государственное автономное профессиональное образовательное учреждение   
Чувашской Республики «Межрегиональный центр компетенций –  
 Чебоксарский электромеханический колледж» Министерства образования и    
молодежной политики Чувашской Республики

ДИПЛОМНЫЙ ПРОЕКТ

Разработка блока управления энергетическим

оборудованием ООО «Пролог-СТ»

на базе GSM-модуля

ДП.Кс1-17.02.00.ПЗ

Студент Иванов И.И.

(Фамилия И. О.)

(подпись) (чч.мм.гггг)

Руководитель Петров П.П.

(Фамилия И. О.)

(подпись) (чч.мм.гггг)

Консультант

по экономике Сидоров С.С.

(Фамилия И. О.)

(подпись) (чч.мм.гггг)

Нормоконтроль Данилов Д.Д.

(Фамилия И. О.)

(подпись) (чч.мм.гггг)

Рецензент Васильев В.В.

(Фамилия И. О.)

(подпись) (чч.мм.гггг)

Зав. отделением Степанов С.С.

(Фамилия И. О.)

(подпись) (чч.мм.гггг)

2021

# АННОТАЦИЯ

B ходе дипломного проекта разработан блок управления энергетическим оборудованием ООО «Пролог-СТ» на базе GSM-модуля. Устройство необходимо для дистанционного контроля и управления температурой в производственном помещении. Оно позволяет понижать или повышать температуру не только в производственном цеху, но также может найти применение в быту.

Устройство имеет жидкокристаллический индикатор для постоянного мониторинга температуры. Имеет защиту от сбоев электроэнергии. Разработанный блок управления легко встраиваемый и расширяемый.

# СОДЕРЖАНИЕ

Изм.

Лист

№ докум.

Подпись

Дата

Лист

2

ДП.Кс1-16.02.00.ПЗ

Разраб.

Батырев М.С.

Разработка блока управления энергетическим оборудованием ООО «Пролог-СТ» на базе GSM-модуля

Лит.

Листов

52

МЦК – ЧЭМК

Пров.

Иванов Л.Б.

Реценз.

Яичников П.Г.

Н.контр.

Терентьева С.Н.

Утв.

Федорова Т.В.

Введение 4

1 Аналитическая часть 5

1.1 5

1.2 Классификация GSM-модулей 8

1.3 Области применения GSM-модулей 8

1.4 Анализ требований к разрабатываемому устройству 8

1.5 Обзор аналогов разрабатываемого устройства 8

2 Конструкторская часть 10

2.1 Разработка схемы электрической структурной 10

2.2 Описание элементной базы 10

2.3 Разработка схемы электрической принципиальной 11

2.4 Расчет надежности и потребляемой мощности схемы 11

3 Технологическая часть 14

3.1 Разработка алгоритма работы управляющей программы 14

3.2 Разработка печатной платы 14

3.3 Описание метода изготовления печатной платы 14

3.3 Расчет печатной платы 14

4 Экономические показатели проекта 15

4.1 Актуальность разработки устройства 15

4.2 Расчет трудоемкости и цены разработки 15

4.3 Расчет затрат на создание устройства 15

4.4 Расчет себестоимости опытного образца проектируемого изделия 15

4.5 Расчет цены единицы изделия 15

4.6 Расчет прибыли от реализации 15

5 Безопасность и экологичность проектного решения 16

5.1 Характеристика объекта разработки и рабочего помещения 16

5.2 Анализ опасных и вредных производственных факторов на предприятиях приборостроения 16

5.3 Мероприятия по устранению или уменьшению влияния вредных факторов при производстве печатных плат в приборостроении 16

5.4 Пожарная безопасность 16

5.5 Экологичность проекта 16

Заключение 17

Список использованных источников 18

Приложение А – Схема электрическая структурная 19

Приложение Б – Схема электрическая принципиальная 20

Приложение В – Чертеж печатной платы 21

Приложение Г – Экономические показатели 22

# ВВЕДЕНИЕ

На сегодняшний день многим людям приходится иметь дело с большим объёмом информации. С распространением различных информационных технологий у человека появилась задача по обработке больших массивов данных. Сведения, представленные в виде обычного текста, требуют очень много концентрации и времени для того, чтобы люди смогли в них разобраться и принять какие-либо решения на основе представленной информации. Для облегчения процесса обработки была придумана визуализация данных. Если показатели из среза необходимых данных будут отображаться не в виде обычного текста, а в виде различных геометрических фигур, то человек сможет гораздо быстрее их проанализировать, потому что информацию, представленную в графическом виде, мозг обрабатывает эффективнее.

Таким образом, разработчикам приложений и сайта для бизнеса требуется часто добавлять в свои продукты средства визуализации данных. Для того, чтобы в каждой новой программе не переписывать код, направленный на решение типовой и часто встречающейся задачи, были придуманы библиотеки. Их использование значительно сокращает время разработки программного продукта.

Целью дипломного проекта является разработка библиотеки, предоставляющей инструменты для визуализации данных.

Исходя из поставленной цели, были определены следующие задачи:

* анализ и оценка конкурентов на рынке;
* составление технического задания на разработку библиотеки;
* разработка модулей библиотеки;
* тестирование модулей;
* публикация библиотеки;
* расчет экономических показателей проекта;
* сопровождение программного продукта.

## 1 Теоретическая часть

### 1.1 Понятие визуализации данных

Визуализация данных – это представление данных в виде, который обеспечивает наиболее эффективную работу человека по их изучению. Визуализация данных – важная функция средств бизнес-анализа и ключ к расширенной аналитике. Визуализация позволяет проводить оценку значения информации или данных. Под визуализацией данных подразумевается представление информации в графической форме, например, в виде круговой диаграммы, графика или визуального представления какого-либо другого типа.

Качественная визуализация данных имеет очень важное значение для анализа данных и принятия решений на их основе. Визуализация позволяет быстро и легко замечать и интерпретировать связи и взаимоотношения, а также выявлять развивающиеся тенденции, которые не привлекли бы внимания в виде необработанных данных. В большинстве случаев для интерпретации графических представлений не требуется специальное обучение, что сокращает вероятность недопонимания. Также данные, представленные в удобном графическом виде, не утомляют пользователя во время взаимодействия и помогают сохранять концентрацию на более долгий срок.

Продуманное графическое представление не только содержит информацию, но и повышает эффективность ее восприятия за счет наглядности, привлечения внимания и удержания интереса в отличие от таблиц и обычных текстовых документов.

Большинство средств визуализации данных могут подключаться к локальным или облачным источникам данных, например, реляционным базам. Таким образом данные извлекаются для анализа. Пользователи могут выбрать наиболее подходящий способ представления данных из нескольких вариантов. Некоторые средства автоматически выводят рекомендации по использованию представлений в зависимости от анализируемых данных.

### 1.2 Диаграмма

Диаграмма – графическое представление данных линейными отрезками или геометрическими фигурами, позволяющее быстро оценить соотношение нескольких величин. Представляет собой геометрическое символьное изображение информации с применением различных приёмов техники визуализации

Благодаря своей наглядности и удобству использования, диаграммы часто используются не только в повседневной работе бухгалтеров, логистов и других служащих, но и при подготовке материалов презентаций для клиентов и менеджеров различных организаций.

Диаграммы в основном состоят из геометрических объектов (точек, линий, фигур различной формы и цвета) и вспомогательных элементов (осей координат, условных обозначений, заголовков и т. п.). Также диаграммы делятся на плоскостные (двумерные) и пространственные (трёхмерные или объёмные). Сравнение и сопоставление геометрических объектов на диаграммах может происходить по различным измерениям: по площади фигуры или её высоте, по местонахождению точек, по их густоте, по интенсивности цвета и т. д. Кроме того, данные могут быть представлены в двухмерной или полярной системе координат.

Основные виды диаграмм:

* Линейные диаграммы – это тип диаграмм, на которых полученные данные изображаются в виде точек, соединённых линиями. Точки могут быть как видимыми, так и невидимыми. Для построения диаграмм-линий применяют двухмерную систему координат.
* Диаграммы с областями – это тип диаграмм, схожий с линейными диаграммами способом построения кривых линий. Отличается от них тем, что область под каждым графиком заполняется индивидуальным цветом или оттенком. Преимущество данного метода в том, что он позволяет оценивать вклад каждого элемента в рассматриваемый процесс, если отображать её составные сегментировано.
* Гистограммы (столбчатые диаграммы). Данный тип в основном используется для наглядного сравнения полученных статистических данных или для анализа их изменения за определённый промежуток времени. Построение столбчатой диаграммы заключается в изображении статистических данных в виде вертикальных прямоугольников или трёхмерных прямоугольных столбиков. Каждый столбик изображает величину уровня данного статистического ряда. Гистограммы также могут изображаться и группами (одновременно расположенными на одной горизонтальной оси с разной размерностью варьирующих признаков).
* Круговые или кольцевые диаграммы. Данный тип диаграмм также называют секторным. В круговых диаграммах сумма всех составляющих представляется в виде круга, а набор значений – в виде секторов этого круга. Этот вид графиков удобно использовать, когда нужно показать долю каждой величины в общем объёме. Сектора могут изображаться как в общем круге, так и отдельно, расположенными на небольшом удалении друг от друга.
* Диаграмма Ганта – это визуальный способ отображения запланированных задач в виде набора столбцов. Диаграммы Ганта позволяют наглядно изобразить составляющие части проекта и разбить его на задания меньшего размера для удобства управления. Получившиеся задания размещаются вдоль временной шкалы диаграммы. Данный тип визуализации очень часто используется во многих приложениях по осуществлению менеджмента какого-либо проекта.

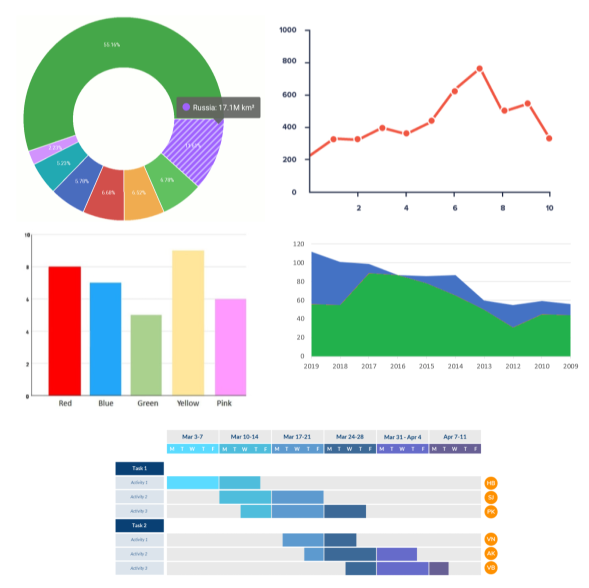


Рисунок 1.1 – Примеры диаграмм

### 1.3 Загружаемые библиотеки в npm

Библиотека – сборник подпрограмм или объектов, используемых для разработки программного обеспечения (ПО). Библиотеки для интерпретируемых языков представляют собой файлы, содержащие либо код на интерпретируемом языке, либо байт-код для виртуальной машины.

Npm (англ. Node Package Manager) – менеджер пакетов для языка программирования JavaScript. Npm является стандартным пакетным менеджером для окружения Node.js.

Менеджер npm состоит из двух частей:

* CLI (интерфейс командной строки) – средство для размещения и скачивания пакетов,
* онлайн-репозитории, содержащие JS пакеты.

С помощью клиента командной строки пользователь может загружать библиотеки других разработчиков, а также публиковать и управлять своими собственными загружаемыми пакетами. Таким образом разработчики могут очень просто загружать готовые алгоритмы для решения своих задач.

Чтобы использовать пакеты, проект должен содержать файл с именем package.json. Внутри этого пакета находятся метаданные, касающиеся проектов.

Метаданные показывают несколько аспектов проекта в следующем порядке:

* Название проекта;
* Первоначальная версия;
* Описание;
* Точка входа;
* Тестовые команды;
* Репозиторий Git;
* Ключевые слова;
* Лицензия;
* Зависимости;
* Зависимости для разработки.

Метаданные помогают идентифицировать проект и служат основным источником информации о проекте.

Исходя из вышеперечисленной информации можно сделать выводы:

* В мире современных информационных технологий большую роль играет визуализация данных;
* Наиболее удобным способом представления данных в графическом виде является использование диаграмм;
* Загружаемые библиотеки значительно ускоряют процесс разработки и внедрения программного обеспчения;
* Библиотека будет загружаться в базу данных менеджера пакетов npm.

## 2 Аналитическая часть

Анализировать большое количество данных очень сложно, когда они представлены в виде обычного текста и чисел. Визуализация значительно облегчает процесс понимания представляемой информации и ускоряет процесс её исследования. Значительную роль играет внешний вид диаграмм и графиков и также интерактивность элементов, представленных на экране. Поскольку визуализация данных используется в большом количестве сайтов и веб-приложений, предназначенных для аналитики, наилучшим решением будет представить компонент в виде загружаемой библиотеки. Таким образом компонент можно будет легко переносить в проекте, использовать его много раз, а также не беспокоиться о работоспособности системы, так как библиотека и основное приложение изолированы друг от друга.

На сегодняшний день рынок предлагает большое количество различных библиотек разработчиков практически любой области. Особенно сильно такой подход распространен в направлении веб-разработки. В интернете можно найти загружаемые компоненты для реализации любой цели, в том числе и для представления данных в графическом виде. Такое обилие предлагаемых продуктов порождает большую конкуренцию.

Конкурентоспособность – свойство субъекта или объекта, определяющее способность превзойти конкурентов в заданных условиях. В нашем случае она определяется эффективностью для других разработчиков.

Выберем наиболее важные критерии конкурентоспособности для загружаемых компонентов.

Первым критерием является перечень предоставляемых функциональных возможностей. Чем больше различного и полезного функционала библиотека может реализовать, тем выгоднее она выглядит в глазах других разработчиков.

Вторым критерием является гибкость настройки. Библиотека должна быть способна выполнять свои функции в любом проекте и предоставлять инструменты простой настройки имеющихся компонентов.

Третий критерии – вес всего загружаемого пакета. Данный критерий очень важен, поскольку библиотека используется в других проектах, и разработчики будут всегда стремиться выбирать варианты с наименьшим весом, чтобы уменьшить размер сайта или приложения, что приведет к увеличению скорости загрузки всех исходных файлов.

Для того, чтобы определить конкурентоспособность библиотеки, следует проанализировать пакеты, предоставляемые другими разработчиками.

Одной из самых популярных библиотек является Apexcharts.js.

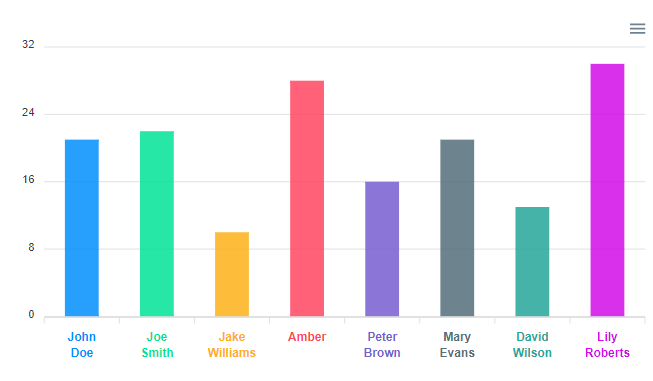


Рисунок 2.1 – Пример гистограммы, созданной на Apexcharts.js

Достоинства:

* возможности настраивать оси графиков;
* используемые цвета заметно отличаются друг от друга;
* большое количество предлагаемых видов графиков.

Недостатки:

* большой размер загружаемого пакета;
* сильные ограничения по количеству используемых цветов;
* невозможность конфигурирования анимаций.

Apexcharts.js позволяет отображать практически все возможные виды диаграмм, что является самым явным плюсом данного пакета. Однако очень большой размер исходных файлов (4 МБ) может оттолкнуть множество пользователей.

Следующим возможным конкурентом является библиотека Chart.js

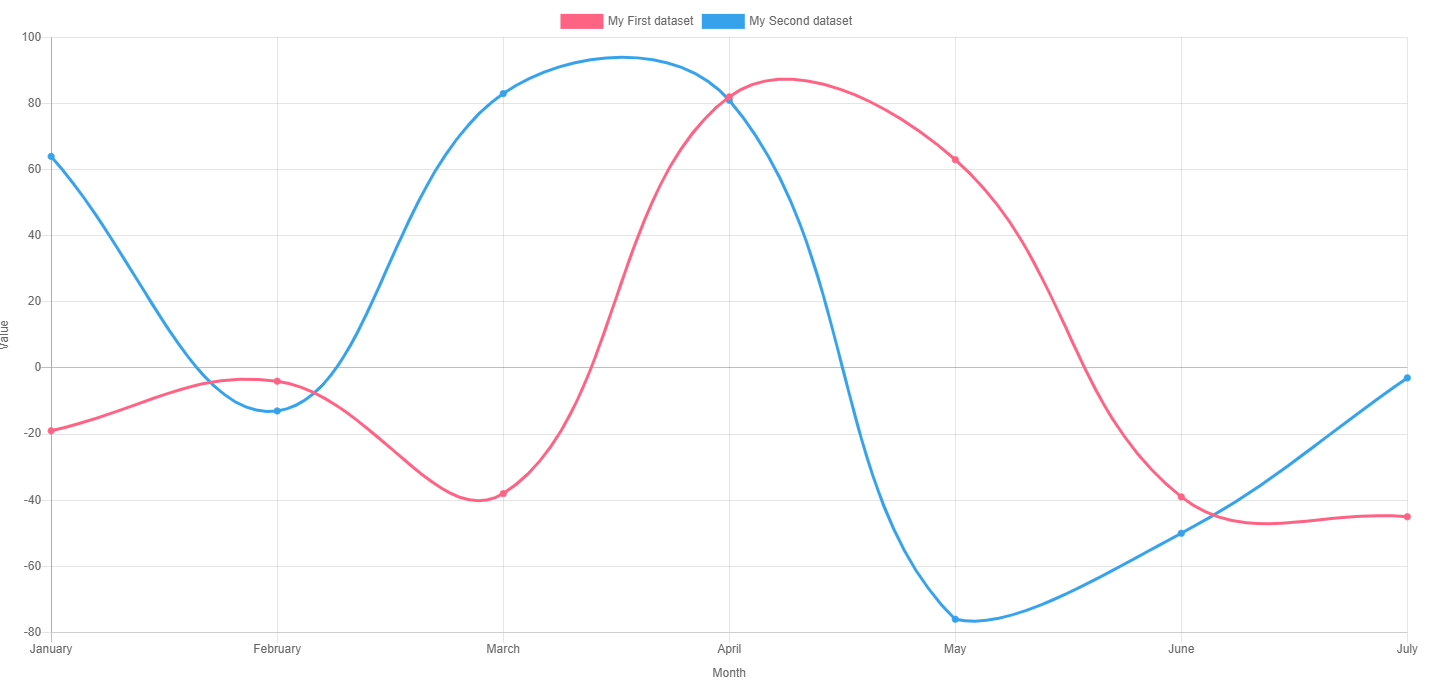


Рисунок 2.2 – Пример линейной диаграммы, созданной на Chart.js

Достоинства:

* высокая скорость обработки большого количества данных;
* возможность сохранить график как картинку

Недостатки:

* отсутствие встроенных настроек стилей;
* небольшое количество предлагаемых типов визуализации;
* неудобный способ установки данных.

Chart.js работает на основе canvas, а не SVG, что сильно ускоряет процесс рендера элементов, но отсутствие базовых настроек стилизации требует от пользователя самостоятельную настройку цветов, что может сказаться на удобстве использования пакетом.

Еще одним конкурентом является библиотека Chartist.

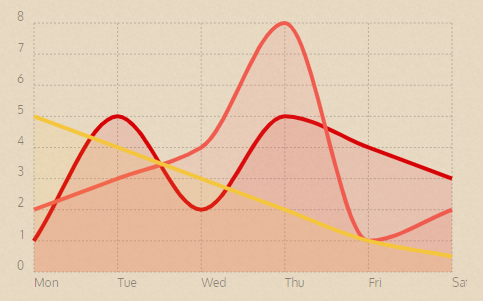


Рисунок 2.3 – Пример диаграммы с областями на Chartist

Достоинства:

* отсутствие зависимостей от сторонних библиотек;
* малый размер;
* простата в использовании.

Недостатки:

* малое количество настроек стилизации;
* встречаются баги в отображении;
* невозможность отображать несколько различных типов визуализации на одном холсте.

Chartist не имеет зависимостей и предоставляет собой очень компактную библиотеку, что делает её идеальным вариантом для рендера небольших и простых графиков. Однако с её помощью не получится отображать сложные графики для больших данных.

Проанализировав трех конкурентов, можно сделать вывод, что основной акцент в библиотеках сделан на внешнем виде, анимациях и на простоте использования (для рендера достаточно заполнить параметры в конфигураторе).

Следующим этапом после анализа конкурентов является изучение средств разработки веб-приложений и компонентов. Одним из самых важных средств являются языки программирования. С их помощью можно описать логику работы всех элементов программы.

Для разработки выбран язык TypeScipt. TypeScript – язык программирования, представленный Microsoft в 2012 году и позиционируемый как средство разработки веб-приложений, расширяющее возможности JavaScript. TypeScript отличается от JavaScript возможностью явного статического назначения типов, поддержкой использования полноценных классов, а также поддержкой подключения модулей.

В ходе разработки достаточно часто используются дополнительные библиотеки для упрощения процесса разработки. Поскольку визуализация данных подразумевает перевод данных в определенный внешний вид, в проекте будет оптимальным выбором использование библиотеки для манипуляции элементами веб-страницы. Для разработки выбрана библиотека D3.js. D3.js – загружаемый пакет, для языков JavaScript и TypeScript, позволяющий управлять документом страницы на основе входных данных.

Также для написания кода используются среды разработки и текстовые редакторы с интеллектуальным автодополнением и подсветкой синтаксиса.

Одним из наиболее распространенных является редактор Atom. Atom –бесплатный текстовый редактор с открытым исходным кодом для Linux, macOS, Windows с поддержкой плагинов, написанных на Node.js, и встраиваемых под управлением Git (см. рисунок 1.4). Atom имеет поддержку большого количества языков, менеджер плагинов, возможность открывать рабочие директории и переключаться между файлами с помощью вкладок. Также данный редактор является достаточно легким и не сильно нагружает систему в ходе работы.

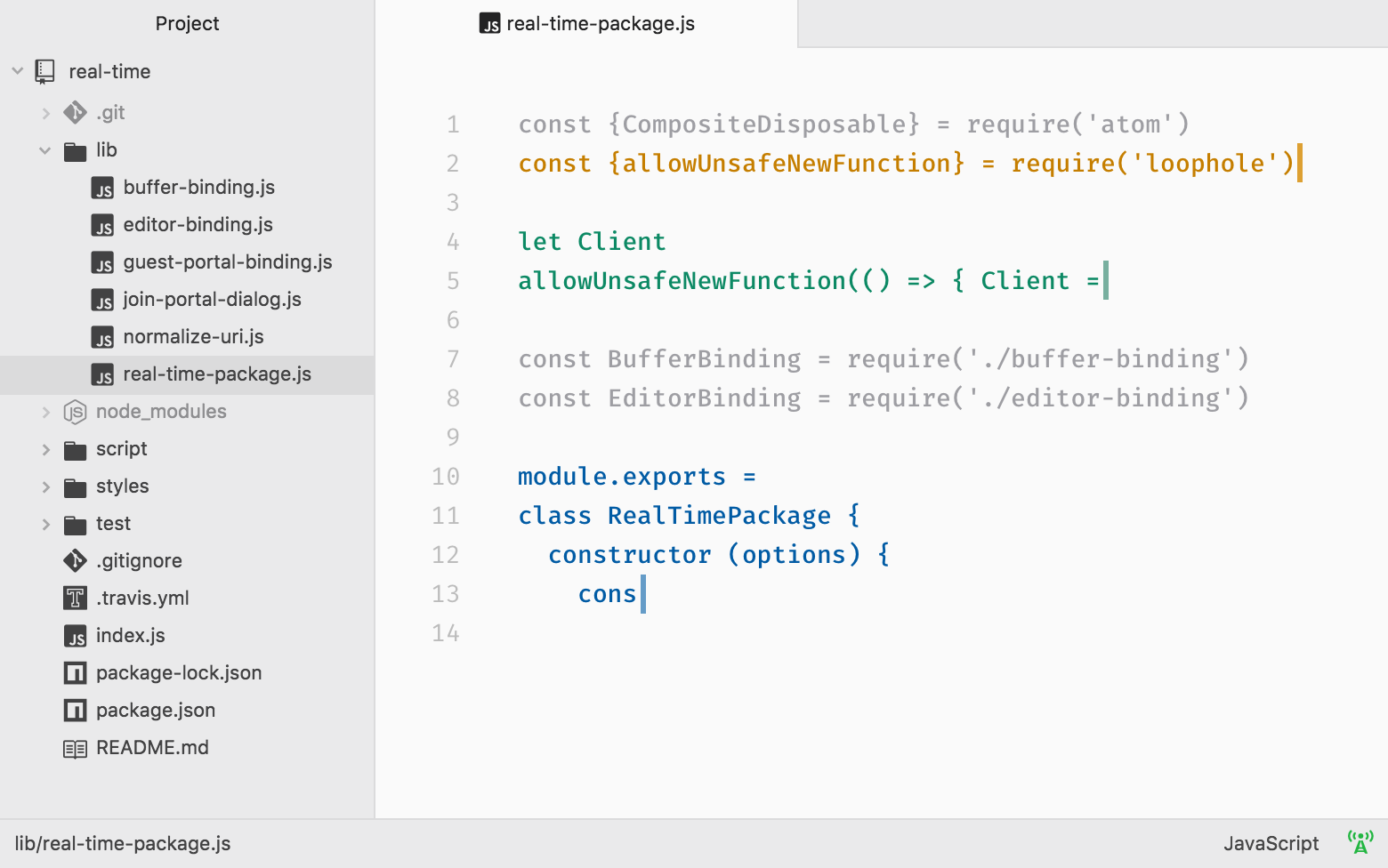


Рисунок 2.4 – Интерфейс редактора Atom

Также одним из самых популярых является редактор Sublime Text. Sublime Text –проприетарный текстовый редактор от разработчика Джона Скиннера (см. рисунок 1.5). Среди предоставляемых возможностей можно выделить быструю навигацию, командную палитру, одновременно редактирование в режиме разделения экрана и пакетный менеджер. Данный редактор является платным.

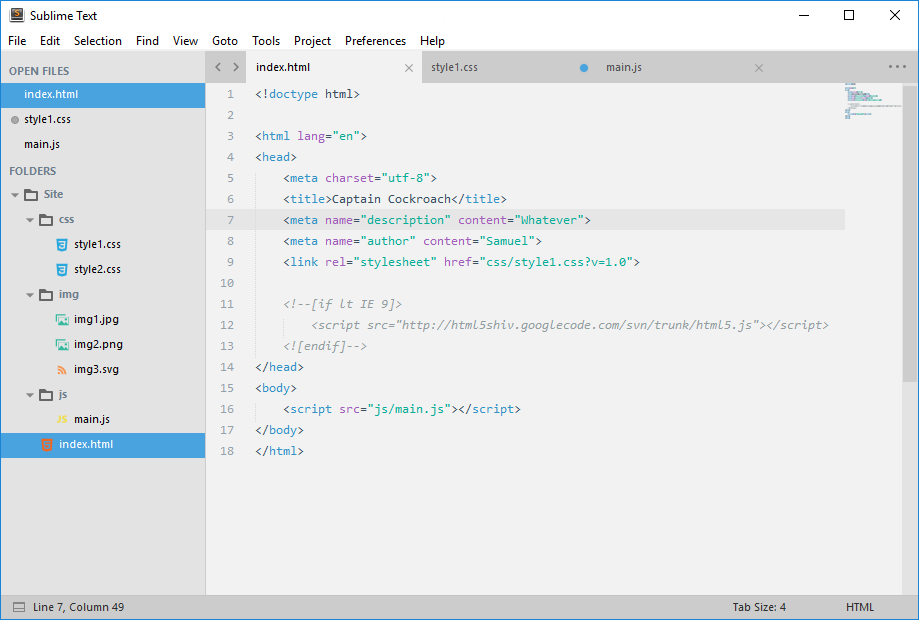


Рисунок 2.5 – Интерфейс редактора Sublime Text

Последний рассматриваемый редактор – Visual Studio Code. Visual Studio Code — редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS (см. рисунок 1.6). Данный редактор имеет поддержку со стороны разработчиков и комьюнити, инструменты отладки рефакторинга и навигации.

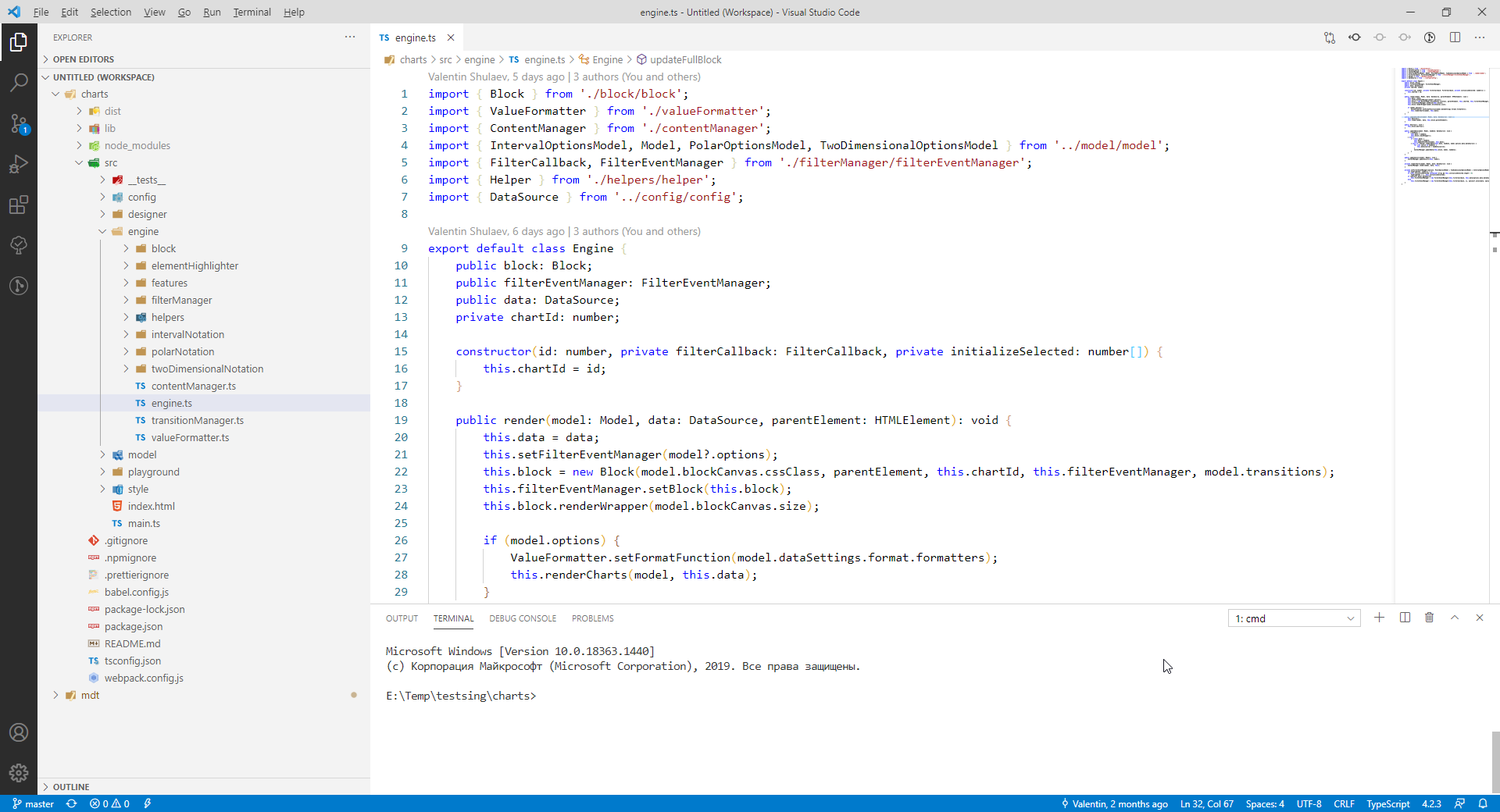


Рисунок 2.6 – Интерфейс редактора Visual Studio Code

Для разработки библиотеки был выбран редактор кода Visual Studio Code, так как он имеет наибольшее количество плагинов, глубокую настройку и множество встроенных помощников для комфортного написания кода.

Выводы по разделу:

а) Основной акцент в библиотеках сделан на внешнем виде и на упрощении процесса использования.

б) Небольшой размер загружаемой библиотеки и количестве настраиваемых элементов привлекает большинство разработчиков.

в) В качестве среды разработки выбран редактор кода Visual Studio Code.

## 3 Проектная часть

Логика работы библиотеки разделена на два больших этапа. Первым этапом является сборка модели на основе входных конфигураторов и данных. Модель заключает в себе всю необходимую информацию для рендера компонентов, а именно: цвета элементов графиков, положение осей, положение легенды, настройки объектов масштабирования, расстояние элементов друг от друга, отображаемые данные, координаты элементов и т. д. Вторым этапом является рендер диаграмм и других компонентов на основе модели. За это отвечает движок библиотеки.

### 3.1 Архитектура библиотеки

Модель и движок инкапсулированы в библиотеке. Пользователь может взаимодействовать с ними через класс Chart. Он является единственным классом, который предоставляет библиотека пользователю. В его конструктор передается 6 параметров: конфигуратор пользователя, конфигуратор дизайнера, отображаемые данные, флаг перерисовки графиков, если изменился размер окна браузера, коллбэк-функция, которая выполняется при выделении компонентов визуализации, а также выбранные для подсветки со старта ключи. Последние три параметра являются необязательными. После инициализации объекта класса Chart, можно вызвать метод render, который принимает родительский элемент. Внутри родительского элемента будет сгенерирован и отображен на странице готовый блок графика. Также класс Chart предоставляет методы для удаления графиков, обновления данных и изменения размера.

Два самых больших компонента библиотеки – сборщик модели и движок. Они находятся в разных директориях, внутри которых расположено множество TypeScript-классов, раскиданных по отдельным файлам.

### 3.2 Логическая модель

В самом начале данные, пользовательский конфигуратор и конфигуратор дизайнера отправляются в сборщик модели. В главном файле сборщика полученная информация раскидывается по различным компонентам, которые представляют собой статические классы. Данные классы предоставляют ряд методов, возвращающие различные части модели. После получения всех частей, методы главного файла собирают их воедино и выдают готовую модель, представляющую собой объект со всей нужной информацией для рендера.

Для получения данных о каждом отображаемом элементе выделен один или два класса в модели. Таким образом в директории расположены классы настроек для элементов графиков, для дополнительных отображаемых элементов, для объектов масштабирования, для используемых цветов, а также для различных деталей самих графиков.

Глобально все типы возможных графиков в библиотеке делятся на три системы исчисления:

* двухмерная система;
* полярная система;
* интервальная система.

К двухмерной системе относятся гистограмма, линейная диаграмма и диаграмма с областями. К полярной относится кольцевая диаграмма. К интервальной относится диаграмма Ганта.

Все эти системы влияют на то, как будут интерпретироваться поступающие данные, чтобы их можно было перевести в конкретные координаты на холсте.

Среди классов для дополнительных элементов присутствуют компоненты модели для легенды, осей и заголовка графика. В данных компонентах происходит расчет координат элементов, текстовое содержимое, их положение, режим отображения и другие детали. Все расчеты происходят на основе данных конфигураторов. Полученные объекты могут дополняться низкоуровневыми свойствами для движка.

Для легенды определяется её положение относительно графика (сверху, справа или снизу), и её размер. Для двухмерной системы исчисления легенда всегда расположена сверху, все её элементы находятся на одной линии и идут друг за другом. Для полярной системы положение легенды определяется размером квадрата, в который вписана кольцевая диаграмма. Если сторона квадрата больше 260 пикселей, а у всего блока холста ширина меньше, чем высота, то легенда будет расположена снизу под самим графиком. В ином случае блок легенда располагается справа от диаграммы. Размер легенды рассчитывается в зависимости от положения её блока относительно графика и ограничен числом, которое задается в конфигураторе дизайнера. Если легенда расположена справа, то её ширина определяется шириной самого длинного её элемента. В расчет идет сам текст элемента цветовой маркер и расстояние между ними. Если легенда расположена снизу, то её высота оценивается исходя из количества всех элементов и размера квадрата, в который вписана кольцевая диаграмма. При расположении сверху размер всегда статичен и ограничивается лишь стандартными показателями, которые идут в расчет во всех случаях (размер шрифта и размер цветового маркера).



Рисунок 3.1 – Варианты расположения легенды

Для осей определяются точные координаты на холсте, расположение каждой оси (сверху, справа, снизу или слева), её видимость, наличие подписей, положение самих подписей, расстояние текста от линии оси, максимальная выделяемая длина для подписи и наличие засечек. Точные координаты левой верхней части оси на холсте рассчитываются исходя из отступа графика от всех соседних элементов и от границы блока (задается в конфигураторе дизайнера). Если ось расположена слева, то её левая координата будет равна отступу графика от левой границы, а верхняя точка складывается из размера легенды (при наличии), заголовка и отступа графика от этих элементов. Расположение оси зависит от настроек, заданных пользователем в конфигураторе. Конфигуратор предоставляет выбор ориентации графика (вертикальная или горизонтальная), а также расположение оси в начале или в конце. Данные настройки работают аналогично флексбоксу из CSS. Таким образом, если ориентация графика вертикальная, и для оси ключей положение задано как «end», то она будет расположена снизу. Следовательно, настройка положения для оси значений в этом случае располагать ось либо слева, либо справа. Наличие засечек определяется настройками из конфигуратора, в модели никаких дополнительных расчетов не производится. Если в конфигураторе ось настроили как невидимую, то при расчете отступа блока с графиками от других элементов размеры подписей оси и её толщина не учитываются.

Также в определенных случаях подписи оси могут быть перевернуты на 90 градусов. Если ось ключей расположена горизонтально, и размер в пикселях, вычисляемый для одного ключа меньше определенного константного значения, то все надписи переворачиваются, чтобы их можно было прочитать (в противном случае они все слишком сильно обрежутся в движке).

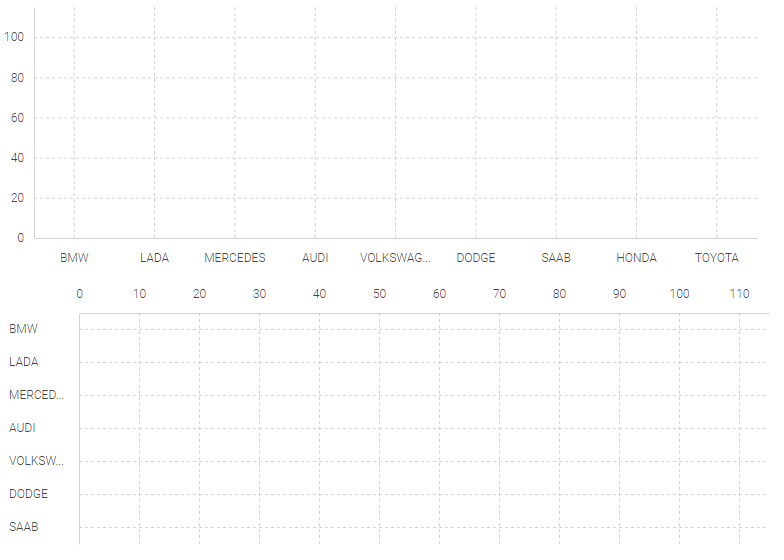


Рисунок 3.2 – Варианты расположения осей

Для заголовка графика высчитывается его положение относительно верхней части родительского блока.

В отдельную группу выделены классы для получения настроек самих графиков. Для каждой системы исчисления свой класс. Данные классы используют уже полученные данные от других компонентов модели. В информации о графиках собирает информация о низкоуровневых объектах масштабирования, типах графиков, читаемых полях из источника данных, CSS-классах, порядковых индексах, и методе отображения полей.

Поскольку в библиотеке предусмотрена возможность отображения нескольких типов графиков на одном холсте, информация о каждом графике собирается в массив объектов. Для каждого объекта индивидуально заносятся все вышеперечисленные параметры. Для расчетов используются данные о дополнительных элементах, упомянутых раннее. Из этих сведений используются показатели размера и отступов для того, чтобы рассчитать точные координаты компонентов самого графика.

Для каждого объекта диаграммы задаются уникальные CSS-классы, чтобы можно было с их помощью определить все компоненты на холсте, принадлежащие ей, и задать определенные стили.

Пользователь может настроить в конфигураторе отображение значений, привязанных к этому графику в тултипе – блоке-подсказке, который всплывает при наведении курсора на какой-либо элемент на странице. Если ни у одного объекта графика не задано отображение тултипа, то он скрывается и не показывается при наведении.

Все используемые цвета для каждой диаграммы собираются в модели. Для этого используется набор базовых цветов, которые задает пользователь в конфигураторе дизайнера. Генерация происходит с помощью библиотеки chroma.js, которая позволяет манипулировать цветами. В функцию передается полученный из конфигуратора массив и количество необходимых цветов. После генерации полученная коллекция подвергается небольшим изменениям в зависимости от типа графика. Так для линейной диаграммы повышается сатурация в 3 раза, чтобы тонкие линии выделялись на фоне других элементов на холсте. Вдобавок среди настроек стилей задается непрозрачность диаграммы. Для всех типов она устанавливается в 100%. Исключение составляет диаграмма с областями, если она является несегментированной, отображает несколько значений, или с ней отображается еще один или несколько графиков на холсте. В таких случаях её непрозрачность составляет 50%.

Также для всех двухмерных графиков в конфигураторе задается опция показа в сегментированном или несегментированном виде. Если график несегментированный, то все поля значений, которые привязаны к нему, отображаются как независимые друг от друга элементы. В противном случае все значения суммируются и полученные компоненты разделяются на категории.

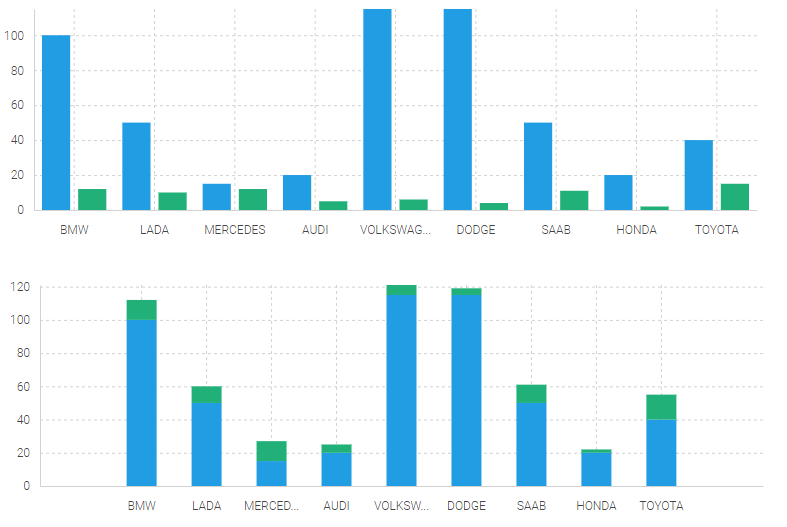


Рисунок 3.3 – Примеры отображения гистограммы в сегментированном и несегментированном виде

Помимо всех вышеперечисленных параметров, для каждого графика также определяются его особенные настройки, характерные только для него.

Для гистограммы можно установить отображение надписей на колонках, если они расположены в горизонтальном виде. Пользователь выбирает в конфигураторе, какую информацию показать. Можно отобразить названия ключей для каждой колонки или значение, которое к нему привязано. Если пользователь выбирает отражение ключей, то выполняется проверка остальных объектов графиков и настройку, отвечающую за встроенные надписи. Если во всех объектах выбрана гистограмма и выбрано отображение ключей, то подписи оси ключей скрываются (при условии, что сама ось настроена, как видимая).

Для линейной диаграммы и для диаграммы с областями можно настроить видимость интерактивных маркеров, расположенных на вершинах точек. Они представляют собой круги, имеющие белую сердцевину и окантовку с тем цветом, который задан для текущего графика.

Для кольцевой диаграммы есть возможность задать расстояние сегментов друг от друга. Значение расстояния никак не преобразовывается в модели и передается в движок в том виде, в котором её задали.

Также в конфигураторе заносятся данные, которые не связаны с компонентами графиков напрямую. Среди них собирается информация об объектах масштабирования и о необходимых преобразованиях для данных, предназначенных для отображения.

Объект масштабирования представляет собой функцию, которая принимает необходимые для визуализации значения и возвращает координаты на холсте для компонента, к которому привязано это значение. В модели задаются настройки, которые будут использоваться при постройке данного объекта в движке. Собираемые опции:

* тип входных данных;
* расстояние, на которое будет проецироваться значение;
* все возможные входные значения (если они дискретные) или диапазон значений (если они непрерывные).

Для каждой оси должен быть свой объект масштабирования.

В начале определяется тип входных данных. Для объекта оси ключей это всегда набор дискретных строк. Поэтому после в поле всех возможных значений заносятся все ключи, переданные в снаружи. Для объекта оси значений входные данные могут представлять собой либо числовые значения, либо даты. Для обоих типов после определения заносится диапазон (самое малое значение и самое большое). Причем для чисел самым малым по умолчанию является нуль, если все переданные числа положительные. В самом конце вычисляется расстояние в пикселях, на которое будут проецироваться все входные в функцию значения. Для горизонтальных осей данное расстояние равно ширине всего блока минус правый и левый отступ, заданные в конфигураторе дизайнера, к данным отступам прибавляется размер текстов, используемых в вертикальных осях и промежутку между подписями и линии оси. Аналогично рассчитывается расстояние для вертикальных осей, однако к верхнему отступу также прибавляется размер легенды (если она отображается), размер заголовка и промежутки между ними. Стоит отметить, что параметры объектов масштабирования вычисляются даже в тех случаях, когда ось не отображается, так как данные объекты нужны для всех компонентов двухмерных графиков.

В модели перед расчетом всех показателей происходит преобразование входных данных. В начале происходит приведение к нужному типу всех значений. Тип задается в объекте графика в настройках отображаемых полей. Так числовые значения, если они переданы в формате строки, парсятся в float тип, а даты преобразовываются с помощью класса Date в языке JavaScript.

После преобразования типов высчитывается количество скрытых записей. Данные записи не будут отображены на холсте, а их количество покажется в специальном сообщении. Оно зависит от типов отображаемых графиков. Если среди выбранных диаграмм есть гистограмма или кольцевая диаграмма, то запускается проверка свободного места и расчет допустимого количества записей.

У гистограммы максимальное допустимое количество записей рассчитывается исходя из минимальной ширины одной колонки, расстоянию между колонками в пределах группы одного ключа (если отображается несколько графиков или у одного несколько полей значений), а также расстояние между группами колонок. Все эти параметры задаются в конфигураторе дизайнера.

Если среди графиков есть гистограмма, то происходит расчет лимита записей. Лимит записей получается из деления имеющегося пространства на ширину одной группы колонок в сумме с расстоянием между ними. Если полученный лимит больше, чем количество отображаемых записей, то в список отображаемых ключей в модели попадают все ключи, а число скрытых записей равно нулю. В противном случае количество записей обрезается до лимита, и оставшиеся ключи заносятся в массив.

Для кольцевой диаграммы обрезка происходит по легенде, если она показывается. Для этого вычисляется положение блока легенды относительно графика и, в зависимости, от результата прикидывается, сколько элементов может поместиться, чтобы при этом сторона квадрата, в который вписана диаграмма оставалась в пределах минимального размера, закрепленного в константе. В расчете создаются виртуальные элементы, которым присваиваются все необходимые CSS-классы. После этого общий размер всех этих элементов суммируется и сравнивается с высотой всего блока, генерируемого библиотекой (если легенда находится справа) или высотой всего блока с вычетом размера заголовка и размера самого графика. После получения максимального значения, в массив также заносятся все допустимые ключи и количество скрытых записей.

В конце полученные данные фильтруются по массиву допустимых ключей. И уже в отфильтрованном виде передаются в остальные компоненты модели и в движок.

### 3.3 Конфигуратор пользователя

Все компоненты конфигуратора занесены в интерфейсы языка TypeScript. Таким образом разработчики, если они используют язык TypeScript, смогут импортировать все интерфейсы из пакета и иметь автодополнение всех свойств каждого объекта.

В конфигураторе есть два самых главных свойства: canvas и options. В свойстве canvas занесена информация о самом вешнем блоке и о SVG элементе, в котором рендерится график.

В свойстве canvas пользователь может задать размер блока, а именно ширину и высоту по отдельности в пикселях. Вторым полем внутри canvas является CSS-класс, который присваивается SVG-блоку. Данное поле представляет собой обычную строку, поэтому пользователь может перечислить несколько классов через пробел.

В свойстве options расположены уже сами настройки графиков и прочих элементов, которые находятся внутри внешнего блока. Доступные поля внутри options меняются в зависимости от выбранной системы исчисления. Поля, внутри options, не зависящие от системы:

* настройки легенды (legend);
* настройки источника данных (data);
* заголовок (title);
* флаг возможности выделения элементов графиков (selectable);
* тип системы исчисления (type);
* настройки содержимого тултипа (tooltip)

В настройках легенды есть единственное свойство – show. Данное свойство относится к булевому типу и отвечает за показ легенды на холсте. В настройках источника данных пользователь заносит названия источника данных (dataSource) и названия поля ключей (keyField). Таким образом модель и движок понимают, какой массив данных из источника взять, и какое поле в объектах записей является ключевым. В поле заголовка задается строка, которая будет отображаться как заголовок графика. Поле selectable относится к булевому типу и при установленном значении true, пользователь сможет на холсте выбирать элементы по полю ключей с помощью клика на кнопку мыши. Данная настройка используется для реализации кросс-фильтрации при построении дашборда – набора графиков, объединенных между собой. В настройках наполнения тултипа содержится поле html, которое является функцией, принимающей один параметр – объект записи. С помощью этого объекта пользователь получает доступ к данным, которые должны быть отображены в тултипе. Возвращает данная функция строку, представляющую собой html верстку тултипа. Таким образом можно настроить показать содержимое в том, виде, которое требуется для конкретного графика. Тип системы исчисления представляет собой строку, которая может быть равна «2d», «polar» или «interval».

Свойство options относится к объединенному типу, состоящему из трех интерфейсов. Это опции для двухмерных графиков, полярных и интервальных. В зависимости от значения поля type объект будет относиться к тому или иному интерфейсу.

Дополнительные поля для options, если выбран двухмерный тип исчисления:

* настройки осей (axis);
* настройки дополнительных элементов (additionalElements);
* ориентация графиков (orientation);
* настройки графиков (charts).

Настройки осей представляют собой объект с полями key и value – ось ключей и ось значений соответственно. Они, в свою очередь, тоже являются объектами, в которых пользователь настраивается видимость с помощью булевого поля visibility, позицию с помощью строки, которая может быть равна «start» или «end», и отображение засечек оси в свойстве flag объекта, который относится к полю ticks. Для оси значений есть еще свойство domain, представляющее собой объект с числовыми полями start и end. Таким образом пользователь может настраивать масштаб, в разрезе которого хочет видеть значения.

Настройки дополнительных элементов – это объект с полем gridLine. В нем пользователь задает отображение линий сетки для каждой оси отдельно с помощью свойств key и value.

Ориентация графиков задается с помощью значений «vertical» или «horizontal».

Поле charts представляет собой массив объектов двухмерного графика. Свойства данного объекта:

* тип графика (type);
* привязанные данные (data);
* тип встроенных в элементы графика подписей (embeddedLabels);
* флаг отображения графика в сегментированном или несегментированном виде (isSegmented);
* отображение маркеров на вершинах графика (markers);
* отображение привязанных данных в тултипе (tooltip)

Поле type может быть равно «bar», «line» или «area» – выбор между гистограммой линейно диаграммой и диаграммой с областями соответственно.

Привязанные данные – объект со свойством valueFields. Данное свойство является массивом объектов, каждый из которых соответствует одному полю значений из источника данных. В объекте одного поля значений находятся свойства для системного имени (имя, которое задано в источнике данных), формата данных, и для псевдонима, который отображается в легенде и в тултипе. В конфигураторе поля называются name, format и title соответсвенно.

Тип встроенных в элементы графика подписей – строка, которая может быть равна «none», «key» или «value». Если задано значение «none», то данные подписи не отображаются. Если задано «key» или «value», то будут демонстрироваться ключи или значения соответственно. Если ориентация вертикальная или сам график является не гистограммой, то в модели данная опция будет принудительно переведена в «none».

Флаг отображения графика в сегментированном или несегментированном виде – булевое поле. Если оно равно true, то все поля значений, который привязаны к диаграмме, будут визуализироваться не в виде самостоятельного элемента, а в виде сегмента одного большого стака. Данную опцию следует устанавливать в true, когда требуется наглядно отобразить общую сумму по нескольким показателям, а также долю каждого отдельного значения.

Отображение маркеров также является булевым полем, значение которого влияет на видимость маркеров на вершинах линейной диаграммы или диаграммы с областями. Данные маркеры являются интерактивными и реагируют на события наведения мыши и клика. Для гистограммы данная опция в модели будет принудительно переведена в false

Последним свойством двухмерного графика является отображение значений в тултипе. Данной свойство является булевым. Если оно равно true, то в тултипе будет отображены значения, которые привязаны к этому графику. Если все во всех объектах графиков в массиве данное свойство равно false, то в движке тултип не будет отображаться.

Единственным дополнительным полем в options в случае, если выбран тип «polar», является свойство chart – объект графика. Свойства графика:

* тип графика (type);
* привязанные данные (data);
* отображение привязанных данных в тултипе (tooltip).

Тип графика является строкой, которая может быть равна только «donut» – единственный доступный график для полярной системы исчисления.

Привязанные данные – объект, в котором задаются настройки для поля значений. Данное поле значений имеет такие же настройки, что и для двухмерного графика (имя, формат, псевдоним). Настройки отображения тултипа полностью идентичны двухмерным графикам.

Дополнительные поля для options, если выбран интервальный тип данных:

* настройки осей (axis);
* настройки дополнительных элементов (additionalElements);
* ориентация графиков (orientation);
* настройки графиков (charts);

Поля имеют те же названия, что у настроек двухмерных графиков, но некоторые из них имеют другой тип данных.

Для оси значений отсутствует настройка выбора минимального и максимального значения, так как для диаграммы Ганта данная функция не нужна. Дополнительные элементы, отображение привязанных данных в тултипе и ориентация полностью аналогичны опциям двухмерного графика.

Свойства интервального графика:

* тип графика (type);
* привязанные данные (data);
* отображение привязанных данных в тултипе (tooltip).

Тип графика – строка, у которой одно доступное значения – «gantt». Настройки отображения тултипа полностью идентичны двухмерным и полярным диаграммам.

Привязанные данные в интервальном графике – объект с двумя свойствами: valueField1 и valueField2. Для обозначения полей, по значению которых начинается и заканчивается элемент диаграммы Ганта соответственно.

### 3.4 Конфигуратор дизайнера

Конфигуратор дизайнера используются для хранения статических настроек для всех графиков дашборда. Если обычный конфигуратор предназначен для предоставления его пользователю, то конфигуратор дизайнера доступен либо для специальных пользователей в системе, либо настроен прямо в коде.

Доступные свойства:

* настройки элементов холста (canvas);
* настройки форматирования данных (dataFormat);
* опции стилей графиков (chartStyle);
* статические настройки элементов (elementsOptions);
* длительности анимаций (transitions).

Настройки элементов холста – объект со следующими свойствами:

* параметры подписей осей (axisLabel);
* управление отступом области графиков от границ холста (chartBlockMargin);
* настройки блока легенды (legendBlock);
* статические настройки для определенных типов графиков (chartOptions).

В параметрах подписей осей находится числовое поле maxSize. Данному значению равна максимальная ширина подписей вертикальных осей. Также это максимальная ширина подписей горизонтальной оси, если они повернуты на 90 градусов. Если блок с текстом становится шире данного значения, то содержимое обрезается.

Свойство chartBlockMargin представляет собой объект с полями: top, bottom, left, right. Все они являются числовыми и отражают отступ области графиков с каждой стороны от границы холста. В конфигураторе дизайнера пользователь задает базовые минимальные отступы. Данные значения в модели могут меняться в зависимости от элементов, расположенных с каждой стороны. Например, отступ сверху в конечном итоге будет состоять из базового значения из конфигуратора, высоты заголовка, высоты легенды (при наличии), а также расстоянию этих элементов от границ и друг от друга.

В объекте настроек блока легенды находится одно числовое свойство для обозначения максимальной ширины блока легенды. Данное значение используются только тогда, когда легенда расположена справа. Если текст какого-либо элемента легенды шире, чем заданное значение, то он переносится.

Статические настройки для определенных типов графиков содержат конкретные параметры для элементов некоторых диаграмм. В конфигураторе дизайнера можно задать настройки для гистограммы (bar) и для кольцевой диаграммы (donut).

Параметры для гистограммы содержат следующие свойства:

* минимальная ширина одной колонки (minBarWidth);
* максимальная ширина одной колонки (maxBarWidth);
* минимальное расстояние между группами колонок (groupMinDistance);
* максимальное расстояние между группами колонок (groupMaxDistance);
* расстояние между колонками в одной группе (barDistance).

Первые два параметра относятся к группам колонок. В одной группе содержатся все колонки, привязанные к одному ключу. Если диаграмма несегментированная и имеет несколько полей значений, или в массиве объектов графиков есть несколько гистограмм, то колонки будут раскиданы по группам. Минимальное и максимальное расстояние между группами колонок используются для разграничения групп в случаях, когда ширина блока слишком высока или наоборот достаточно мала. Расстояние между колонками в одной группе строгое, и в движке всегда используется тот размер, что задан в конфигураторе. Если в группе находится только одна колонка, то данное значение никак не влияет на положение элементов.

Статические настройки для кольцевой диаграммы содержат следующие свойства:

* расстояние сегментов друг от друга (padAngle);
* минимальная толщина (minThickness);
* максимальная ширина (maxThickness);
* отступ агрегатора, находящегося в середине кольца, от самого графика (aggregatorPad).

Расстояние сегментов друг от друга – число с плавающей точкой, которое передается в генератор кольцевой диаграммы в движке. Оно равно доле от радиуса кольца. Полученное после расчетов число представляет собой расстояние между сегментами в фиксированных пикселях.

Минимальная и максимальная толщина задаются для сохранения пропорций графика для разных размеров квадрата, в который вписана диаграмма. Когда величина стороны становится меньше определенного числа, заданного в движке в константе, то толщина генерируемого кольца становится равна минимальной. Во всех остальных случаях она равна максимальной.

Отступ агрегатора от графика – число, обозначающее отступ текстового блока от внутренних границ кольцевой диаграммы. Данный блок расположен посередине кольцевой диаграммы и занимает внутреннее свободное пространство. Чем больше данное расстояние, тем меньше размер шрифта текста агрегатора.

Следующим полем в конфигураторе дизайнера являются настройки форматирования данных. В данном объекте в свойстве formatters задается функция форматирования данных. У этой функции два параметра: значение и дополнительные опции. Дополнительные опции являются объектом, в который пользователь может заносить какую-либо информацию для более глубокого форматирования. Внутри движка во время вызова данной функции передается привязанное значение и объект, внутри которого есть поле type. Значение этого поля подхватывается из конфигуратора (в объекте каждого поля пользователь задает формат значений). На выходе функция выдает строку, представляющую собой отформатированное значение.

В опциях стилей графиков в поле с базовыми цветами (baseColors) передается массив строк, представляющие собой названия цветов. Названия можно передавать в шестнадцатеричном формате, либо в формате RGB, либо в формате RGBA. На основе данных цветов в модели генерируется палитра, использующаяся для окраски элементов графиков и легенды.

Статические настройки элементов представляют собой объект, в котором есть свойство для хранения настроек тултипа (tooltip). В нем пользователь может выбрать способ позиционирования тултипа. Данное поле является строковым, которое может принимать значения «fixed» или «followCursor». Если задано значение «fixed», то тултип будет зафиксирован в определенном положении относительно элемента графика. Если выбрано значение «followCursor», то тултип будет отображаться всегда рядом с курсором с небольшим отступом от его положения.

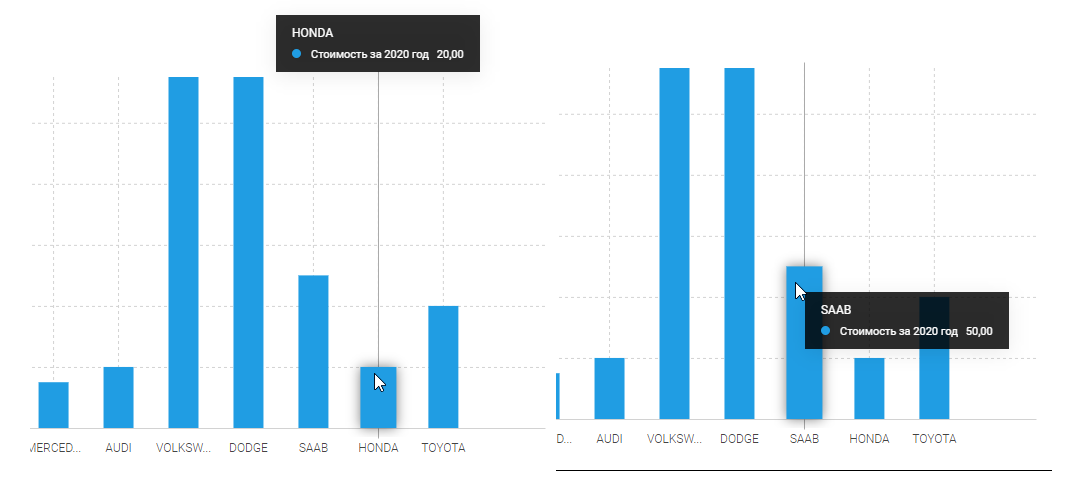


Рисунок 3.4 – Способы позиционирования тултипа

Последним свойством в конфигураторе дизайнера являются опции управления длительностью анимаций. В объекте этих опции есть поля для установления длительности для следующих анимаций:

* обновления графиков после получения новых данных (chartUpdate);
* исчезновение ненужных элементов после получения новых данных путем уменьшения непрозрачности (elementFadeOut);
* переход фиксированного тултипа в новое положение (tooltipSlide);
* увеличение и уменьшение сегментов кольцевой диаграммы во время выделения (donutHover);
* увеличение и уменьшение маркеров линейной диаграммы и диаграммы с областями во время выделения (markerHover).

Длительности всех анимации задаются в миллисекундах. Также каждое поле является необязательным. Если значение какого-либо поля не задано или оно меньше нуля, то в движке устанавливаются длительности по умолчанию. Если в качестве значения продолжительности анимации задан нуль, то анимация не будет проигрываться, элементы будут менять свой внешний вид мгновенно.

### 3.5 Движок

Все внешнее взаимодействие с движком происходит с помощью класса Engine. Данный класс отражает состояние привязанного к нему холста. Все внутренние классы содержатся в папке engine в исходном коде. Всего в неё содержится 50 файлов, в каждом из которых задан один класс. Если есть группа классов, объединенных между собой логически или если они отвечают за один компонент холста, то они заносятся в одну подпапку. Также есть перечень классов, используемых для осуществления каких-либо дополнительных расчетов в ходе рендера. Они все имеют слово helper в названии. Для каждого компонента графика есть как минимум один класс, в котором содержится все взаимодействие со страницей и один вспомогательный класс с суффиксом helper в наименовании.

В директории движка существуют компоненты, которые являются глобальными и используются во множестве методов других классов. Они делятся на общих помощников (helpers), на менеджеров состояния холста и на инструменты трансформации элементов графиков.

Все классы в папке движка можно разделить на следующие группы:

* классы для взаимодействия с блоком холста;
* классы для управления графиками (для каждого типа по отдельности);
* классы для управления дополнительными элементами холста;
* инструменты трансформации элементов графиков;
* классы-помощники, которые предоставляют методы для реализации распространенных типовых задач в движке;
* прочие классы, не попадающие ни в одну из групп.

В группу классов для взаимодействия с блоком холста относятся два компонента: block и blockHelper. Экземпляр класса block содержится как отдельное поле в Engine и передается в почти все остальные классы, так как через него можно получить доступ к HTML и SVG блокам холста.

В конструктор класса block передается уникальный идентификатор холста, родительский элемент, CSS-классы из модели, экземпляр класса менеджера инструментов кросс-фильтрации и длительности анимаций, заданные в конфигураторе дизайнера. Данный класс предоставляет методы для рендера, удаления и получения доступа к HTML элементу-обёртке и SVG элементу, котором находятся все составляющие графиков. Также через данный класс происходит генерация элемента, ограничивающего область видимости элементов и также тег defs, в котором хранятся элементы фильтрации для SVG. Доступ к длительности анимаций и массиву ключей, принадлежащие к которым элементы должны быть визуально выделены, также осуществляется через класс block через свойства transitionManager и filterEventManager.

Класс blockHelper представляет статические методы для получения размеров области видимости графиков и для генерации CSS-классов для элемента-обертки.

Компоненты для управления графиками расположены в трех директориях: twoDimensionalNotation, polarNotation и intervalNotation. В каждой директории находятся классы для типов графиков, используемых в конкретной системе исчисления.

В папке twoDimensionalNotation находятся классы для взаимодействия с гистограммами, линейными диаграммами и диаграммами с областями. Также в этой директории расположен класс twoDimensionalManager, в котором происходит вызов рендера и обновления всех графиков и дополнительных элементов.

Классы для графиков имеют названия area (диаграмма с областями), line (линейная диаграмма) и bar (гистограмма). У каждого из них есть помощник, в котором происходят дополнительные расчеты. Каждый класс двухмерных графиков содержит публичные статические методы для рендера, обновления положения после получения новых данных и обновления цветов. Bar также содержит метод, предоставляющий выборку всех колонок, расположенных на холсте, который передается в качестве параметра.

Поскольку двухмерные графики могут отображаться сегментированно, у каждого класса есть приватные методы для рендера в сегментированном (renderSegmented) и несегментированном (renderGrouped) виде. Аналогично присутствуют методы updateSegmented и updateGrouped. Их вызов происходит в функциях render и update.

В классах-помощниках для линейной диаграммы и диаграммы с областями находятся методы для получения генераторов. Генераторы представляют собой функцию, которая возвращает координаты для атрибута d элемента path внутри SVG-блока на основе полученных данных. Реализация генераторов происходит с помощью объектов масштабирования, которые передаются из twoDimensionalManager. Для сегментированного и несегментированного графика генераторы отличаются, так как в первом случае данные преобразовываются и принимают иной вид, из-за чего доступ к значению по названию поля происходит другим образом.

Класс-помощник для гистограммы предоставляет методы для получения атрибутов колонок на основе входных данных. Они используют объекты масштабирования и возвращают объект со следующими полями: x, y, width, height. Каждое поле представляет собой функцию, которая преобразовывает данные в координаты, а также корректирует полученные числа, если колонка в группе не одна. Внутренняя реализация данных функций отличается в зависимости от того, является ли график сегментированным. Также на значение полей объекта влияет ориентация графика и позиция оси ключей. В случае, если график не является сегментированным и при этом имеет несколько поле значений, перед генерацией функции происходит дополнительный расчет отступа колонки от начала группы. Отступ равен порядковому номеру колонки, умноженному на сумму её ширины и расстояния между колонками внутри группы.

В папке polarNotation находятся классы для взаимодействия с кольцевой диаграммой и класс polarManager, в котором, как и в twoDimensionalManager, происходит вызов рендера и обновления всех графиков и дополнительных элементов.

Класс для кольцевой диаграммы называется donut, и помимо стандартных методов рендера, обновления координат и обновления цветов, он предоставляет методы для получения всех сегментов холста и все их клонов. Клонирование используется для визуального выделения сегмента.

Все клоны создаются в отдельной группе и объединяются внутри тега g SVG-блока. В качестве приватных методов в Donut предоставляются манипуляторы данной группой (рендер и перемещение по верстке), и инструмент рендера новых сегментов, который выделен в отдельную прослойку, чтобы его можно было использовать при первом генерировании графика и его последующем обновлении.

Класс-помощник кольцевой диаграммы предоставляет множество методов для получения какой-либо информации, необходимой для правильного отображения сегментов. Среди них есть функции расчета толщины кольца, координат центра выбранного сегмента, внутреннего и внешнего радиуса, генераторы сегментов на основе радиусов и на основе входных данных.

Как было сказано выше, генераторы сегментов подразделяются на два объекта. Генератор на основе радиусов применяется для элемента g и создает группу, внутри которой все отображаемые круговые элементы имеют определенные радиусы. Генератор на основе входных данных применяется для элемента path, который рендерится внутри группы и имеет конкретные координаты, занесенные в атрибут «d».

Также класс-помощник предлагает метод для манипулирования поступающих данных во время обновления. Он анализирует новые данные и старые и объединяет их в один большой массив, в котором находятся новые записи, старые записи, которые должны быть удалены и старые записи, которые должны быть обновлены. Это используется для осуществления плавного обновления. Во время проигрывания анимации сегменты, привязанные к удаленным записям, плавно сужаются и после полного завершения трансформации удаляются с холста.

В папке intervalManager находятся класс для взаимодействия с диаграммой Ганта и класс intervalManager, используемый для рендера интервальных графиков.

Класс для диаграммы Ганта называется Gantt и он, в отличие, от других графиков, предоставляет только один публичный метод – render, так как обновление диаграммы Ганта полностью удаляет график и производит повторный рендер с новыми данными.

Класс-помощник диаграммы Ганта имеет один метод для получения атрибутов элементов. Атрибуты схожи с теми, которые используются для гистограммы. Отличие заключается в том, что начало и конец элементов графика зависит от входных данных, так как они не имеют единого основания на оси. Для получения начала столбика используются данные, заключенные в объекте data.valueField1 объекта графика из модели, а для конца – данные из объекта data.valueField2.

Классы для управления дополнительными компонентами находятся в директории features и подразделяются на следующие группы:

* агрегатор (aggregator);
* объекты масштабирования (scale);
* оси (axis);
* встроенные в график надписи (embeddedLabels);
* линии сетки (gridLine);
* легенда (legend);
* маркеры (markDots);
* индикатор переполнения записей (recordOverflowAlert);
* область для перехвата событий мыши (tipBox);
* заголовок (title);
* тултип (tooltip).

Класс агрегатора предоставляет публичные методы render и update. В обоих методах в первую очередь создается объект, реализующий интерфейс AggregatorInfo. В данном объекте заполняются поля с названием агрегатора, которое на холсте отображается ниже самого значения (name), самим значением (value) и форматом данных (format). Далее этот объект передается в приватные функции renderText и updateText. Внутри renderText происходит проверка свободного внутри кольцевой диаграммы. Если её внутренний радиус больше 50 пикселей, то происходит генерация всех элементов верстки на холсте. В updateText в первую очередь выполняется поиск элементов агрегатора. Если они были найдены, запускается анимация обновления значения. Во время выполнения анимации значение агрегатора меняется с помощью инкрементирования или декрементирования, пока текст элемента не станет равным новому значению. В конце обоих методов получившийся размер текста сравнивается с внутренним радиусом кольцевой диаграммы с учетом заданного в конфигураторе дизайнера отступа агрегатора от стенок диаграммы. В зависимости от результата, корректируется размер шрифта. Он ограничен и колеблется в диапазоне от 15 до 60 пикселей.

Одними из самых важных компонентов движка являются объекты масштабирования, так как от их настроек зависит позиционирование всех элементов графика и осей. Все взаимодействие с объектами происходит с помощью класса Scale. Его метод getScales возвращает на основе полученных из модели настроек объект, реализующий интерфейс Scales, с полями key и value для масштабирования относительно поля ключей и относительно полей значений соответственно.

Тип объекта масштабирования зависит от значения поля type, полученного из модели. Для поля ключей объект может иметь либо тип «band», либо тип «point». Их отличие состоит в том, где находятся крайние точки оси ключей. Если выбран тип «band», то они расположены с отступом от границ области графиков, равным половине расстояния между ключами. Если выбран тип «point», то крайние ключи расположены у самой границы области графиков. Тип «band» задается, когда среди отображаемых графиков есть гистограмма, в противном случае всегда задается тип «point».

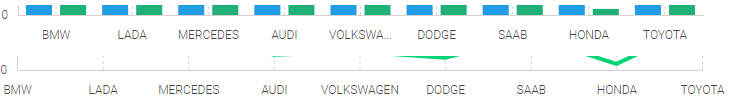


Рисунок 3.5 – Отличия между типами объекта масштабирования ключей на примере оси

Для типа «band» также происходят дополнительные расчеты на основе настроек для гистограммы из конфигуратора дизайнера. В зависимости от минимального и максимального расстояния между группами корректируется параметры paddingInner и paddingOuter (внутренний и внешний отступ). Если свободное пространство, выделенное под один ключ больше, чем требуется для группы колонок, то внутренний отступ увеличивается, пока размеры не станут нужными или пока значение отступа не станет больше числа, заданного в конфигураторе дизайнера. Если внутренний отступ достиг своего максимального предела, а размер свободного пространства под одну группу все еще слишком большое, происходит увеличение внешнего отступа. Таким образом, в случаях, когда размер холста слишком большой, а количество отображаемых записей мало, все ключи будут сконцентрированы в середине холста, а не разбросаны по всей ширине.

Также класс Scale предоставляет публичные методы для получения ширины группы одного ключа, размера шага между ключами в пикселях и точных координат на основе входных данных.

Для работы с осями в движке предоставлено пять классов:

* класс для общего взаимодействия с осями из других компонентов движка (Axis);
* класс-помощник для взаимодействия с холстом (AxisDomHelper);
* класс-помощник для получения результатов расчетов и генераторов (AxisHelper);
* класс-помощник для взаимодействия с подписями готовыми подписями оси на холсте (AxisLabelHelper);
* менеджер событий для подписей (AxisLabelsEventManager).

В классе Axis расположены публичные методы render и update. В render происходит проверка, установлен ли параметр оси visibility из модели как true. Если да, то вызывается метод renderAxis. Данный алгоритм выполняется для каждой оси по отдельности. В приватном методе renderAxis вызывается генератор оси, который применяется для объекта g. У данного генератора устанавливаются параметры отображение засечек и форматирование подписей. После создания, в зависимости от оси, производится преобразование подписей. Для оси ключей проверяется положение подписей, равное строке. Если в модели оно равно «rotated», то они переворачиваются на 90 градусов. После происходит проверка длины подписи в пикселях со свободным местом. Если длина больше, то происходит обрезка текста. В случае, если ось ключей расположена вертикально, перед обрезкой проверяется шаг между каждой подписью, если он достаточно большой, длинный текст переносится на вторую строку. В конце на подписи вешаются события мыши.

Оставшиеся четыре класса предоставляют методы, которые используются внутри Axis. Класс AxisDomHelper содержит метод для обновления оси с анимацией, путем вызова у выборки группы функции-генератора. Также внутри он обновляет атрибут transform у группы, чтобы скорректировать позицию оси на холсте. Вторым методом класса AxisDomHelper является функция восстановления стандартных параметров оси. Он используется во время обновления для случаев, когда количество записей изменилось, что привело к тому, что перевернуть подписи или вернуть их в исходное положение.

Класс AxisHelper предоставляет инструментарий для получения и изменения параметров генератора оси. Генератор с базовыми настройками возвращает функция getAxisByOrient. Необходимые начальные настройки – это положение оси (сверху, слева, справа или слева) и объект масштабирования для неё. Положение оси определяет, с какой стороны относительно линии будут расположены подписи, а объект масштабирование – значения и количество этих подписей. Если ось предназначена для числового типа данных, то для её генератора также применяется метод, устанавливающий форматирование отображаемых значений. Оно зависит от выводимых чисел. Если максимальное число из диапазона больше тысячи, то устанавливается форматирование с помощью функции format библиотеки D3.js.

Класс AxisLabelHelper содержит методы для обработки и трансформации уже отренедеренных подписей на странице. Функция setTitle задает подписям стандартный тултип-подсказку с помощью тега title. Это используется для отображения полной версии обрезанного текста при наведении на него. Также класс предоставляет метод для переворачивания подписей на 90 градусов. К ним применяется трансформация с помощью атрибута transform с правилом rotate(-90), а также для них переопределяются координаты. Методы выравнивания применяются к осям в зависимости от их положения. Если ось расположена вертикально, то все подписи выравниваются по левому краю. Для горизонтальных осей применяется более сложный алгоритм. Все подписи, кроме самых крайних, всегда выравнены посередине. К крайней левой подписи устанавливается выравнивание по правому краю, также смещение в правую сторону с помощью атрибута transform. У самой правой подписи – выравнивание по левому краю и смещение в левую сторону. Размер смещения ограничен шагу между засечками оси и размером самой подписи. Если их текст слишком короткий, или по бокам достаточно много места, то манипуляции с выравниванием не производятся.

Менеджер событий предназначен для того, чтобы вешать на подписи различные события. Среди них есть события передвижения курсора мыши на элементах (mousemove), покидание курсора мыши элемента (mouseleave) и клика левой кнопкой мыши (click). В обработчиках происходит принудительный вызов события на элемент области перехвата событий (tipBox). В качестве класса события используется CustomEvent. В его свойство detail передается объект, в котором перечислены дополнительные детали, необходимые для использования в обработчике другого элемента. В mousemove это значение ключа, к которому привязана подпись и координаты курсора на холсте, а в click – также значение ключа и булевый флаг, сигнализирующий о том, была ли нажата клавиша Ctrl, если пользователь использует операционные системы Window или Linux или клавиша Command – если пользователь использует операционную систему Mac OS.

Все обработчики событий регистрируются в методе setEvents. Когда происходит обновление графиков, события все удаляются с помощью метода removeEvents и заново вешаются после того, как закончится обновление анимации всех элементов оси.

Встроенные надписи – это элементы с тегом text, которые отображаются поверх элементов графиков и предназначены для более удобного чтения данных.

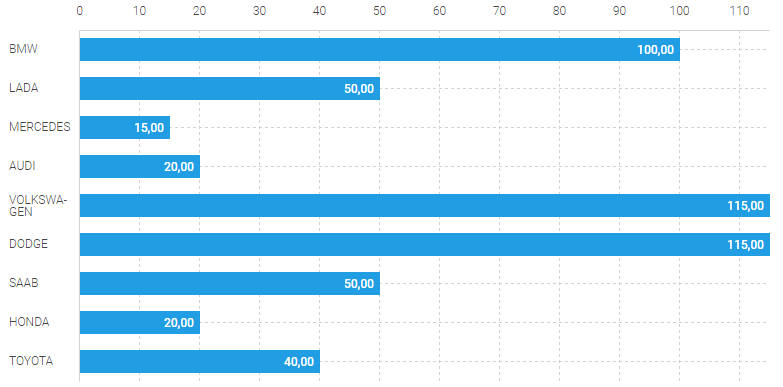


Рисунок 3.6 – Пример гистограммы со встроенными надписями

Все файлы, относящиеся к ним, находятся в директории embeddedLabels. Для работы с этими надписями предоставлено три класса:

* класс для взаимодействия с надписями из других компонентов (EmbeddedLabels);
* класс-помощник для взаимодействия с холстом (EmbeddedLabelsDomHelper);
* класс-помощник для проведения расчетов (EmbeddedLabelsHelper).

Класс EmbeddedLabels предоставляет публичные методы для рендера (render), восстановления надписей после обновления (restoreRemoved), удаления неиспользуемых надписей после обновления данных (removeUnused), обновления положения (update) и перемещение надписей на передний план (raiseGroups).

Внутри метода render происходит обход всех колонок и генерация надписи для каждой колонки. К надписям привязывается строка, данные которой они отображают. После считываются координаты колонки на холсте. Эти данные передаются в методы класса EmbeddedLabelsHelper, чтобы получить параметры отображения: положение внутри или снаружи колонки и координаты. Если надписиь расположена не внутри колонки, а снаружи, то дополнительно для неё рендерится белый полупрозрачный прямоугольник, чтобы черная линия при наведении на элементы графиков не сливалась с текстом надписи. Если элемент отображается внутри колонки, то цвет его текста становится белым, а если снаружи, то он становится черным. Аналогичным образом происходит обновление существующих надписей.

Методы restoreRemoved и removeUnused вызываются во время обновления гистограммы. RemoveUnused используется для уничтожения надписей, привязанных к тем колонкам, которые во время обновления были удалены. restoreRemoved предназначена для разрешения конфликта анимаций, возникающий во время обновления графика при неактивной вкладке. Поскольку анимации реализуются с помощью функции языка JavaScript requestAnimationFrame, они не проигрываются в случаях, когда вкладка браузера неактивна. Вместо этого они скапливаются в специальном списке и сместе реализуются после того, как пользователь заново переключился на вкладку. В ходе этого некоторые лишние надписи могут быть из-за конфликта удалены. restoreRemoved производит рендер недостающих элементов.

Метод raiseGroups выводит группы, в которых объединены надписи на передний план. Это осуществляется с помощью перемещения групп в конец верстки внутри SVG, потому что индекс элемента по оси Z в SVG определяется его положением в верстке.

Класс EmbeddedLabelsDomHelper предоставляет методы для обновления координат одной надписи с анимацией и обрезке текстов. В методе обрезки происходит проверка свободного места для надписи отновительно того, где она расположена. Если надпись элемент расположен внутри колонки, то максимальный размер свободного пространства равно ширине этой колонки, в противном случае размер равен размеру области графиков с вычетом ширины колонки.

Класс EmbeddedLabelsHelper предназначен для сбора информации, необходимой для рендера на основе различных атрибутов других элементов холста и конфигуратора. Таким образом, он предоставляет методы для получения позиции надписи, свободного места в пикселях, низкоуровневых атрибутов, и данных, используемых для отображения.

Метод для получения позиции возвращает строку, которая может быть равна «outside» или «inside». Размещение снаружи колонки или внутри неё соответственно. В начале сравнивается высота колонки с константой. Если высота колонки меньше значения константы, то положение надписи всегда будет снаружи. В противном случае происходит проверка свободного места внутри колонки, свободного места снаружи колонки и ширины надписи. Если места внутри меньше, чем ширина текста, и при этом размер свободного места снаружи больше, чем внутри, то метод вернет «outside». Во всех остальных случаях будет возвращаться «inside».

Размер свободного места рассчитывается исходя из позиции надписи. Если она расположена внутри колонки, то будет возвращена её ширина. Иначе метод вернет ширину холста с вычетом ширины колонки и отступ области графиков.

Метод для получения атрибутов надписи возвращает объект, реализующий интерфейс LabelAttrs. Данный объект имеет поля x, y и textAnchor. Все эти значения применяются к одноименным атрибутам элемента.

Для управления линиями сетки осей используются классы GridLine и GridLineHelper. Каждая линия привязывается к подписи оси. Если ось не отображается, то линии не рендерятся.

Класс GridLine предоставляет публичные методы для ренедера (render) и обновления (update). Для обновления не используется никакая анимация, так как во время отрисовки они привязываются к группам меток оси. Таким образом анимация передвижения линий лежит на плечах класса Axis. Перед обновлением происходит полное удаление с холста и повторный рендер уже для новых меток. Среди приватных методов используются функции для добавления элемента line на холст и применения атрибутов, полученных из класса-помощника и удаление тех линий, которые совпадают со смежной осью.

Класс GridLineHelper содержит методы для получения длины линии и низкоуровневых атрибутов. Длина линии высчитывается на основе ориентации оси, размеров области графиков и размеров отступов этой области от границ холста. Если ось вертикальная, то длина её линий сетки будут равняться высоте холста с вычетом верхнего и нижнего отступа. Если ось горизонтальная, то длина равна ширине холста с вычетом левого и правого отступов. Метод для получения атрибутов возвращает объект, поля которого предназначены для заполнения атрибутов элемента: x1, x2, y1, y2.

Легенда графиков хранится внутри SVG блока в элементе foreignObject, внутри которого расположена стандартная HTML разметка. Каждая строчка легенды – это элемент div, внутри которого находится два эелемента с тегом span. Один используется для цветового маркера. Другой – для самого текста.

Классы для работы с легендой хранятся в директории legend. Всего их 4 штуки:

* класс для взаимодействия с легендой из других компонентов (Legend);
* класс-помощник для обеспечения взаимодействия элементов легенды с холстом (LegendDomHelper);
* класс-помощник для получения значения вычисляемых свойств (LegendHelper);
* класс для управления событиями, навешиваемыми на элементы легенды (LegendEventsManager).

Класс Legend предоставляет четыре публичных метода: рендер (render), обновление элементов (update), обновление цветов (updateColors) и получение элементов легенды по массиву ключей из отображаемых данных (getItemsByKeys).

В методе render происходит проверка позиции легенды, которая передается из модели. Если она не равна значению «off», то выполняются приватные методы renderObject для добавления на холст элемента foreignObject и установки контента внутрь данного элемента (setContent). В методе update повторяется логика из render, только перед установки контента происходит его удаление с помощью приватного метода removeContent.

Метод updateColors используется для обновления цветовых маркеров. Внутри происходит получение новых цветов и приведение его к массиву с помощью функции из класса LegendHelper getMarksColors. После в переменную записываются все элементы, отображённые на холсте, которые передаются в метод setItemsColors из класса LegendDomHelper.

Метод getItemsByKeys предназначен для использования в других компонентах движка для получения элементов легенда для их дальнейшего преобразования.

Рендер всего внутреннего контента легенды происходит в методах setContent и renderContent. В setContent происходит сборка массива всего текстового содержимого элементов, массива цветов для маркеров и способ отображение элементов (в строку или в столбик). После вся полученная информация отправляется в метод renderContent, а на полученные элементы на холсте вешаются события или тултип-подсказка для отображения обрезанного текста.

Внутри функции renderContent создается контейнер, служащий оберткой для элементов легенды. В зависимости от их способа отображения контейнеру задаются определенные стили. После происходит рендер элементов, внутри которых расположены маркер и текст. Маркерам присваиваются цвета из полученного массива, а текст, если элементы отображаются в строку проверяется на длину, и обрезается в случаях, если ширина всей строки с легендой больше, чем ширина холста.

Класс LegendDomHelper содержит методы для установки тултипа-подсказки для элементов, установки цветов для маркеров и обрезки элементов, если они стоят в ряд. В методах установки тултипа-подсказки и цветов задаются атрибут title и CSS свойство background-color соответственно. В методе обрезки текстов элементов происходит обход всех элементов, снятие их показателей ширины отступа друг от друга. Далее сумма всех значений сравнивается с шириной холста. Если сумма больше ширины холста, то запускается массив с предусловием, который останавливается, когда сумма ширин и отступов станет меньше ширины холста или пока текст не обрежется полностью. Обрезке подвергается текстовое содержимое только тех элементов, показатель ширины которых больше, чем значение ширины холста, поделенное на количество элементов. На каждой итерации цикла происходит удаление одного символа с конца и его замена на троеточие (…).

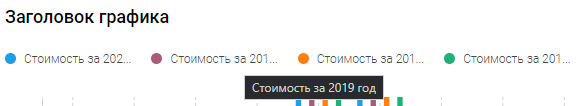


Рисунок 3.7 – Пример отображения элементов легенды в строку

Основные методы класса LegendHelper – это функции для получения текстового содержимого элементов, цветов для маркеров, CSS классов для элементов, способа представления и низкоуровневых координат.

Текстовое содержимое собирается на основе информации из конфигуратора и входных данных. Если отображаемые графики принадлежат двухмерному или интервальному типу, то наполнение представляет собой набор псевдонимов полей значений из конфигуратора. Для кольцевой диаграммы текстовое содержимое представляет собой набор всех ключей из массива записей.

Набор цветов для маркеров тоже зависит от системы исчисления отображаемых графиков. Если они относятся к двухмерным или интервальным, то собираются цвета из каждого объекта графика из модели. Для кольцевой диаграммы просто возвращается набор цветов из объекта единственного графика.

Способ представления зависит от позиции блока легенды. Если блок расположен справа или снизу, то элементы располагаются в столбик. Если легенда расположена сверху, то элементы отображаются в строку. Способ представления влияет на то, какие CSS классы будут применены для каждого элемента. Также на список применяемых классов влияет расположение блока легенды.

Класс для управления событиями предоставит методы для навешивания обработчиков событий мыши на элементы легенды. Обработчики устанавливаются только том случае, когда отображается кольцевая диаграмма. В приватном методе setHoverListeners задаются обработчики на события наведения мыши (mouseover), передвижения курсора по элементу (mousemove) и покидания курсора элемента (mouseleave). Внутри обработчиков происходит принудительный вызов соответствующих событий у секторов диаграммы. Сопоставление элемента легенды и сектора происходит по ключу. Аналогично в методе setClickListeners происходит установка обработчика события клика на элемент легенды.

Для управления маркерами линейной диаграммы и диаграммы с областями используются классы MarkDot и MarkDotHelper. Маркеры представляют собой элементы circle, которые имеют белый цвет и окантовку того же цвета, что и график, к которому они привязаны. Класс MarkDot предоставляет публичные методы для рендера (render), обновления координат (update), обновления цветов окантовки (updateColors), получения маркеров для конкретного графика (getMarkDotForChart).

В методе render происходит привязка данных, получение атрибутов и добавление всех созданных элементов в группу графиков. Данный метод вызывается во время обхода каждого поля значения внутри классов Line и Area. Если данные маркеры отключены, то после рендера они скрываются, так как при наведении или клике на область графиков элементы должны отображаться.

Во время обновления устаревшие точки удаляются с холста, существующим присваиваются новые атрибуты с анимацией, а новые сразу появляются внутри области графиков. Если в конфигураторе маркеры отключены, то новым также присваивается специальный CSS класс для сокрытия точке.

В методе updateColors происходит записывание всех точек данного графика в константу и обновление цвета окантовки с помощью метода setChartElementColor глобального класса-помощника DomHelper.

Метод getMarkDotForChart используется для получения всех точек для определенного графика для трансформации в других компонентах. Он принимает экземпляр класса block для работы с холстом и CSS классы данного графика, по которым можно сделать выборку.

Класс markDotHelper предоставляет единственный метод getDotAttrs, который возвращает объект, реализующий интерфейс DotAttrs. В данном объекте хранится два поля: cx и cy. Они представляют собой функции, которые получают строку данных и выдают число – конкретные координаты, сгенерированные на основе объектов масштабирования и дополнительных расчетов, связанных с отступом области графиков от границ холста.

Функция в поле cx используется для получения координат по оси X, а cy – по оси Y. Их результаты являются значениями соответствующих атрибутов элемента circle.

В модели в ходе преобразования данных некоторые записи могут быть обрезаны. В таких случаях на холсте отображается специальное сообщение, используемое для оповещения о том, что некоторые записи скрыты.

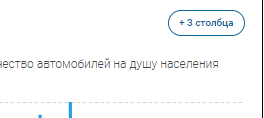


Рисунок 3.8 – Блок оповещения о скрытых записях

Для работы с данным блоком в движке выделен класс RecordOverflowAlert. Он предоставляет два метода – render и update. Первый вызывается, когда происходит первый рендер холста. Второй – во время обновления графиков. Обращение к данному классу происходит только в том случае, если в модели значение поля, содержащее в себе количество скрытых записей не равно нулю.

Также в классе содержатся методы для расчета координат и их подгонке по легенду. Координаты зависят от системы исчисления графиков холста. Для двухмерных и интервальных графиков блок расположен в правом верхнем углу. Для кольцевой диаграммы координата x равна соответствующей координате блока легенды. Следовательно, он может быть расположен как справа от графика, так и снизу. Нижняя координата в этих случаях всегда равна 20 пикселям.

Текстовое наполнение зависит от системы исчисления и ориентации графиков. Для полярных графиков используется слово «категория» с разными окончаниями. Для двухмерных и интервальных графиков используется слово «столбец», если графики вертикальны, и слово «строка», если графики горизонтальны.

В случаях, когда на холсте отображены двухмерные или интервальные графики, следует выводить тултип и перехватывать клики мыши по всей области графиков, без «прицеливания» на конкретные элементы. Для решения такой задачи используется область перехвата событий мыши.

Область перехвата событий мыши представляет собой обычный прямоугольник, создаваемый с помощью элемента rect. Он не имеет заливки и расположен поверх всех графиков. Его размеры равны размерам холста с вычетом отступов области графиков от границ, однако прямоугольник расширен во все стороны на 5 пикселей.

Для управления областью перехвата событий используются классы tipBox и tipBoxHelper. Класс tipBox предоставляет публичные методы renderOrGet, get и clearEvents. Метод renderOrGet предназначен для получения элемента данной области, однако если его не существует, происходит получение атрибутов элемента, его рендер и возврат. Функция get просто возвращает сам элемент области. Метод clearEvents предназначен для очищения событий передвижения курсора мыши по области (mousemove), покидание курсором данной области (mouseleave) и клика левой кнопкой мыши по ней (click). Единственным приватным методом является renderBlock, который предназначен для рендера элемента на холсте.

Класс tipBoxHelper предоставляет вспомогательные методы для класса tipBox, а также методы для поиска ключей из источника данных на основе координат курсора мыши в этой области. Метод для получения атрибутов возвращает объект с полями x, y, width и height. Значения данных полей предназначены для заполнения соответствующих атрибутов элемента rect. Все координаты и размеры вычисляются на основе размера холста и отступа области графиков от границ.

Метод для поиска ключей на основе координат называется getKeyValueByPointer. Он принимает координаты курсора мыши, ориентацию графиков, размеры холста, объект масштабирования для поля ключей, а также тип данного объекта. В зависимости от типа объекта масштабирования область делится на части, с разными размерами. Если тип равен «point», то самая первая часть и самая последняя равны половине всех остальных. Если тип равен «band», то все части равны. Количество частей равно количеству записей из источника данных (без обрезанных). Далее на основе полученных размеров высчитывается индекс доли данной области. По данному индексу находится нужный ключ из массива всех ключей. Его можно получить с помощью метода domain объекта масштабирования. Данный ключ нужен для поиска всех элементов, которые к нему привязаны, чтобы их визуально выделить, а также вызвать функцию-коллбэк, которая передается в класс Chart и применяется для реализации кросс-фильтрации.

Для взаимодействия с заголовком графика используется класс Title. Он предоставляет единственный метод render для добавления его на холст. Внутри данного метода происходит получение координат, максимальной ширины заголовка, а также значение атрибута dominant-baseline элемента text. Координаты применяются для заполнения атрибутов x и y. Максимальная ширина используется для обрезки текста в случаях, когда он слишком длинный. Обрезка происходит с помощью функции cropSvgLabels глобального класса-помощника DomHelper. В самом конце в элемент text добавляется элемент title, чтобы отображать тултип-подсказку при наведении на заголовок. Это используется для получения полной версии заголовка, если на холсте он обрезан.

Для работы с тултипами в движке предоставлены 4 класса:

* класс для взаимодействия с тултипами из других компонентов (Tooltip);
* класс-помощник для обеспечения взаимодействия с холстом (TooltipDomHelper);
* класс-помощник для получения вычисляемых значений (tooltipHelper);
* класс для управления компонентами тултипа (tooltipComponentManager).

Класс Tooltip предоставляет два публичных метода – render и hide. Hide предназначен для простого сокрытия всех компонентов тултипа. В методе render происходит рендер обертки тултипа и проверка модели на наличие графиков с отображаемыми тултипами. Далее, в зависимости от системы исчисления генерируются события на элементы и сами компоненты.

Генерация тултипа для двухмерных и интервальных графиков происходит в методе renderLineTooltip. В начале генерируются составляющие тултипа: внешний блок, контент и линия. Создается переменная, хранящая название ключа, элементы которого должны быть выделены. После вызывается функция renderOrGet класса TipBox, на полученную область перехвата событий вешаются два обработчика событий: передвижение курсора по области (mousemove) и покидание курсора этой области (mouseleave). Внутри обработчика события mousemove в самом начале у объекта самого события проверяется наличие поля keyValue внутри свойства detail. Данное поле будет иметь значение в том случае, если событие было принудительно вызвано при нажатии на подпись оси. В противном случае происходит получение значения ключа с помощью метода getKeyValueByPointer класса TipBoxHelper. После блока тултипа отображается на странице, если он был скрыт. Полученный ключ используется для поиска текущей записи и генерации текстового содержимого блока контента. В зависимости от позиционирования блока ему присваиваются разные координаты (статично конце области графиков или около курсора). После задаются координаты линии. Они должны быть такие, чтобы она указывала на выбранный ключ на оси. В конце происходит обход всех графиков и выделение его элементов с помощью инструментов класса ElementHighlighter. Во время обхода каждого графика также проверяется текущий ключ на его наличие в списке выделенных ключей. От этого зависит способ выделения.

Внутри обработчика mouseleave происходит сокрытие элементов тултипа и снятие выделения у элементов, принадлежащих текущему ключу. Переменная, хранящая ключ, очищается.

Генерация тултипа для кольцевой диаграммы происходит немного по-иному сценарию. В начале генерируется внешний блок, блок контента. Если тултип позиционируется фиксировано, то у него также генерируется стрелка.

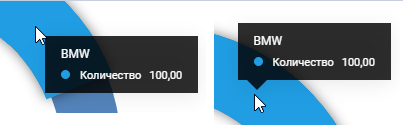


Рисунок 3.9 – Различия внешнего вида тултипа в зависимости от позиционирования

Обработчики события вешаются на каждый сегмент кольцевой диаграммы. Слушатели создаются для событий наведения курсора на сегмент (mouseover) и покидания курсором сегмента (mouseleave). Также, если тултип следует за курсором, создается обработчик события передвижения мыши по сегменту (mousemove). Это используется только для обновления координат блока тултипа, чтобы он всегда был около курсора. В обработчике mouseover обновляется содержимое блока контента. Если тултип отображается фиксировано, то ему задаются такие координаты, чтобы нижняя стрелка указывала на середину сегмента. В конце также происходи визуальное выделение сегмента с помощью инструментов класса ElementHighlighter.

В обработчике события mouseleave происходит скрытие тултипа с холста и, если сегмент привязан к ключу, не находящимся в списке выделенных, происходит снятие всех стилей выделения.

В классе TooltipDomHelper содержатся публичные методы для переопределения координат тултипа (setRecalcedCoordinate) и рендера содержимого для блока контента (fillFor2DCharts и fillForPolarChart). Каждая строка внутри блока контента – это элемент div. Если пользователь в конфигураторе не задал свое собственное содержимое, то в верху тултипа отображается ключ, а ниже – сведения из полей значений. При этом ширина тултипа ограничена в 500 пикселей. Если строка слишком длинная, то происходит перенос слов. Если же пользователь задал свое собственное наполнение, то происходит вызов функции-коллбэка, которая возвращает уже готовую верстку. При этом сохраняются все пробелы и табуляции, проставляемые пользователем.

Все методы класса ToolipHelper направлены для расчета и предоставления координат для тултипа. Для расчета координат используются размеры холста и размеры видимой части браузера пользователя. Если холст находится близко к границам окна, а тултипа должен быть расположен так, что его блок выезжает за пределы видимой части, происходит перерасчет координат. Методы получения координат возвращают объект с полями left, top, bottom, right. Все эти значения отображают отступ от соответствующей стороны на указанное число пикселей. Они заполняют одноименные CSS правила элементов. Причем если с одной стороны задано определенное значение, то с противоположной будет задано значение null. Метод для получения координат линии возвращает объект с полями x1, x2, y1 и y2. Данные значения используются для одноименных атрибутов элемента line.

Класс TooltipComponentManager предназначен для взаимодействия со всеми элементами, из которых состоит тултип (обертка, блок контента, стрелка и линия). Среди них методы для переключения режима видимости элемента, методы для рендера каждого компонента, а также методы для установки координат, полученных из класса TooltipHelper.

Визуальное выделение элементов графиков и интернатов играют очень большую роль в удобстве использования визуализаций. Для того, чтобы менять режим отображения элементов графов, в движке выделены два класса:

* общий класс для управление внешним видом компонентов (ElementHighlighter);
* класс для управления выделением при элементов для кросс-фильтрации на основе инструментов класса ElementHighlighter (SelectHighlighter).

Методы класса ElementHighlighter предназначены для изменения атрибутов элементов и присвоения им определенных CSS классов. Для того, чтобы отобразить элемент неактивным, используется функция toggleActivityStyle. В нем происходит присвоение или удаление CSS класса, который делает элемент полупрозрачным. Для управления тенью выделенных компонентов используются методы setShadowFilter и removeShadowFilter. Для кольцевой диаграммы при выделении создается клон элемента, чтобы отобразить его поверх остальных элементов. Если выделение снимается, то клон должен быть уничтожен. Для данных целей используются методы makeArcClone и removeCloneForElem. Чтобы увеличить сегмент кольцевой диаграммы и его клона или вернуть их в исходное состояние, используется метод toggleDonutHighlightState. Чтобы переключить видимость и размер маркеров линейной диаграммы и диаграммы с областями, используются методы toggleMarkDotVisible и toggleDot. Для управления шириной колонок гистограммы используется метод toggleBar. Для более удобного использования последних трех методов создана функция toggle2DElements.

Класс SelectHighlighter предоставляет всего два метода: click2DHandler и clickPolarHandler. Они предназначены для удобного переключения режима отображения элементов при событии клика на них. Данное выделение нужно для того, чтобы пользователь смог определить в главном графике, по каким ключам отфильтрованы данные в подчиненном графике. Оба метода вызываются только внутри класса менеджера кросс-фильтрации (FilterEventManager).

Менеджер кросс-фильтрации предназначен для визуального выделения элементов графиков, по которым будет происходить фильтрация. Поскольку каждый холст обособлен и не знает ничего другого про другие холсты, все взаимодействие обеспечивается с помощью функции коллбэка. Во время обновления списка выделенных ключей происходит вызов данной функции, содержимое которой полностью задает пользователь. Таким образом можно внутри коллбэка изменить данные другого холста и вызвать у него метод обновления данных.

Экземпляр класса FilterEventManager содержится в качестве свойства внутри класса block. FilterEventManager содержит список всех выделенных ключей холста и задает слушатели события клика на область перехвата событий для двухмерных графиков и на сегменты для полярных графиков. Внутри слушателя проверяется, была ли нажата клавиша Ctrl или Command во время клика, проверка, был ли выделен ключ на момент клика, вызов функции коллбэка и выделение элементов с помощью инструментов класса SelectHighlighter. Если во время клика были нажаты клавиши Ctrl или Command, то происходит множественное выделение, для выбора нескольких ключей. Если ключ уже выделен на момент клика, то он удаляется из списка выделенных ключей и элементы, относящиеся к нему, меняют свой внешний вид.

Четыре класса в движке являются глобальными классами-помощниками. Они содержат методы для реализации алгоритмов, которые часто встречаются в разных компонентах движка. К ним относятся следующие классы:

* класс-помощник для получения вычисляемых свойств и значений (Helper);
* класс-помощник для взаимодействия с холстом (DomHelper);
* класс-помощник для форматирования значений (ValueFormatter);
* класс-помощник для генерации наименований ID и CSS классов для элементов (NamesHelper).

Класс Helper предоставляет группы методов для трансформации строк, задаваемых в качестве CSS классов (получение списка классов через точку из массива, разбиение строки из классов на массив, конкатенация массивов CSS классов), взаимодействия с записями (получение значений ключей из записей, получение записей по ключам, получение ключей по уникальным идентификаторам, глубокое сравнение двух наборов данных), а также для операций над числами (получение суммы из массива чисел, преобразование строки в число, подсчет знаков после запятой, получение значения, если оно больше нуля).

Основная задача класса DomHelper – облегчить процесс получения элементов по каким-либо атрибутам или получениях самих атрибутов. Также данный класс предоставляет методы для задания каких-либо общих свойств для элементов (цвет, CSS классы). Самыми часто используемыми являются методы для получения элементов конкретного графика на основе его CSS классов и фильтрация элементов из готовой выборки на основе ключей, которые привязаны к ним. Последний метод активно используется в классе ElementHighlighter, чтобы получать элементы, ключи которых хранятся или не хранятся в списке выделенных. Также в компоненте присутствуют методы для получения числовых атрибутов выборки и CSS свойств элементов на холсте. Для работы с текстом предлагается метод cropSvgLabels, который принимает выборку элементов, содержащих текст и максимальную ширину, которую они могут иметь. Внутри функции происходит обрезка текста до тех пор, пока ширина элемента не станет меньше или равной максимальной ширине.

Класс NamesHelper предоставляет два метода. Первым является метод генерации CSS класса. Класс складывается из префикса, который хранится в константе и постфикса, приходящего в качестве параметра. Вторым методов является функция для получения id. Идентификатор получается в результате слияния в одну строку префикса, постфикса и идентификатора холста, который можно получить из свойства экземпляра класса Block.

Класс ValueFormatter предоставляет также два метода: setFormatFunction и formatField. Первый метод приравнивает приватному свойству ссылку на функцию форматирования, приходящую из конфигуратора дизайнера. Второй метод принимает значение и его формат, чтобы передать данные параметры в функцию форматирования с помощью приватного свойства.

Чтобы управлять анимациями на холсте, используется класс TransitionManager. Его экземпляр хранится в качестве свойства класса Block. Конструктор данного класса принимает ссылку на холст и объект, содержащий в себе длительности анимаций, заданных в конфигураторе дизайнера. Если объект отсутствует или не заданы какие-то значения у его полей, то анимациям присваиваются стандартные значения, заданные внутри класса. Данный класс предоставляет единственный метод interruptTransitions, который принудительно прерывает все активные анимации, которые проигрывались внутри холста. Данный метод используется перед обновлением элементов и удалением холста.

### 3.6 Дополнительные компоненты библиотеки

Помимо движка и модели среди исходных файлов также расположены CSS файлы и автоматизированные тесты.

Все файлы стилей хранятся в папке style. Их всего три штуки: develop.css, charts-main.css и charts-main.less. В файле develop.css хранятся все стили, используемые в ходе разработки. Именно его содержимое используется в плейграунде. После окончания разработки какого-либо компонента созданные в develop стили перемещаются в файлы charts-main. Данные файлы уже переносятся в директорию lib, которая содержит готовые файлы к публикации на npm.

Все автоматизированные тесты выполнены на языке TypeScript и расположены в директории \_\_tests\_\_. Тесты предназначены для проверки методов компонентов модели и движка. У движка проверяются классы-помощники, функции которых не затрагивают элементы на холсте напрямую. Все тесты созданы с помощью фреймворка для тестирования Jest. Он помогает группировать тесты, просто выполнять их, а также обновлять входные данные с помощью группы методов. Для запуска всех тестов в репозитории выделен скрипт npm run test, который запустит команду jest. По итогам в консоли генерируется список со всеми успешными и проваленными тестами.

Для демонстрации результатов разработки в репозитории хранятся файлы плейграунда. Он представляет собой страницу, на которой расположены примеры графиков, а также различные поля ввода для быстрого изменения полей конфигураторов. Все файлы расположены в папке playground. Внутри находятся:

* JSON-файл с тестовыми входными данными;
* примеры конфигураторов в форматах .ts и .json;
* файл с исходным кодом плейграунда listeners.ts.

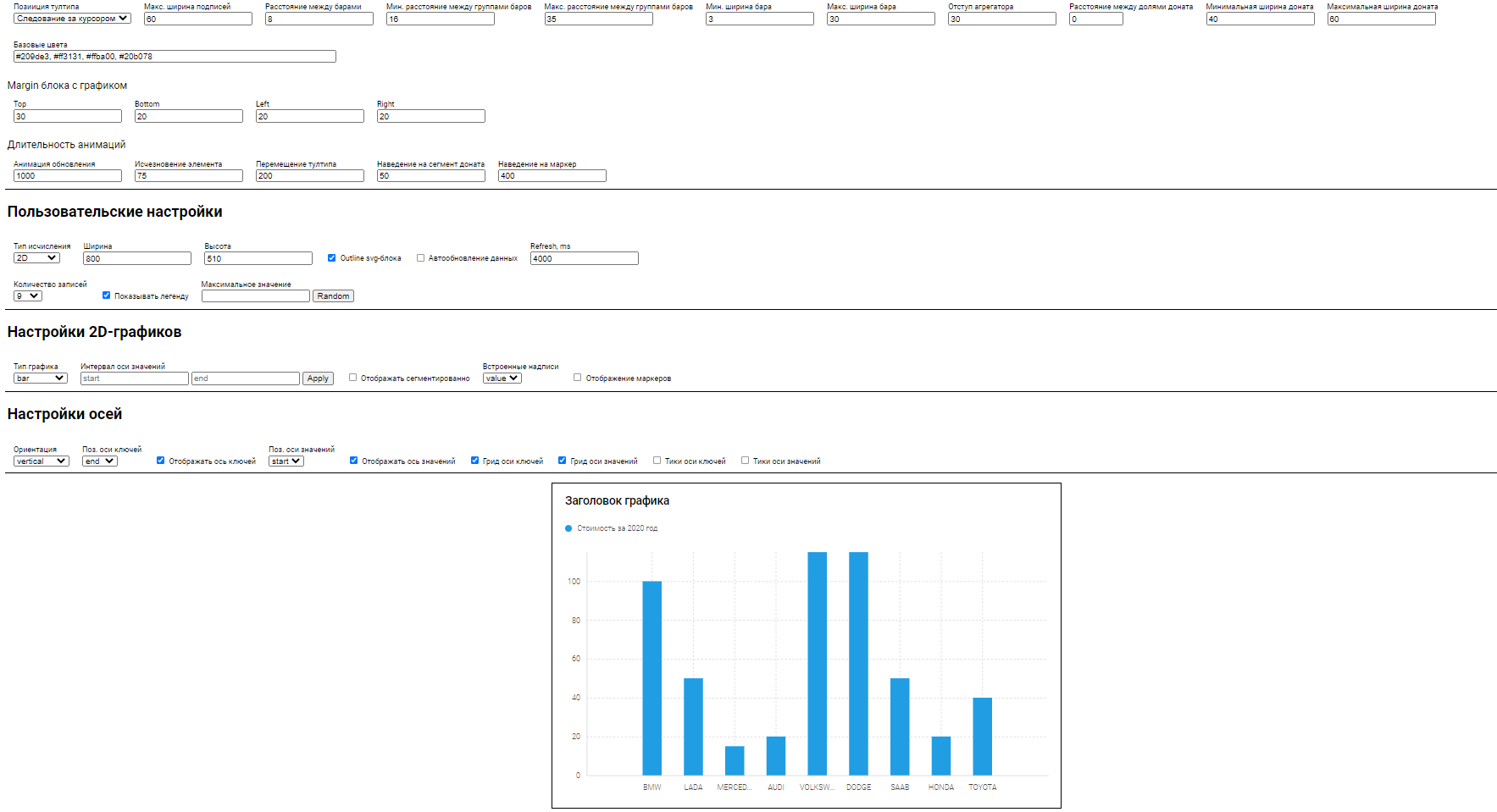


Рисунок 3.10 – Внешний вид плейграунда

Плейграунд доступен по ссылке: <https://vishulaknow.github.io/charts-demo/>.

Исходя из всего вышеописанного, можно сделать следующие выводы:

* Пользователь может управлять графиками с помощью пользовательского конфигуратора и конфигуратора дизайнера;
* Пользовательский конфигуратор предназначен для настройки динамических свойств, которые в ходе работы могут меняться для каждого холста индивидуально, а конфигуратор дизайнера – для настройки статических свойств, которые являются общими для всех холстов на странице;
* Библиотека разделена на две большие части: модель и движок;
* Модель предназначена для вычисления всех показателей, рассчитанных на основе входных данных и содержимого конфигураторов;
* Движок отвечает за генерацию всего контента на странице и использует данные из модели;
* Все внешнее взаимодействие с графиками происходит через класс Chart;
* Класс Chart предоставляет методы для рендера, обновления данных, удаления холста, обновления размеров холста и обновления цветовой палитры элементов, а также принимает пользовательскую функцию для реализации кросс-фильтрации.

## 4 Технико-экономическая часть

### 4.1 Трудовые затраты

Для грамотной оценки всех временных и материальных затрат проекта следует определить трудоемкость работ.

Трудоемкость работ – это показатель, характеризующий затраты живого труда, выраженные в рабочем времени, затраченном на производство продукции. Трудоемкость измеряется, как правило, в нормо-часах (фактических часах работы, затраченных на производство единицы работы).

Чтобы установить длительность реализации проекта, необходимо разделить реализацию на этапы и оценить длительность их выполнения.

В таблице 4.1 представлен список всех работ и их длительность в днях и часах.

Таблица 4.1 – Расчет трудоемкости проекта

|  |  |  |  |
| --- | --- | --- | --- |
| № | Наименование задачи | Длительность, дней | Длительность, часов |
| 1 | Сбор требований | 2 | 16 |
| 2 | Проектирование | 5 | 40 |
| 3 | Разработка | 40 | 320 |
| 4 | Тестирование | 5 | 40 |
| 5 | Внедрение | 3 | 24 |
| 6 | Сопровождение | 5 | 40 |
| Итого | | 60 | 480 |

Для определения трудоемкости произведенных работ была необходимость составления перечня всех ее этапов и определения трудоемкости каждого из них. Трудоемкость всей работы рассчитываться путем суммирования трудоемкостей ее отдельных этапов.

### 4.2 Материальные и технические затраты

Себестоимость продукции - это затраты предприятия на ее производство и реализацию, выраженные в денежной форме.

Стоимость проекта была определена по следующим статьям:

* заработная плата;
* страховые взносы во внебюджетные фонды;
* прочие расходы (затраты на электроэнергия, Интернет).

Затраты заработной плате рассчитываются следующим образом. Расчет трудоемкости проекта производится в часах и за основу выбирается восьмичасовой рабочий день. Трудоемкость работ рассчитывается по формуле 4.1

T=k×n, (4.1)

где k – продолжительность работы, дней;

n – продолжительность рабочего дня, ч.

Для вычисления основной зарплаты требуется знать ставку программиста за час работы, её вычисление происходит по формуле 4.2.

Зпр/ч=Стпр Фвм, (4.2)

где Cтпр — ставка программиста, руб;

Фвм — фонд рабочего времени в месяц, ч.

Ставка программиста-стажера составляет приблизительно 25000 рублей, на основании изучения вакансий работы в городе Чебоксары (без опыта работы). В среднем в одном месяце 22 рабочих дня, значит ставка программиста за 1 час работы составляет: 25000 руб. / 22 дн. / 8 ч. = 142,04 руб. – ставка программиста за час работы.

Вычислить основную зарплату можно по формуле 4.3.

Зос=Зпр./ч×Т (4.3)

Подставив данные, можно выявить основную заработную плату, которая в итоге составляет: 142,04 руб./ч. × 480 ч. = 68179,2 руб.

Следующим шагом определяется размер отчислений в фонд социального страхования, которые составляют 30,2 %, по формуле 4.4.

ФСС=Зобщ×0,302 (4.4)

Тогда размер отчислений в фонд социального страхования будет составлять:

68179,2 руб. × 0,3 = 20453,76 руб.

Все этапы работы над проектом предполагают работу за компьютером, следовательно, для расчета сметы затрат и оценки стоимости проекта требуется учесть стоимость затраченного электричества.

Поскольку в разработке данного проекта были затраты на электроэнергию, приведем в таблице 4.2 о стоимости электроэнергии.

Таблица 4.2 – Данные о стоимости электроэнергии

|  |  |  |  |
| --- | --- | --- | --- |
| Исходные показатели | Единицы измерения | Обозначение | Значение |
| Стоимость 1 кВт электроэнергии в 2021 г. | руб. | Скв | 3,48 |
| Мощность компьютера | кВт/ч | Мк | 0,35 |

Произведем расчеты по формуле 4.5.

Сэл=Мк×Скв×T (4.5)

Вычисляем: 3,48 руб. × 0,35 кВт./ч. × 480 ч. = 584,64 руб., получим итоговую стоимость электроэнергии.

Во время работы над проектом использовался интернет. Интернет провайдер — «Ростелеком». Согласно данным с сайта месячная стоимость по тарифу составляет 890 руб. в месяц.

Так работа над проектом продолжалась 60 дней, то плата за Интернет составила: (890руб. / 30дн.) × 60дн. = 1780 руб.

Все программное обеспечение, кроме операционной системы, бесплатное. При разработке проекта самым оптимальным вариантом будет использование Windows 10 Pro. Стоимость на операционную систему Windows 10 Pro составляет 15000 руб.

Исходя из того, что стоимость компьютера, с учетом основного и периферийного оборудования принимается за 37542 руб. и в течение года компьютер используется 247 рабочих дня по 8 часов, тогда произведем вычисления:

37542руб. / (3года×8часов × 247 дн.) × 480ч. = 3039,83 руб.

Все результаты расчета себестоимости проекта представлены в таблице 4.3.

Таблица 4.3 – Расчет себестоимости и смета затрат на проект программного продукта

|  |  |
| --- | --- |
| Статья затрат | Значение, руб. |
| Заработная плата программиста | 68179,2 |
| Страховые взносы на внебюджетные фонды (ФСС) | 20453,76 |
| Затраты на электроэнергию | 584,64 |
| Затраты на услуги Интернет | 1780 |
| Расходы на обслуживание ЭВМ | 3039,83 |
| Итого | 94037,43 |

Исходя из результатов произведенных расчетов, можно сделать следующие выводы:

а) Для завершения всех этапов разработки потребуется 60 дней при 8-часовом рабочем дне;

б) Полная стоимость данного проекта составляет 94037,43 рублей, что является нормальной ценой для рынка программного обеспечения на момент 2021 года.

в) Разработка данного программного продукта является индивидуальным заказом и установленная денежная стоимость работы полностью устраивает заказчика.

## 5 Охрана труда

Охрана труда - система законодательных актов, социально-экономических, организационных, технических, гигиенических и лечебно-профилактических мероприятий и средств, обеспечивающих безопасность, сохранение здоровья и работоспособности человека в процессе труда.

### 5.1 График работы специалистов

Рабочий день составлял 8 часов. Время начала работы – 9:00. Время окончания – 18:00.

Согласно Федеральному закону от 30.03.1999 г. Л?52-Ф3 (ред от 19 07.2018) «О санитарно-эпидемиологическом благополучии населения» условия работы с машинами, механизмами, установками, устройствами. аппаратами. которые являются источниками физических факторов воздействия на человека (шума, вибрации, ультразвуковых, инфразвуковых воздействий, теплового, понтирующего, неионизирующего и иного излучения), не должны окалывать вредное во (действие на человека, порядок организации работы лиц. труд которых связан с компьютерной техникой регулируется Постановлением Главного государственного санитарного врача РФ от 03.06.2003 г №118 (в ред. от 21.06.2016 г ) «О введении в действие санитарно-эпидемиологических правил и нормативов СанПин 2 2.2/2 4 1340-03».

Согласно Постановлению, организации работы с компьютерной техникой осуществляется в такие имеет и от вида и категории трудовой деятельности.

Виды трудовой деятельности подразделяются на следующие:

* группа А работа по считыванию информации с экрана по сделанному запросу;
* группа Б – работа по вводу информации;
* группа В – творческая работа в режиме диалога с персональной электронной вычислительной машиной (ПЭВМ).

Для видов трудовой деятельности устанавливаются определенные категории тяжести и напряженности. Для группы А – по суммарному числу считываемых знаков за рабочую смену, но не более 60 000 знаков за смену. Для группы Б – по суммарному числу считываемых или вводимых знаков за рабочую смену, но не более 40 000 знаков за смену. Для труппы В по суммарному времени непосредственной работы с ПЭВМ за рабочую смену, но не более 6 ч за смену.

В ходе работы специалисты использовали перерывы по 15 минут через каждые 2 часа работы.

### 5.2 Организация рабочего пространства

Площадь на одно рабочее место пользователей ПЭВМ должно составлять не менее 4.5 м2 Мри использовании ПВЭМ без вспомогательных устройств (например, принтера сканера), отвечающих требованиям международных стандартов безопасности компьютеров, с продолжительностью работы менее четырех часов в день допускается минимальная площадь 4.5 м2 на одно рабочее место.

Для внутренней отделки интерьера таких помещений должны использоваться диффузно отражающие материалы со следующим коэффициентом отражения:

* для потолка 0.7-0.8;
* для стен 0.5 0.6;
* для пола 0.3 0.5.

Помещения с ПЭВМ должны быть оборудованы защитным заземлением в соответствии с техническими требованиями по эксплуатации

Экран монитора должен находиться от глаз пользователя на расстоянии 600-700 мм, но не ближе 500 мм.

При планировке в помещение должно быть естественное н искусственное освещение. Окна в помещениях преимущественно должны быть ориентированы на север и северо-восток. Оконные проемы оборудуются регулируемым устройствами (жалюзи. внешних козырьков и др.).

Конструкция рабочего стола должна обеспечивать оптимальное размещение на рабочей поверхности используемого оборудования с учетом его количества и конструктивных особенностей, характера выполняемой работы При этом допускается использование рабочих столов различных конструкций. отвечающих современным требованиям эргономики.

Высота рабочей поверхности стола для сотрудников должна регулироваться в пределах 680-800 мм (при отсутствии такой возможности – должна составлять 725 мм).

### 5.3 Характеристики используемого помещения

Освещение помещения бывает естественным и искусственным. Естественный свет наиболее ценен, так как человеческий глаз к нему максимально приспособлен. Такой свет поступает внутрь помещения через окна или прозрачные строительные конструкции.

В основном, условия освещения определяются диффузорным светом небосвода, а также отраженным светом, полученным в результате отраженного излучения от земной поверхности. Помещения, в которых происходит рабочий процесс, обязательно должно иметь естественное освещение.

Для того, чтобы обеспечить условия, необходимые дтя зрительного комфорта га рабочем месте, в системе освещения должны быть реализованы следующие требования:

* однородное освещение;
* оптимальная яркость;
* отсутствие бликов;
* соответствующая контрастность;
* правильная цветовая гамма;
* отсутствие стробоскопического эффекта или мерцания света.

Если осветительные приборы размещены так, чтобы обеспечить достаточную освещенность в зоне работ, то это называют общим искусственным освещением. Если помимо общего освещения на рабочих местах используются дополнительные осветительные приборы, то это считается комбинированным искусственным освещением.

Расчет искусственного освещения связан с определением типа и количества светильников для обеспечения нормируемой величины освещенности рабочих мест. Для этого может быть использованы три метола:

* метод коэффициента использования светового потока;
* точечный метод;
* метод удельной мощности.

### 5.4 Пожарная безопасность

Пожарная безопасность – состояние объекта, при котором исключается возможность пожара, а в случае его возникновения предотвращается воздействие на людей опасных ею факторов и обеспечивается защита материальных ценностей.

Пожары в помещениях, оснащенных компьютерной техникой, представляют особую опасность, так как сопряжены с большими материальными потерями Характерная особенность таких помещений – небольшая их площадь, и наличие факторов, необходимых для возникновения пожара (взаимодействие горючих веществ, окисление и наличие источников зажигания). При нарушении техники эксплуатации или сбоя работы, образуются перегретые элементы, электрические искры и дуги. Все это способно вызвать возгорание

Согласно СП 12.13130.2009 «Определение категорий помещений, зданий и наружных установок по взрывопожарной и пожарной опасности», помещение. в котором размещены компьютеры или серверы. по категориям пожарной опасности относится к категории В.

Одной из наиболее важных задач пожарной защиты является защита помещений от разрушений и обеспечение их достаточной прочности в условиях воздействия высоких температур при пожаре. Основной причиной возникновения пожаров являются короткие замыкания оборудования. перепады напряжения и т.д.

Для тушения пожаров на начальных стадиях широко применяются огнетушители. в случае с компьютерной техникой – углекислотные огнетушители. Такие огнетушители возможно использовать лаже в том случае, если электроустановка не обесточена сразу.

При возникновении пожара или другой нештатной ситуации необходимо следовать положению но охране труда и выполнять действия, предписанные данным документом. В большинстве случаев это связано с принятием мер по эвакуации людей, тушению пожара и сохранности материальных ценностей. Эвакуация людей осу шест ваяется согласно плану эвакуации.

План эвакуации – документ, в котором указаны эвакуационные пули и выходы, установлены правила поведения людей, а также порядок и последовательность действий обслуживающего персонала на объекте при возникновении чрезвычайной ситуации.

План эвакуации подразумевает графическую и текстовые части. Текстовая часть содержит информацию о способах оповещения, порядок и последовательности эвакуации, обязанности и действия людей, порядок аварийной остановке оборудования, порядок включения автоматики. На графической части показан план помещения и пути эвакуации с указанием эвакуационных выходов, огнетушителей, средств защиты и т.д.

Охрана труда является одной из важнейших составляющих полноценного трудового процесса.

Работа с сотрудниками организации по охране труда является одним из основных направлений деятельности, обеспечивающей безопасность, надежность и эффективность работы организации.

В рамках дипломного проекта рассмотрены основные положения работ с компьютерной техникой:

* определен график рабочего дня с учетом интенсивности работ и предоставлением перерывов;
* организация рабочего пространства с определением минимальной площади рабочего пространства;
* определены нормы освещенности помещения с расчетом количества осветительных приборов на соответствующую площадь помещения;
* изучены норма по созданию комфортного микроклимата для помещения;
* изучены вопросы по созданию комфортного микролимата для помещения при чрезвычайных нештатных ситуациях.

Соблюдение норм по охране труда позволит обеспечить не только комфортные условия для труда, повысить его производительность, но и сохранить здоровье сотрудников.

# ЗАКЛЮЧЕНИЕ

Цель дипломного проекта была полностью достигнута: была разработана библиотека для визуализации данных.

B ходе выполнения дипломного проекта были выполнены следующие задачи:

* анализ и оценка конкурентов на рынке;
* составление технического задания на разработку библиотеки;
* разработка модулей библиотеки;
* тестирование модулей;
* публикация библиотеки;
* расчет экономических показателей проекта;
* сопровождение программного продукта.

В экономической части дипломного проекта был произведен расчет себестоимости программного продукта, которая составила 94037,43 руб. Длительность всего процесса проектирования и разработки библиотеки составила 60 рабочих дней.

Библиотека позволяет разработчикам легко отображать на странице различные виды визуализации данных. Управление настройками происходит с помощью конфигуратора пользователя и конфигуратора дизайнера.

Также в результате работ был создан плейграунд, предназначенный для демонстрации возможностей библиотеки. Плейграунд доступен по ссылке: https://vishulaknow.github.io/charts-demo/.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

СДЕЛАТЬ

# ПРИЛОЖЕНИЕ А

## Техническое задание на разработку

|  |  |
| --- | --- |
| Общие сведения | |
| Назначение документа | Настоящее Техническое задание определяет требования и порядок разработки библиотеки для построения визуализации данных |
| Заказчик | ООО «Интеллектуальные корпоративные решения» |
| Исполнитель | Шулаев Валентин Николаевич |
| Основание для разработки библиотеки | Задание на дипломное проектирование |
| Плановые сроки начала и окончания работ по созданию библиотеки | Начало работ: 19.04.2021 Окончание работ: 11.06.2021 Содержание и длительность отдельных этапов работ приведены в разделе настоящего ТЗ. Сроки, состав и очередность работ являются ориентировочными и могут изменяться по согласованию с Заказчиком |
| Порядок оформления и предъявления результатов работ | Работы по созданию сайта производятся и принимаются поэтапно. По окончании каждого из этапов работ, перечисленных в разделе настоящего ТЗ, Исполнитель представляет Заказчику соответствующие результаты, и стороны подписывают Акт сдачи-приемки работ |
| Назначение и цель создания библиотеки | |
| Назначение библиотеки | Библиотека должна предоставлять инструментарий для генерации визуализации входных данных. Настройка диаграмм происходит с помощью пользовательского конфигуратора и конфигуратора дизайнера. |
| Цель создания библиотеки | Цель создания библиотеки – облегчить процесс создания диаграмм в приложении и переиспользования алгоритма генерации визуализации данных в разных модулях приложения или сайта. |
| Основные требования к библиотеки | |
| Структура библиотеки | Библиотека принимает входные данные, которые необходимо визуализировать и два конфигуратора. Исходные файлы библиотеки относятся к математической модели или к движку. Модель на выходе предоставляет объект с информацией, необходимой движку. |
| Требования к дизайну | |
| Общие требования | Стиль генерируемых диаграмм можно описать как современный. Используемые цвета должны генерироваться на основе базовых цветов, получаемых из используемой темы в приложении. |
| Требования к информационному обеспечению | |
| Требования к языкам программирования | Для реализации процесса построения диаграмм используются языки программирования TypeScript и JavaScript. Верстка элементов осуществляется с помощью языка разметки HTML. Для описания стилей используется формальный язык описания внешнего вида документа CSS. Также при разработке используются инструменты библиотек D3.js и chroma.js. |
| Требования к программному обеспечению | Программное обеспечение клиентской части должно удовлетворять следующим требованиям:  Веб-браузер Firefox 35 или выше, Google Chrome 18 или выше, Edge 79 или выше. |
| Порядок контроля и приемки библиотеки | |
| Виды, состав, объем и методы испытаний | Сдача-приемка выполненных работ должна осуществляться при предъявлении Исполнителем комплектов соответствующих документов и завершаться оформлением акта сдачи-приемки, подписанного Исполнителем и утвержденного Заказчиком. Тестирование веб-приложения проводится силами Исполнителя. |
| Общие требования к приемке библиотеки | Приемка библиотеки должна проводиться приемочной комиссией, в состав которой должны входить представители Заказчика и Исполнителя, в течение трех рабочих дней после завершения работ. Результаты работы комиссии должны оформляться актом, подписанным членами комиссии и утверждённым Заказчиком. |

# ПРИЛОЖЕНИЕ Б

## Исходный код интерфейсов конфигураторов

Исходный код интерфейсов пользовательского конфигуратора:

type DataType = 'integer' | 'decimal' | 'date' | 'money' | 'string';

export type AxisPosition = 'start' | 'end';

export type ChartOrientation = 'vertical' | 'horizontal';

export type ChartNotation = '2d' | 'polar' | 'interval';

export type ChartType = 'bar' | 'line' | 'area' | 'donut' | 'gantt';

export type TwoDimensionalChartType = 'line' | 'bar' | 'area';

export type PolarChartType = 'donut';

export type IntervalChartType = 'gantt';

export type EmbeddedLabelType = 'none' | 'key' | 'value';

export type TooltipHtml = (dataRow: DataRow) => string;

export type DataRow = {

[field: string]: any

}

export interface DataSource {

[source: string]: DataRow[];

}

export interface Config {

canvas: ChartBlockCanvas;

options: PolarOptions | TwoDimensionalOptions | IntervalOptions;

}

export interface ChartBlockCanvas {

size: Size;

class: string;

}

export interface Size {

width: number;

height: number;

}

interface Options {

legend: Legend;

data: DataOptions;

title: string;

selectable: boolean;

tooltip?: TooltipOptions;

}

export interface TwoDimensionalOptions extends Options {

type: '2d';

axis: TwoDimensionalAxis;

additionalElements: AdditionalElements;

charts: TwoDimensionalChart[];

orientation: ChartOrientation;

}

export interface PolarOptions extends Options {

type: 'polar';

chart: PolarChart;

}

export interface IntervalOptions extends Options {

type: 'interval';

axis: IntervalAxis;

chart: IntervalChart;

additionalElements: AdditionalElements;

orientation: ChartOrientation;

}

export interface Legend {

show: boolean;

}

export interface DataOptions {

dataSource: string;

keyField: Field;

}

interface Field {

name: string;

format: DataType;

}

export interface ValueField extends Field {

title: string;

}

export interface TooltipOptions {

html: TooltipHtml;

}

export interface AdditionalElements {

gridLine: GridLineOptions;

}

interface GridLineOptions {

flag: GridLineFlag;

}

interface GridLineFlag {

key: boolean;

value: boolean;

}

export interface TwoDimensionalAxis {

key: DiscreteAxisOptions;

value: NumberAxisOptions;

}

export interface AxisOptions {

visibility: boolean;

position: AxisPosition;

ticks: AxisTicks;

}

interface AxisTicks {

flag: boolean;

}

export interface NumberAxisOptions extends AxisOptions {

domain: NumberDomain;

}

export interface NumberDomain {

start: number;

end: number;

}

export interface DiscreteAxisOptions extends AxisOptions { }

export interface IntervalAxis {

key: DiscreteAxisOptions;

value: DateAxisOptions;

}

interface DateAxisOptions extends AxisOptions { }

interface ChartSettings {

tooltip: Tooltip;

}

interface Tooltip {

show: boolean;

}

export interface TwoDimensionalChart extends ChartSettings {

type: TwoDimensionalChartType;

data: TwoDimensionalChartData;

embeddedLabels: EmbeddedLabelType;

isSegmented: boolean;

markers: MarkersOptions;

}

export interface PolarChart extends ChartSettings {

type: PolarChartType;

data: PolarChartData;

}

export interface IntervalChart extends ChartSettings {

type: IntervalChartType;

data: IntervalChartData;

}

export interface TwoDimensionalChartData {

valueFields: ValueField[];

}

interface MarkersOptions {

show: boolean;

}

export interface PolarChartData {

valueField: ValueField;

}

interface IntervalChartData {

valueField1: ValueField;

valueField2: ValueField;

}

Исходный код интерфейсов конфигуратора дизайнера:

export type DataType = 'integer' | 'decimal' | 'date' | 'money' | 'string';

export type DataTypeOptions = {

[option: string]: any

}

export type Formatter = (value: any, options?: any) => string;

export type TooltipPosition = 'followCursor' | 'fixed';

export interface DesignerConfig {

canvas: Canvas;

dataFormat: DataFormat;

chartStyle: ChartStyleConfig;

elementsOptions: ElementsOptions;

transitions?: Transitions;

}

interface Canvas {

axisLabel: AxisLabelCanvas;

chartBlockMargin: BlockMargin;

legendBlock: LegendBlockCanvas;

chartOptions: ChartOptionsCanvas;

}

export interface AxisLabelCanvas {

maxSize: AxisLabelSize;

}

interface AxisLabelSize {

main: number;

}

interface BlockMargin {

top: number;

bottom: number;

left: number;

right: number;

}

export interface LegendBlockCanvas {

maxWidth: number;

}

interface ChartOptionsCanvas {

bar: BarOptionsCanvas;

donut: DonutOptionsCanvas;

}

export interface BarOptionsCanvas {

minBarWidth: number;

maxBarWidth: number;

groupMinDistance: number;

groupMaxDistance: number;

barDistance: number;

}

export interface DonutOptionsCanvas {

padAngle: number;

minThickness: number;

maxThickness: number;

aggregatorPad: number;

}

interface DataFormat {

formatters: Formatter;

}

export interface ChartStyleConfig {

baseColors: string[];

}

export interface ElementsOptions {

tooltip: TooltipSettings;

}

export interface TooltipSettings {

position: TooltipPosition;

}

export interface Transitions {

chartUpdate?: number;

elementFadeOut?: number;

tooltipSlide?: number;

higlightedScale?: number;

markerHover?: number;

}

# ПРИЛОЖЕНИЕ В

## Примеры графиков, генерируемых библиотекой



Рисунок В.1 – Пример кольцевой диаграммы

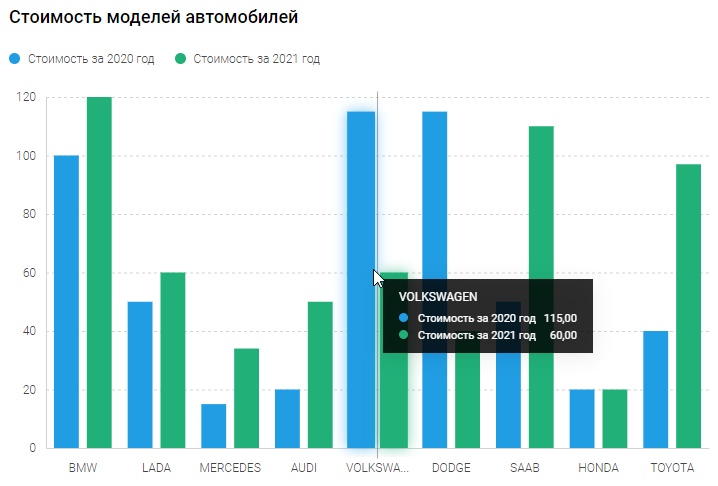


Рисунок В.2 – Пример столбчатой диаграммы

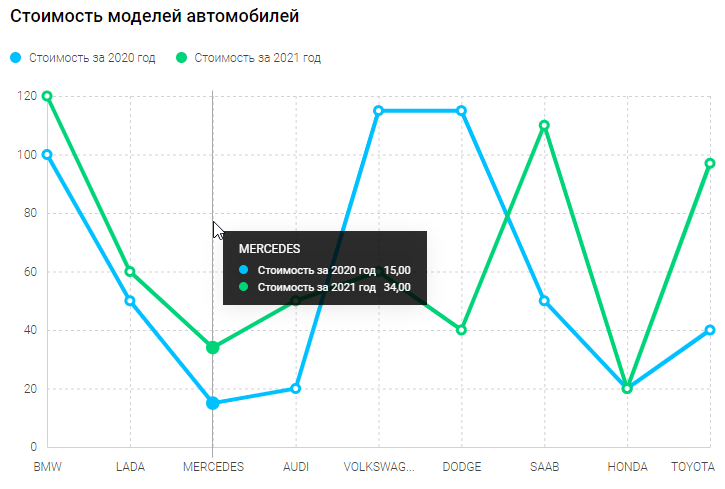


Рисунок В.3 – Пример линейной диаграммы

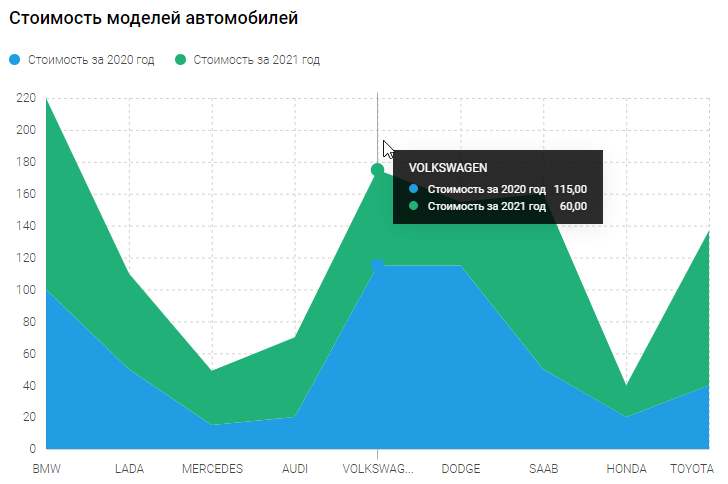


Рисунок В.4 – Пример диаграммы с областями

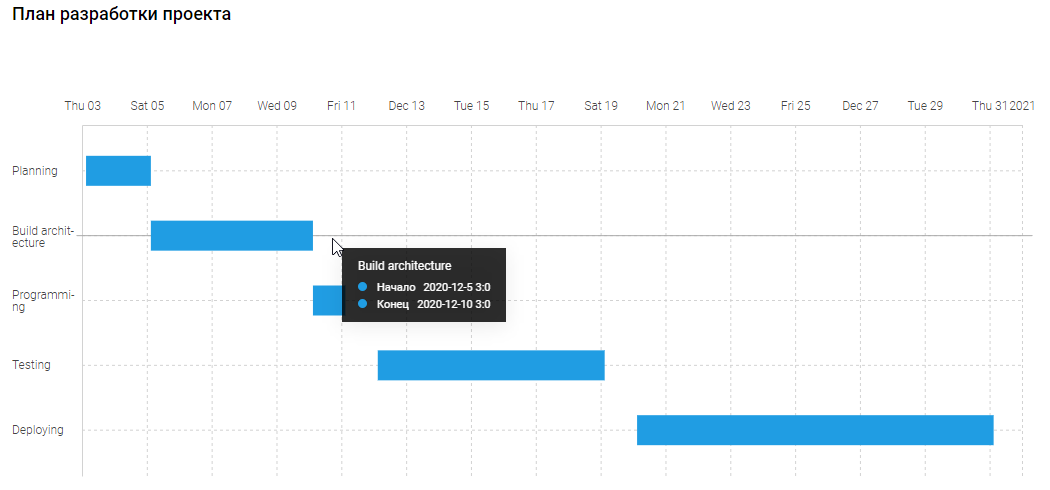


Рисунок В.5 – Пример диаграммы Ганта

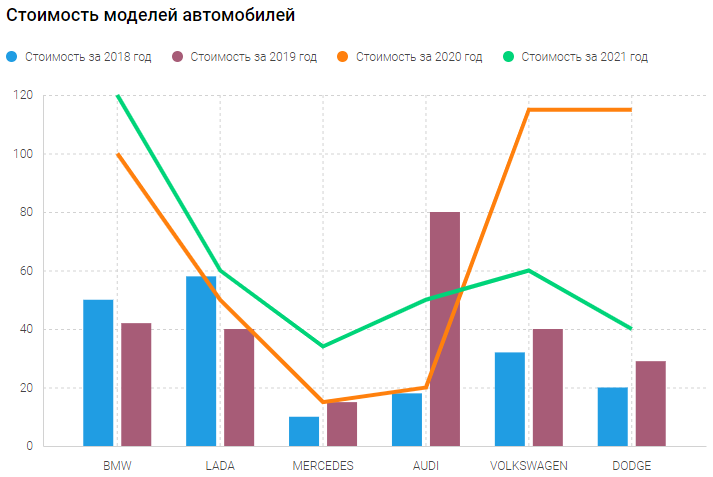


Рисунок В.6 – Пример комбинированных диаграмм (столбчатая и линейная)