МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

Web-приложение «EasyRoad»

Выполнил студент Сивак Михаил Николаевич

(Ф.И.О.)

Руководитель проекта асс. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер асс. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2021

Содержание

[Введение 3](#_Toc90450338)

[1 Постановка задачи 4](#_Toc90450339)

[1.1 Обзор аналогов 4](#_Toc90450340)

[1.2 Исходные данные к проекту 6](#_Toc90450341)

[2 Проектирование программного средства 7](#_Toc90450342)

[2.1 Выбор платформы 7](#_Toc90450343)

[2.1.1 Серверная часть 7](#_Toc90450344)

[2.1.2 Клиентская часть 7](#_Toc90450345)

[2.1.3 База данных 7](#_Toc90450346)

[2.2 Диаграмма вариантов использования 8](#_Toc90450347)

[2.3 Проектирование базы данных 8](#_Toc90450348)

[2.4 Описание структуры проекта 12](#_Toc90450349)

[3 Разработка программного средства 16](#_Toc90450350)

[3.1 Проектирование серверной части приложения 16](#_Toc90450351)

[3.2 Проектирование клиентской части приложения 19](#_Toc90450352)

[4 Тестирование программного средства 21](#_Toc90450353)

[5 Руководство пользователя 24](#_Toc90450354)

[Заключение 29](#_Toc90450355)

[Список используемых источников 30](#_Toc90450356)

[Приложение А 31](#_Toc90450357)

[Приложение Б 32](#_Toc90450358)

## **Введение**

Сегодня очень большую роль в жизни каждого пользователя играет сеть интернет. Почти каждый человек на земле пользуется обширными возможностями интернета, как в познавательных целях, так и для развлечений. Возможность пользоваться таким полезным изобретением человечества стало неотъемлемой частью многих рабочих должностей. К примеру, ни один разработчик программного обеспечения не обходится без сети интернет, так как там много полезной информации для повышения квалификации, ответы на все волнующие вопросы, касаемые сферы программирования. Даже, чтобы найти физический источник знаний, можно легко найти в интернете информацию о книгах, посмотреть адреса, где можно их можно купить и даже заказать, не выходя из дома, и дожидаться курьера, занимаясь своими делами дальше.

Существуют также задачи, не ограничивающиеся лишь поиском в интернете. В свободном доступе находятся очень удобные и значительно упрощающие жизнь сервисы, например, сервисы поиска транспорта для поездок в рамках как города, так и для путешествий хоть по всему миру. Подходя вплотную к теме моего проекта, необходимо сказать, что с недавнего времени стали очень популярны сервисы для поиска поездок в рамках города, а также в рамках страны.

Исходя из этого, тема моего проекта подразумевает собой веб-приложение для поиска водителей и попутчиков в рамках определённого города определённой страны, что может значительно сократить время и материальные расходы людей, нуждающихся в частых поездках из одной точки города в другую.

# **Постановка задачи**

## **1.1 Обзор аналогов**

Чтобы разработать качественное приложение, которым реально будут пользоваться люди, необходимо провести анализ аналогов с целью понять, как они работают и почему люди отдают им своё предпочтение.

1. Смайлбус

Смайлбус — сервис для осуществления поездок из города в город в рамках страны. Сервис получил широкое распространение благодаря удобному сайту, который позволяет пользователям осуществлять поездки в необходимые им города страны. Ежедневно данным сервисом пользуется огромное количество людей.

На главной странице сервиса находится поле для поиска маршруток для осуществления поездок и прочая информация, описывающая назначение сервиса, скриншот представлен на рисунке 1.1.

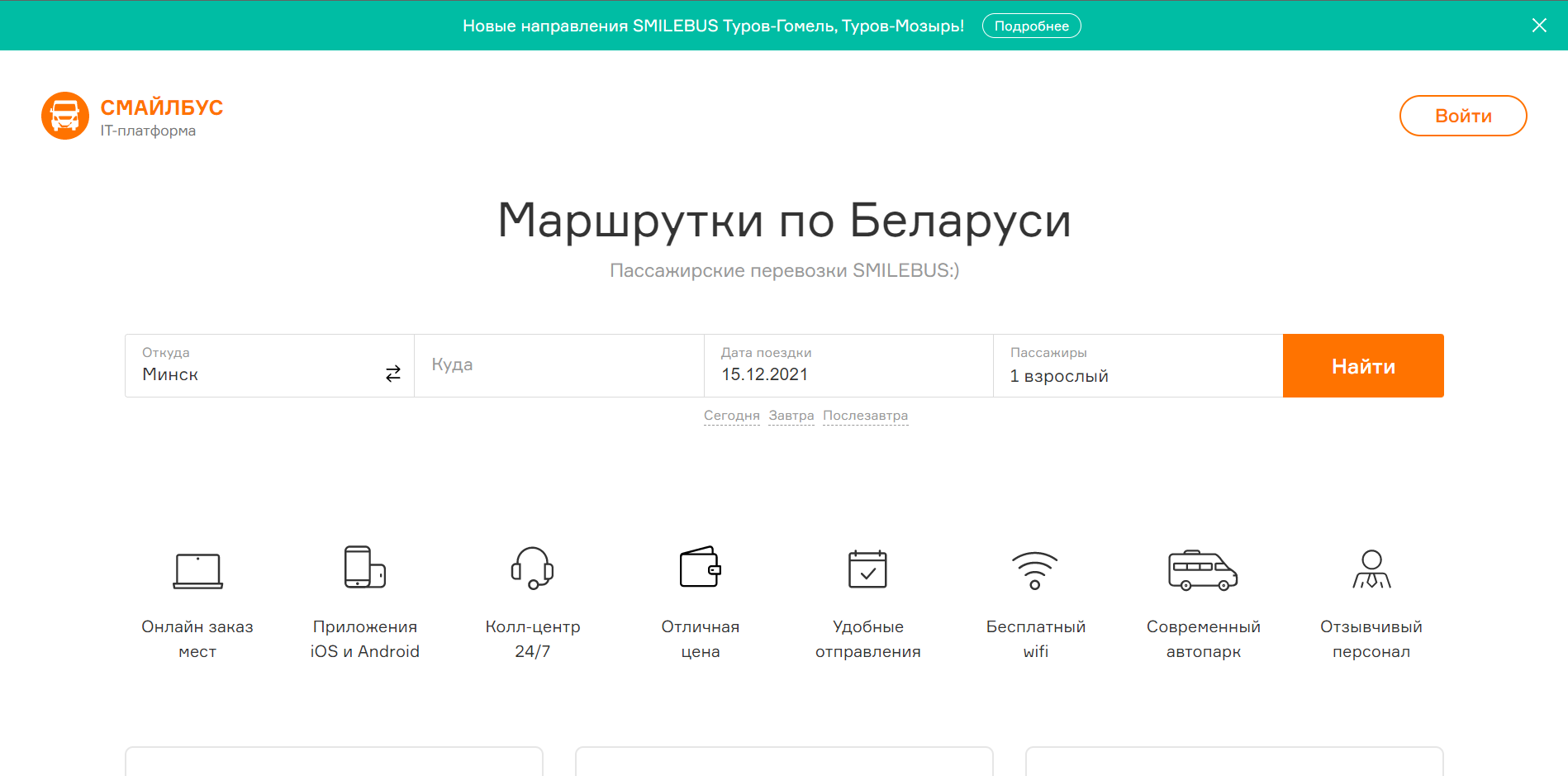


Рисунок 1.1 – Главная страница Смайлбус

Также на главной странице размещён блок с популярными поездками с возможностью открыть полный список всех направлений, по которым водители данного сервиса осуществляют поездки. Скриншот со списком популярных поездок представлен на рисунке 1.2.

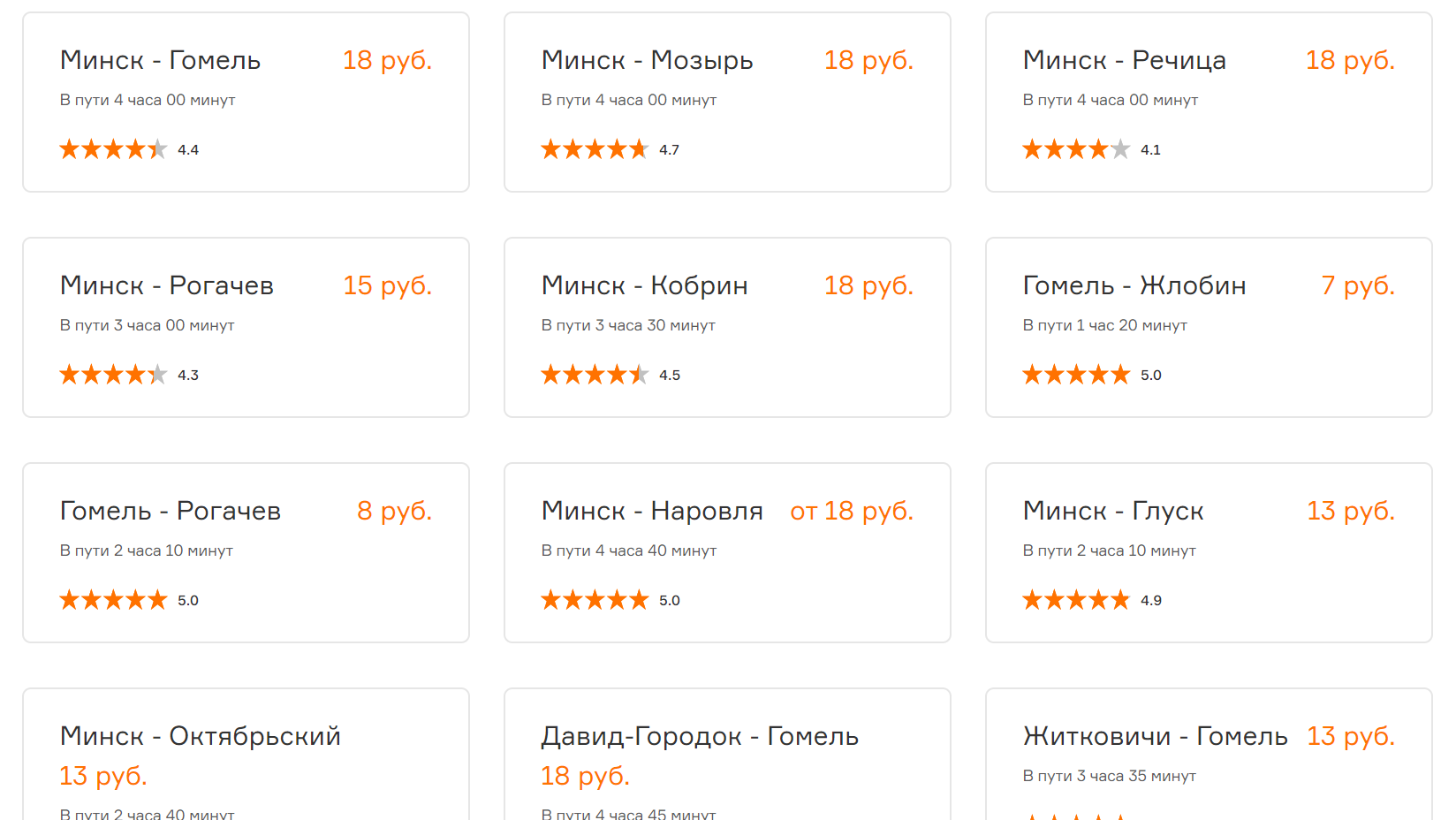


Рисунок 1.2 – Список популярных поездок

Исходя из анализа данного сайта, можно выделить следующее: на главной странице можно увидеть поле для поиска необходимых маршрутов для поездок; пользователь может найти необходимый ему маршрут и осуществить бронирование места в маршрутке.

1. BlaBlaCar

BlaBlaCar – ведущий мировой туристический сервис с активным сообществом путешественников. Куда бы вы ни собирались, на автобусе или с попутчиками, вы сможете найти свою идеальную поездку среди множества маршрутов и доехать по самой низкой цене

Функционал данного сайта схож с предыдущим, однако стоит выделить следующее: есть возможность осуществлять поездки не только в рамках страны. Пользователи могут осуществлять как поиск водителей с частным автомобилем, так и другие виды транспорта, осуществляющие поездки в указанном клиентом маршруте. Идея создания моего веб-приложения лежала в основе данного сервиса. Скриншот главной страницы сервиса представлен на рисунке 1.3.

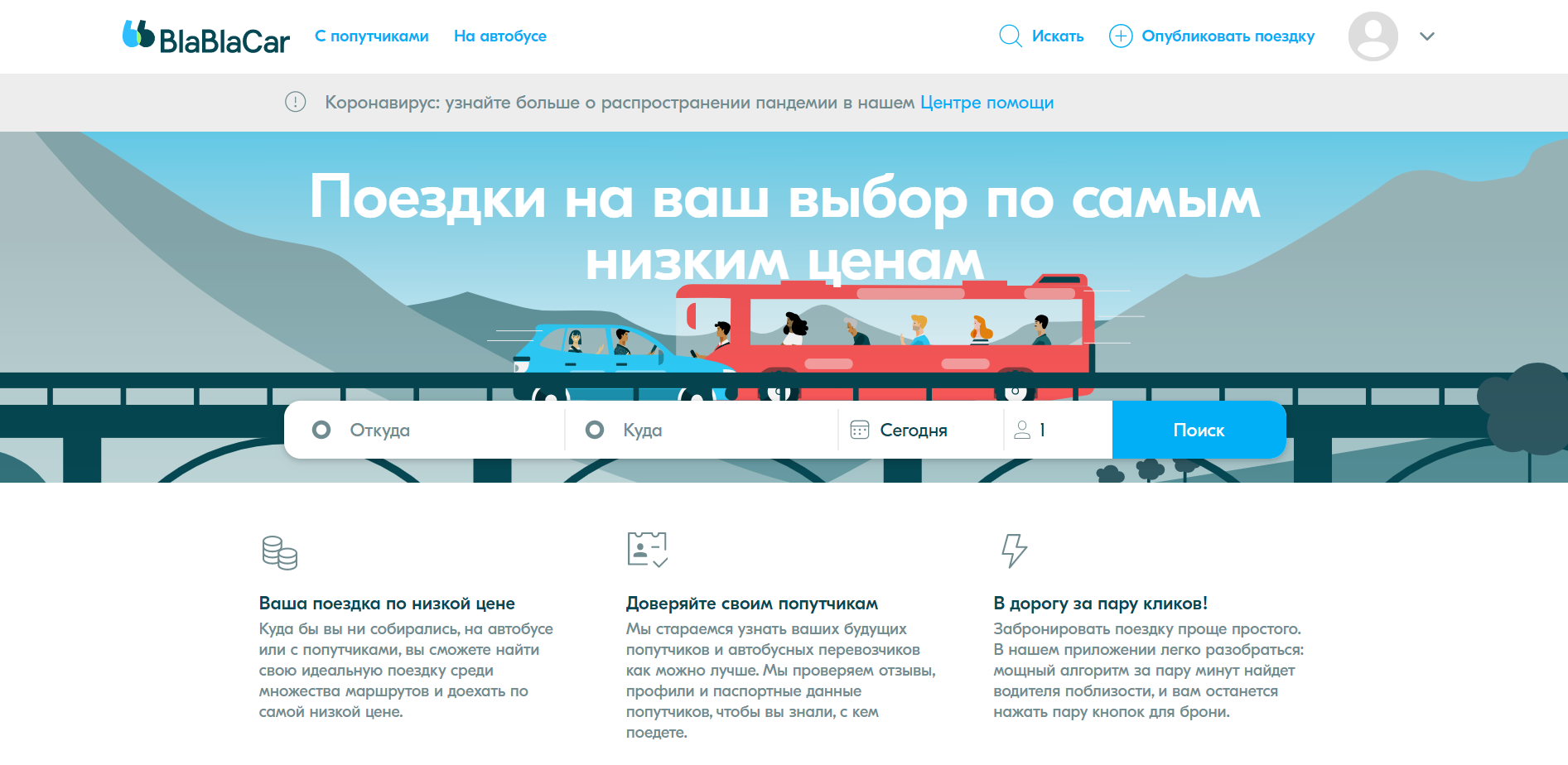


Рисунок 1.3 – Главная страница BlaBlaCar

Пользователь может зарегистрироваться и полноценно пользоваться сервисом, а также после регистрации и прохождения ряда проверок он может не только бронировать поездки, но и сам публиковать их. Ещё одно существенное отличие от предыдущего аналога в том, что нет явного разделения ролей, это значительно удобнее, так как пользователь получает больше возможностей. Но, очевидно, всё зависит от конечной цели.

## **1.2 Исходные данные к проекту**

Программное средство должно быть выполнено с использованием асинхронного программирования, взаимодействовать с базой данных, реализовано под разными платформами. Отображение, бизнес логика и хранилище данных должны быть максимально независимы друг от друга для возможности расширения. Диаграммы вариантов использования, классов реализации задачи, последовательности разработать на основе UML. Язык разработки проекта JavaScript, платформа Node.js. Программное средство должно представлять собой web-приложение c асинхронным UI (React). Web-приложение должно быть логически завершенным. Управление программой должно быть интуитивно понятным и удобным. Листинги проекта должны содержать комментарии. Также программное средство должно: обеспечивать реализацию 2 явных ролей: пользователь и администратор; обеспечивать регистрацию и авторизацию пользователей; обеспечивать поиск водителей для поездки; обеспечивать создание объявлений в качестве водителя; обеспечивать создание объявлений в качестве попутчика; обеспечивать бронирование мест на поездку; обеспечивать просмотр списка объявлений; позволять попутчикам оставлять комментарии о водителях; позволять администратору блокировать и разблокировать пользователей; позволять администратору просматривать список всех пользователей.

# **Проектирование программного средства**

## **2.1 Выбор платформы**

Для реализации проекта необходимо построить структуру проекта и выбрать платформу разработки.

Структура проекта стоит из трех компонентов:

* серверная часть;
* клиентская часть;
* база данных.

## **Серверная часть**

Для реализации серверной части была выбрана платформа Node.js. Node или Node.js ­­– программная платформа, основанная на движке V8, превращающая JavaScript из узкоспециализированного языка в язык общего назначения [1].

Для работы также понадобится NPM (node package manager), который позволяет устанавливать необходимые библиотеки для реализации обработки запросов.

Для обработки запросов существует множество разный библиотек и фреймворков, таких как Koa.js, Nest.js, Http, Meteor.js, Socket.io, Sails.js, Total.js. В данном приложении будет использоваться фреймворк Express.js. Express – это минималистичный и гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений.

## **Клиентская часть**

Для реализации клиентской части была выбрана JavaScript библиотека React.  React — это декларативная, эффективная и гибкая JavaScript-библиотека для создания пользовательских интерфейсов. Она позволяет вам собирать сложный UI из маленьких изолированных кусочков кода, называемых «компонентами».[2]

## **База данных**

В качестве СУБД был выбран PostgreSQL. Это может быть обосновано рядом причин:

* поддержка БД неограниченного размера;
* мощные и надёжные механизмы транзакций и репликации;
* расширяемая система встроенных языков программирования и поддержка загрузки C-совместимых модулей;
* наследование;
* легкая расширяемость
* экономия материальных средств при использовании на хостинге относительно многих других СУБД.

Для организации баз данных PostgreSQL использует реляционную модель. Реляционная база данных – это набор данных с предопределенными связями между ними.

## **Диаграмма вариантов использования**

Для начала разработки необходимо разработать диаграмму вариантов использования. Диаграмма вариантов использования описывает динамические аспекты поведения системы в виде блок-схемы, которая отражает бизнес-процессы, логику процедур и потоки работ — переходы от одной деятельности к другой. Диаграмма вариантов использования представлена на рисунке 2.1.

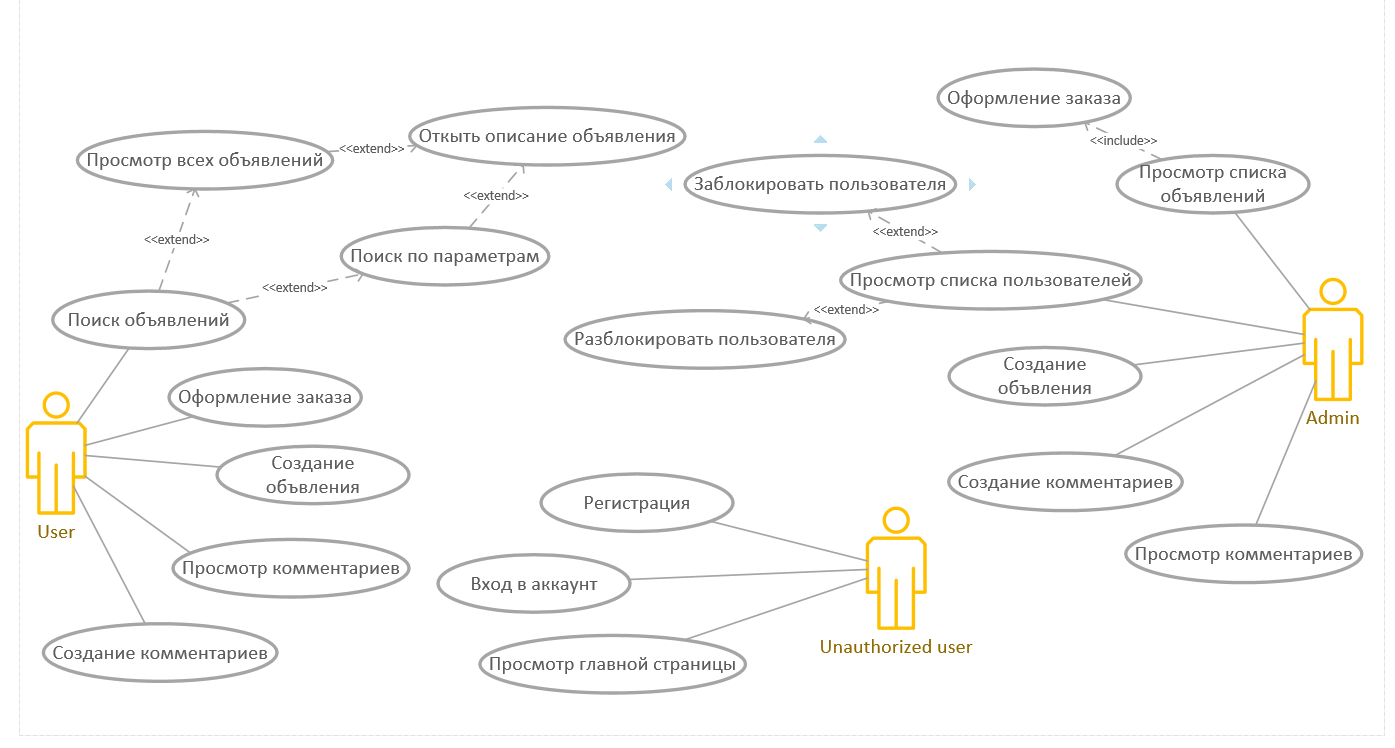


Рисунок 2.1 – Диаграмма вариантов использования

Благодаря данной диаграмме в процессе разработки будет легче понимать, что из себя должно представлять веб-приложение.

## **Проектирование базы данных**

Для начала разработки проекта необходимо спроектировать базу данных. Чтобы составить визуальную взаимосвязанную структуру базы данных, нам необходимо продумать, какая информация будет храниться в этих таблицах, после этого создать связи с помощью первичных и внешних ключей.

Диаграмма базы данных представляет собой таблицы, которые содержат названия столбцов таблиц, а также тип, который будут принимать столбцы. В ходе проектирования была создана диаграмма базы данных со связями, которая продемонстрирована на рисунке 2.2.

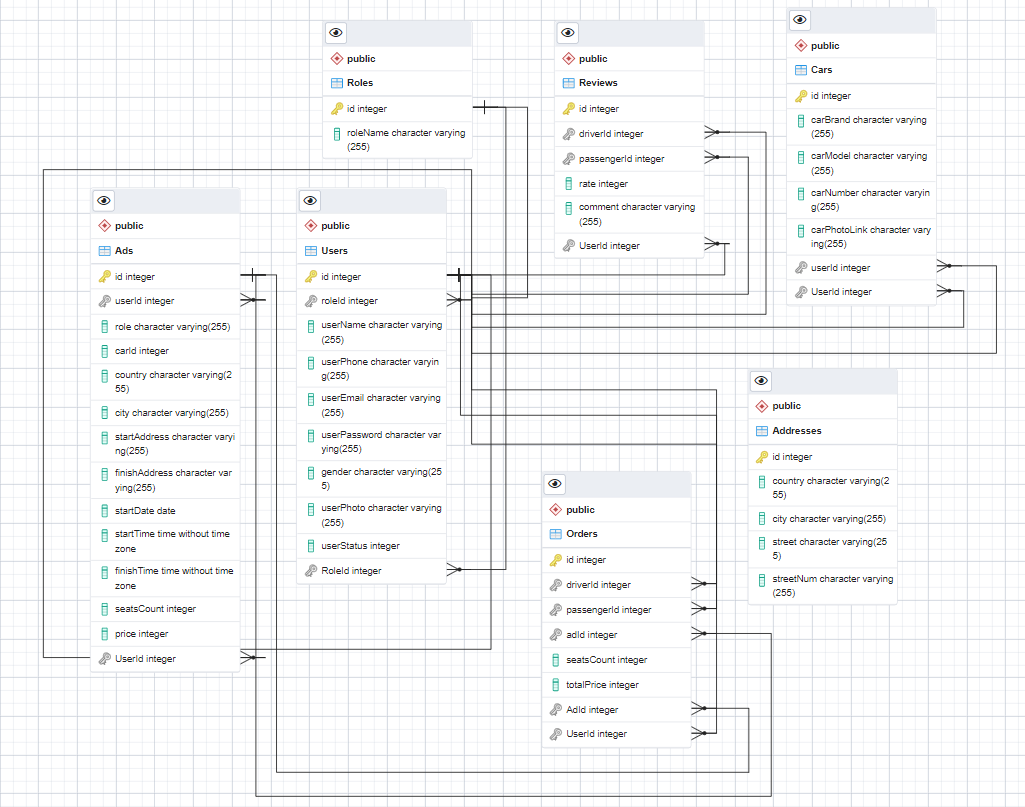


Рисунок 2.2 – Диаграмма базы данных

База данных приложения состоит из 7 таблиц, взаимосвязанных между собой внешними ключами:

* Addresses;
* Ads;
* Cars;
* Orders;
* Reviews;
* Roles;
* Users.

Таблица Addresses представляет собой таблицу с адресами, продемонстрирована в таблице 2.1.

Таблица 2.1 – Столбцы таблицы Addresses

|  |  |
| --- | --- |
| Наименование | Описание, тип |
| id | Идентификатор адреса, тип number, первичный ключ |
| country | Название страны, тип text |
| city | Название города, тип text |
| street | Название улицы, тип text |
| streetNum | Идентификатор здания, тип text |

Таблица Users представляет собой таблицу с пользователями, зарегистрированными в сервисе, продемонстрирована в таблице 2.2.

Таблица 2.2 – Столбцы таблицы Users

|  |  |
| --- | --- |
| Наименование | Описание, тип |
| id | Идентификатор пользователя, тип number, первичный ключ |
| roleId | Идентификатор роли пользователя, вторичный ключ, тип number |
| userName | Уникальное имя пользователя, тип text |
| userPhone | Номер телефона проьзователя, тип text |
| userEmail | Почта пользователя, тип text |
| userPassword | Хеш пароля пользователя, тип text |
| gender | Пол пользователя, тип text |
| userPhoto | Ссылка на фотографию пользователя, тип text |
| userStatus | Статус пользователя, тип number |

Таблица Roles представляет собой таблицу с ролями пользователей, продемонстрирована в таблице 2.3.

Таблица 2.3 – Столбцы таблицы Roles

|  |  |
| --- | --- |
| Наименование | Описание, тип |
| id | Идентификатор роли, тип number, первичный ключ |
| roleName | Название роли, тип text |

Таблица Reviews представляет собой таблицу с отзывами и комментариями пользователей, продемонстрирована в таблице 2.4.

Таблица 2.4 – Столбцы таблицы Reviews

|  |  |
| --- | --- |
| Наименование | Описание, тип |
| id | Идентификатор отзыва, тип number, первичный ключ |
| driverId | Идентификатор пользователя-водителя, вторичный ключ, тип number |
| passengerId | Идентификатор пользователя-пассажира, вторичный ключ, тип number |
| rate | Оценка пользователя, тип number |
| comment | Комментарий пользователя, тип text |

Таблица Orders представляет собой таблицу с заказами пользователей, продемонстрирована в таблице 2.5.

Таблица 2.5 – Столбцы таблицы Orders

|  |  |
| --- | --- |
| Наименование | Описание, тип |
| id | Идентификатор заказа, тип number, первичный ключ |
| driverId | Идентификатор пользователя-водителя, вторичный ключ, тип number |
| passengerId | Идентификатор пользователя-пассажира, вторичный ключ, тип number |
| adId | Идентификатор объявления, вторичный ключ, тип number |
| seatsCount | Количество мест, тип number |
| totalPrice | Цена за поездку, тип number |

Таблица Cars представляет собой таблицу с автомобилями пользователей, продемонстрирована в таблице 2.6.

Таблица 2.6 – Столбцы таблицы Cars

|  |  |
| --- | --- |
| Наименование | Описание, тип |
| id | Идентификатор автомобиля, тип number, первичный ключ |
| carBrand | Название марки авто, тип text |
| carModel | Название модели авто, тип text |
| carNumber | Номер автомобиля, тип text |
| carPhotoLink | Ссылка на фотографию авто, тип text |
| userId | Идентификатор владельца авто, вторичный ключ, тип number |

Таблица Ads представляет собой таблицу с объявлениями пользователей, продемонстрирована в таблице 2.7.

Таблица 2.7 – Столбцы таблицы Ads

|  |  |
| --- | --- |
| Наименование | Описание, тип |
| id | Идентификатор объявления, тип number, первичный ключ |
| userId | Идентификатор создателя объявления, вторичный ключ, тип number |
| role | Промежуточная роль пользователя, тип text |
| carId | Идентификатор автомобиля, тип number |
| country | Страна, тип text |
| city | Город, тип text |
| startAddress | Адрес отправления, тип text |
| finishAddress | Адрес прибытия, тип text |
| startDate | Дата поездки, тип date |
| startTime | Время отправления, тип time |
| finishTime | Время прибытия, тип time |
| seatsCount | Количество мест, тип number |
| price | Цена за поездку, тип number |

## **Описание структуры проекта**

Проект представляет собой два приложения: клиент и сервер.

Для реализации клиента будет использоваться JavaScript-библиотека React. Ключевая возможность React — это композиция компонентов. Компоненты, написанные разными людьми, должны хорошо работать вместе. Для нас важно, чтобы вы могли добавлять в компонент функциональность, не вызывая волну изменений по всему коду. Паттерн Container/Component [4]. Компоненты контейнера могут в первую очередь называться родительскими элементами других компонентов в приложении React. Они служат мостом между обычными компонентами, которые отображают пользовательский интерфейс, и логикой, которая делает компоненты пользовательского интерфейса интерактивными и динамичными.

Наиболее распространенной функцией компонента контейнера является получение данных. Получение данных не означает традиционный способ извлечения данных из конечной точки API, это также связано с логикой компонента React.

Как только процесс выполнения логики или получения данных в компоненте будет завершен, он отображает соответствующий компонент. Иногда компонент контейнера может выполнять две функции (т. е. отображать пользовательский интерфейс и удерживать логику).

Для реализации сервера будет использоваться программная платформа Node.js, а также фреймворк Express.js.

Возможности Express.js:

* отличная производительность. Node был разработан для оптимизации пропускной способности и масштабируемости веб-приложений и является хорошим решением для многих распространенных проблем веб-разработки (например, веб-приложений реального времени);
* код написан на "простом старом JavaScript", что означает, что меньше времени тратится на "сдвиг контекста" между языками, когда вы пишете код как на стороне клиента, так и на стороне сервера;
* javaScript является относительно новым языком программирования и выигрывает от улучшений в языковом дизайне по сравнению с другими традиционными языками веб-серверов (например, Python, PHP и т. Д.). Многие другие новые и популярные языки компилируются/преобразуются в JavaScript, поэтому вы также можете использовать TypeScript, CoffeeScript, ClojureScript, Scala, LiveScript и т.д;
* диспетчер пакетов узлов (NPM) предоставляет доступ к сотням тысяч повторно используемых пакетов. Он также обладает лучшим в своем классе разрешением зависимостей, а также может использоваться для автоматизации большей части цепочки инструментов сборки;
* node.js является портативным. Он доступен в ОС Microsoft Windows, macOS, Linux, Solaris, FreeBSD, OpenBSD, WebOS и NonStop. Кроме того, он хорошо поддерживается многими поставщиками веб-хостинга, которые часто предоставляют специальную инфраструктуру и документацию для размещения узлов сайтов;
* у него очень активная сторонняя экосистема и сообщество разработчиков, с большим количеством людей, которые готовы помочь.

Также для реализации будет использоваться паттерн MVC, Repository, DTO.

Паттерн MVC. Паттерн MVC включает ряд компонентов:

* модели определяют структуру и логику используемых данных;
* представления (views) определяет визуальную часть, как данные будут отображаться;
* контроллеры обрабатывают входящие http-запросы, используя для обработки модели и представления, и отправляет в ответ клиенту некоторый результат обработки, нередко в виде html-кода.

Система маршрутизации как дополнительный компонент сопоставляет запросы с маршрутами и выбирает для обработки запросов определенный контроллер.

В общем случае, когда к приложению приходит запрос, система маршрутизации выбирает нужный контроллер для обработки запроса. Контроллер обрабатывает запрос. В процессе обработки он может обращаться к данным через модели и для рендеринга ответа использовать представления. Результат обработки контроллера отправляется в ответ клиенту. Нередко ответ представляет html-страницу, которую пользователь видит в своем браузере. Схема работы продемонстрирована на рисунке 2.3.

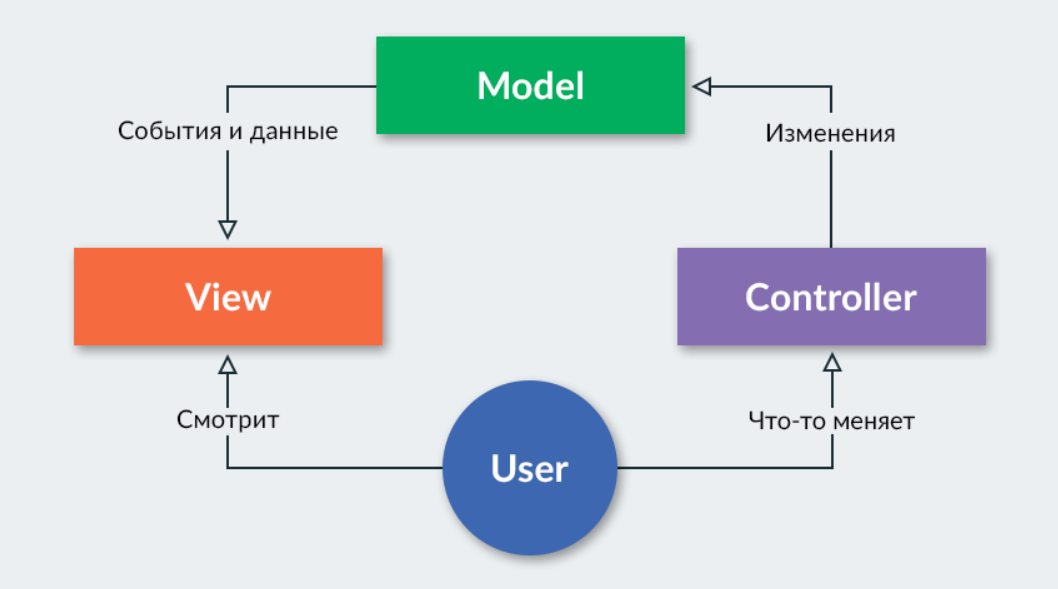


Рисунок 2.3 – Схема работы паттерна MVC

Паттерн Repository. Говоря о паттерне Repository, это представление, в котором вы можете хранить все операции своей базы данных (например, операции создания, чтения, обновления и удаления) в одном локальном для каждого бизнес-объекта, когда вам нужно выполнять операции с базой данных, не вызывайте напрямую драйверы базы данных, и если у вас есть несколько баз данных или разные базы данных для одной транзакции, ваше приложение вызывает только метод репозитория и прозрачно для тех, кто вызывает. Паттерн Repository отделяет логику, которая извлекает данные и сопоставляет их с моделью сущности, от бизнес-логики, которая действует на модель. Паттерн Repository скрывает детали того, как именно данные извлекаются или сохраняются из/в базу данных Паттерн Repository работает между уровнем источника данных и бизнес-уровнями приложения. В шаблоне хранилища логика доступа к данным и бизнес-логика взаимодействуют друг с другом с помощью интерфейсов.[5] Схема работы продемонстрирована на рисунке  2.4.

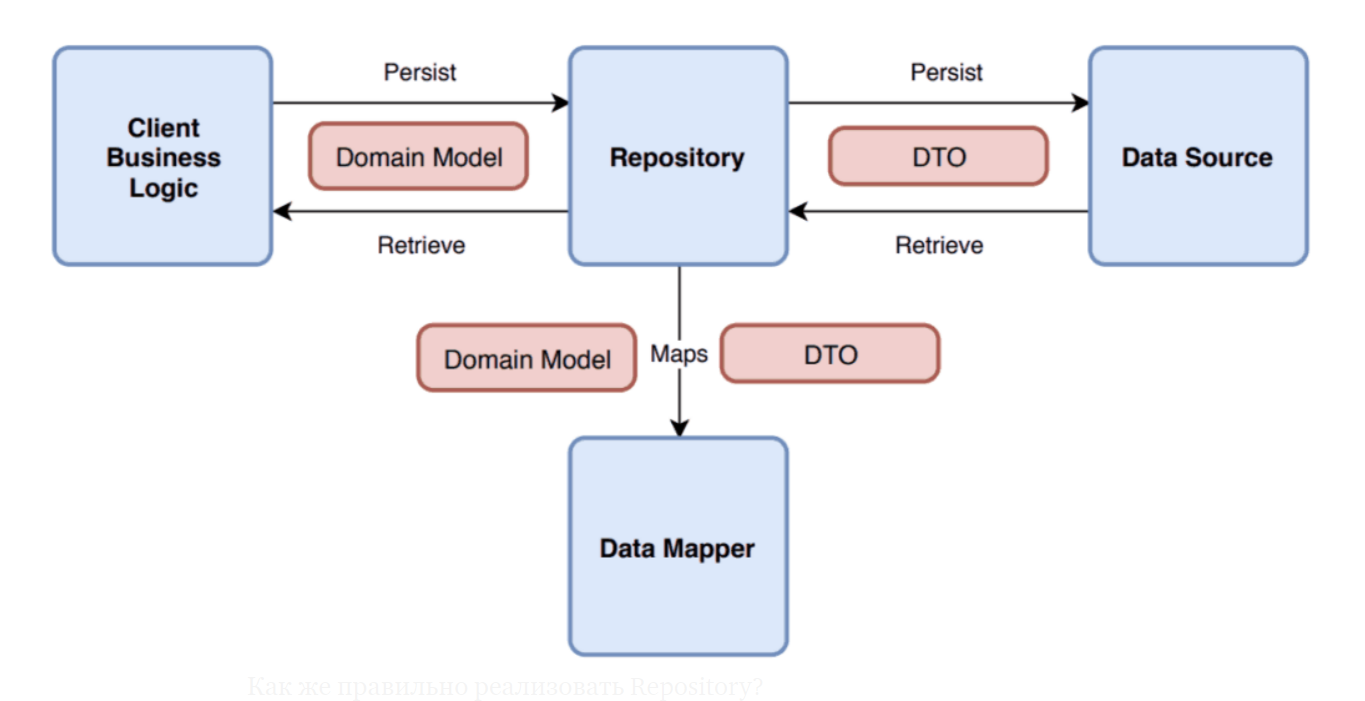


Рисунок 2.4 – Схема работы паттерна Repository

Паттерн DTO (Data Transfer Object). Зачастую, в клиент-серверных приложениях, данные на клиенте (слой представления) и на сервере (слой предметной области) структурируются по-разному. На стороне сервера это дает нам возможность комфортно хранить данные в базе данных или оптимизировать использование данных в угоду производительности, в то же время заниматься “user-friendly*”* отображением данных на клиенте, и, для серверной части, нужно найти способ как переводить данные из одного формата в другой. Конечно, существуют и другие архитектуры приложений, но мы остановимся на текущей в качестве упрощения. DTO-подобные объекты могут использоваться между любыми двумя слоями представления данных. Схема работы продемонстрирована на рисунке 2.5.

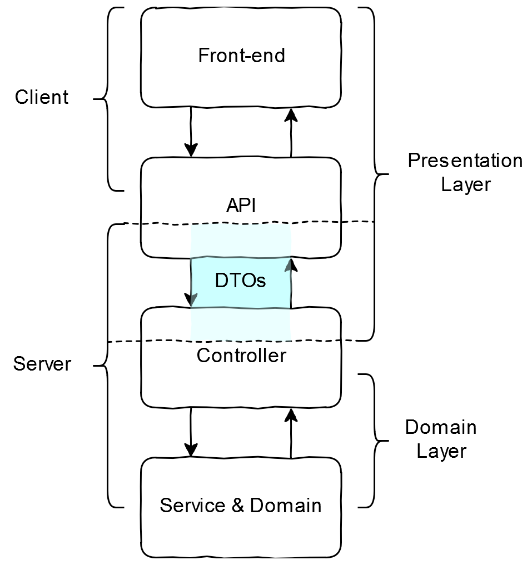


Рисунок 2.5 – Схема работы паттерна DTO

После того, как приложение было спроектировано, можно приступать к его разработке.

## **Разработка программного средства**

При проектировании программного средства в качестве программной платформы для сервера был выбран Node.js. Основной веб-фреймворк для приложения – Express.js.

## **Проектирование серверной части приложения**

Структура серверной части продемострирована на рисунке 3.1.

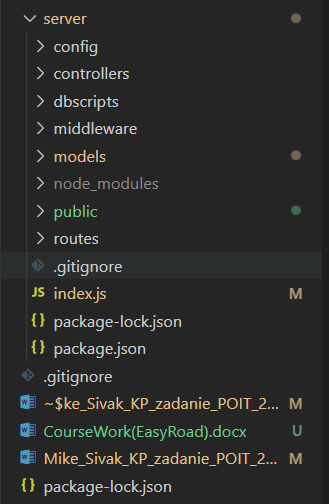


Рисунок 3.1 – Структура серверной части

Директория «node\_modules» хранит необходимые библиотеки для работы приложения. Директория «config» хранит конфигурационные файлы с настройкой подключения к базе данных и секретным ключом для верификации jwt-токена при запросах пользователей. Директория «controllers» содержит в себе файлы, обрабатывающие входящие http-запросы и вызывающие определенные методы работы с данными. Директория «dbscripts» хранит sql-файл с операциями заполнения базы данных для ускорения процесса разработки. Директория «models» хранит модели ORM для генерации структуры базы данных приложения. Директория «middlewares» хранит файлы с middleware-функциями. Директория «public»содержит в себе статические файлы. Директория «routes» хранит модули маршрутизации. Листинги маршрутизаторов будут представлены в приложении А

Для реализации аутентификации была использована библиотека jsonwebtoken. Реализация аутентификации представлена на листинге 3.1.

|  |
| --- |
| verifyToken = (req, res, next) => {    let token = req.headers["x-access-token"];    if (!token) {      return res.status(403).send({        message: "No token provided!"      });    }    jwt.verify(token, config.secret, (err, decoded) => {      if (err) {        return res.status(401).send({          message: "Unauthorized!"        });      }      req.userId = decoded.id;      next();    });  };  isAdmin = (req, res, next) => {    User.findByPk(req.userId)      .then(user => {        if(user.roleId == 1){          console.log("Admin: " + user.userName);          next();        }        else{          res.status(403).send("You have not access to admin");        }      });    return;  }; |

Листинг 3.1 – Реализация аутентификации с помощью jsonwebtoken

Для работы с базой данных PostgreSQL использовалась библиотека Sequelize. Использование библиотеки представлена на листинге 3.2.

|  |
| --- |
| const dbConfig = require('../config/db.config.js');  const Sequelize = require('sequelize');  // connection to created database  const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASSWORD, {      host: dbConfig.HOST,      // port: dbConfig.PORT, -- is not require for some databases      dialect: 'postgres',      operatorsAliases: 0,  });  // create object for save database entities  const db = {}  db.Sequelize = Sequelize;  db.sequelize = sequelize;  db.Roles = require('./Roles.model.js')(sequelize, Sequelize);  db.Users = require('./Users.model.js')(sequelize, Sequelize);  db.Addresses = require('./Addresses.model.js')(sequelize, Sequelize);  db.Cars = require('./Cars.model.js')(sequelize, Sequelize);  db.Ads = require('./Ads.model.js')(sequelize, Sequelize);  db.Orders = require('./Orders.model.js')(sequelize, Sequelize);  db.Reviews = require('./Reviews.model.js')(sequelize, Sequelize); |

Листинг 3.2 – Использование библиотеки Sequelize

Также была необходимость в написании такого кода, который проверял бы статус пользователя для предоставления доступа к сервису. Для этого использовался код, который приведен в листинге 3.3.

|  |
| --- |
| exports.checkStatus = (req, res, next) => {      User.findByPk(req.userId)      .then(user => {        if(user.userStatus == 1){          console.log("User Status: " + user.userStatus + " - You have access to application :)");          next();        }        else{          console.log("User Status: " + user.userStatus + " - You have not access to application :(");          res.status(403).send("Ваш профиль был заблокирован!"); |

Листинг 3.3 – Реализация изменение роли пользователя

Для того, чтобы проверять доступ к функциям сервиса, которые предназначены только для администратора, используется промежуточная функция, которая проверяет роль пользователя, пример использования продемонстрирован в листинге 3.4.

|  |
| --- |
| isAdmin = (req, res, next) => {    User.findByPk(req.userId)      .then(user => {        if(user.roleId == 1){          console.log("Admin: " + user.userName);          next();        }        else{          res.status(403).send("You have not access to admin");        }      });    return;  }; |

Листинг 3.4 – Использование middleware-функции checkRoles

## **Проектирование клиентской части приложения**

Для реализации клиентской части используется JavaScript-библиотека React, а также для хранения состояния используется локальное хранилище браузера — localstorage.

Структура клиентской части продемонстрирована на рисунке 3.2.

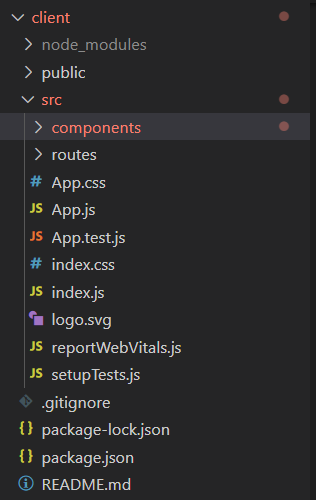


Рисунок 3.2 – Структура клиентской части

Директория «node\_modules» хранит необходимые библиотеки для работы приложения. Директория «src» хранит необходимую логику для работы приложения. Директория «public» хранит публичный файлы приложения. Директория «components» хранит директории и файлы с React-компонентами. Директория «routes» хранит составной компонент маршрутизации на стороне клиента.

Для создания объявлений был реализован компонент NewAd, листинг которого продемонстрирован в приложении Б.

## **Тестирование программного средства**

Для обеспечения корректности работы программы, обрабатываются различные ошибки, возникающие в процессе работы. Данное программное средство использует подключение к базе данных, следовательно, неправильно введенные данные или же их отсутствие может повлечь за собой неработоспособность приложения.

В курсовом проекте организована валидация полей на случай неочевидного или неправильного поведения пользователя.

Валидация приложения – это один из основных гарантов надёжности приложения и всякий программист должен предусмотреть и предотвратить непредвиденное поведение пользователя.

При авторизации обрабатываются вводимые данные пользователя. Если пользователь оставил поля пустыми, на форме появится сообщение с ошибкой, как показано на рисунке 4.1.

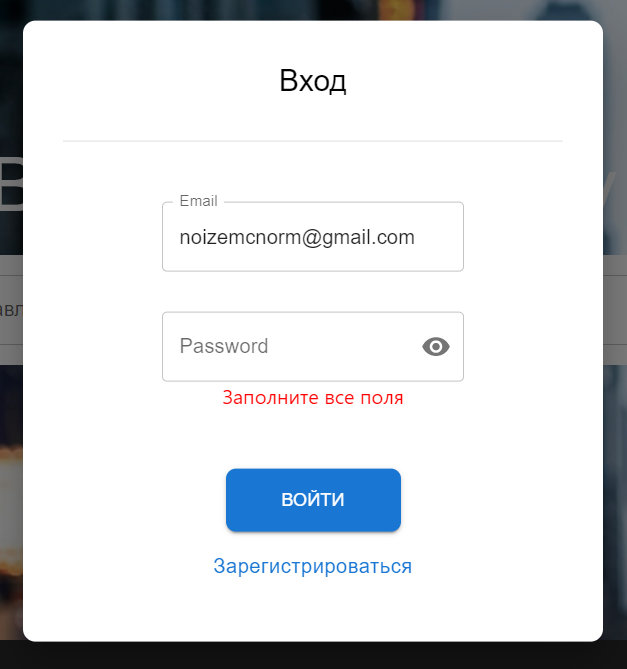


Рисунок 4.1 – Сообщение о незаполненных данных

Также, в случае если пользователь без регистрации попытается перейти по защищённому маршруту, введя в адресную строку какой-либо маршрут, пользователя перенаправит на страницу с ошибкой 401–Unauthorized, это представлено на рисунке 4.2.

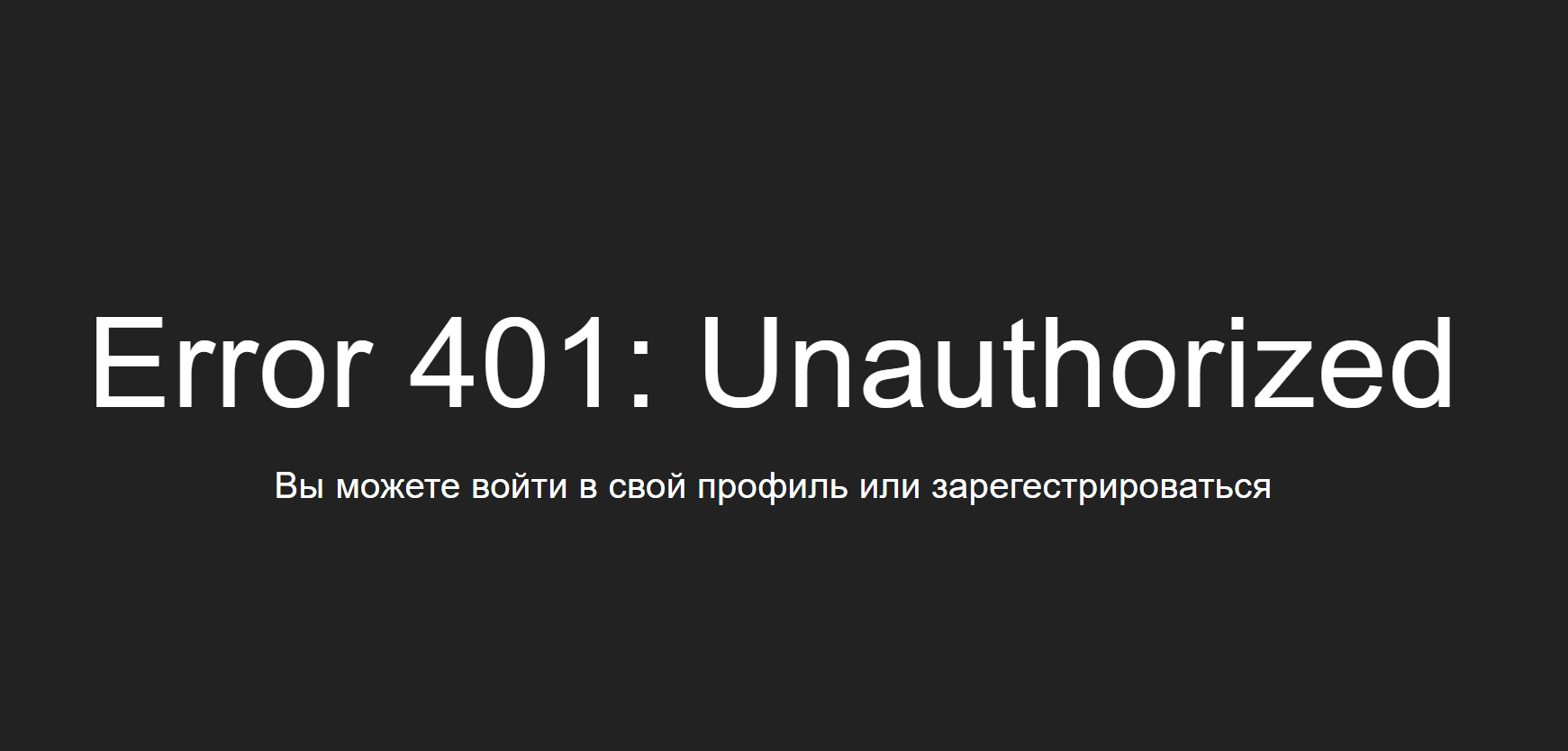


Рисунок 4.2 – Ошибка 401–Unauthorized

Когда зарегистрированный пользователь будет создавать объявление в качестве водителя или принимать заявки попутчиков, будет проведена проверка на наличие автомобиля в профиле пользователя. Если пользователь не добавил своё авто в профиль, на экране появится окно с соответствующим сообщением, пример уведомления об ошибке предоставлен на рисунках 4.3, 4.4.

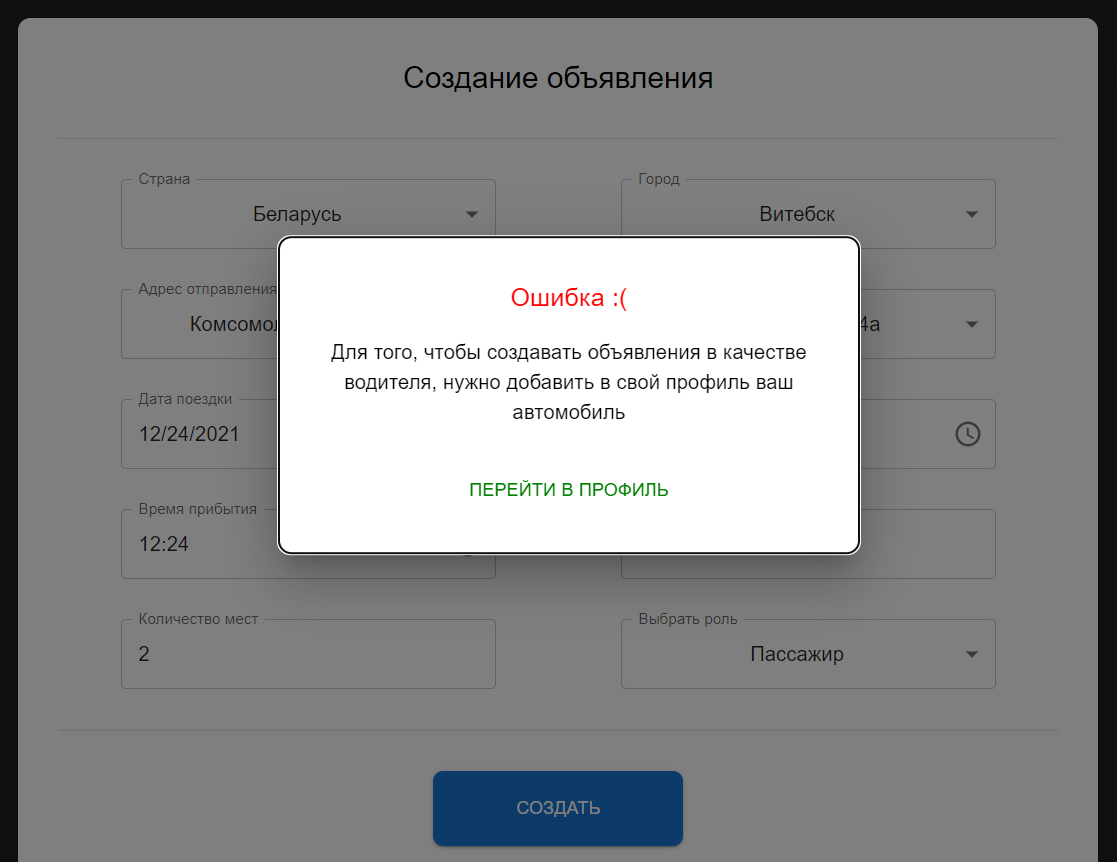


Рисунок 4.3 – Ошибка при попытке создать объявление в качестве водителя

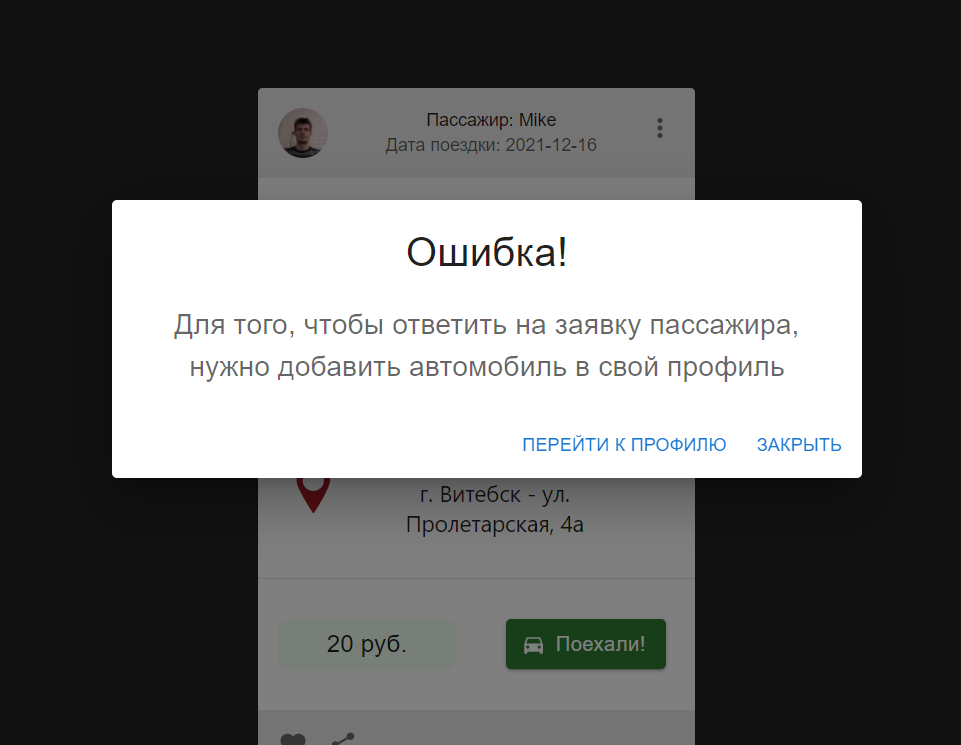


Рисунок 4.4 – Ошибка при попытке принять заявку пассажира при отсутствии авто

Если заблокированный пользователь попытается что-либо сделать на сайте, его перенаправит на страницу со статусом ошибки 403–You have not access, пример ошибки предоставлен на рисунке 4.5

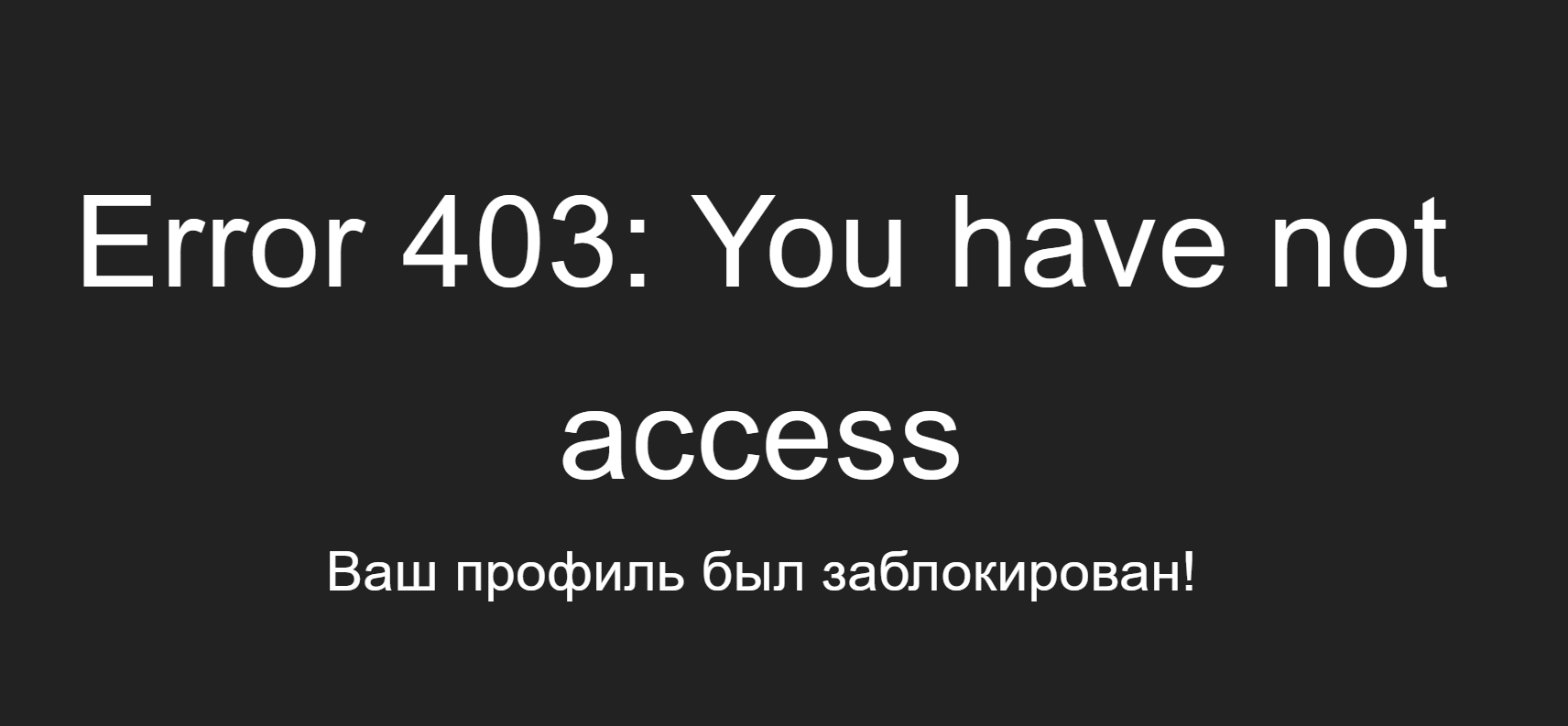


Рисунок 4.5 – 403–You have not access

Тестирование необходимо для того, чтобы исключить все вероятные ошибки пользователей при работе с приложением. В ходе тестирования были проведена проверка на:

* попытка входа с не введенными данными;
* некорректный ввод логина при регистрации;
* попытка входа на страницы, предназначенных для определенных ролей;
* удаление чужого поста.

## **Руководство пользователя**

Для работы с приложением, необходимо авторизоваться или зарегистрироваться, окна авторизации и регистрации представлены на рисунках 5.1, 5.2.

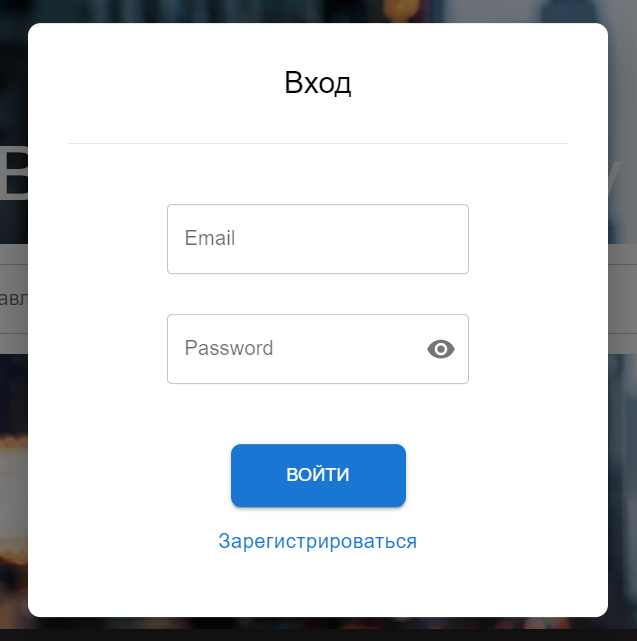


Рисунок 5.1 – Страница авторизации

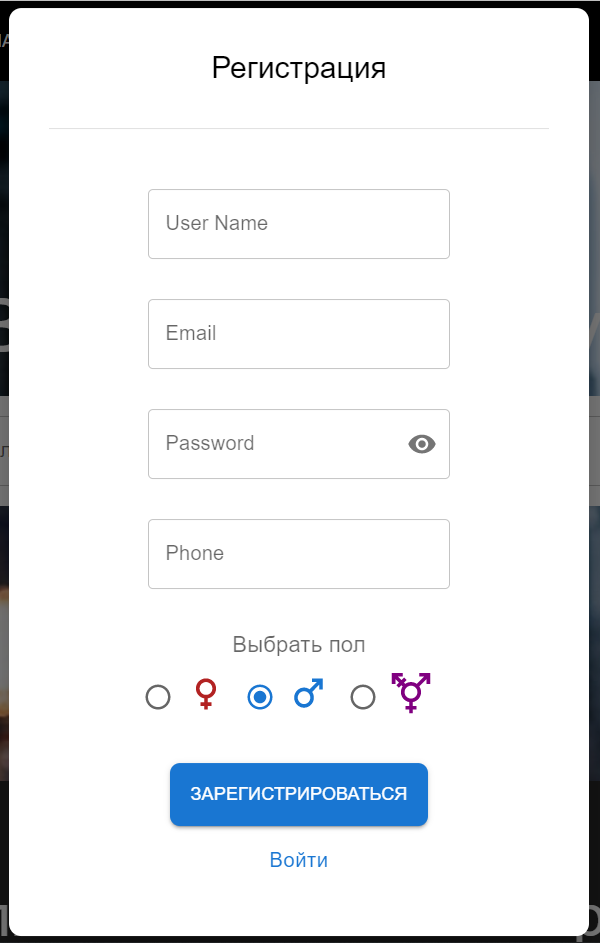


Рисунок 5.2 – Страница регистрации

На странице объявлений можно увидеть список всех объявлений и осуществить заказ, а также на главной странице можно вести поиск объявлений. Страница с объявлениями и процесс поиска представлены на рисунках 5.3, 5.4.

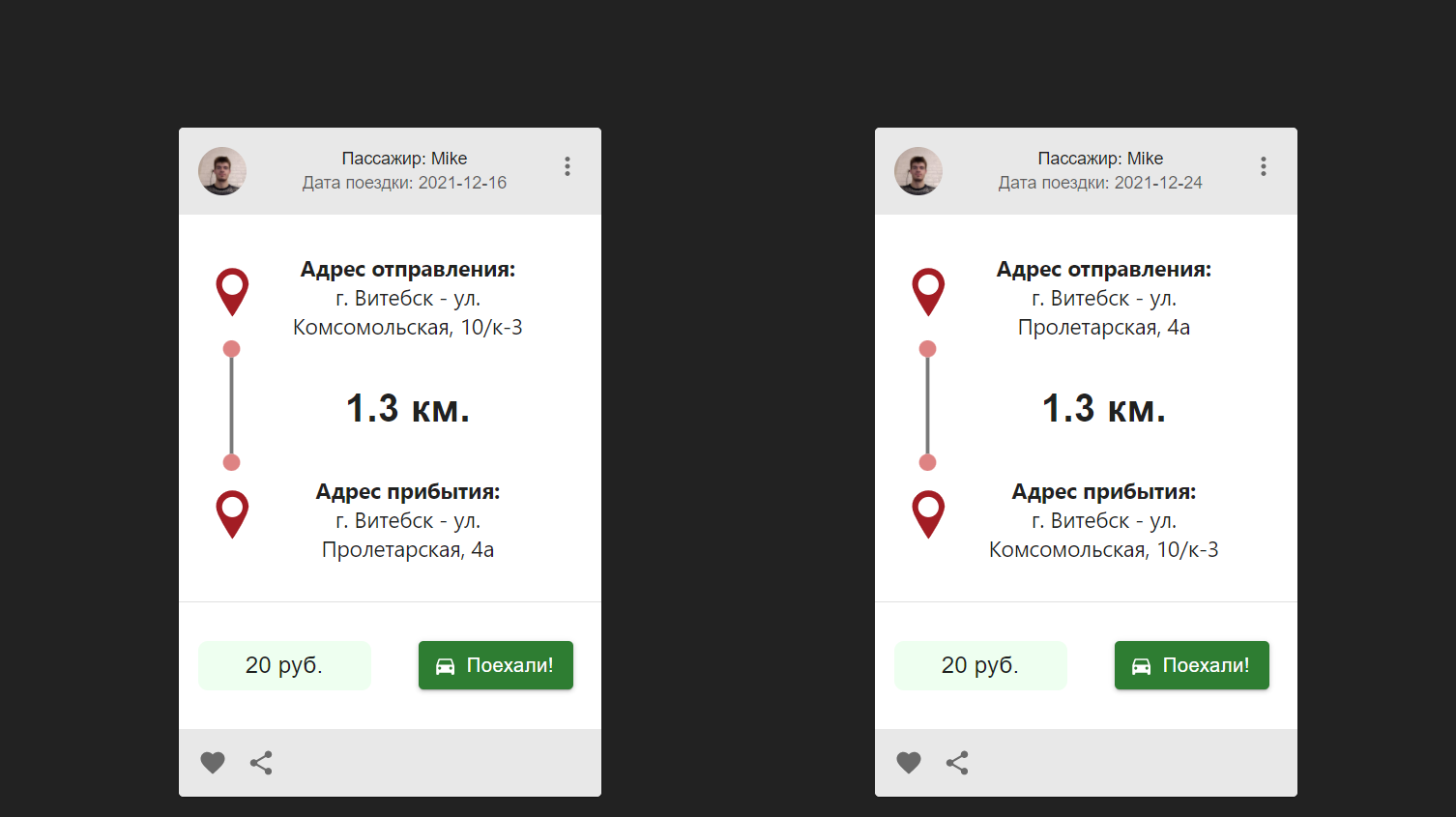


Рисунок 5.3 – Страница объявлений

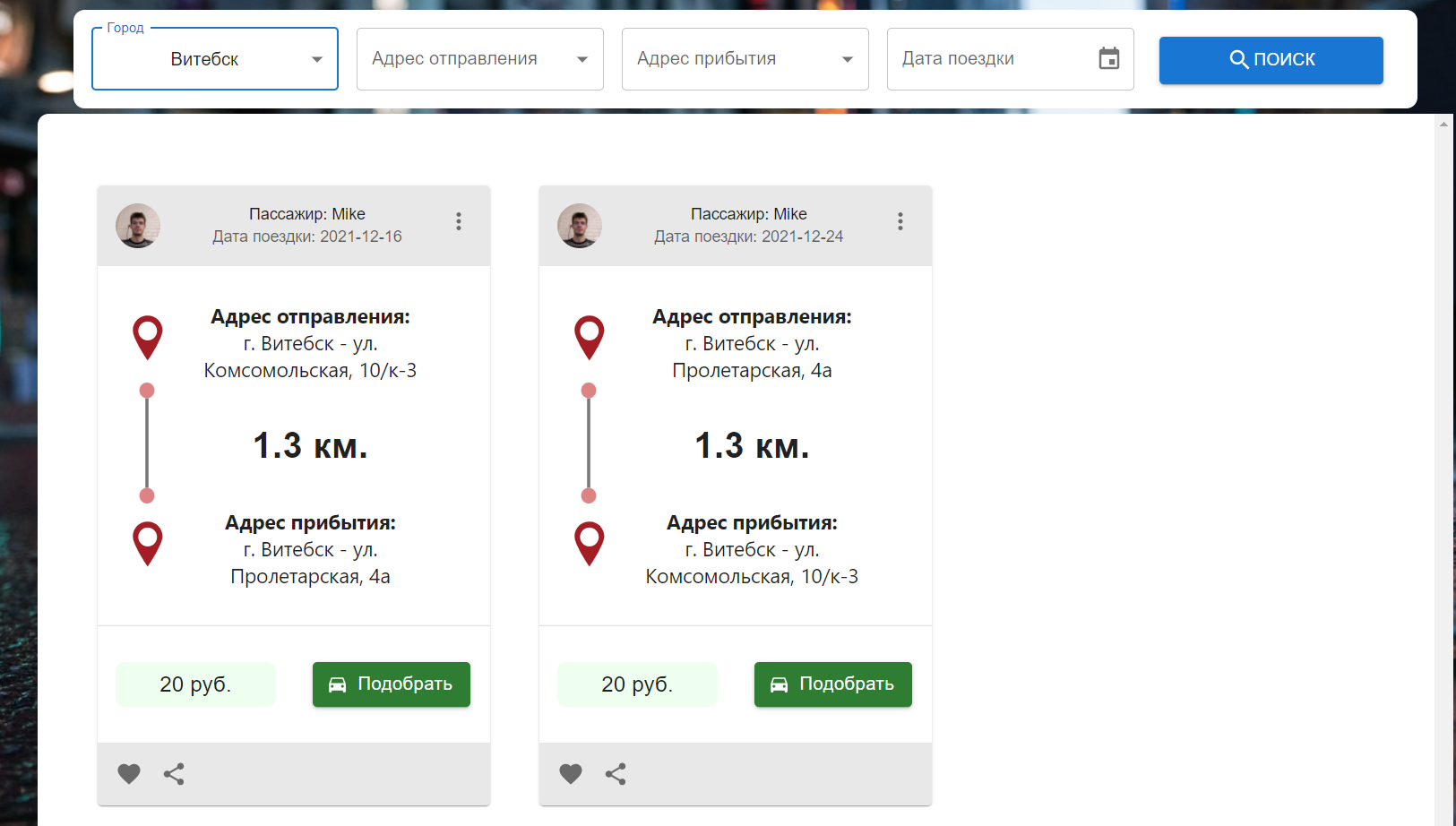


Рисунок 5.4 – Процесс поиска объявлений

На сайте есть страница профиля, где можно посмотреть всю информацию своего профиля, представлено на рисунке 5.5.

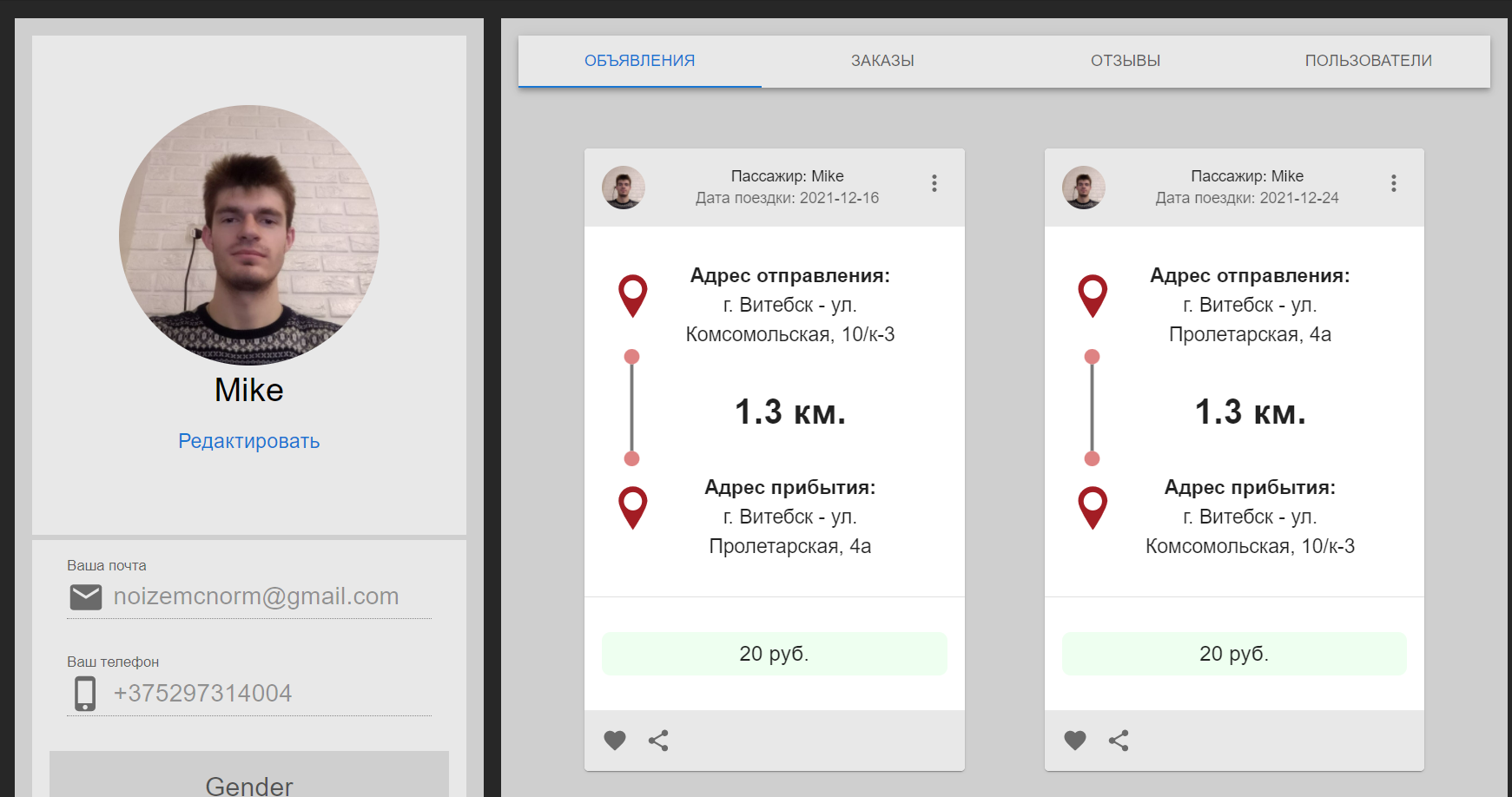


Рисунок 5.5 – Страница профиля

Страница профиля содержит следующие элементы:

* информацию о пользователе;
* объявления пользователя;
* заказы пользователя;
* отзывы о пользователе;
* выход из аккаунта;
* для администратора доступна панель администратора;

При нажатии на «Заказы» пользователь попадет на вкладку со своими заказами. Вкладка с заказами представлена на рисунке 5.6.

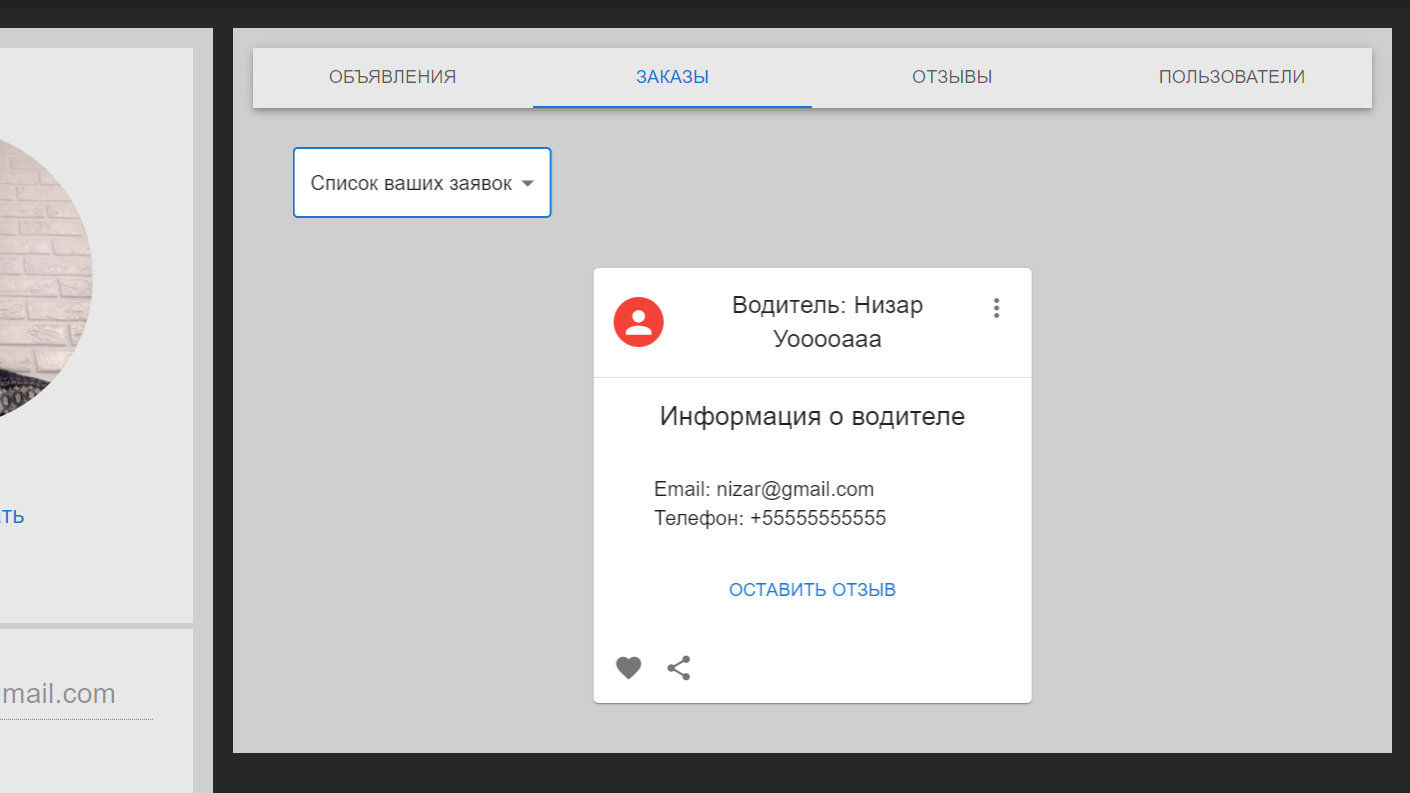


Рисунок 5.6 – Вкладка «Заказы»

При нажатии на «Отзывы» пользователь попадет на вкладку с отзывами. Вкладка с отзывами представлена на рисунке 5.7.

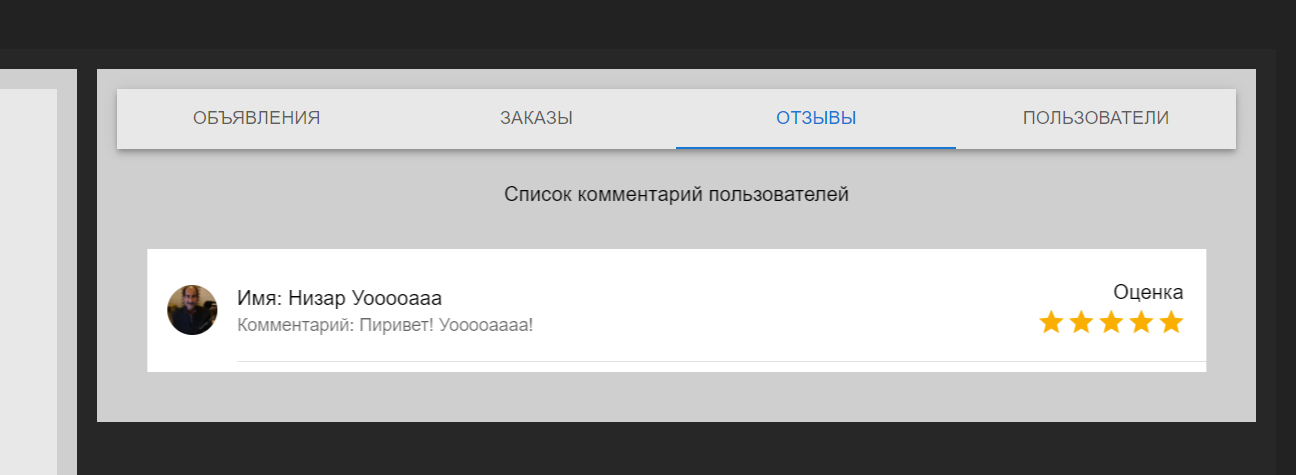


Рисунок 5.7 – Вкладка с отзывами

Когда пользователь открывает список своих заказов в качестве пассажира, он сможет оставить свой комментарий водителю. Окно создания комментария представлено на рисунке 5.8.

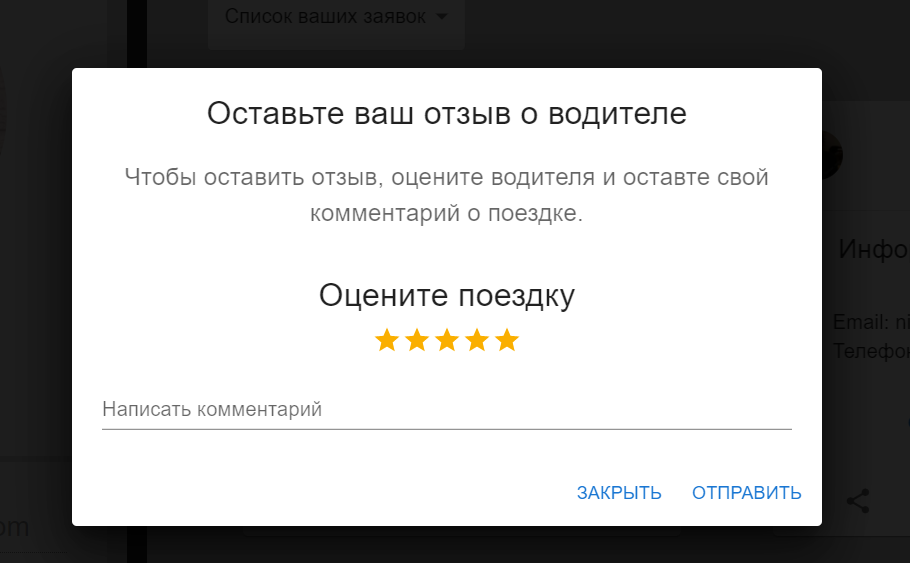


Рисунок 5.8 – Окно создания отзыва о водителе

При нажатии на «Пользователи» администратор может просматривать список всех пользователей, а так же заблокировать или разблокировать пользователей. Вкладка с пользователями представлена на рисунке 5.9.

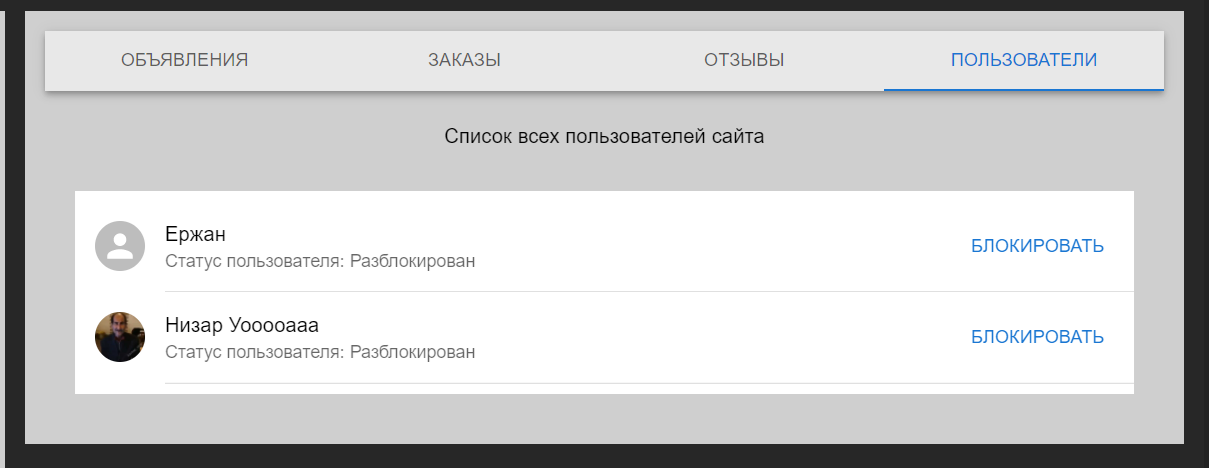


Рисунок 5.9 – Вкладка с пользователями

Если пользователю нужно выйти из системы, то он может нажать на кнопку «Выйти», которая расположена в шапке сайта. Кнопка Logout представлена на рисунке 5.10.

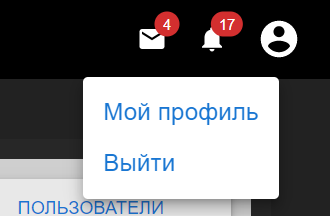


Рисунок 5.16 – Кнопка Log Out

## **Заключение**

В ходе выполнения работы были проанализированы аналоги сервисов для поездок. Выявлены их сильные и слабые стороны, и на основе этого были поставлены задачи.

В данном курсовом проекте были разработаны база данных и сервис для поиска водителей и попутчиков. Были разработаны клиентская и серверная части. Для реализации серверной части использовались: программная платформа Node.js, фреймворк Express.js, а также средства и библиотеки Sequelize, jsonwebtoken, Bcrypt. Для реализации клиентской части использовались библиотеки, такие как React, Material UI. Использовались паттерны Component/Container, MVC. Для работы с базой данных использовалась СУБД PostgreSQL, а также утилита для работы с PostgreSQL Pg Admin 4.

В данном приложении реализованы следующие требования: обеспечивать реализацию 2 явных ролей: пользователь и администратор; обеспечивать регистрацию и авторизацию пользователей; обеспечивать поиск водителей для поездки; обеспечивать создание объявлений в качестве водителя; обеспечивать создание объявлений в качестве попутчика; обеспечивать бронирование мест на поездку; обеспечивать просмотр списка объявлений; позволять попутчикам оставлять комментарии о водителях; позволять администратору блокировать и разблокировать пользователей; позволять администратору просматривать список всех пользователей.

В соответствии с полученным результатом работы программы, можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

## **Список используемых источников**

1. Node.js // Официальная документация [Электронный ресурс]. – Режим доступа: https://nodejs.org/en/docs/ . – Дата доступа: 19.11.2021.
2. React.js // Официальная документация [Электронный ресурс]. – Режим доступа https://ru.reactjs.org/docs/getting-started.html. – Дата доступа: 19.11.2021.
3. Sequelize // Официальная документация [Электронный ресурс]. – Режим доступа https://sequelize.org/. – Дата доступа: 23.11.2021.
4. Container/Component Pattern // Section.io [Электронный ресурс]. – Режим доступа https://www.section.io/engineering-education/container-components-in-react/ – Дата доступа: 19.11.2021.
5. Repository Pattern // Dev.to [Электронный ресурс]. – Режим доступа https://dev.to/fyapy/repository-pattern-with-typescript-and-nodejs-25da. – Дата доступа: 19.11.2021.

## **ПРИЛОЖЕНИЕ А**

**навзание**

|  |
| --- |
| //Admin routes  admin\_router.get('/', admin\_controller.getAdminProfile);  admin\_router.get('/users', admin\_controller.getAllUsers);  admin\_router.post('/user/block', admin\_controller.blockUser);  admin\_router.post('/user/unblock', admin\_controller.unblockUser);  admin\_router.post('/user/delete', admin\_controller.deleteUser);  admin\_router.get('/ads', ads\_controller.getAllAds);  admin\_router.post('/ad/delete', admin\_controller.deleteAd);  //Ads routes  ads\_router.get('/', ads\_controller.getAllAds);  ads\_router.post('/searchAds', ads\_controller.searchAds);  ads\_router.get('/userAds', ads\_controller.getUserAds);  ads\_router.get('/countries', ads\_controller.getCountries);  ads\_router.get('/cities/:country', ads\_controller.getCities);  ads\_router.get('/allcities', ads\_controller.getAllCities);  ads\_router.get('/addresses/:city', ads\_controller.getAddresses);  ads\_router.get('/cars', ads\_controller.getCars);  ads\_router.post('/create', ads\_controller.createAd);  //Auth routes  app.use(function(req, res, next) {      res.header(        "Access-Control-Allow-Headers",        "x-access-token, Origin, Content-Type, Accept"      );      next();    });    app.post(      "/auth/signup",      [        verifySignUp.checkDuplicateUsernameOrEmail,      ],      controller.signup    );    app.post("/auth/signin", controller.signin);  //Orders routes  orders\_router.get('/getreviews', orders\_controller.getUserComments);  orders\_router.get('/:roleOrder', orders\_controller.getUserOrders);  orders\_router.post('/addreview', orders\_controller.createReview);  orders\_router.post('/create', orders\_controller.createOrder);  //Profile routes  profile\_router.get('/', [authJwt.verifyToken], profile\_controller.getProfileInfo);  profile\_router.post('/update', profile\_controller.updateProfileInfo);  profile\_router.post('/addCar', [authJwt.verifyToken], profile\_controller.addCar); |

## **Приложение Б**

|  |
| --- |
| <Box style={{ backgroundColor: '#222222' }}>                      <Modal                          open={open}                          onClose={handleClose}                          aria-labelledby="modal-modal-title"                          aria-describedby="modal-modal-description"                      >                          <Box sx={modalError}>                              <Typography id="modal-modal-title" style={{ textAlign: 'center', color: 'red' }} variant="h6" component="h2">                                  Ошибка :(                              </Typography>                              <Typography id="modal-modal-description" sx={{ mt: 2, textAlign: 'center' }}>                                  Для того, чтобы создавать объявления в качестве водителя, нужно добавить в свой профиль ваш автомобиль                              </Typography>                              <Box style={{ textAlign: 'center', padding: '2em 0 0 0' }}>                                  <Button variant="text" onClick={goToProfile} style={{ margin: '0 auto', color: 'green' }}>Перейти в профиль</Button>                              </Box>                          </Box>                      </Modal>                      <Box>                          <Container style={stylesNewAd.mainContainer} maxWidth='lg'>                              <Box sx={stylesNewAd.newAddBox}>                                  <Typography id="modal-modal-title" sx={{ mb: '2rem' }} variant="h5" component="h1">                                      Создание объявления                                  </Typography>                                  <Divider />                                  <Grid container xs={12}>                                      <Grid item xs>                                          <FormControl sx={{ mx: '1rem', mt: '2rem', maxWidth: 500, minWidth: 300 }} variant='outlined'>                                              {/\* <TextField                                              type={'text'}                                              id="outlined-required-country"                                              label="Страна"                                          /> \*/}                                              <InputLabel id="demo-simple-select-label">Страна</InputLabel>                                              <Select                                                  labelId="demo-simple-select-label"                                                  id="demo-simple-select"                                                  value={country}                                                  label="Страна"                                                  onChange={handleCountryChange}                                              >                                                  {coutries.map((country) => (                                                      <MenuItem value={country.country}>{country.country}</MenuItem>                                                  ))}                                              </Select>                                          </FormControl>                                      </Grid>                                      <Grid item xs>                                          <FormControl sx={{ mx: '1rem', mt: '2rem', maxWidth: 500, minWidth: 300 }} variant='outlined'>                                              {/\* <TextField                                              type={'text'}                                              id="outlined-required-city"                                              label="Город"                                          /> \*/}                                              <InputLabel id="demo-simple-select-label">Город</InputLabel>                                              <Select                                                  labelId="demo-simple-select-label"                                                  id="demo-simple-select"                                                  value={city}                                                  label="Город"                                                  onChange={handleCityChange}                                              >                                                  {cities.map((city) => (                                                      <MenuItem value={city.city}>{city.city}</MenuItem>                                                  ))}                                              </Select>                                          </FormControl>                                      </Grid>                                      <Grid item xs>                                          <FormControl sx={{ mx: '1rem', mt: '2rem', maxWidth: 500, minWidth: 300 }} variant='outlined'>                                              <InputLabel id="demo-simple-select-label">Адрес отправления</InputLabel>                                              <Select                                                  labelId="demo-simple-select-label"                                                  id="demo-simple-select"                                                  value={startAddress}                                                  label="Адрес отправления"                                                  onChange={handleStartAddressChange}                                              >                                                  {addresses.map((address) => (                                                      <MenuItem value={address.street + ', ' + address.streetNum}>{address.street + ', ' + address.streetNum}</MenuItem>                                                  ))}                                              </Select>                                          </FormControl>                                      </Grid>                                      <Grid item xs>                                          <FormControl sx={{ mx: '1rem', mt: '2rem', maxWidth: 500, minWidth: 300 }} variant='outlined'>                                              <InputLabel id="demo-simple-select-label">Адрес прибытия</InputLabel>                                              <Select                                                  labelId="demo-simple-select-label"                                                  id="demo-simple-select"                                                  value={finishAddress}                                                  label="Адрес прибытия"                                                  onChange={handleFinishAddressChange}                                              >                                                  {addresses.map((address) => (                                                      <MenuItem value={address.street + ', ' + address.streetNum}>{address.street + ', ' + address.streetNum}</MenuItem>                                                  ))}                                              </Select>                                          </FormControl>                                      </Grid>                                      <Grid item xs>                                          <FormControl sx={{ mx: '1rem', mt: '2rem', maxWidth: 500, minWidth: 300 }}>                                              <LocalizationProvider dateAdapter={AdapterDateFns}>                                                  <Stack spacing={3}>                                                      <DesktopDatePicker                                                          label="Дата поездки"                                                          inputFormat="MM/dd/yyyy"                                                          value={date}                                                          onChange={dateChange}                                                          renderInput={(params) => <TextField {...params} />}                                                      />                                                  </Stack>                                              </LocalizationProvider>                                          </FormControl>                                      </Grid>                                      <Grid item xs>                                          <FormControl sx={{ mx: '1rem', mt: '2rem', maxWidth: 500, minWidth: 300 }} variant='outlined'>                                              <LocalizationProvider locale={ru} dateAdapter={AdapterDateFns}>                                                  <TimePicker                                                      label="Время отправления"                                                      value={startTime}                                                      inputFormat="HH:mm"                                                      onChange={(newValue) => {                                                          setStartTime(newValue)                                                      }}                                                      renderInput={(params) => <TextField {...params} />}                                                  />                                              </LocalizationProvider>                                          </FormControl>                                      </Grid>                                      <Grid item xs>                                          <FormControl sx={{ mx: '1rem', mt: '2rem', maxWidth: 500, minWidth: 300 }} variant='outlined'>                                              <LocalizationProvider locale={ru} dateAdapter={AdapterDateFns}>                                                  <TimePicker                                                      label="Время прибытия"                                                      value={finishTime}                                                      onChange={(newValue) => {                                                          setFinishTime(newValue)                                                      }}                                                      renderInput={(params) => <TextField {...params} />}                                                  />                                              </LocalizationProvider>                                          </FormControl>                                      </Grid>                                      <Grid item xs>                                          <FormControl sx={{ mx: '1rem', mt: '2rem', maxWidth: 500, minWidth: 300 }} variant='outlined'>                                              <TextField                                                  type={'text'}                                                  id="outlined-required"                                                  label="Цена поездки"                                                  value={price}                                                  onChange={handlePriceChange}                                              />                                          </FormControl>                                      </Grid>                                      <Grid item xs>                                          <FormControl sx={{ mx: '1rem', mt: '2rem', mb: '2rem', maxWidth: 500, minWidth: 300 }} variant='outlined'>                                              <TextField                                                  type={'text'}                                                  id="outlined-required"                                                  label="Количество мест"                                                  value={seatsCount}                                                  onChange={handleSeatsCountChange}                                              />                                          </FormControl>                                      </Grid>                                      <Grid item xs>                                          <FormControl sx={{ mx: '1rem', mt: '2rem', maxWidth: 500, minWidth: 300 }}>                                              <InputLabel id="demo-simple-select-label">Выбрать роль</InputLabel>                                              <Select                                                  labelId="demo-simple-select-label"                                                  id="demo-simple-select"                                                  value={role}                                                  label="Выбрать роль"                                                  onChange={handleRoleChange}                                              >                                                  <MenuItem value={'driver'}>Водитель</MenuItem>                                                  <MenuItem value={'passenger'}>Пассажир</MenuItem>                                              </Select>                                          </FormControl>                                      </Grid>                                      <Grid item xs style={{ display: carField }}>                                          <FormControl sx={{ mx: '1rem', pb: '2em', maxWidth: 800, minWidth: 300 }}>                                              <InputLabel id="demo-simple-select-label">Выбрать авто</InputLabel>                                              <Select                                                  labelId="demo-simple-select-label"                                                  id="demo-simple-select"                                                  value={carId}                                                  label="Выбрать авто"                                                  onChange={handleCarIdChange}                                              >                                                  {cars.map((car) => (                                                      <MenuItem value={car.id}>{car.carBrand} {car.carModel} {car.carNumber}</MenuItem>                                                  ))}                                              </Select>                                          </FormControl>                                      </Grid>                                  </Grid>                                  <Divider />                                  <FormControl sx={{ mx: '1rem', mt: '2rem', maxWidth: 500, minWidth: 300 }} variant='outlined'>                                      <Button                                          variant="contained"                                          sx={{ height: '60px', minWidth: 200, maxWidth: 400, borderRadius: '8px', margin: '0 auto' }}                                          onClick={createAd}                                      >                                          Создать                                      </Button>                                  </FormControl>                              </Box>                          </Container>                      </Box>                  </Box> |