Архитектура приложения

Приложение реализовано на базе фреймворка Spring с использованием компонентов Spring MVC, Spring Context и Spring Security. Для доступа к базе данных используется библиотека MyBatis.

Приложение имеет 3-хслойную арихетуктуру:

* слой представления (Controllers, JSP страницы, javascript, CSS и т.д.)
* слой бизнес логики (Widgets, FilterServices)
* слой доступа к данным (Repositories, MyBatis Mappers)

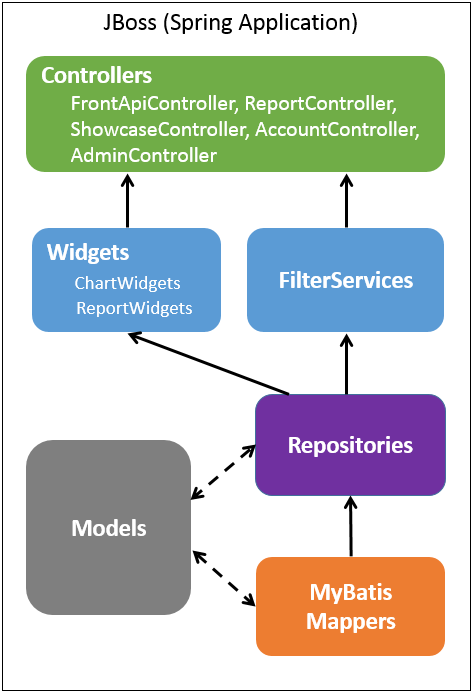


Рис. 1. Схема взаимодействия компонентов каждого слоя

Взаимодействие между слоя реализовано с помощью Dependency Injection и Spring IoC (Inversion Of Control) контейнера. Для конфигурации компонентов, сервисов и репозиториев используется Spring AOP (Aspect Oriented Programming).

Далее мы рассмотрим каждый из трех слоев подробнее.

# Слой представления

Слой представления составляет бо́льшую часть приложения и включает в себя контроллеры Spring, jsp страницы (витрины), CSS стили и javascript код.

Контроллеры (пакет «controller») выполняют следующий функционал:

1. FrontApiController (@RestController) принимает все Ajax запросы к данным для фильтров и виджетов и перенаправляет их соответствующим компонентам.
2. ReportController отвечает за выгрузку отчета в виде Excel файла.
3. ShowcaseController отвечает за отображение jsp страниц.
4. AccountController выполняет все действия связанные с авторизацией
5. AdminController помогает осуществлять администрирование приложения

Код на Javascript включает в себя сторонние библиотеки, фреймворки и собственный мини-фреймворк, на основе которого отображаются витрины.

## Сторонние JavaScript библиотеки и фреймворки

На клиентской части используются следующие библиотеки и фреймворки:

* [Bootstrap](http://getbootstrap.com/) – HTML, CSS и JS фреймворк.
* [Bootstrap Datepicker](https://github.com/eternicode/bootstrap-datepicker) – DatePicker, компонент для выбора дат.
* [Bootstrap Select](http://silviomoreto.github.io/bootstrap-select/) – компонент для реализации выпадающего списка.
* [Font Awesome](https://fortawesome.github.io/Font-Awesome/) – набор шрифтов-иконок.
* [HighCharts](http://www.highcharts.com/) – библиотека для отображения графиков.
* [JQuery](https://jquery.com/) – библиотека для манипуляций с DOM, выполнения Ajax запросов и обработки событий.
* [JQuery cookie](https://github.com/carhartl/jquery-cookie) – библиотека для удобной работы с cookies.
* [JSON2](https://github.com/douglascrockford/JSON-js) – добавляет поддержку JSON для старых браузеров (IE 8 и ниже)
* [Kendo UI](http://www.telerik.com/kendo-ui) – фреймворк с большим количеством виджетов. Из них используются только Grid (таблицы) и NumericTextBox.
* [Knockout](http://knockoutjs.com/) – библиотека, позволяющая реализовать Model-View-View Model (MVVM) паттерн.
* [Knockout Validation](https://github.com/Knockout-Contrib/Knockout-Validation) – валидация ввода для Knockout.
* [Lodash](https://lodash.com/) – библиотека для удобной работы с коллекциями и массивами.
* [Moment](http://momentjs.com/) – библиотека для удобной работы с датами.

## JavaScript мини-фреймворк

Мини-фреймворк написан на javascript и помогает минимизировать количество «рутинного» кода клиентской части.

В основе фреймворка лежит идея компонентов (Web Components), которая реализована при помощи библиотеки Knockout (<http://knockoutjs.com/documentation/component-overview.html>). Все интерактивные элементы интерфейса – фильтры, графики, таблицы, вкладки и прочее, являются компонентами.

Каждый компонент состоит из HTML разметки (template) и JS модели, в которой находится вся логика компонента (view model). Таким образом весь функционал компонента изолирован от всего остального, что позволяет быстро и легко вносить изменения.

Для того чтобы компоненты могли взаимодействовать между собой, необходима глобальная View Model. В мини-фреймворке такой view model выступает DynamicViewModel. Называется она так, потому что динамически создает саму себя, используя при этом JSON конфигурацию.

DynamicViewModel состоит из одной или нескольких групп (FilterGroup). Группа – это логическое объединение интерактивных элементов, в основном фильтров и графиков. По умолчанию, группа всего одна и называется «default».

Группы между собой могут иметь отношение master -> slave, что позволяет реализовывать общие группы фильтров, или, например, создать фильтр, который влияет только на определенный график, не затрагивая при этом остальные. Slave-группы получают все параметры master-групп.

Каждая FilterGroup содержит следующие свойства:

* *filters* – список фильтров
* *tabStrips* – список групп вкладок
* *charts* – список графиков
* *slaves* – список slave-групп
* *currentAjaxRequests* – список текущий ajax запросов
* *processingCharts* – флаг, указывающий на то, что в данный момент происходит создание/обновление графиков
* *drillDownLevel* – текущий уровень drill down. По умолчанию, 0.
* *filterData* – [computed observable](http://knockoutjs.com/documentation/computedObservables.html), содержит текущие параметры фильтров, параметры фильтров master-группы, если такая есть, и параметры drill down, если они заданы.

Cписок *currentAjaxRequests* используется для определения состояния элементов витрины, например, кнопки «Обновить». После начала выполнения Ajax запрос добавляется в этот список, а после завершения – удаляется.

Инициализация группы проходит следующим образом:

1. Анализ JSON конфигурации.
2. Строится дерево зависимостей фильтров.
3. Проверка на наличие циклических зависимостей. В случае если такая зависимость найдена, выполнение прерывается.
4. Выполняется инициализация каждого фильтра в нужном порядке. Каждый тип фильтров (например, «select» или «datepicker») имеет соответствующий builder. Этот builder создает все необходимые [observables](http://knockoutjs.com/documentation/observables.html) и, если, нужно выполняет запрос к серверу для получения данных. В первую очередь обрабатываются фильтры, которые ни от кого не зависят. Затем, используя дерево зависимостей, выполняется инициализация для всех остальных фильтров.
5. Инициализация вкладок (*tabStrips*).
6. Инициализация правил валидации.
7. Инициализация свойства *filterData*.

Как только все группы закончили свою инициализацию, выполняется привязка DynamicViewModel к HTML коду витрины (ko.applyBindings), что, в свою очередь, запускает механизм компонентов.

Во время загрузки специальный [componentLoader](http://knockoutjs.com/documentation/component-loaders.html) обрабатывает параметры компонента, находит соответствующую view model и выполняет ее инициализацию. Затем, получившийся результат, встраивается на определенное место на витрине.

Свойство группы *charts* заполняется в процессе инициализации компонентов. Chart view model так же отвечает за обновление внешнего вида графика (например, индикация процесса загрузки данных) и отображение ошибок.

И наконец, когда все компоненты были созданы, при необходимости (например, на dashboard), выполняется отображение графиков с текущими параметрами фильтров.

Отображение графиков – это следующая последовательность действий:

1. Валидация состояния фильтров. Если есть какие-нибудь ошибки, выполнение прерывается.
2. Текущие значения фильтров сохраняются в cookies.
3. Формируется список графиков, которые в данный момент доступны для отображения, т.е. не скрыты.
4. Графики группируются по свойству *dataSourceUrl*. Это необходимо для того, чтобы уменьшить количество Ajax запросов и получить данные для однотипных графиков за один раз.
5. Для каждой группы графиков выполняется запрос к серверу.

### JSON Configuration

JSON-конфиг – основной способ настройки функционала и внешнего вида витрины. Помимо него есть еще одна точка расширения (см. параметр *extendDynamicViewModel* функции *app.init*), для более гибкой настройки, которая не вписывается в стандартный функционал.

JSON-конфиг располагается на соответствующей jsp странице. Пример внешнего можно найти в Дополнении №1.

#### DynamicViewModel

|  |  |  |  |
| --- | --- | --- | --- |
| Ключ | Тип значения | По умолчанию | Описание |
| groups | Array | - | Список настроек FilterGroup. Должен содержать как минимум одну группу. |
| cookies | Boolean | String | «OnlyDrillDown» | Режим работы с cookies:  true – всегда получать значения фильтров из cookies  false – не использовать cookies  «OnlyDrillDown» – использовать значения из cookies только при drilldown на эту витрину |
| forceShowCharts | Boolean | false | Отображать графики сразу после загрузки витрины (обычно используется для витрин-дэшбордов) |

#### FilterGroup

|  |  |  |  |
| --- | --- | --- | --- |
| Ключ | Тип значения | По умолчанию | Описание |
| name | String | «default» | Имя группы. Если группа одна в списке, этот параметр не обязательный и задается имя по умолчанию. |
| filters | Object  *{*  *filter1: {},*  *filter2: {}*  *}* | - | Список настроек фильтров (см. Filter) |
| tabStrips | Object  (по аналогии с filters) | - | Список настроек групп вкладок (см. TabStrip) |
| charts | Object  (по аналогии с filters) | - | Список настроек графиков (см. Chart) |
| slaves | Array | - | Список slave-групп |
| forceShowCharts | Boolean | false | Отображать графики сразу после загрузки витрины (обычно используется для витрин-дэшбордов) |
| dontShowAfterMaster | Boolean | false | Не отображать графики этой группы, при отображении master-группы. |

#### Filter

Настройки фильтров зависят от его типа (*type*). У всех фильтров есть несколько общих параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Ключ | Тип значения | По умолчанию | Описание |
| type | String | - | Тип фильтра. Обязательный параметр. Может принимать следующие значения:   1. «CheckBox» 2. «DatePicker» 3. «NumericTextBox» 4. «Select» |
| title | String | - | Названия фильтра. Не обязательный параметр. |

##### Checkbox

Нет никаких дополнительных параметров

##### DatePicker

|  |  |  |  |
| --- | --- | --- | --- |
| Ключ | Тип значения | По умолчанию | Описание |
| notAfter | String | - | Имя фильтра с типом DatePicker, значения которого не может быть больше текущего. При попытке установить большее значения, значении текущего фильтра будет так же увеличено. Обычно используется вместе с параметров *notBefore* для фильтров «Период, с» и «Период, по». |
| notBefore | String | - | Делает то же что и параметр *notAfter* только наоборот. |
| width | Number | 130 | Ширина, px. |
| defaultValue | Date | Текущая дата | Начальное значение. Используется, если режим использования cookies не равен *true*. |
| disableIf | String | - | Имя фильтра, значение которого влияет на доступность DatePicker. Если значения фильтра не равно 0, empty или undefined, то DatePicker доступен. |
| disableIfNot | String | - | Делает то же что и параметр *disableIf* только наоборот. |
| lastDayOfMonth | Boolean | false | Если true, то при выборе любого дня месяца, значение будет установлено как последний день выбранного месяца. |
| datepickerOptions | Object | {  language: “ru”,  orientation: top”,  autoclose: true  } | Параметры плагина Bootstrap Datepicker. Используются для тонкой настройки внешнего вида и функционала. Подробнее тут  http://bootstrap-datepicker.readthedocs.org /en/stable/options.html |

##### NumericTextBox

|  |  |  |  |
| --- | --- | --- | --- |
| Ключ | Тип значения | По умолчанию | Описание |
| width | Number | 80 | Ширина, px. |
| min | Number | 0 | Минимальное значение |
| max | Number | 100 | Максимальное значение |
| step | Number | 1 | Шаг изменения значения |
| format | String | «0» | Формат значения |
| decimals | Number | 0 | Количество знаков после запятой |

##### Select

|  |  |  |  |
| --- | --- | --- | --- |
| Ключ | Тип значения | По умолчанию | Описание |
| dataSource | Object | - | Источник данных для Select. Подробное описание можно найти ниже. |
| multiple | Boolean | false | Множественный выбор. |
| withGroups | Boolean | false | Группировка элементов выпадающего списка. Требует получения данных в специальном формате. |
| defaultValue | Number | Array | - | Начальное значение. Используется, если режим использования cookies не равен *true*.  В случае, если это SingleSelect (т.е. параметр *multiple* = false), то значением по умолчанию будет первый элемент в списке. |
| optionsCaption | String | «Все», только для  MutliSelect | Текст-placeholder, который отображается если ничего не выбрано. |
| selectedTextFormat | String | «count>1» | Формат отображения выбранного элемента(-ов). [http://silviomoreto.github.io/bootstrap-select/#titleMultiplesFormat](http://silviomoreto.github.io/bootstrap-select/%23titleMultiplesFormat) |
| size | Number | 10 | Количество одновременно отображаемых значений в выпадающем списке. |
| width | Number | 250 | Ширина, px |
| enableSearch | Boolean | false | Поиск по выпадающему списку. |
| enableClear | Boolean | false | Кнопка «Очистить», которая очищает список выбранных элементов. |
| enableSelectAll | Boolean | false | Кнопка «Выбрать все», которая выбирает все доступные элементы. |
| disableIfNull | String | - | Имя фильтра. Если значение этого фильтра null, undefined либо пустой массив, то Select становится недоступным. |
| disableIfComputed | Object  {  params: [],  jsFunc: function(ctx){  }  } | - | Параметр для гибкой настройки доступности Select. Принимает объект с двумя параметрами:   * params – список строк с названиями фильтров * jsFunc – функция с единственным параметром *context*. Этот *context* содержит текущее значение перечисленных в списке *params* фильтров. Возвращает true, если фильтр недоступен. |

##### DataSource

DataSource может работать в двух режимах: «Ajax» и «Local».

В режиме «Ajax» данные для фильтра возвращаются с сервера в виде JSON. Список параметров для ajax запроса формируется автоматически на основании свойства *params*. Каждый раз, при изменении значения любого из фильтров, указанных в списке *params*, выполняется очередной Ajax запрос. Если свойство *params* не указано, то запрос выполнится всего один раз.

В режиме «Local» данные задаются сразу в конфигурации фильтра и никаких запросов к серверу не происходит. Это удобно, если значений фильтра всего несколько штук. Данные задаются в виде списка объектов с двумя обязательными полями: «id» и «name».

В обоих режимах работы DataSource может быть задано свойство *computed*, которое позволяет выполнить пост-обработку полученных значений.

|  |  |  |  |
| --- | --- | --- | --- |
| Ключ | Тип значения | По умолчанию | Описание |
| url | String | - | URL метода FilterService. Если этот параметр задан, считается что режим работы – «Ajax». В противном случае режим работы считается «Local». |
| params | Array | - | Список названий фильтров, значений которых нужно передать как параметры к Ajax запросу. Не обязательный параметр. |
| data | [] | - | Используется в режиме работы «Local», |
| computed | Object | - | Объект, который содержит следующие свойства:   * params – список названий фильтров * jsFunc – функция с единственным параметром context, который содержит текущее значение выбранных фильтров и данные, полученные источником данных. Функция должна вернуть отфилтрованный/видоизменный список данных. |

#### TabStrip

TabStrip – это группа вкладок (Tab). Разметка вкладок может как одинаковой (см. параметр *sameMarkup*), так и разной, и содержать интерактивные элементы вместе с обычным HTML.

В случае, если разметка каждой вкладки задается отдельно (без параметра *sameMarkup*), количество вкладок в конфиге должно совпадать с количеством вкладок в разметке витрины (компоненте).

Если параметр *sameMarkup* используется, то разметка всех вкладок описывается один раз и используется как шаблон. В каждый график на шаблоне передаются параметры, описанные в свойстве *customParams*. Так же в шаблоне происходит замена всех строковых значений {groupName} на значения свойства *group*.

|  |  |  |  |
| --- | --- | --- | --- |
| Ключ | Тип значения | По умолчанию | Описание |
| sameMarkup | Boolean | false | Позволяет задать описание всех вкладок при помощи одного шаблона. |
| tabs | Array<String> |  Array<Object> | - | Список с описанием вкладок. Может содержать строки (названия вкладок) или Objects. |
| header | String | - | Заголовок для группы вкладок. |
| reportUrl | String | - | Ссылка на ReportService для выгрузки отчета в Excel. Не обязательный параметр. |
| dynamicConfig | Object | - | Этот параметр позволяет динамически настроить видимость и название каждой вкладки в зависимости от значений фильтров.  Работает по принципу остальных computed параметров. Содержит свойства:   * params – список названий фильтров * jsFunc – функция, принимающая параметр *context*, который содержит список вкладок и текущие значений фильтров. |

#### Tab

Разметка вкладки задается не в конфиге, а на самой витрине.

|  |  |  |  |
| --- | --- | --- | --- |
| Ключ | Тип значения | По умолчанию | Описание |
| title | String | - | Название вкладки. Может меняться при помощи функции dynamicConfig. |
| group | String | Группа, к которой относится TabStrip. | Название FilterGroup. Подставляется в виде параметра на все графики данной вкладки. Используется вместе с параметром sameMarkup. |
| customParams | Object | - | Object вида key-value. Используется вместе с параметром sameMarkup. |

#### Chart

|  |  |  |  |
| --- | --- | --- | --- |
| Ключ | Тип значения | По умолчанию | Описание |
| jsFunc | Function | - | Обязательный параметр. Функция, которая содержит всю логику создания графика. Принимает следующие параметры:   * $container – jQuery wrapper поверх элемента-контейнера * filterData – список с текущими значениями фильтров * jsonData – данные, полученные в результате выполнения Ajax запроса. * customParams – список с дополнительными параметрами. |
| dataSource | String | - | Обязательный параметр. URL WidgetService. |
| customParams | Object | - | Object вида key-value. Нужен для передачи дополнительных параметров, если несколько графиков используют одну и ту же jsFunc. |

### API

Мини-фреймворк практически не создает переменных в глобальной области видимости. Для этого все его модули обвернуты в closure.

Единственная переменная, которая используется для доступа к API – *app* (window.app), которая представляет собой класс App.js.

App.js имеет следующие методы:

* *parseNameAndGroup (nameAndGroup)* – вспомогательная функция, используется во многих компонентах фреймворка. Позволяет распарсить строку или объект и получить информацию о имени фильтра/вкладки и группе, к которому фильтр/вкладка относятся.
* *showAlert (message, title)* – отображает модальное окно с сообщением об ошибке.
* *init (config, extendDynamicViewModel)* – основная функция, создает DynamicViewModel. *extendDynamicViewModel* – точка расширения, позволяет добавить специфические для данной витрины свойства и функции.

# Слой бизнес логики

Слой бизнес логики состоит в основном из сервисов:

* Виджеты (пакет «widget») – сервисы, которые отвечают за получение, преобразование и передачу данных для виджетов (графиков, таблиц)
* FilterServices (пакет «filter») – сервисы, которые отвечают за получение, преобразование и передачу данных для фильтров.

На уровне сервисов происходит преобразование данных (Models), полученных из репозиториев (Repositories) в требуемое для каждого виджета/фильтра представление.

Виджеты делается на 3 вида:

1. Графики – используются для отображения HighCharts графиков. Наследуются от класса BaseChart.
2. Таблицы – используются для отображения таблиц (KendoUI). Наследуют BaseWidget.
3. Отчеты – отвечают за выгрузку данных в Excel файл. Наследуют BaseReport.

# Слой доступа к данным

Слой доступа данных делится на:

* MyBatis mappers (пакет «mapper») – интерфейсы доступа к данным из БД, которые MyBatis использует для SQL запросов.
* Repositories – (пакет «repository») репозитории расширяют интерфейс мапперов и добавляют необходимый функционал.

В качестве DTO (Data Transfer Object) выступают POJO- классы (Plain Old Java Object), которые находятся в пакете «Model».

Каждый маппер должен иметь соответствующий xml файл с конфигурацией. Этот файл затем используется MyBatis для автоматической реализации интерфейса маппера. Пример такого файла находится во вложении №2.

# Авторизация

Авторизация работает по принципу Single Sign-On, т.е. пользователю не нужно дополнительно вводить логин/пароль. Для проверки прав доступа и получения необходимой информации о пользователях, приложение выполняет запрос к серверу Active Directory, используя механизм SPNEGO (Simple and Protected GSSAPI Negotiation Mechanism).

Запрос к Active Directory выполняется под специальным пользователем, у которого есть права поиск и получение всей необходимой информации об учетной записи пользователя («memberOf», «displayName» и т.д.). Затем, на основе списка ролей Active Directory (поле «memberOf»), приложение получает соответствующий им внутренний список ролей (SimpleGrantedAuthority из компонента Spring Security) и далее работает только с ним.

В случае, если у пользователя отсутствуют необходимые права доступа, ему отображается страница с сообщением об этом.

# Прочее

Отдельно от этих 3х слоев можно рассматривать пакет «infrastructure», который содержит сервисы и классы, облегчающие разработку и поддержку проекта.

Например, в пакете «infrastructure.chart» находятся классы ChartResult, Series и Point, которые передаются в JSON формате и используются при отображении HighCharts графиков.

# Дополнения

## Дополнение №1. Фрагмент JSON-конфига.

Витрина «DashBoard Персонал»

**var *config*** = {  
 **forceShowCharts**: **true**,  
 **groups**: [{  
 **filters**: {  
 **startDate**: {  
 **type**: **"DatePicker"**,  
 **title**: **"Год"**,  
 **width**: 90,  
 **datepickerOptions**: {  
 **format**: **"yyyy"**,  
 **minViewMode**: 2  
 },  
 **defaultValue**: **moment**().**startOf**(**'year'**).**toDate**(),  
 **forceShowCharts**: **true** }  
 },  
 **charts**: {  
 **kpi2Chart**: {  
 **jsFunc**: *createKpi2\_3\_4\_6Chart*,  
 **dataSource**: **"LeaderBoardKpi2\_3\_4\_6Chart"**,  
 **customParams**: {**kpiCode**: **"KPIOB~2"**}  
 },  
 **kpi3Chart**: {  
 **jsFunc**: *createKpi2\_3\_4\_6Chart*,  
 **dataSource**: **"LeaderBoardKpi2\_3\_4\_6Chart"**,  
 **customParams**: {**kpiCode**: **"KPIOB~3"**}  
 },  
 **kpi4Chart**: {  
 **jsFunc**: *createKpi2\_3\_4\_6Chart*,  
 **dataSource**: **"LeaderBoardKpi2\_3\_4\_6Chart"**,  
 **customParams**: {**kpiCode**: **"KPIOB~4"**}  
 },  
 **kpi5Chart**: {  
 **jsFunc**: *createKpi5Chart*,  
 **dataSource**: **"LeaderBoardKpi5Chart"** },  
 **kpi6Chart**: {  
 **jsFunc**: *createKpi2\_3\_4\_6Chart*,  
 **dataSource**: **"LeaderBoardKpi2\_3\_4\_6Chart"**,  
 **customParams**: {**kpiCode**: **"KPIOB~6"**}  
 }  
 },  
 **slaves**: [{  
 **name**: **"kpi1"**,  
 **filters**: {  
 **divisionGroupId**: {  
 **type**: **"Select"**,  
 **multiple**: **false**,  
 **title**: **"Группа подразделений"**,  
 **dataSource**: {  
 **url**: **"leaderBoardFilter/DivisionGroups"**,  
 **params**: [**"startDate"**]  
 },  
 **defaultValue**: 800022430857, *// Операционный Блок* **width**: 220,  
 **forceShowCharts**: **true** }  
 },  
 **charts**: {  
 **kpi1Chart**: {  
 **jsFunc**: *createKpi1Chart*,  
 **dataSource**: **"LeaderBoardKpi1Chart"** }  
 }  
 }]  
 }]};

**var *color*** = {  
 **prevValue**: **"#91e8e1"**,  
 **currentValue**: **"#7cb5ec"**,  
 **planValue**: **"#f7a35c"**};  
  
**function** *createKpi1Chart*($container, filterData, jsonData) {  
 **var** chart = jsonData[0],  
 series = chart.**series**,  
 title = chart.bag.kpiName;  
  
 **if** (series.length === 0 || series[0].data.length === 0) {  
 $container.highcharts({  
 **title**: {**text**: title},  
 **chart**: {  
 **height**: 100  
 }  
 });  
 **return**;  
 }  
  
 filterData.**endDate** = **moment**(filterData.startDate).add(1, **"y"**).add(-1, **"d"**).**toDate**();  
 filterData.**timeUnitId** = 4;  
  
  
 series[0].**color** = ***color***.**prevValue**;  
 series[1].**color** = ***color***.**currentValue**;  
 series[2].**color** = ***color***.**planValue**;  
  
 $container.highcharts({  
 **chart**: {  
 **type**: **'column'**,  
 **height**: 320  
 },  
 **title**: {  
 **text**: title  
 },  
 **xAxis**: {  
 **categories**: [  
 **"Январь"**, **"Февраль"**, **"Март"**, **"Апрель"**,  
 **"Май"**, **"Июнь"**, **"Июль"**, **"Август"**,  
 **"Сентябрь"**, **"Октябрь"**, **"Ноябрь"**, **"Декабрь"** ],  
 **crosshair**: **true** },  
 **yAxis**: {  
 **min**: 0,  
 **max**: 100,  
 **title**: {**text**: **""**},  
 **labels**: {  
 **format**: **'{value}%'** }  
 },  
 **tooltip**: {  
 **headerFormat**: **'<span style="font-size:10px">{point.key}</span><table>'**,  
 **pointFormat**: **'<tr><td style="padding:2px">{series.name}: </td>'** +  
 **'<td style="padding:0"><b>{point.y:.2f}%</b></td></tr>'**,  
 **footerFormat**: **'</table>'**,  
 **shared**: **true**,  
 **useHTML**: **true** },  
 **legend**: {  
 **verticalAlign**: **'top'**,  
 **y**: 25  
 },  
 **plotOptions**: {  
 **column**: {  
 **dataLabels**: {  
 **inside**: **true**,  
 **enabled**: **true**,  
 **rotation**: 270,  
 **style**: {  
 **"fontSize"**: **"12px"**,  
 **"fontWeight"**: **"bold"** },  
 formatter: **function**() { **return this**.*y* < 4 ? **null** : (**this**.*y*.toFixed(2) + **"%"**); }  
 }  
 }  
 },  
 **series**: series  
 });  
}

## Дополнение №2. Пример xml конфигурации MyBatis mapper

*<?****xml version="1.0" encoding="UTF-8"*** *?>****<!DOCTYPE mapper  
 PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">****<****mapper namespace="ru.alfabank.dmpr.mapper.leaderBoard.LeaderBoardMapper"****>  
 <****cache****/>  
  
 <****select id="getKPIs" resultType="ru.alfabank.dmpr.model.BaseEntityWithCode"****>  
 <![CDATA[  
 select  
 id,  
 name,  
 ccode as code  
 from table(pkg\_leaderboard\_api.get\_KPIs)  
 ]]>  
 </****select****>  
  
 <****select id="getKpiData" useCache="false"  
 parameterType="ru.alfabank.dmpr.model.leaderBoard.LeaderBoardOptions"  
 resultType="ru.alfabank.dmpr.model.leaderBoard.KpiDataItem"****>  
 <![CDATA[  
 select  
 VALUE\_DAY as calcDate, -- Отчетная дата  
 KPI\_UK as kpiId, -- Показатель (UK)  
 KPI\_CCODE as kpiCode, -- Показатель (CCODE)  
 KPI\_NAME as kpiName, -- Показатель (Наименование)  
 DivisionGroup\_UK as divisionGroupId, -- Группы подразделений (UK)  
 DivisionGroup\_CCODE as divisionGroupCode, -- Группы подразделений (CCODE)  
 DivisionGroup\_NAME as divisionGroupName, -- Группы подразделений (Наименование)  
 PREV\_FACTVALUE as prevValue, -- Предыдущее факт. значение показателя   
 FACTVALUE as currentValue, -- Фактическое значение показателя  
 PLANVALUE\_HIHBOUND\_CNT as planValue, -- Плановое значение показателя  
 Fact\_Description as description  
 from table(PKG\_LEADERBOARD\_API.get\_kpi\_data (  
 p\_DateFrom => #{startDate},  
 p\_DateTill => #{endDate},  
 p\_KPIIDs => #{kpiIds},  
 p\_DateIntervalType => #{dateIntervalType},  
 p\_DivisionGroupIDs => #{divisionGroupId}  
 ))  
 ]]>  
 </****select****>  
</****mapper****>*