3 РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1 Выбор средств реализации

Разрабатываемая система будет иметь клиент-серверную архитектуру. Клиентская часть приложения будет разрабатываться на языке JavaScript с использованием библиотеки Vue.js, библиотека управления состоянием – Vuex, разметка HTML и SCSS стили, серверная часть – NodeJS+ExpressJS с использованием REST-архитектуры. Приложение будет использовать MVC (модель-вид-контроллер) архитектуру. Для работы с базами данных будет использоваться СУБД MongoDB и ODM (Object-Document Mapping) библиотека Mongoose. Для сборки клиентской части приложения будет использоваться сборщик Webpack.

Рассмотрим подробнее выбранные средства программной реализации дипломного проекта.

MVC (Model-view-controller) – схема использования нескольких шаблонов проектирования, с помощью которых модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные. Данная схема проектирования часто используется для построения архитектурного каркаса, когда переходят от теории к реализации в конкретной предметной области.

Основная цель применения этой концепции состоит в разделении бизнес-логики от её визуализации. За счет такого разделения повышается возможность повторного использования кода. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения.

К одной модели можно присоединить несколько видов, при этом не затрагивая реализацию модели. Например, некоторые данные могут быть одновременно представлены в виде электронной таблицы, гистограммы и круговой диаграммы.

Не затрагивая реализацию видов, можно изменить реакции на действия пользователя (нажатие мышью на кнопке, ввод данных), для этого достаточно использовать другой контроллер.

Ряд разработчиков специализируется только в одной из областей: либо разрабатывают графический интерфейс, либо разрабатывают бизнес-логику. Поэтому возможно добиться того, что программисты, занимающиеся разработкой бизнес-логики (модели), вообще не будут осведомлены о том, какое представление будет использоваться.

Концепция MVC позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента:

− Модель (англ. Model). Модель предоставляет знания: данные и методы работы с этими данными, реагирует на запросы, изменяя своё состояние. Не содержит информации, как эти знания можно визуализировать.

− Представление, вид (англ. View). Отвечает за отображение информации (визуализацию). Часто в качестве представления выступает форма (окно) с графическими элементами.

− Контроллер (англ. Controller). Обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

Важно отметить, что как представление, так и контроллер зависят от модели. Однако модель не зависит ни от представления, ни от контроллера. Тем самым достигается назначение такого разделения: оно позволяет строить модель независимо от визуального представления, а также создавать несколько различных представлений для одной модели.

REST (Representational State Transfer) – архитектурный подход к созданию взаимодействия компонентов распределённого приложения. В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждый URL в свою очередь имеет строго заданный формат.

JavaScript (JS) – прототипно-ориентированный скриптовый язык программирования, который в большинстве своём используется для разработки клиентской части веб-приложений. JavaScript имеет C-подобный синтаксис, но, несмотря на это, имеет коренные отличия от этого языка, такие как: тип данных объект, динамическая типизация, функции как объекты первого класса и т.д. JS за время своего существования претерпел множество изменений и в последних стандартах языка появилась возможность использовать принцип модульности, который до этого существовал только в специальных библиотеках (например, RequireJS). Последний стандарт языка имеет маркировку ES7 (ECMAScript 2016).

Javascript при загрузке в браузер компилируется специальным механизмом, называемым движком, в машинный код, который потом исполняется на стороне клиента. Javascript содержит в себе мощный механизм по управлению объектной моделью документа и событиями, происходящими на веб-странице.

Существует огромное количество фреймворков, библиотек и готовых модулей для реализации задач различного типа на языке JavaScript. Несмотря на то, что данный язык не взаимодействует с файловой системой, потоками ввода/вывода и не имеет стандартных интерфейсов для работы с веб-сервисами и БД, существуют

фреймворки и библиотеки, которые позволяют также создавать серверные части приложений на данном языке. Это NodeJS, ExpressJS, Meteor.

Также появилась возможность создавать нативные (мобильные) приложения с использованием библиотек JS таких как Weex, React Native, Cordova Phonegap, Angular 2 и так далее.

JavaScript предоставляет инструмент асинхронности, который позволяет нескольким процессам выполняться не поочерёдно, а вместе, что ускоряет обработку данных, а при неудачной обработке отодвигает процесс во времени, предоставляя ресурс на выполнение другому процессу [4].

В последнее время для создания клиентских частей приложений больше всего используется компонентный подход, поэтому для реализации клиентской части своего дипломного проекта мой выбор пал на развивающуюся многофункциональную библиотеку Vue.js.

Библиотека Vue – это прогрессивный фреймворк для создания пользовательских интерфейсов. В отличие от фреймворков-монолитов Vue создан пригодным для постепенного внедрения. Его ядро в первую очередь решает задачи уровня представления (view), что упрощает интеграцию с другими библиотеками и существующими проектами. С другой стороны, Vue полностью подходит и для создания сложных одностраничных приложений (SPA, Single-Page Applications), если использовать его совместно с современными инструментами и дополнительными библиотеками [5].

Данная библиотека довольно проста в изучении и использовании. Главным подходом в разработке с Vue являются компоненты. Компоненты – это элементы Vue, написанные разработчиком. Они представляют собой Vue файлы, в которых содержится код, который при компиляции в браузере превращается в обычный HTML, и методы для обработки событий, и логики приложения.

Также для рендера элементов Vue использует технологию Virtual DOM. Virtual DOM — это виртуальное представление объектной модели документа DOM, которое строится на основе Vue компонентов. Vue хранит данное представление и при изменении данных в компонентах сверяет Virtual DOM с реальным DOM и изменяет его. Данная технология позволяет экономить ресурсы и время, так как не требуется полная отрисовка страницы заново, а лишь перерисовка некоторых элементов.

Библиотека Vue предоставляет мощный инструментарий по управлению жизненным циклом компонент, например, отслеживание монтирования компоненты или изменения её параметров. Данный инструмент позволяет максимально оптимизировать поведение веб-страницы, сэкономить ресурсы и время.

Внутри компонента может использоваться «состояние» (state) для динамического изменения данных на странице.

Также данная библиотека предоставляет возможность композиции, то есть добавление более мелких компонент для создания более крупных.

Vuex – это паттерн управления состоянием и библиотека для приложений на Vue.js. Он служит центральным хранилищем данных для всех компонентов приложения и обеспечивает предсказуемость изменения данных при помощи определённых правил.

HTML (HyperText Markup Language) – язык гипертекстовой разметки данных, который используется для разметки веб-страниц. Язык HTML интерпретируется браузерами; полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства.

HTML был разработан в конце 80-х годов 20 века. HTML создавался как язык для обмена научной и технической документацией, пригодный для использования людьми, не являющимися специалистами в области вёрстки.

Последним стандартом HTML является HTML 5.1. Это теговый язык разметки. Любой документ представляет собой набор тэгов. Тэги могут содержать либо не содержать внутри себя информацию, а также могут быть одиночными либо парными. Кроме элементов в HTML также присутствуют сущности – специальные символы, которые начинаются с амперсанда и далее имеют код либо имя символа (например, запись &lt при рендеринге превратится в знак «меньше <»). В каждом тэге можно передавать атрибуты – специальные свойства, которые являются либо просто информативными, либо описывают поведение тэга. Так, например, если передать в тэг <input> атрибут type=”email”, это будет означать, что при введение в поле невалидного email адреса встроенный обработчик выдаст ошибку. Атрибут class присвоит текущему тэгу класс, который потом будет использоваться в качестве CSS-селектора.

Для HTML есть расширение XHTML, которое является смесью XML и HTML и имеет более строгое определение разметки документа, чем обычный HTML. Он обязательно требует наличие тэгов <html>, <head>, <body> и в случае ошибок (незакрытых тэгов, неправильного их написания) не отрисуется.

SCSS – это диалект языка SASS, который, в свою очередь, является метаязыком на основе CSS (Cascade StyleSheets), предназначенный для увеличения уровня абстракции CSS-кода и упрощения написания каскадных таблиц стилей. SCSS, как и SASS, написан на языке Ruby, соответственно SCSS-код компилируется Ruby-компиллятором в валидный CSS-код [6].

SCSS завоевал свою популярность благодаря тому, что он расширяет возможности классического CSS, добавив в него такие вещи, как объявление переменных, циклы, вложенность, обращение к «родителю» и т.д.

Использование переменных перевернуло мир вёрстки, упростив процесс стилизации страниц. Например, вам не придётся запоминать название или, тем более,

нумерацию цвета, ведь его можно просто объявить в переменной с запоминающимся названием и использовать где угодно.

Также SCSS позволяет «вкладывать» стили друг в друга, что уменьшает количество написанного кода благодаря тому, что не приходится дублировать селекторы, просто вкладывая друг в друга наследуемые селекторы и стили.

Особое место занимают математические операции в написании стилей. Например, задав какую-то стандартную величину, можно её использовать в любом месте документа, умножая/разделяя на любое число и получая таким образом нужные размеры.

Node.js — программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API (написанный на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль веб-сервера, но есть возможность разрабатывать на Node.js и десктопные оконные приложения.

В данном проекте NodeJS будет использоваться для создания и функционирования REST-сервера. Взаимодействие с различными модулями позволяет быстро и эффективно написать сервер на NodeJS. Для создания эффективного сервера будем использовать фреймворк ExpressJS.

Также NodeJS предоставляет инструмент npm (Node Package Manager), менеджер пакетов, который позволяет быстро устанавливать и управлять зависимосятми для приложения.

ExpressJS – веб-фреймворк для приложений NodeJS, который позволяет создавать API для вашего веб-приложения. Express изначально работает с протоколами HTTP/HTTPS, но в качестве расширений можно использовать модули для работы с websocket и другими протоколами [7].

Express предоставляет простой API для реализации маршрутизации и взаимодействия с различными ресурсами, а также предоставляет возможность рендеринга на стороне сервера.

Особенностью Express являются промежуточные обработчики (middleware). Данные обработчики являются функциями, которые имеют доступ к объекту запроса, объекту ответа и к следующей функции обработки в цикле «запрос-ответ» приложения. Функции промежуточной обработки в основном используются для логгирования вызовов сервера, обработки маршрутов, использования встроенных обработчиков и т.д.

Express также предоставляет возможность интеграции с базами данных (как реляционными, так и NoSQL) через нативные клиенты либо через высокоуровневые

библиотеки (ORM), такие как Mongoose (MongoDB) или Sequelize (PostgreSQL, MS SQL, MariaDB, SQLite, MySQL). В своём дипломном проекте я буду использовать именно Sequelize для взаимодействия с базой данных.

MongoDB – документоориентированная система управления базами данных (СУБД) с открытым исходным кодом, не требующая описания схемы таблиц. Классифицирована как NoSQL, использует JSON-подобные документы и схему базы данных. Написана на языке C++.

Данная СУБД была выбрана мной на дипломный проект благодаря её современности и популярности в веб-разработке, наличию типа данных Массив и JSON, что позволит упростить решение задачи хранения некоторых данных, а также благодаря большому порогу памяти на хранение данных.

Для визуализации работы с базой данных будет использоваться клиент Robo 3T, предоставляемый разработчиками MongoDB.

Mongoose – это ODM библиотека для приложений на NodeJS, которая поддерживает работу с MongoDB. Предоставляет возможность работать и настраивать базы данных в проекте без использования SQL кода (хотя можно и с ним) через специально написанные методы, которые сокращают работу по написанию запросов в базу данных [8].

Данная ODM построена на основе Promise (инструмент Javascript для обработки асинхронных вызовов), что позволяет быстро и легко обрабатывать ответы базы данных.

Эта библиотека предоставляет широкий инструментарий для описания моделей (сущностей) БД, установки опций полям (например, валидация, возможность иметь нулевое значение), работы с этими моделями (создание отношений (связей) между моделями, CRUD операции), создание хуков – функций жизненного цикла приложения, который в определённый момент выполнения кода позволяют совершить какое-либо действие (например, изменение записи таблицы перед её сохранением), а также возможность создавать и использовать скоупы – опции модели, которые позволяют хранить в себе наиболее часто используемые запросы к данной модели.

Webpack – утилита для сборки проектов на языке Javascript и других ресурсов фронтенда. Логика работы данной утилиты заключается в том, что Webpack принимает один или несколько файлов, используя пути из файла конфигурации. Сборщик во время обработки оборачивает каждый модуль в функцию, таким образом возвращая контекст этого модуля, сжимает его и возвращает в конечном итоге так называемый бандл, который содержит в себе все модули Javascript, и запускает его.

Webpack позволяет также автоматически сжимать и объединять файлы стилей, разметки и т.д.

Webpack содержит в себе встроенный dev-сервер, который во время разработки позволяет автоматически обновлять бандлы и мгновенно подгружать их в работающее приложение.

Автоматический сборщик позволяет быстро упаковать пользовательские модули и зависимости для полноценной работы приложения, а также имеет возможность описывать и исполнять пользовательские конфигурации, например, настройка транспайлера Babel, настройка юнит-тестов и путей к ним, настройка путей к хранилищам файлов в приложении, выставление вендорных префиксов в CSS-файлах и так далее. Все эти конфигурации можно прописывать, установив соответствующие загрузчики (loaders) и расширения (plugin).

Для нормального функционирования системы требуется:

− программное обеспечение:

а) клиентское приложение – веб-браузеры последних версий (Google Chrome 65+, Microsoft Edge 41+, Mozilla Firefox53+), операционная система не важна;

б) веб-сервер – NodeJS не ниже 7 версии, ExpressJS 5, ОС – Windows 7 и выше, Linux 15 и выше;

в) база данных – СУБД MongoDB 3.5+, ОС – Windows 7 и выше, Linux 15 и выше.

− аппаратное обеспечение:

а) клиентское приложение – 1Гб оперативной памяти, процессор 2x1.2 ГГц и больше;

б) веб-сервер – 1Гб оперативной памяти, 2Гб места на жёстком диске, процессор 2x1.2 ГГц и больше;

в) база данных – 1Гб оперативной памяти, 4Гб места на жёстком диске, процессор 2x1.2 ГГц и больше.

3.2 Разработка архитектуры приложения

Архитектура приложения состоит из серверной и клиентской части. Рассмотрим их. Архитектура файлов клиентского приложения представлена на рисунке 3.1. Ниже приведено описание файлов и каталогов.

– build – каталог, в котором хранятся минифицированные файлы, собранные сборщиком проекта Webpack;

– config – каталог, созданный утилитой vue-cli, в котором хранятся файлы конфигураций сборщика проекта;

– node_modules – каталог, в котором содержатся модули, которые были установлены при помощи пакетных менеджеров npm;

– static – каталог, в котором хранятся файлы публичного доступа (изображения, доступные всем пользователям);

– src – каталог, в котором содержатся исходные коды приложения, такие как Vue-компоненты, Vuex-модули, CSS-стили и так далее;

– .gitignore – файл, содержащий список игнорируемых системой контроля GIT файлов и директорий;

– .postcssrc.js – файл конфигурации CSS-стилей;

– .eslintrc.js – файл конфигурации JS-код-стилей;

– index.html – основной файл разметки веб-приложения;

– package.json – файл конфигурации для сторонних модулей и пакетных менеджеров npm;

– .eslintignore – файл, содержащий список игнорируемых системой контроля качества eslint файлов и директорий;

– README.md – файл с описанием программы.

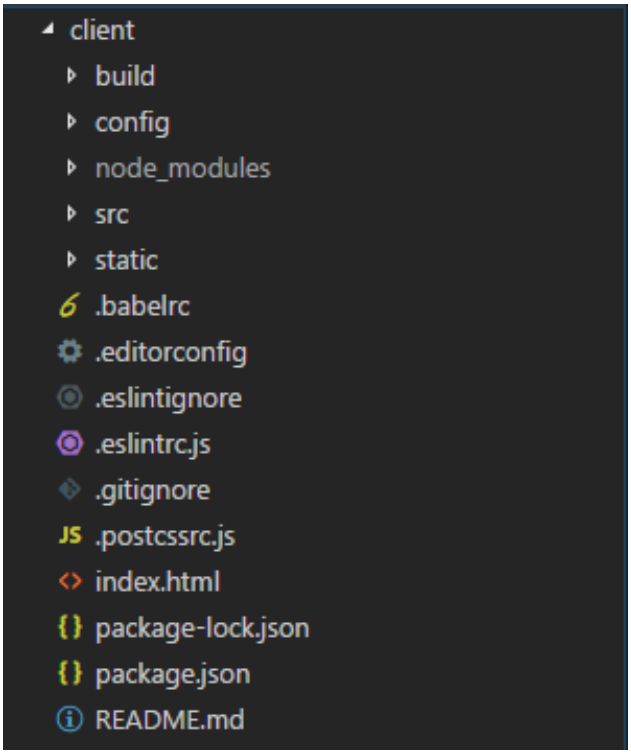

Рисунок 3.1 – Архитектура файлов клиентского приложения

Архитектура каталогов и файлов веб-сервера представлена на рисунке 3.2

– api – каталог, в котором хранятся обработчики маршрутизатора ExpressJS;

– constants – каталог, в котором содержатся файлы, хранящие константы приложения;

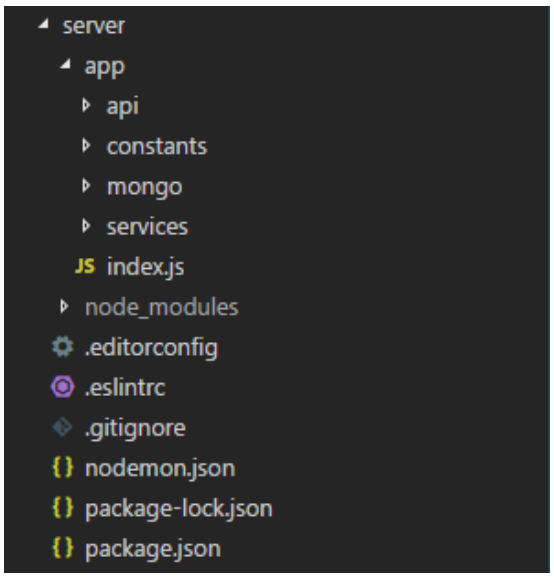

Рисунок 3.2 – Архитектура файлов веб-сервера

– mongo – каталог, в котором содержатся файлы, которые описывают конфигурацию базы данных;

– services – каталог, содержащий пользовательские промежуточные обработчики, а также описания взаимосвязи моделей базы данных;

– node_modules – каталог, в котором содержатся модули, которые были установлены при помощи пакетных менеджеров npm;

– package.json – файл конфигурации для сторонних модулей и пакетных менеджеров yarn и npm;

– index.js – файл, в котором подключаются все промежуточные обработчики, участвующие в работе веб-сервера, а также конфигурируется порт, на котором будет запущен сервер;

– .editorconfig – файл содержащий конфигурации для среды разработки;

– nodemon.json – файл конфигурации запуска серверного приложения с поддержкой быстрой перезагрузки.

Разработаем приложение согласно схеме программы представленной на чертеже БрГТУ.141144 - 07 92 00.

3.3 Разработка базы данных

В данном дипломном проекте в качестве СУБД используется MongoDB. Для оперирования данными на сервере используется ORM Mongoose. В качестве сервера используется сервис mlab.com. Для управления базой данных используется программа Robo 3T.

Для начала требуется создать базу данных. Для этого требуется подключиться к серверу MongoDB в программе Robo 3T (смотри рисунок 3.3), авторизоваться, после этого требуется создать базу данных (смотри рисунок 3.4).

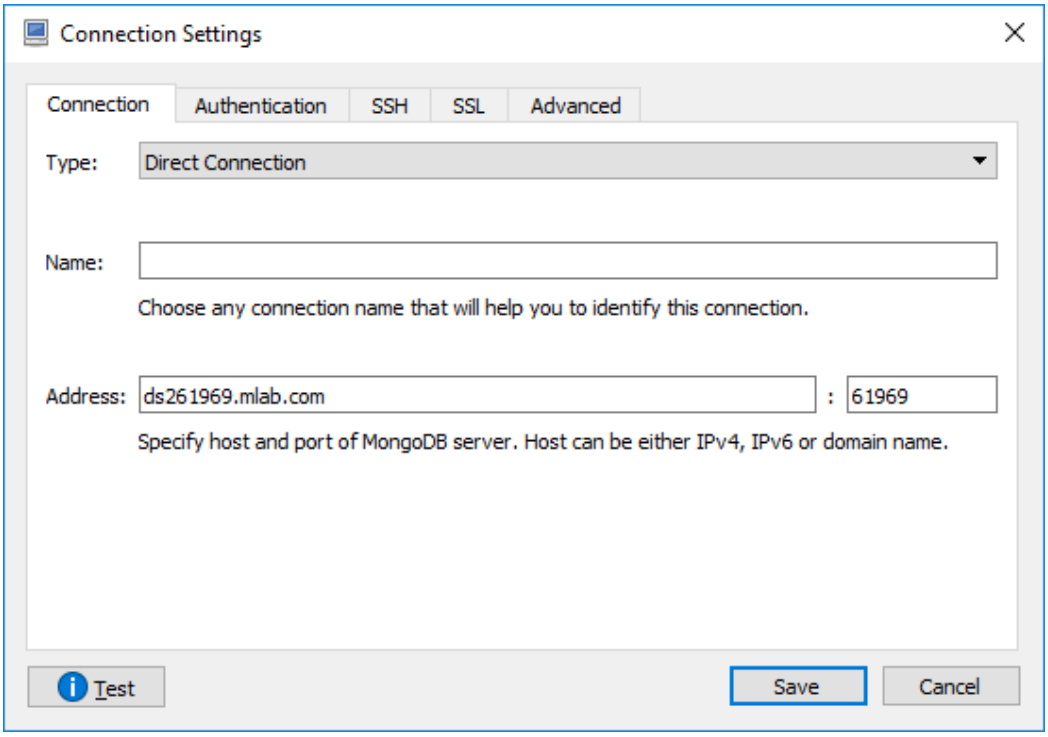

Рисунок 3.3 – Подключение к базе данных MongoDB

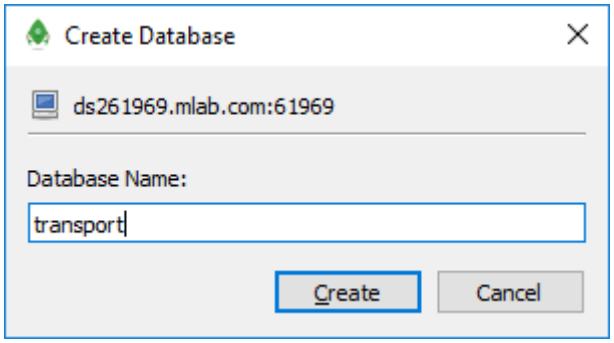

Рисунок 3.4 – создание новой базы данных в Robo 3T

Для управления базой данных используется ORM Mongoose. Для того, чтобы сервер смог работать с конкретной базой данных, требуется сконфигурировать её на стороне сервера и подключить её в файле index.js (смотри рисунок 3.5).

```
import mongoose from 'mongoose';

mongoose.connect('mongodb://user:password@ds261969.mlab.com:61969/transport');
```

Рисунок 3.5 – конфигурирование базы данных transport

Для создания таблиц используется метод mongoose.model(), который создаёт модель таблицы и позволяет ей управлять программно. Первым параметром метода идёт название таблицы, вторым – поля таблицы с типом данных и опциями, например, валидацией, третьим – опции модели, например, описание методов экземпляра модели. Пример модели можно увидеть на рисунке 3.6.

```
import mongoose from '../../mongo';

const RideSchema = new mongoose.Schema({
  user: {
    type: String,
    required: true,
  },
  vehicle: {
    type: String,
    required: true,
  },
  from: {
    type: String,
  },
  to: {
    type: String,
  },
  payment: {
    type: Number,
    default: 0.5,
  },
  date: {
    type: Date,
    default: Date.now,
  },
});

export default mongoose.model('Ride', RideSchema);
```

Рисунок 3.6 – Пример создания модели поездок

Для CRUD операций в Mongoose представлены следующие методы, указанные в таблице 3.1.

Таблица 3.1 – Методы CRUD операций в Mongoose

Метод	Описание	Аналог SQL
1	2	3
Model.find(options)	Поиск всех записей по указанным опциям	SELECT * FROM model WHERE options
Model.findOne(options)	Поиск одного элемента по указанным опциям	SELECT * FROM model WHERE options LIMIT 1

Продолжение таблицы 3.1

1	2	3
Model.create(data)	Создание новой записи с данными data	INSERT INTO model data
Model.updateOne (options, data)	Изменение выбранных данных data у элемента, найденному по параметрам options	UPDATE model SET data WHERE options
Model.deleteOne(options)	Удаление выбранной записи модели	DELETE FROM model WHERE options

На рисунке 3.7 приведен пример операций с MongoDB для остановок.

```
import StopSchema from './stopSchema';


export default class Stop {
  static addStop(name, longitude, latitude) {
    const stop = new StopSchema({
      name,
      longitude,
      latitude,
    });

    return stop.save()
      .then(() => stop);
  }

  static editStop(data) {
    const { _id } = data;

    return StopSchema.findOneAndUpdate({ _id }, data, { new: true })
      .then(stop => (stop || null));
  }

  static deleteStop(name) {
    return StopSchema.findOneAndRemove({ name });
  }
}
```

Рисунок 3.7 – Пример операций с MongoDB

Перечислим функции базы данных для работы с остановками:

– Функция «addStop» выполняет добавление новой остановки. Для этого в функцию передаются параметры «name», «longitude» и «latitude». Обязательным параметром является только «name». Принадлежность параметров к обязательным указывается на уровне схемы;

– Функция «editStop» выполняет редактирование существующей остановки. Она принимает объект, который хранит все данные об остановке. Из этого объекта извлекается идентификатор остановки, по которому происходит выборка и, если остановка с заданным идентификатором найдена, то она будет обновлена;

– Функция «deleteStop» выполняет удаление остановки по имени. Как и в случае с редактированием, сначала происходит выборка, а затем при наличии результата удаление. Поле «name» является уникальным, поэтому удаление не той остановки исключено.

3.4 Разработка серверного приложения

Для работы с базой данных на стороне сервера необходимо описать модели используемых сущностей. Рассмотрим список моделей, используемых в приложении:

– поездка – содержит поездки пользователей: идентификатор пользователя, начальная и конечная точки, оплата, дата и время поездки;

– маршрут – хранит информацию о маршрутах;

– остановка – хранит данные об остановках: их названия и координаты;

– пользователь – хранит данные о пользователе: его имя, номер карты, состояние счета и т.д;

– транспортное средство – хранит данные о транспортных средствах: номер ТС и ссылка на маршрут.

Рассмотрим схему описания модели «Маршрут» более подробно. Она содержит в себе следующие поля:

– номер маршрута – строковый формат, обязательное поле;

– тип ТС – строковый формат, может быть одним из трех вариантов (автобус, троллейбус или трамвай), обязательное поле;

– остановки в прямом направлении – массив ссылок на объекты типа «Остановка», по которым можно извлечь все зависимости;

– остановки в обратном направлении – массив ссылок на объекты типа «Остановка», по которым можно извлечь все зависимости.

Для управления моделями используются контроллеры. На рисунке 3.8 приведен пример такого контроллера для модели «Остановка».

Из тела запроса извлекаются данные, которые затем используются в операциях добавления, редактирования, удаления и т.д.

При добавлении новой записи производится попытка извлечения из БД записи с соответствующими именем. Если запись с таким именем уже существует, то пользователю будет отправлен ответ с кодом ошибки 403. В случае, если записи

с переданным именем не существует в БД, она будет создана, ответу устанавливается код успешного завершения операции – 200.

```
import StopService from '../../services/stop';

export function addStop(req, res) {
  const { name, longitude, latitude } = req.body;

  StopService.addStop(name, longitude, latitude)
    .then((stop) => {
      if (!stop) res.status(403).send('Stop already exist');
      else res.status(200).send('Stop successfully added');
    })
    .catch(() => res.status(403).send('Server error'));
}
```

Рисунок 3.8 – Контроллер для модели «Остановка»

Такие контроллеры регистрируются для всех моделей. Регистрация же самих роутов осуществляется в главном файле приложения. На рисунке 3.9 представлен механизм подключения роутов к серверному приложению.

```
import { Router } from 'express';
import AuthRouter from './auth';
import UserRouter from './user';
import StopRouter from './stop';
import RouteRouter from './route';
import VehicleRouter from './vehicle';
import { ROUTES } from '../constants';
import authMiddleware from './middlewares/authMiddleware';
const router = new Router();

// The authentication route and middleware must be the first
router.use(ROUTES.auth, AuthRouter);

router.use(authMiddleware);

router.use(ROUTES.user, UserRouter);
router.use(ROUTES.stop, StopRouter);
router.use(ROUTES.route, RouteRouter);
router.use(ROUTES.vehicle, VehicleRouter);

// 404 Not Found
router.all('*', (req, res) => res.sendStatus(404));

export default router;
```

Рисунок 3.9 – Механизм подключения роутов

Приложение должно обеспечивать поддержку возможности одновременной работы нескольких людей. Для этого используется механизм авторизации на основе веб-токенов с помощью технологии JSON Web Token (JWT).

JWT обеспечивает способ проверки подлинности каждого запроса без необходимости поддерживать сеанс или повторно передавать учетные данные для входа на сервер. JWT состоит из трех основных компонентов: объект заголовка, объект данных и подпись (смотри рисунок 3.10). Эти три свойства кодируются с помощью base64, а затем объединяются при помощи разделителей. Эти три части объединяются и добавляются к заголовку запроса в соответствующее поле.

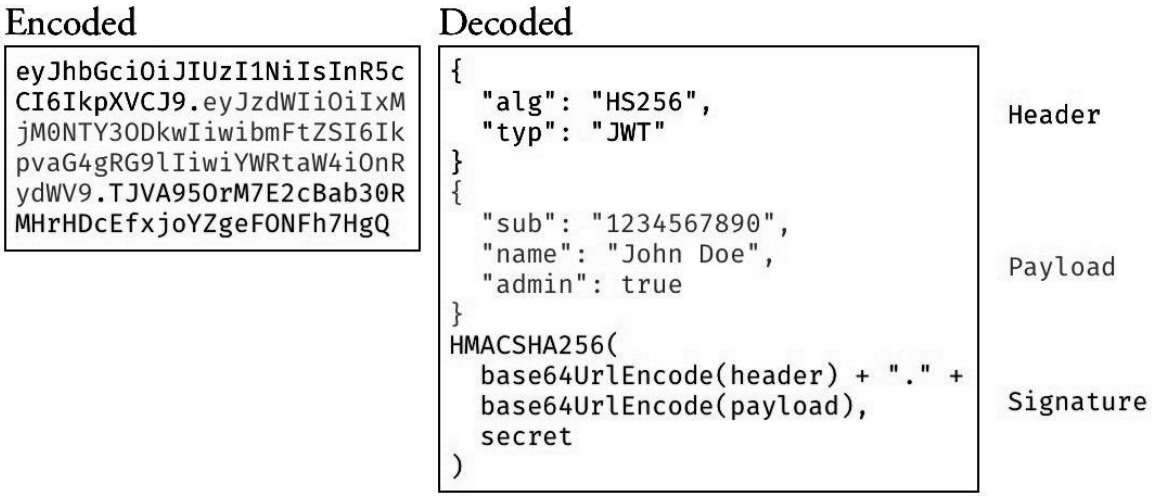

Рисунок 3.10 – Пример работы JWT

3.5 Разработка клиентского приложения

Как было указано ранее, на клиентской части используется фреймворк Vue.js в связке с модулями Vue Router (для осуществления навигации по приложению) и Vuex (для работы с данными).

Механизм работы данного приложения основывается на том, что существуют несколько состояний приложения, которые будут выполнять определенные задачи.

Следует обратить внимание на то, что клиентское приложение содержит большое количество модулей, загрузка которых по отдельности вызывает дополнительные задержки. Для этого было принято решение при помощи сборщика Webpack собирать «.js»-файлы в один и отправлять его пользователю.

Причиной этого стало то, что при загрузке большого числа файлов тратится очень много времени на ожидание ответа от сервера. Пример случая с большим количеством файлов приведен на рисунке 3.11.

Из рисунка видно, что общее время загрузки файлов с размером 8Kb занимает примерно 1,67 секунд, что для такого объема очень велико.

Приведем пример статистики загрузки оптимизированных файлов, объединенных в один на рисунке 3.12.

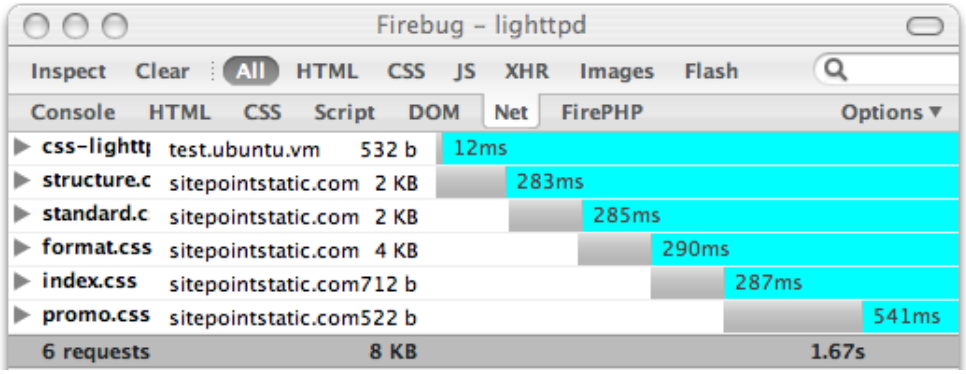

Рисунок 3.11 – Загрузка большого числа файлов

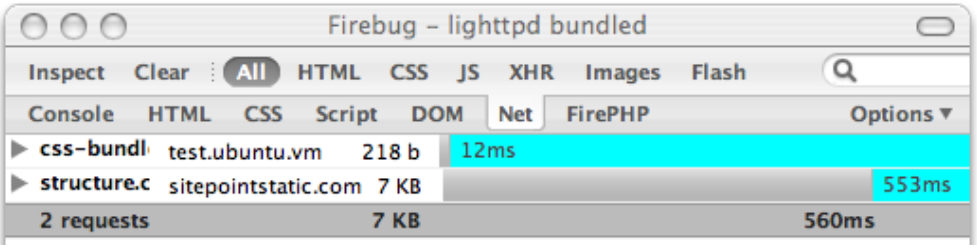

Рисунок 3.12 – Загрузка одного объединенного файла

Загрузка одного объединенного файла занимает 560 ms, что дает почти трехкратное увеличение скорости загрузки.

Для выполнения запросов используется библиотека Axios. Запросы бывают нескольких типов: GET – для получения данных, POST – для добавления новых данных, PUT – для изменения существующих данных, DELETE – для удаления существующих данных. На рисунке 3.13 представлен пример запросов для работы с маршрутами.

Рассмотрим функции, выполняющие запросы для маршрутов:

– getRoutes – отправляет GET запрос на получение всех доступных маршрутов;

– getRoute – отправляет GET запрос на получение конкретного маршрута с заданным идентификатором;

– addRoute – отправляет POST запрос на добавление нового маршрута, в теле запроса содержится вся информация о маршруте;

– editRoute – отправляет PUT запрос на изменение существующего маршрута, в теле запроса содержится вся информация о маршруте;

– deleteRoute – отправляет DELETE запрос на удаление конкретного маршрута по заданному идентификатору;

– getRoutesByStops – отправляет POST запрос на получение маршрутов, которые проходят через остановки, указанные в теле запроса.

```
import axios from 'axios';
import { HOST, PORT } from './config';

export default {
  getRoutes = () => axios.get(`${HOST}:${PORT}/route/all`),
  getRoute = (id) => axios.get(`${HOST}:${PORT}/route/${id}`),
  addRoute = (data) => axios.post(`${HOST}:${PORT}/route`, data),
  editRoute = (data) => axios.put(`${HOST}:${PORT}/route`, data),
  deleteRoute = (id) => axios.delete(`${HOST}:${PORT}/route`, { data: { id } }),
  getRoutesByStops = (data) => axios.post(`${HOST}:${PORT}/route/by-stops`, data),
};
```

Рисунок 3.13 – Функции для запросов с использованием библиотеки Axios

Для управления состоянием приложения используется Vuex. Vuex – это паттерн управления состоянием и библиотека для приложений на Vue.js. Он служит центральным хранилищем данных для всех компонентов приложения и обеспечивает предсказуемость изменения данных при помощи определённых правил. Схема работы Vuex представлена на рисунке 3.14.

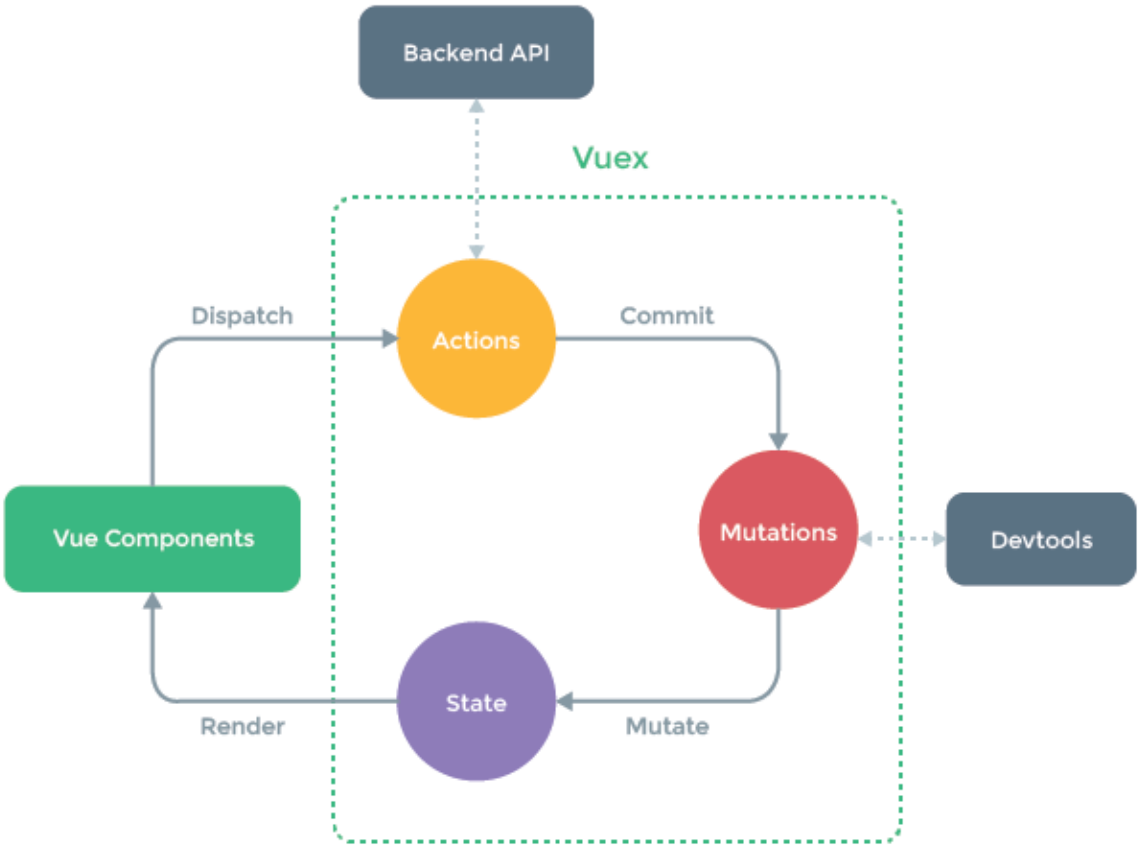

Рисунок 3.14 – Схема работы Vuex

На рисунке 3.15 представлен пример модуля Vuex для остановок. Поле «namespaced» задаёт область видимости. Поле «state» хранит текущее состояние. В поле «actions» описываются действия, после которых путем вызова мутаций должно измениться состояние. В поле «mutations» описываются функции для изменения состояния.

```
import api from '@/api';

export default {
  namespaced: true,
  state: { stops: [] },
  actions: {
    getStops: ({ commit }) => {
      api.stops.getStops()
        .then((response) => {
          commit('SET_STOPS', response.data);
        });
    },
  },
  mutations: {
    SET_STOPS: (state, stops) => { state.stops = stops },
  },
};
```

Рисунок 3.15 – Модуль хранилища для остановок

Аналогичным образом создаются модули для работы с пользователями, поездками, маршрутами и транспортом.

Для определения местоположения используется API геолокации браузера. API геолокации позволяет пользователю предоставлять свое местоположение web-приложению, если пользователь согласится предоставить его. Из соображений конфиденциальности у пользователя будет запрошено разрешение на предоставление информации о местоположении. Функция получения географических координат представлена на рисунке 3.16.

```
navigator.geolocation.getCurrentPosition(function(position) {

  do_something(position.coords.latitude, position.coords.longitude);

});
```

Рисунок 3.16 – Пример получения данных о локации

3.6 Сборка проекта

Существует два наиболее популярных инструмента сборки – Webpack и Gulp. Но есть и другие, менее распространенные варианты. Сборщик Webpack был выбран по причине быстродействия, простоты и большого количества дополнительных модулей (плагинов).

Последовательность решаемых задач в проекте следующая:

– запуск веб-сервера, контроль за его перезагрузкой;

– преобразование SASS в CSS;

– обновление браузера автоматически при сохранении файлов;

– сборка всех файлов (CSS, JS, шрифты и изображения) для использования на клиентской стороне (файлы одного типа собираются в один файл для ускорения загрузки сайта).

Webpack является инструментом, который помогает в нескольких важных задачах веб-разработки. Он часто используется для решения следующих задач:

– запуск веб-сервера;

– перезагрузка браузера автоматически при изменении файлов проекта;

– использование CSS-препроцессоров вроде SASS.;

– оптимизация размера при работе с CSS, JavaScript и изображениями.

Это неполный список задач, которые Webpack может выполнять (список приведен для текущего проекта). Webpack является чрезвычайно мощным инструментом веб-разработки.

Такие инструменты, как Webpack, часто называются инструментами для сборки, потому что они являются комплексным способом для запуска задач при создании веб-приложений.

CSS-препроцессор – это надстройка над CSS, которая добавляет ранее недоступные возможности для CSS, с помощью новых синтаксических конструкций. На сегодняшний день можно выделить такие препроцессоры как SASS, Less и Stylus.

В качестве основного инструмента для быстрой и продуктивной разработки таблиц стилей для приложения был выбран SASS. SASS – это самый популярный препроцессор с огромным сообществом и мощным функционалом.

SASS является препроцессором языка CSS. Файлы SASS не могут использоваться напрямую в браузере и должны быть обработаны путем конвертирования в другой язык, который может быть использован в большинстве браузеров. SASS транслируется в CSS.

SASS имеет два синтаксиса. Первый – с расширением «.sass» и второй – «.scss». Оригинальный синтаксис представляет собой разделенный синтаксис со вложенностью, который более легок по сравнению со стандартным синтаксисом

CSS. Вдобавок к этому он имеет поддержку переменных.

В проекте используется расширение «.scss», поскольку оно является классическим. Перечислим основные возможности препроцессора SASS, которые были задействованы в ходе разработки:

– Переменные дают возможность хранить цвета, стеки шрифтов или любые другие значения CSS, которые вы хотите использовать. Чтобы создать переменную в Sass нужно использовать символ «$»;

– Вложенности позволяют вкладывать CSS селекторы таким же образом, как и в визуальной иерархии HTML. Но следует отметить, что чрезмерное количество вложенностей делает документ менее читабельным и воспринимаемым, что считается плохой практикой;

– Импорт препроцессора SASS берет идею импорта файлов через директиву «@import», но вместо создания отдельного HTTP-запроса SASS импортирует указанный в директиве файл в тот, где он вызывается, т.е. на выходе получается один CSS-файл, скомпилированный из нескольких фрагментов;

– Миксины позволяют создавать группы деклараций CSS, которые придется использовать по нескольку раз на сайте. Хорошо использовать миксины для вендорных префиксов. Для создания миксина используется директива «@mixin» + название. После того, как миксин был создан, его можно использовать в качестве параметра CSS, начиная вызов с «@include» и имени миксина;

– Фрагменты содержат в себе небольшие отрывки CSS, которые можно будет использовать в других SASS-файлах. Это отличный способ сделать CSS модульным, а также облегчить его обслуживание.

Пример фрагмента стилей с использованием данного препроцессора представлен на рисунке 3.17.

После запуска препроцессора данный код будет преобразован в CSS-синтаксис. Для этого необходимо выполнить команду «sass <имя файла>». Результат представлен на рисунке 3.18.

Основным языком для написания сценариев в данном проекте был выбран JavaScript. Он использовался как на клиентской стороне, так и на стороне сервера. Благодаря JavaScript улучшается взаимодействие с пользователем, отслеживаются и обрабатываются события в браузере.

JavaScript является интерпретируемым языком. Он выполняется непосредственно в момент запуска интерпретатором. В настоящее время он имеет множество модификаций.

Последней официальной спецификацией языка является ECMA-262, которая также известна под именем ES-2015. Данная спецификация вносит улучшения в язык, непосредственно ускоряя разработку на нем.

Поскольку спецификация требует некоторого времени на освоение и внедрение

в браузеры, необходимо использовать более старую версию в сочетании с новой. Для преобразования языка из новой версии в старую используются трансплайеры. В данном проекте используется Babel.

```
<style lang="scss" scoped>
  @import '../../assets/styles/palette';

  @mixin bs($color) {
    background: $color;
    transition-duration: 1s;
  }

  .primary {
    @include bs($blue)
  }

  .success {
    @include bs($green)
  }

  .danger {
    @include bs($red)
  }

  .alert {
    position: absolute;
    bottom: 10px;
    right: 10px;
    width: 300px;
    height: 100px;
    &.hidden {
      display: none;
    }
  }

</style>
```

Рисунок 3.17 – Стили с использованием препроцессора SASS

Babel является популярным JavaScript-трансплайером, который был установлен более восьми миллионов раз, что показывает его популярность. Он используется в компаниях Facebook, Netflix, Spotify, Yahoo!.

Благодаря этому транспайлеру написанный JavaScript-код будет работать во всех современных браузерах.

```
<style scoped>

  .primary {
    background: blue;
    transition-duration: 1s;
  }

  .success {
    background: green;
    transition-duration: 1s;
  }

  .danger {
    background: red;
    transition-duration: 1s;
  }

  .alert {
    position: absolute;
    bottom: 10px;
    right: 10px;
    width: 300px;
    height: 100px;
  }

  .alert.hidden {
    display: none;
  }

</style>
```

Рисунок 3.18 – Результат работы препроцессора SASS

Схема взаимодействия программ разработанного программного обеспечения представлена на чертеже БрГТУ.141144 - 07 93 00.

С помощью вышеописанных средств в рамках данного дипломного проекта было разработано программное обеспечение, которое представлено в документе БрГТУ.141144 - 07 12 00.