (последнее предположительно скоро будет `deprecated`).

Так же, учитывая ролевые особенности родительских атласов и `ds_res`, вы можете разместить файлы стилей `_global.css` в каждом из них и по итогу получить несколько подключений файлов стилей, по приоритету:

- сначала стили `ds_res` (если есть - будут подключаться к каждому атласу и его дешбордам)
- затем цепочка стилей родительских атласов (только для дочерних атласов и их дешбордов)
- стили текущего атласа (для всех дешбордов текущего атласа)
- стили для текущего дешборда текущего атласа будут дополнительными стилями. Такой файл для дешборда с `id = 1` будет называться `styles.dboard=1.css`

Чтобы добавить на страницу кастомный шрифт - загрузите в ресурсы атласа `ds_res` файлы веб-шрифта (`woff`,`woff2`,`ttf`,`otf`,`svg`) и в файле стилей, например `_global.css` укажите `font-face` для шрифта и примените шрифт к `html`, `body` или нужному вам селектору на странице

Пример:

```
@font-face {
  font-family: 'DIN Pro, Arial, sans-serif';
  font-style: normal;
  font-weight: lighter;
  src: local('DIN Pro'),
    url('/srv/resources/ds_res/fonts/dinpro_light.otf') format('opentype');
}

....

html, body {
   font-family: 'DIN Pro, Arial, sans-serif';
}
```

При написании стилей привязывайтесь к селекторам по человекопонятным классам, а не атомарным, так как последние имеют свойство меняться.

13.6 Работа с темами

В данный момент интерактивный раздел для кастомизации и тонкой настройки тем находится в разработке, однако вы можете менять и создавать темы при помощи создания файла `themes.json`.

Каждый ключ верхнего уровня в таком файле - идентификатор темы. По умолчанию в коробке Luxms BI две темы: светлая (`light`) и темная (`dark`).

Ключи внутри соответствующих тем используют в качестве значения какие-либо CSS-свойства или объекты, необходимые для конфигурации специфических разделов фронтенда (карты, графики на `Apache Echarts`).

В результате работы с конфигом тем веб-клиент создает `набор CSS-переменных` (их можно увидеть у тега `body` в инспекторе браузера). Потому, что именно хранить в значении свойства определяется допустимыми значениями, принимаемыми CSS-переменными.

13.6.1 Список базовых свойств, повторяющийся для всех тем

```
color1: "Базовый цвет 1";
color2: "Базовый цвет 2";
color3: "Базовый цвет 3";
color4: "Базовый цвет 4";
color5: "Базовый цвет 5";
color6: "Базовый цвет 6";
color7: "Базовый цвет 7";
color8: "Базовый цвет 8";
color9: "Базовый цвет 9";
```

```
color10: "Базовый цвет 10";
color11: "Базовый цвет 11";
color12: "Базовый цвет 12";
color13: "Базовый цвет 13";
color14: "Базовый цвет 14";
color15: "Базовый цвет 15";
color16: "Базовый цвет 16";
color17: "Базовый цвет 17";
color18: "Базовый цвет 18";
color19: "Базовый цвет 19";
color20: "Базовый цвет 20";
color_conditions: "Цвет условий";
system_panel: "Системный цвет (панели)";
system_background: "Системный цвет (бекграунд)";
system_effect: "Системный цвет (эффект)";
system_selection: "Системный цвет (Выделение)";
system_element: "Системный цвет (Элементы)";
neutral_300: "Нейтральный цвет (насыщенность 300)";
neutral_400: "Нейтральный цвет (насыщенность 400)";
neutral_500: "Нейтральный цвет (насыщенность 500)";
neutral_600: "Нейтральный цвет (насыщенность 600)";
neutral_700: "Нейтральный цвет (насыщенность 700)";
neutral_800: "Нейтральный цвет (насыщенность 800)";
neutral_900: "Нейтральный цвет (насыщенность 900)";
active_firstdefault: "Цвет заливки для основных кнопок";
active_firsthover: "Цвет заливки для оснвных кнопок (отображение эффекта наведен
  ия)";
active_firstcontrast: "Цвет для контрастных элементов на первостепенных кнопках
  (текст, иконки)";
active_secondcontrast: "Основной цвет для второстепенных элементов (кнопки без з
  аливки, кнопки с обводкой, ссылки)";
active_seconddefault: "Цвет заливки для второстепенных кнопок";
active_secondhover: "Цвет заливки для второстепенных кнопок (эффект наведения)";
state_success: "Цвет состояния 'Успешно выполнено'";
state_warning: "Цвет состояния 'Предупреждение'";
state_error: "Цвет состояния 'Ошибка'";
themeBuilder: "Объект, хранящий доп.настройки для коробочных графиков на ECharts"
```

Это те свойства, которые любая тема будет иметь по умолчанию. Значения зависят от темы, разумеется. Однако вы можете добавлять и свои свойства в этот набор.

Рассмотрим пример файла `themes.json`:

```
{
  "dark":{
    "color7": "rgba(151, 151, 196, 0.3)",
    "color8": "#0070BA",
    "color20": "#fff",
    "color19": "#212D3C",
    "color11": "rgba(255, 255, 255, 0.1)",
    "system_panel": "#273445",
    "system_background": "#273445",
    "system_effect": "#3D4958",
```

```
    "system_selection": "#212D3D",
    "system_element": "#8EA4CE",
    "neutral_300": "#F3F4F5",
    "neutral_400": "#D4D6DA",
    "neutral_500": "#A6ACB8",
    "neutral_600": "#828894",
    "neutral_700": "#707580",
    "neutral_800": "#505661",
    "neutral_900": "#3E444F",
    "themeBuilder":{
      "color":[
        "#ADD4DE",
        "#F6F6F6",
        "#F1D1D7",
        "#E88898",
        "#EDADB7",
        "#F1D1D7",
        "#F6F6F6",
        "#DAE5F1",
        "#BED4EC",
        "#A1C2E7",
        "#85B1E2"
      ]
    }
  },
  "light":{
    "color7": "rgba(151, 151, 196, 0.3)",
    "color8": "#0070BA",
    "color20": "#fff",
    "color19": "#F2F2F8",
    "color11": "rgba(151, 151, 196, 0.1)",
    "system_panel": "#FFFFFF",
    "system_background": "#878787",
    "system_effect": "#F6F6F9",
    "system_selection": "#E9E9F5",
    "system_element": "#9797C4",
    "neutral_300": "#363636",
    "neutral_400": "#4F4F4F",
    "neutral_500": "#686868",
    "neutral_600": "#828282",
    "neutral_700": "#9C9C9C",
    "neutral_800": "#CECECE",
    "neutral_900": "#E8E8E8",
    "themeBuilder":{
      "color":[
        "#54A2AE",
        "#FFF1E6",
        "#E36378",
        "#E88898",
        "#EDADB7",
        "#F1D1D7",
        "#F6F6F6",
        "#DAE5F1",
```

```
            "#BED4EC",
            "#A1C2E7",
            "#85B1E2"
        ]
    }
  }
}
```

Как вы видите, здесь не полный перечень свойств, указанный выше. Но это и не нужно. Файл `themes.json` мержится веб-клиентом с тем конфигом тем, который есть из коробки. Т.е. вам достаточно указать только те свойства, которые вы хотите изменить или добавить в конкретной теме, а остальные останутся как есть.

Вам лишь нужно добавить новые свойства, отсутствующие по умолчанию в наборе во все ваши темы. Чтобы при переключении темы в интерфейсе ваши React-компоненты, которые будут использовать эту переменную для стилизации не получили некорректное значение.

13.6.2 Общая настройка темы графиков у коробочных визуализаций

Уникальный ключ `themeBuilder` для темы хранит конфиг графика на библиотеке `Apache ECharts`, чей движок используется на платформе по умолчанию, за исключением узкого круга специфических визуализаций.

Этот блок будет автоматически наполняться в интерактивном разделе, но пока вы можете использовать для этого нативный раздел библиотеки тут:

https://echarts.apache.org/en/theme-builder.html

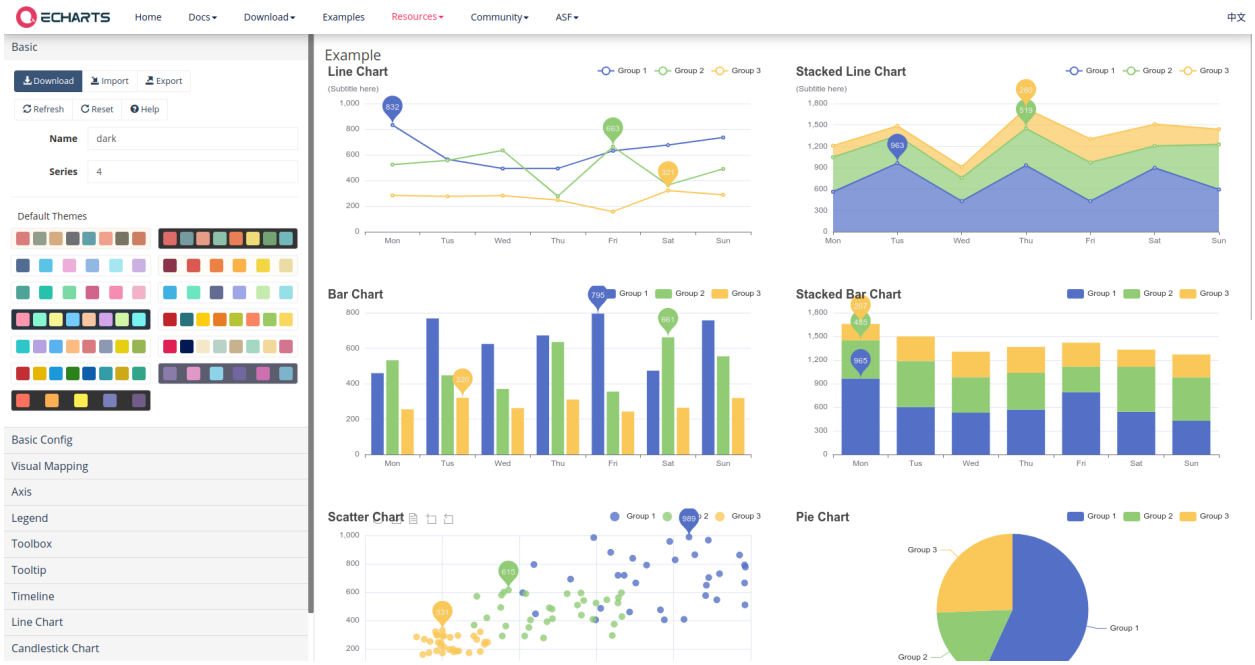


Рис. 13.2 **themebuilder.png**

Вам нужно будет провести нужные вам настройки для соответствующей темы и по окончании кликнуть по кнопке `Download`. Затем перейти в таб `JSON version`

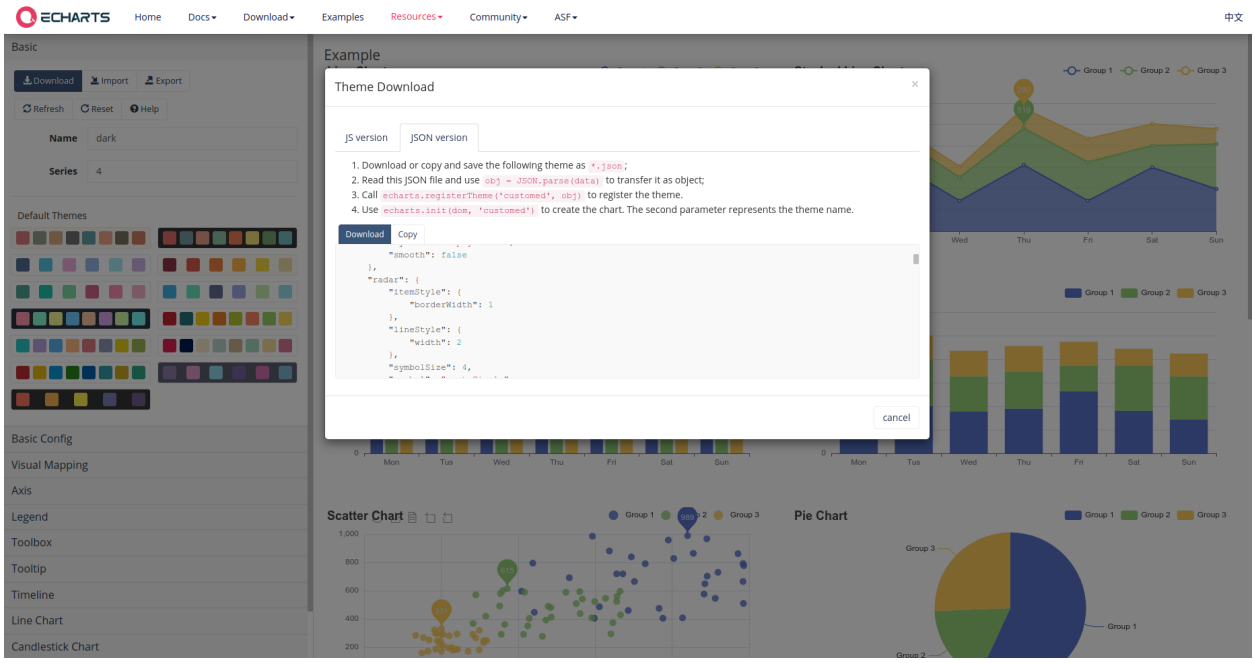


Рис. 13.3 **screenshot-echarts.apache.org-2023.05.04-16_01_08.png**

И нажать `Copy`. Вы скопируете готовый для вставки в качестве значения в `themeBuilder` объект.

Импорт такого файла не предусмотрен.

Такой готовый объект для вставки будет использован веб-клиентом с возможными ограничениями, связанными с настройками конкретного дешлета или заложенной логикой веб-клиента на конкретные свойства конкретных типов графиков, однако большая часть свойств будет применена.

На примере `themes.json` выше `themeBuilder` содержит ключ `color` с массивом hex-цветов. Этот массив определяет палитру цветов, которую по умолчанию станут использовать графики при включенной теме. Пока еще эта палитра не является замкнутой, то есть как только серий на графике станет больше чем число цветов палитры - для выходящих за рамки будут использоваться дефолтные цвета палитры веб-клиента.

А это вот такой список:

```
"#AA6FAC",
"#E85498",
"#4F4F9B",
"#4AB6E8",
"#E07921",
"#5FB138",
"#F05045",
"#F2BB05",
"#9797C4"
```

13.6.3 Создание новых и отключение существующих тем

Если указать напротив какого либа ключа темы `null`, то вы таким образом выключите ее. Если в результате остается всего одна тема - вы не увидите в тулбаре страницы `dashboards` иконки смены текущей темы.

```
{
 "light": null
}
```

Чтобы создать новую тему просто добавьте еще один ключ верхнего уровня и в базовом случае пустой объект

```
{
 "pink": {}
}
```

У вас создатся тема `pink`, которая по умолчанию будет наследовать базовые значения темы `light` из коробки (не того, что вы переопределили здесь, в файле `themes.json`!) и, памятуя об этом, переопределите те свойства, которые вам потребуются для уникализации новой темы, вплоть до полного переопределения всех свойств.

Иерархическая логика переопределения тем из ресурсов `ds_res`, родительского атласа(если есть) и к текущему атласу сохраняется. То есть на конкретном атласе могут быть темы, не присутствующие на остальных, или вообще единственная тема.

13.6.4 Сервис для работы с темами ThemeVC

`ThemeVC` - `Singleton` `observable` сервис, который предоставляет вам список существующих тем и их итоговых после всех мержей свойств, а также методы подписки на смену темы, методы эту смену осуществляющие

13.6.4.1 Модель

```
{
  error: null, // объект возможной ошибки
  loading: false, // индикатор готовности сервиса (true значит, что сервис в процессе загрузки модели)
  themes: {"light": {...}, "dark": {...}}, // доступные темы
  currentTheme: {...}, // объект со свойствами текущей темы
  currentThemeId: 'light', // идентификатор текущей темы
}
```

13.6.4.2 Метод setTheme для выставления темы

```
public setTheme(currentThemeId: string) {
/*
 данный метод установит вам тему нативным образом, сохранив выбранный ключ темы в localStorage вашего браузера. В будущем, возможно, это будет конфиг пользователя, чтобы тема сохранялась за пользователем вне зависимости от браузера.
*/
  let currentTheme = this._model.themes[currentThemeId];
    if (currentTheme) {
      localStorage.setItem('theme', currentThemeId);
      this._applyThemeVariables(currentTheme, currentThemeId);

      // Данный метод уведомит всех подписчиков сервиса о смене темы
      this._updateModel({
        currentTheme,
        currentThemeId
      });
    }
}
```

13.6.4.3 Подписка и программное изменение темы в React-компонентах

В классовом

```
import React from 'react';
import {ThemeVC} from 'bi-internal/services';

class MyComponent extends React.Component {
  private _themeVC: ThemeVC;
 public state: {
    currentTheme: any;
    currentThemeId: string;
  };

  public constructor(props) {
   super(props);
    this.state = {
     currentTheme: null;
      currentThemeId: "light"
    }
  }

  public componentDidMount() {
    this._themeVC = ThemeVC.getInstance();
  this._themeVC.subscribeUpdatesAndNotify(this._onThemeUpdated)
  }

  private _onThemeUpdated = (model) => {
   if (model.loading || model.error) return;
```

```
    this.setState({
      currentTheme: model.currentTheme,
      currentThemeId: model.currentThemeId
    });
  }

 private _onChangeTheme = () => {
    const {currentThemeId} = this.state;
    this._themeVC.setTheme(currentThemeId === 'light' ? 'dark' : 'light');
  }

  public render() {
    const {currentTheme} = this.state;

   return (
      <>
      {currentTheme &&
       <div className="ThemeChangerDemo">
          <div style={{color: currentTheme.color1}}>Привет мир!</div>}
          <button onClick={onChangeTheme}>Сменить тему</button>
      </div>
   }
      </>
    );
  }
}

export default MyComponent;
```

В функциональном

```
import React, {useState} from 'react';
import {useServiceItself, useService, ThemeVC} from 'bi-internal/services;

const MyComponent = (props) => {
  const themeVC = useServiceItself<ThemeVC>(ThemeVC);

 const onChangeTheme = () => {
  const currentThemeId = themeVC.getModel().currentThemeId;
    themeVC.setTheme(currentThemeId === 'light' ? 'dark' : 'light');
  }

  if (themeVC.getModel().loading || themeVC.getModel().error) return null;

  return (
   <div className="ThemeChangerDemo">
     <div style={{color: themeVC.getModel().currentTheme.color1}}>Привет мир!</div>
   <button onClick={onChangeTheme}>Сменить тему</button>
    </div>
  )
```

```
}

export default MyComponent;
```

В данных компонентах я меняю свойство `color` текста засчет значения переменной темы `color1` (я мог выбрать любую переменную) и методом `setTheme` делаю переключение для темы со светлой на темную и обратно.

Хук `useServiceItself` вернет мне инстанс сервиса. Таким образом я могу получить и модель и использовать методы.

Если бы я писал функциональный компонент, который только получал бы значение `color1` и устанавливал его без возможности сменить в интерфейсе этого же компонента, то я бы мог использовать хук `useService`, который вернул бы мне не инстанс сервиса, а только лишь значение его модели (`themeVC.getModel()`). Тогда методы сервиса мне были бы уже недоступны.

13.6.4.4 Использование темы при стилизации

Лучше всего будет завести специальный файл типа `vars.scss` в котором хранить все переиспользуемые CSS-переменные, в том числе переменные отвечающие за тему.

```
$color1: var(--color1);
$color2: var(--color2);
$color3: var(--color3);
$color4: var(--color4);
$color5: var(--color5);
$color6: var(--color6);
$color7: var(--color7);
$color8: var(--color8);
$color9: var(--color9);
$color10: var(--color10);
$color11: var(--color11);
$color12: var(--color12);
$color13: var(--color13);
$color14: var(--color14);
$color15: var(--color15);
$color16: var(--color16);
$color17: var(--color17);
$color18: var(--color18);
$color19: var(--color19);
$color20: var(--color20);
$system_panel:  var(--system_panel);
$system_background:  var(--system_background);
$system_effect:  var(--system_effect);
$system_selection: var(--system_selection);
$system_element:  var(--system_element);
$neutral_300:  var(--neutral_300);
$neutral_400:  var(--neutral_400);
$neutral_500:  var(--neutral_500);
```

```
$neutral_600:  var(--neutral_600);
$neutral_700:  var(--neutral_700);
$neutral_800:  var(--neutral_800);
$neutral_900:  var(--neutral_900);
$active_firstdefault:  var(--active_firstdefault);
$active_firsthover:  var(--active_firsthover);
$active_firstcontrast: var(--active_firstcontrast);
$active_secondcontrast:  var(--active_secondcontrast);
$active_seconddefault:  var(--active_seconddefault);
$active_secondhover:  var(--active_secondhover);
$state_success:  var(--state_success);
$state_warning: var(--state_warning);
$state_error:  var(--state_error);
```

А в файле стилей вашего компонента использовать их так:

```
// Подключили переменные темы
@import "./vars.scss";

.MyComponent {
  width: 100%;
  height: 100%;
  position: relative;
  background-color: $system_background;

  &__graphic {
    position:absolute;
    z-index: 0;
    width: 100%;
    height: 100%;
  }
  &__onClickText {
    position:absolute;
    z-index: 1;
    left: 0;
    right: 0;
    top: 1.5rem;
    text-align: center;
    color: $color1;
  }
}
```

13.7 Работа с локализацией

Язык локализации по умолчанию определяется своим ключом из файла `settings.js` и наличием файла `.json` с кратким ключом выбранного языка в обвязке (по умолчанию, в обвязке есть только `en.json` и `ru.json` для английского и русского языков соответственно).

Расширение языкового набора возможно, если вы разместите в ресурсах атласа `ds_res` папку `lux-store/locales/` и поместите туда переведенную версию одного из существующих файлов локализации.

13.7.1 Пример файла локализации:

Взять один из сущетвующих файлов локализации вы можете по адресу `имя_вашего_сайта/assets/locales/ru.json` или `имя_вашего_сайта/assets/locales/en.json`.

```
{
  "LuxmsBI": "Luxms BI",
  "home": "Главная",
  "back": "Назад",
  "close": "Закрыть",
  "collapse": "Скрыть",
  "add": "Добавить",
  "Between": "Между",
  "More": "Больше",
  "Less": "Меньше",
  "contacts": "Контакты",
  "header": "Заголовок",
  "Header": "Заголовок",
  "create": "Создать",
  "Create": "Создать",
  "copy": "Копировать",
  "docs": "Документация",
  "error": "Что-то пошло не так...",
  "expand": "Подробнее",
  "log_in": "Войти",
  "log_out": "Выйти",
  "Log out": "Выйти",
  "log_in_wrong": "Неверный логин или пароль",
  "code_wrong": "Неверный код",
  "log_in_too_much_requests": "Вы превысили количество возможных запросов.",
  "login": "Логин",
  "Login": "Логин",
  "no": "Нет",
  "No": "Нет",
  "norm": "Границы",
  "ok": "ОК",
  "cancel": "Отмена",
  "open": "Открыть",
  "password": "Пароль",
  "Password": "Пароль",
  "Password once again": "Пароль ещё раз",
  "password_policy": "Парольная политика",
  "Password policy": "Парольная политика",
  "Change password after": "Сменить пароль после",
  "print": "Печать",
  "rest": "Остальные",
  "check": "Проверить",
  "result": "Результат",
  "Results": "Результаты",
  "resources": "Ресурсы",
  "save": "Сохранить",
  "Save": "Сохранить",
```

```
  "save-": "Сохранить",
  "save_as": "Сохранить как",
  "save-png": "Сохранить как png",
  "save-arrow": "Сохранить arrow",
  "save-xls": "Сохранить как xls",
  "save-parquet": "Сохранить parquet",
  "save-csv": "Сохранить как csv",
  "save-pdf": "Сохранить как pdf",
  "switch_theme": "Переключить тему",
  "Save changes": "Сохранить изменения",
  "search": "Искать",
 // много-много ключей
}
```

13.7.2 Добавление нового языка

Для добавления, например, французского языка вы должны поместить в раздел ресурсов атласа в иерархии `ds_res` => `родительский атлас` => `текущий атлас` файл `fr.json`, загрузив его на сервер или средствами проекта `BMR` или вручную, через drag'n'drop, и изменив его название на полный путь до папки выше: `lux-store/locales/fr.json`.

Т.е. по итогу у вас должен существовать файл по адресу

`имя_вашего_сайта/srv/resources/имя_атласа_в_иерерахии/lux-store/locales/fr.json`

Допустим поместили вы файлы переводов в `ds_res`, тогда ваши файлы будут тут

`/srv/resources/ds_res/lux-store/locales/fr.json`

Для существующих языков вроде `ru` и `en` вы можете создать аналогичным образом файл `.json`, но уже размесить туда не полный перевод, а только те ключи и их значения ,которые хотите поменять.

Тогда клиент, найдя одноименные файлы в ресурсах, просто смержит их между собой.

13.7.3 Замена строк в существующей локализации

Допустим , вы хотите поменять слово `дешборд` на `интерактивная панель визуализаций`.

Открываете файл исходных данных `имя_вашего_сайта/assets/locales/ru.json`.

Создаете в ресурсах устраивающего вас атласа файл `lux-store/locales/ru.json`

Поиском по тексту в исходном перевода ищете слово `дешборд` и копируете все ключи в файл с переводом, который вы разместили в ресурсах.

И в нем пишете:

```
{
  "go_to_dashboards": "Перейти к интерактивным панелям визуализаций",
  "startup_dashboard": "Стартовая интерактивная панель визуализаций",
  "dashboard": "Интерактивная панель визуализаций",
  "dashboards": "Интерактивные панели визуализаций",
  "hideDashboard": "Скрыть интерактивную панель визуализаций",
  "header_Dashboard": "Формат строки контекста для экрана интерактивной панели в
  изуализаций",
  "new_dashboard": "Новая интерактивная панель визуализаций"
}
```

и так для всего, что хотите поменять в локализации. Напоминаю, что полный перечень ключей вам тут не нужен. Это будет необходимо, только если вы создаете ранее несуществующий в коробке перевод.

13.7.4 Компонент для локализации L10n

Является компонентом-оберткой на вашим React-компонентом при разработке в рамках проекта `bi-magic-resources`.

Подразумевается, что все служебные тексты вы пишете на английском, а их перевод храните в файлах типа `ru.json` и подобным.

Можно оборачивать как конкретный ключ локализации, так и весь компонент. Заменяются текстовые блоки, аттрибуты title, description и placeholder.

Слишком высоко поднимать такой компонент тоже не стоит. Иначе можете получить перевод данных и идентификаторов кубов.

Пример использования:

```
import React, { useEffect, useState } from "react";
import {L10n, L10nVC} from 'bi-internal/ui';
import MySomeCustomComponent from './MySomeCustomComponent';

const MyComponent = (props) => {
  return (
   <React.Fragment>
      <button className="MyComponentButton">
     <L10n>go_to_dashboards</L10n>
      </button>
      <L10n><div title="some_key"></div></L10n>
      <L10n>
       <MySomeCustomComponent {...props}/>
      </L10n>
    </React.Fragment>
  );
}
```

```
export default MyComponent;
```

13.7.5 Сервис для работы с локализацией L10nVC

`L10nVC` - `Singleton` `observable` сервис, который предоставляет вам список существующих локализаций и их итоговых после всех мержей свойств, а также методы подписки на смену локализации, методы эту смену осуществляющие

13.7.5.1 Модель

```
{
  error: null, // объект возможной ошибки
  loading: false, // индикатор готовности сервиса (true значит, что сервис в процессе загрузки модели)
  locales: {ru: {...}, en:{...}}, // доступные локализации
  name: 'ru', // идентификатор локали
  current: {LuxmsBI: 'Luxms BI', home: 'Главная', back: 'Назад',...} // доступные ключи текущей локализации
}
```

13.7.5.2 Методы setLocale, setLocaleAndSave для выставления локализации

```
public setLocale(name: string) {
/*
 данный метод установит вам выбранную локализацию. Но после перезагрузки страницы вернется дефолтная локаль из settings.js
*/

}
```

```
public setLocaleAndSave(name: string) {
/*
 данный метод установит выбранную локализацию в localStorage и вернет результат setLocale
*/

}
```

```
import {lang} from 'bi-internal/utils';

function lang(key: string, defaultValue?: string) {
```

```
/*
 данный метод вернет defaultValue, если сервис локализации не загружен или содержит ошибку, переведенное значение ключа в рамках текущей локали и если такого ключа нет - вернет его.

  Кроме того key может быть и массивом строк, тогда будет переведен каждый ключ массива по правилам выше и сконкатенирован через символ ''
*/

}

const example = {
 title: lang('Data source'),
  //...
}
```

13.7.5.3 Подписка и программное изменение локализации в React-компонентах

Подписка на изменение локализации выглядит аналогично тому, как это выглядит при работе с темами.

Приведем пример только функционального компонента

```
import React, {useState} from 'react';
import {useServiceItself, useService} from 'bi-internal/services';
import {L10n, L10nVC} from 'bi-internal/ui';

const MyComponent = (props) => {
  const l10nVC = useServiceItself<L10nVC>(L10nVC);

 const onChangeLocale = () => {
  const name = l10nVC.getModel().name;
    l10nVC.setLocaleAndSave(name === 'ru' ? 'en' : 'ru');
  }

  if (l10nVC.getModel().loading || l10nVC.getModel().error) return null;

  // "Click or drag and drop to upload" - это ключ локализации, который имеет соответственно
  // два перевода для двух существующих локализаций
  return (
   <div className="LocaleChangerDemo">
     <div><L10n>Click or drag and drop to upload</L10n></div>
   <button onClick={onChangeLocale}>Сменить язык</button>
    </div>
  )
}

export default MyComponent;
```

13.8 Работа с иконками

13.8.1 Спрайт с доступными иконками

Для того, чтобы увидеть все доступные иконки октройте консоль браузера и наберите `shell.showSprite()`.

Вы увидите попап со списком иконок

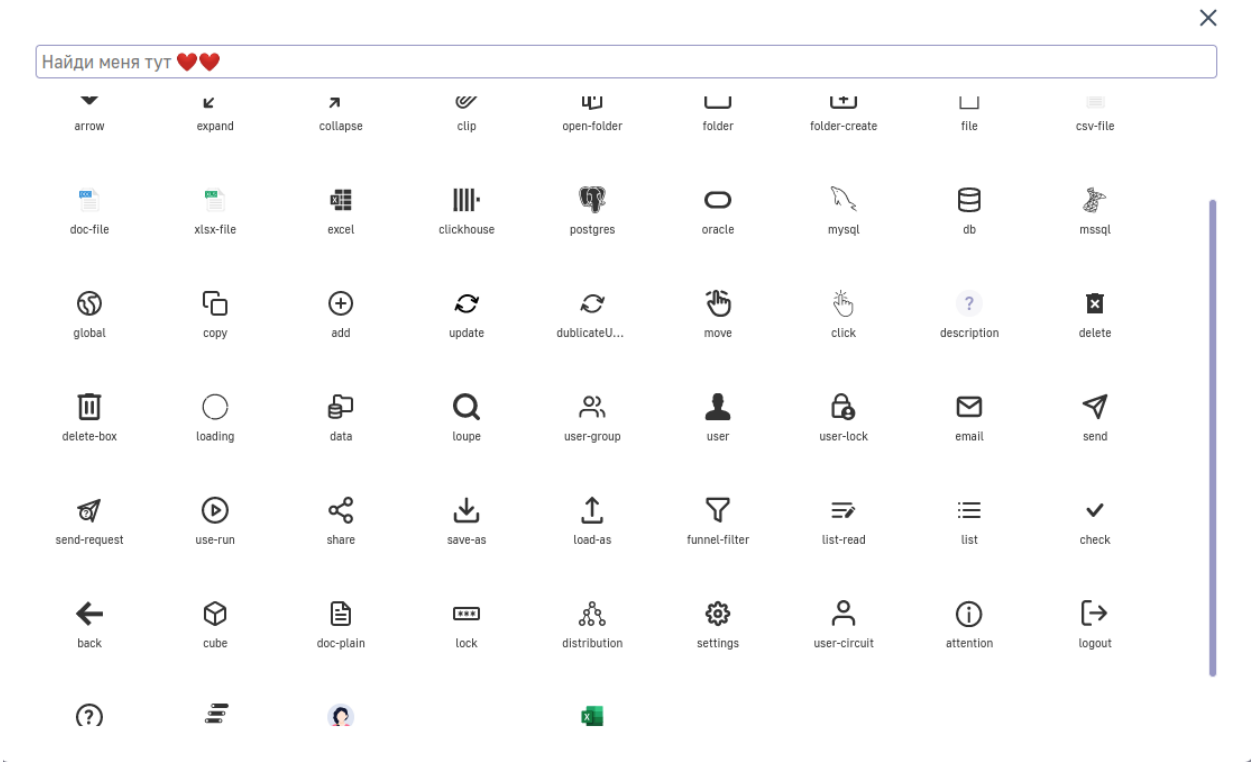


Рис. 13.4 **sprite.png**

Подписи под каждой иконкой есть идентификаторы которые вам будет нужно указать в компоненте для отрисовки иконок `SVGIcon`.

Это классический `svg`-спрайт, где каждая иконка - `symbol`

```
<svg xmlns="http://www.w3.org/2000/svg">
  <symbol viewBox="0 0 20 18" id="atlas">
    <path fill-rule="evenodd" clip-rule="evenodd" fill="currentColor"
          d="M1.6665 0.5C1.11422 0.5 0.666504 0.947715 0.666504 1.5V14C0.666504
  14.5523 1.11422 15 1.6665 15H7.49984C7.89766 15 8.27919 15.158 8.5605
  15.4393C8.8418 15.7206 8.99984 16.1022 8.99984 16.5C8.99984 17.0523 9.44755
  17.5 9.99984 17.5C9.99989 17.5 9.99995 17.5 10 17.5C10.5523 17.5 11 17.0523
  11 16.5C11 16.1022 11.158 15.7206 11.4393 15.4393C11.7206 15.158 12.1022 15
  12.5 15H18.3333C18.8856 15 19.3333 14.5523 19.3333 14V1.5C19.3333 0.947715
  18.8856 0.5 18.3333 0.5H13.3333C12.1841 0.5 11.0819 0.956546 10.2692
  1.7692C10.1746 1.86385 10.0847 1.96242 9.99992 2.06454C9.91509 1.96242
  9.82528 1.86385 9.73063 1.7692C8.91798 0.956546 7.81577 0.5 6.6665
```

```
  0.5H1.6665ZM8.99984 4.83333V13.3377C8.53541 13.1174 8.02368 13 7.49984
  13H2.6665V2.5H6.6665C7.28534 2.5 7.87884 2.74583 8.31642 3.18342C8.754 3.621
  8.99984 4.21449 8.99984 4.83333ZM12.5 13C11.9762 13 11.4644 13.1174 11
  13.3377V4.83333C11 4.21449 11.2458 3.621 11.6834 3.18342C12.121 2.74583
  12.7145 2.5 13.3333 2.5H17.3333V13H12.5Z"
    />
  </symbol>
  <symbol viewBox="0 0 24 24" id="book">
    <path fill-rule="evenodd" clip-rule="evenodd"
          d="M2 2C1.44772 2 1 2.44772 1 3V18C1 18.5523 1.44772 19 2
  19H9C9.53043 19 10.0391 19.2107 10.4142 19.5858C10.7893 19.9609 11 20.4696 11
  21C11 21.5523 11.4477 22 12 22C12.5523 22 13 21.5523 13 21C13 20.4696 13.2107
  19.9609 13.5858 19.5858C13.9609 19.2107 14.4696 19 15 19H22C22.5523 19 23
  18.5523 23 18V3C23 2.44772 22.5523 2 22 2H16C14.6739 2 13.4021 2.52678
  12.4645 3.46447C12.2962 3.63275 12.1411 3.81178 12 3.99997C11.8589 3.81178
  11.7038 3.63275 11.5355 3.46447C10.5979 2.52678 9.32608 2 8 2H2ZM13
  17.5359C13.6029 17.1878 14.2918 17 15 17H21V4H16C15.2044 4 14.4413 4.31607
  13.8787 4.87868C13.3161 5.44129 13 6.20435 13 7V17.5359ZM11 17.5359V7C11
  6.20435 10.6839 5.44129 10.1213 4.87868C9.55871 4.31607 8.79565 4 8
  4H3V17H9C9.70823 17 10.3971 17.1878 11 17.5359ZM4 9C4 8.44772 4.44772 8 5
  8H9C9.55228 8 10 8.44772 10 9C10 9.55228 9.55228 10 9 10H5C4.44772 10 4
  9.55228 4 9ZM15 8C14.4477 8 14 8.44772 14 9C14 9.55228 14.4477 10 15
  10H19C19.5523 10 20 9.55228 20 9C20 8.44772 19.5523 8 19 8H15ZM4 13C4 12.4477
  4.44772 12 5 12H9C9.55228 12 10 12.4477 10 13C10 13.5523 9.55228 14 9
  14H5C4.44772 14 4 13.5523 4 13ZM15 12C14.4477 12 14 12.4477 14 13C14 13.5523
  14.4477 14 15 14H19C19.5523 14 20 13.5523 20 13C20 12.4477 19.5523 12 19 12H15Z"
          fill="currentColor"/>
  </symbol>
  <symbol viewBox="0 0 20 20" id="dashboard">
    <path fill-rule="evenodd" clip-rule="evenodd" fill="currentColor"
          d="M11 2V6L18 6V2L11 2ZM10 4.17233e-07L19 0C19.2652 0 19.5196
  0.105357 19.7071 0.292893C19.8946 0.48043 20 0.734784 20 1V7C20 7.55228
  19.5523 8 19 8H10C9.73478 8 9.48043 7.89464 9.29289 7.70711C9.10536 7.51957 9
  7.26522 9 7V1C9 0.447716 9.44771 4.17233e-07 10 4.17233e-07ZM2 2L2 6L5 6L5
  2L2 2ZM1 7.7486e-07L6 5.96046e-07C6.55228 5.36442e-07 7 0.447716 7 1L7 7C7
  7.55229 6.55229 8 6 8L1 8C0.734784 8 0.48043 7.89464 0.292894
  7.70711C0.105357 7.51957 2.98023e-07 7.26522 2.38419e-07 7L0 1C0 0.447716
  0.447715 8.34465e-07 1 7.7486e-07ZM2 12L2 18H8L8 12L2 12ZM1 10L9 10C9.26522
  10 9.51957 10.1054 9.70711 10.2929C9.89464 10.4804 10 10.7348 10 11L10 19C10
  19.2652 9.89464 19.5196 9.70711 19.7071C9.51957 19.8946 9.26522 20 9
  20H1C0.447716 20 8.34465e-07 19.5523 7.7486e-07 19L4.17233e-07 11C4.17233e-07
  10.4477 0.447716 10 1 10ZM12.2929 10.2929C12.4804 10.1054 12.7348 10 13
  10H19C19.5523 10 20 10.4477 20 11V19C20 19.5523 19.5523 20 19 20H13C12.4477
  20 12 19.5523 12 19L12 11C12 10.7348 12.1054 10.4804 12.2929 10.2929ZM14
  12L14 18H18V12H14Z"
    />
  </symbol>
  <symbol viewBox="0 0 17 15" id="edit-pen">
    <path fill-rule="evenodd" clip-rule="evenodd"
          d="M13.627 1.57895C13.3455 1.57895 13.0848 1.68197 12.8999
  1.85156L2.38704 11.4956L1.93876 13.1405L3.83095 12.7066L14.3541
  3.0531C14.4451 2.96961 14.5145 2.87313 14.5608 2.77052C14.6071 2.66805
  14.6301 2.56008 14.6301 2.45233C14.6301 2.34458 14.6071 2.23661 14.5608
```

```
  2.13414C14.5145 2.03153 14.4451 1.93505 14.3541 1.85156C14.263 1.76797
  14.1521 1.6993 14.0268 1.65166C13.9013 1.604 13.7654 1.57895 13.627
  1.57895ZM11.7906 0.701469C12.2856 0.247358 12.947 0 13.627 0C13.9647 0
  14.3001 0.0609677 14.6146 0.180443C14.929 0.299938 15.2179 0.476237 15.4634
  0.701469C15.709 0.926795 15.9067 1.19694 16.0425 1.49782C16.1785 1.79885
  16.2492 2.12334 16.2492 2.45233C16.2492 2.78132 16.1785 3.10581 16.0425
  3.40684C15.9067 3.70773 15.709 3.97787 15.4634 4.20319L4.78218
  14.0017C4.67862 14.0967 4.55124 14.1634 4.41284 14.1951L0.994855
  14.979C0.717941 15.0425 0.427169 14.9596 0.228937 14.7607C0.0307057 14.5617 -
  0.0458907 14.2758 0.0271763 14.0077L0.881673 10.8722C0.920474 10.7298 0.99935
  10.6009 1.10938 10.5L11.7906 0.701469ZM7.69049 14.2105C7.69049 13.7745
  8.05292 13.421 8.50001 13.421H16.1905C16.6376 13.421 17 13.7745 17 14.2105C17
  14.6465 16.6376 15 16.1905 15H8.50001C8.05292 15 7.69049 14.6465 7.69049
  14.2105Z"
          fill="currentColor"/>
    <!-- И многие другие -->
  </symbol>
```

Со спрайтом по умолчанию можно ознакомиться по адресу `site.ru/assets/icons/sprite.svg` или набрать в консоли `shell.showSprite()` и открыть раздел `Network` браузера. В появившемся запросе за спрайтом будет актуальный его контент.

Чтобы изменить спрайт на свой или заменить лишь некоторые иконки (здесь тоже есть мерж с оригинальным файлом, потому достаточно указать один-два интересующих вас символа спрайта) создаем файл `svg`, схожий с примером выше и помещаем его в раздел ресурсов нужного вам по иерархии атласа, например в `ds_res`.

Обратите внимание на значение поля `fill` у `path`. `currentColor` означает, что иконка будет брать в качестве цвета значение свойства `color` у родительского элемента по правилам `CSS`.

13.8.2 Компонент для отрисовки иконок SVGIcon

`SVGIcon` - `React`- компонент, который может рисовать иконки при помощи различных источников `svg`-контента.

Пример:

```
import React from 'react';
import {SVGIcon} from 'bi-internal/ui';

const MyComponent = (props) => {
  return (
   <div className="IconDemo">
     <SVGIcon
       style={{position: 'absolute', display: 'none'}}
       className="SVGSprite"
        path="#atlas"
      />
```

```
    </div>
  )
}

export default MyComponent;
```

Ключевым `props` компонента является `path`. Он может принимать:

- `#atlas` - ссылка на элемент спрайта
- Настоящий svg вроде `<svg><path d=""></svg>`
- ссылку на файл `.svg`

и по итогу рисует иконку как послушный зайка.

13.9 Написание кастомных React-компонентов

13.9.1 React-компонент на Apache ECharts, фильтрующий данные по клику, следящий за ресайзом окна дешборда

Рассмотрим конфиг дешлета для которого этот компонент используется:

```
{
  url: 'res:MyComponent.js',
  frame: {
    h: 8,
    w: 12,
    x: 0,
    y: 0,
  },
  dataSource: {
    koob: 'BeerDataSource.beerKoob',
    style: {
      measures: {
        count_units: {
          color: 'red',
        },
        count_quantity: {
          color: 'green',
        },
      },
    },
    xAxis: 'category',
    yAxis: 'measures',
    measures: [
      'count(units):count_units',
      'count(quantity):count_quantity',
    ],
    dimensions: [
```

```
      'category',
    ],
  },
  onClickDataPoint: "lpe:setKoobFilter('','category',['=',category])",
  view_class: 'internal',
  title: 'Test',
}
```

Код самого компонента:

```
import React, {useEffect, useRef, useState} from "react";
import  * as echarts from 'echarts';

/*
KoobFiltersService - Observable сервис, управляющий фильтрами для кубов (по умолчанию его использует упр.дешлет)

useService, useServiceItself - специальные хуки для получения только модели или всего инстанса какого-либо observable сервиса. Принимает по умолчанию класс нужного сервиса и если это не singleton то еще и идентификатор (через запятую)
*/

import {KoobFiltersService, useService, useServiceItself} from 'bi-internal/services';

// специальный класс-наблюдатель за размерами элемента, благодаря ему будем ресайзить график
import {GeometryObserver} from 'bi-internal/face';

import './MyComponent.scss';

const MyComponent = (props) => {
  const { cfg, subspace, dp } = props;

  // для наглядности покажем на какую точку (Y,X) кликнули
  const [clickedPointName, setClickedPointName] = useState<string>("");

  // Получили инстанс сервиса фильтров
  // через метод koobFiltersService.getModel() можем получить его модель,
  // а через метод koobFiltersService.setFilter(koobId, dimensionId, valueArray) // ("", "category", ["=", "Beer"])
  // можем фильтровать данные дешлетов, которые подписаны в своих блоках filters на изменение этого дименшна (содержат "category": true)
  const koobFiltersService = useServiceItself<KoobFiltersService>(KoobFiltersService);

  // Храним реф-ссылку на контейнер для графика, сам инстанс Echarts и опции, которые ему передаем
  let containerRef = useRef(null);
  let chart = null;
  let options = {};
```

```
35    // Обрабатываем клик по точке графика. Проверяем, есть ли в конфиге дешлета св
      ойство onClickDataPoint, отвечающая
36    // в коробке за логику клика на точку по умолчанию
37    // если нет - просто реализована произвольная логика выставления текущего знач
      ения дименшна в фильтр + для наглядности
38    // показываем в интерфейсе строчку с "координатами" кликнутой точки

40    const onChartClick = (params): void => {
41  // о том, что входит в params можно подглядеть тут https:
      //echarts.apache.org/en/api.html#events.Mouse%20events
42      if (cfg.getRaw().hasOwnProperty('onClickDataPoint')) {
43        // Формируем объект информации о точке для встроенного контроллера обработ
      ки клика по точке
44        const vcpv = {m: undefined, l: undefined, p: undefined, z: undefined, y:
      params.data.y, x: params.data.x, v: params.value};
45        cfg.controller.handleVCPClick(params.event, vcpv)
46      } else {
47        const koobFiltersModel = koobFiltersService.getModel();
48        if (koobFiltersModel.loading || koobFiltersModel.error) return;
49        koobFiltersService.setFilter('', params.data.x.axisIds[0], ["=",
      params.name]);
50      }
51      setClickedPointName(`${params.data.y.title} ${params.data.x.title}`);
52    }
53    const resize = () => {
54      if (chart) {
55        chart.resize();
56      }
57    }
58    const renderChart = (data) => {
59      // На инит рефа создаем с нуля или обновляем существующий инстанс Echarts и
      подаем ему опции на вход
60      // конфигурацию графиков Echarts смотрите тут https:
      //echarts.apache.org/en/option.html#title
61      if (containerRef.current && data.length) {
62        if (!chart) {
63          chart = echarts.init(containerRef.current, null, {renderer: 'svg'});
64          GeometryObserver.getInstance().addSubscription(containerRef.current,
      resize);
65        }
66        options = {
67          title: {
68            show: false
69          },
70          tooltip: {
71            trigger: 'item',
72            appendToBody: true,
73            show: true
74          },
75          xAxis: {
76            type: 'category',
77            data: subspace.xs.map(x => x.title)
78          },
```

```
      yAxis: {
        type: 'value'
      },
      series: subspace.ys.map((y, yIndex) => ({
        data: subspace.xs.map((x, xIndex) => ({
          name: x.title,
          itemStyle: {
            // контроллер, который получает информацию о цвете автоматически, исходя из контекста
            color: cfg.getColor(y, null, yIndex),
          },
          x,
          y,
          value:  data[yIndex][xIndex] // мы получали матрицу YX
        })),
        name: y.title,
        type: 'bar', // я задал этот тип явно, но это можно прочитать из конфига дешлета
                     //как переменную cfg.getRaw().chartType например
        showBackground: true,
      })),
      legend: {
        show: true,
        data: subspace.ys.map((y, yIndex) => ({
          name: y.title,
          icon: 'circle',
          itemStyle: {
            // контроллер, который получает информацию о цвете автоматически, исходя из контекста
            color: cfg.getColor(y, null, yIndex),
          },
        }))
      },
    };
    chart.setOption(options);
    chart.resize(); // принудительно заставляем расшириться на весь контейнер
    chart.on('click', 'series', onChartClick); // Обрабатываем клик по серии, если нужно
  }
}

useEffect(() => {
  // Получаем полное декартово произведение для указанного конфига в дешлете
  // ожидаем матрицу [subspace.ys.length][subspace.xs.length]

  dp.getMatrixYX(subspace).then(dataArr => {
    renderChart(dataArr);
  });
  return () => {
    GeometryObserver.getInstance().removeSubscription(containerRef.current, resize);
  }
},[cfg, subspace]);
```

```
  return (
    <div className="MyComponent">
      {clickedPointName != "" && <div className="MyComponent__onClickText">Вы кликнули на {clickedPointName}</div>}
      <div ref={containerRef} className="MyComponent__graphic"></div>
    </div>
  );
}
export default MyComponent;
```

Стили:

```
// Подключили переменные темы
@import "./vars.scss";

.MyComponent {
  width: 100%;
  height: 100%;
  position: relative;

  &__graphic {
    position:absolute;
    z-index: 0;
    width: 100%;
    height: 100%;
  }
  &__onClickText {
    position:absolute;
    z-index: 1;
    left: 0;
    right: 0;
    top: 1.5rem;
    text-align: center;
    color: $color1;
  }
}
```

Этот компонент строит по указанным из конфига данным график типа `bar` по правилам Apache ECharts. Умеет обрабатывать клик по точке графика, а именно выставлять в качестве фильтра на дименшн `category` значение этого дименшна в данной точке (за это отвечает запись вида `setKoobFilters` в блоке `onClickDataPoint` конфига). Это значит, что все соседние дешлеты, если содержат в своих конфигах запись вида

```
filters: {
 category: true
}
```

автоматически перезапросят данные с указанным вашим кликом фильтром. Для наглядности мы еще показываем “координаты” кликнутой точки вверху графика.

Имейте, ввиду: если вы укажите запись вида `category: true` как в фильтрах выше у теку-

щего дешлета, то получите фильтрацию дешлетом самого себя. Автоматически вернуться к прежнему состоянию без перезагрузки вы не сможете. Будьте внимательны с тем, чтобы четко определять какой дешлет должен выступать фильтрующим, а какой фильтруемым.

Ответ сервера для компонента:

```
{"category":"Cider","count_units":230,"count_quantity":230}
{"category":"Beer","count_units":852,"count_quantity":852}
```

Результат рендера компонента выше:

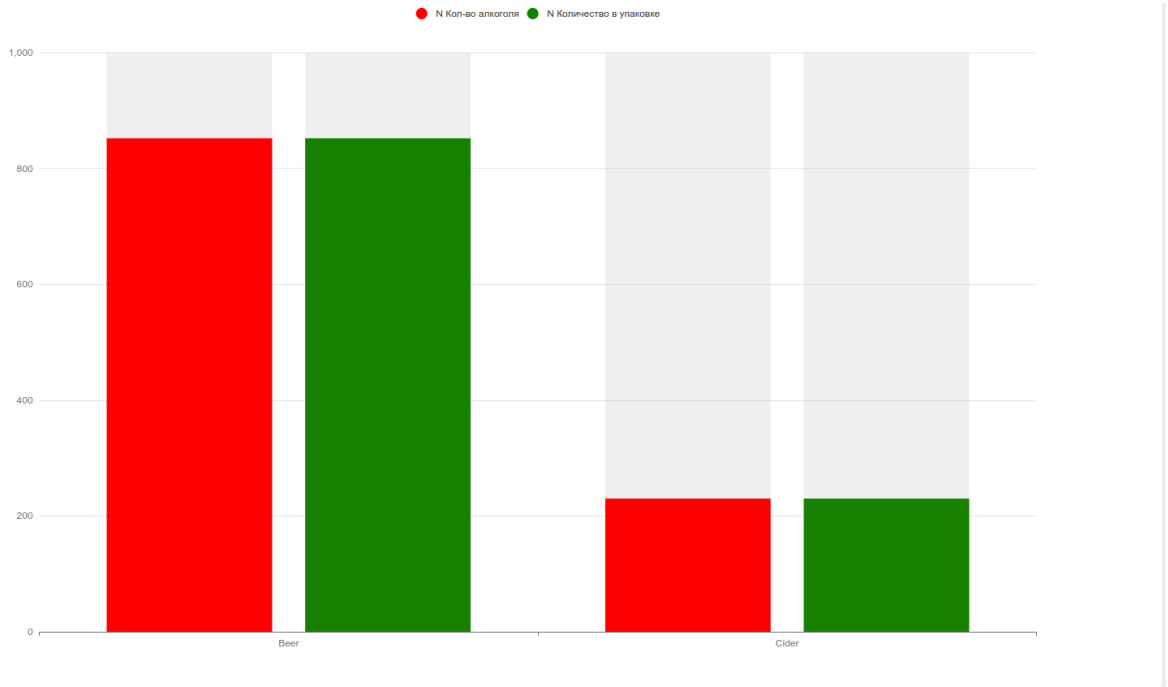


Рис. 13.5 **component.png**

13.9.2 Написание компонента-наследника базовых классов, реализующих коробочные визуализации на ECharts

Основной класс из коробки, который формирует React-компонент, который вставляет график с готовым конфигом и функционалом Echarts - это `BaseVizelEcharts`

Он реализует несколько абстрактных методов, которые все его наследники определяют под свои нужды:

```
protected abstract _getEchartsConfig(vm) {
 /*
  принимает вью-модель текущего дешлета и на основе данных из нее и логики текущ
  его графика формирует и возвращает объект конфигурации графика на Echarts.
  При написани  своего компонента-наследника - именно этот метод вам и нужно пер
  еопределить, чтобы компонент сразу был готов к использованию
```

```
   */
}

protected __createChart(vm: VM): any {
 /*
  Основной метод, формирующий инстанс Echarts, навешивающий обработчики и применяющий темы к графикам, берет конфиг, возвращаемый _getEchartsConfig
  */
}
/*
 Далее обработчики событий Echarts из __createChart
*/
protected _onChartCreated(chart: any): void {
  /*
  График заинициализирован и конфиг был применен
  */
}
protected _onChartClick = (event): void => {
  // обрабатывает событие click и tap по графику
  // https://echarts.apache.org/en/api.html#events.Mouse%20events.click
}
protected _onChartMouseOver = (params): void => {
  // обрабатывает событие mouseover по графику
  // https://echarts.apache.org/en/api.html#events.Mouse%20events.mouseover
}
protected _onLegendClick = (event): void => {
  // обрабатывает событие legendselectchanged графика
  // https:
  //echarts.apache.org/en/api.html#events.Mouse%20events.legendselectchanged
}
protected _onDataZoom(params, vm: any): any {
  // обрабатывает событие datazoom графика
  // https://echarts.apache.org/en/api.html#events.Mouse%20events.datazoom
}
protected _onChartDidMount(): any {
  // обрабатывает событие rendered графика
  // https://echarts.apache.org/en/api.html#events.Mouse%20events.datazoom
}

protected abstract _updateChart(vm, preVM) {
 /*
  Если произошла смена осей или еще чего-то в конфиге, то принимает вью-модель текущую и ее версию до смены. Можно использовать для формирования нового конфига ECharts и его установки в инстансе с графиком
  */
}
```

Немного об объекте типа `vm` (вью-модель):

это объект, описываемый интерфейсом `IVizelXYVM` (не все поля, только значимые)

```

interface IVizelXYVM {
  loading?: boolean; // загружена ли вью-модель
```

```
  error?: string; // есть ли ошибка
  schema_name: string; // имя схема атласа
  vAxes: IVAxis[]; // массив осей, которые сформировались в результате разброса сущностей или работе с единицами измерения
  series: ISerie[]; // список серий, хранящие массивы данных на каждой, название и другое
  categories: IEntity[]; // Категории, которые обычно обозначают
  noData: boolean; // есть ли вообще данные
  noNumericData: boolean; // нет ли числовых данных
  userSortOrder: string; // выбранное пользователем направление сортировки (ASC-DESC)
  userSortBy: string; // выбранная пользователем сортировка по полю
  subspace: ISubspace; // описывает сабспейс, хранящий инфу об элементах на осях, ориентации осей

  // events
  onToggleSort: (userSortBy: string) => void; // вызываем эту функцию для триггера сортировки по указанному полю с уже известным направлением сортировки
}

interface IVAxis {
  index: number;
  id: string;
  unit: IUnit;
  opposite: boolean;
  dataMin?: number;
  dataMax?: number;
}

interface ISerie {
  id: string;
  index: number;
  e: IEntity;
  title: string;
  vAxisIndex: number;
  stackGroup: string;
  values: IValue[];
  strValues: string[];   // string or numerics - formatted
  numValues: number[];   // numerics or nulls
  imgValues?: string[];
  bgColors: string[];
  isColorX?: boolean;
}

interface IEntity {
  id: number | string;
  title: string;
  ids?: Array<string | number>;
  readonly titles?: string[];
  readonly axisId?: string;
  readonly axisIds?: string[];
  readonly formula?: string[];
  readonly children?: IEntity[];
```

```
  readonly parent?: IEntity;
  readonly description?: string;
  config?: any;
  unit?: any;
  color?: string;
}
```

Помимо базового класса `BaseVizelEcharts` есть еще его наследник `EPlot` - это класс, который подготавливает дополнительные расчеты и обработку событий. По умолчанию его используют визуализации типа “Линии” и “Области”.

И `BaseVizelEcharts` и `EPlot` могут быть заимпортированы из модуля `bi-internal/ui`.

Тогда ваша задача при создании своего класса визуализации, который наследует коробочное поведение графиков достаточно унаследоваться от одного из базовых классов (в большинстве случае в это `EPlot`). Те. примерно так:

```
import {EPlot} from 'bi-internal/ui';

class MyComponent extends EPlot {

  protected _getEchartsConfig(vm) {
    let newConfig = super._getEchartsConfig(vm);

    // тут логика по правке конфига под себя, например

    newConfig = {
      ...newConfig,
      series: [
      ...newConfig.series,
        {
         type: "line",
          name: "test",
          data: [123, 46, 67]
        }
      ]
    };

    return newConfig;
  }

}

export default MyComponent;
```

К сожалению ваш кастомный класс не появится в списке встроенных визуализаций в режиме редактирования. Он может быть подключен только через тип визуализации `Внутренний`.

13.9.3 Переопределяем нативные визуализации

Допустим, что вы хотите, чтобы вместо стандартного `Пирог` встраивался ваш пай, который вы сами написали.

Для этого вам нужно создать в ресурсах атласа `ds_res` компонент `React`, чье имя совпадает с одним из нижеперечисленных (регистр важен):

```
board // Доска
tabs // Вкладки
column // Столбики (вертикальные) и штабели
bar // столбики (горизонтальные) и штабели
column1d // Столбики одномерные (две остальные оси фиксированные)
line // Линии
scatter // Точки
spline // Сплайн
area // Области и штабели
bublik // Пончик
pie // Пирог
gauge // Спидометр
halfgauge // Спидометр (половинный)
radar // Радар
radar1d // Одномерный радар
thermometer // Термометр
label // Значение, Текст
funnel // Воронка
plan // Схема (Карта или план здания на svg с предварительной разметкой атрибутами)
waterfall // Водопад
koob-table-simple // тип визуализации Данные
scales // Весы
pivot // Пивот-таблица
tableP // Таблица
whatif // What-if
sankey // Санкей
treemap // Древовидная карта
grid // Сетчатая диаграмма
abc // ABC анализ
axes-selector // Селектор осей
map // Карта (контейнер)
mapdots // Точки на карте (слой на карте)
mapcharts // Графики на карте (слой на карте)
mapareas // Области на карте (слой на карте)
mapheat // тепловая карта (слой на карте)
axes-selector // Селектор осей

// Список скоро пополнится
```

Так вот вам достаточно создать файл `pie.tsx`, который есть наследник `BaseVizelEcharts` по умолчанию (но вообще как вы видели ранее это просто `React.Component`). Где прописать логику отрисовки так, как вы работаете с обычным `React`- компонентом, Однако если это наследник `BaseVizelEcharts`, то такой компонент будет ожидать от вас обязательно

функции `_getEchartsConfig`, которая должна вернуть конфиг графика для `ECharts`, на основании своего аргумента `vm`.