МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ

ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

|  |  |
| --- | --- |
| Факультет | Информационных технологий |
| Кафедра | Программной инженерии |
| Специальность | 1–40 01 01 Программное обеспечение информационных технологий |
| Специализация | 1-40 01 01 10 Программирование интернет-приложений |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к дипломному проекту на тему:**

«Веб-приложение “*EasyRoad* ”»

|  |  |
| --- | --- |
| Дипломник Сивак Михаил Николаевич |  |
| Руководитель проекта ст.преп. И.Г. Сухорукова |  |
| Заведующий кафедрой к.т.н., доц. Н. В. Пацей |  |
| Консультанты: асс. А. С. Пахолко |  |
| асс. А. С. Соболевский |  |
| Нормоконтролер асс. М. В. Макарчук |  |

Дипломный проект защищен с оценкой

Председатель ГЭК к.т.н., доц. В. С. Садов

Минск 2022

Лист задания 2 стороны

**Реферат**

*ФИО*

*Подпись*

*Дата*

Лист

1

4

*БГТУ 00.00.ПЗ*

Разраб.

*Сивак М.Н.*

Провер.

.

*Сухорукова И.Г.*

Консульт.

*Сухорукова И.Г.*

Н. контр.

*Макарчук М.В*

Утв.

*Пацей Н.В.*

*я*

Реферат

Лит.

Листов

1

74417021, 2022

Пояснительная записка содержит 87 страниц, 43 рисунка, 12 таблиц, 11 источников литературы, 8 приложений.

ВОДИТЕЛЬ, ПАССАЖИР, ЗАКАЗ, ОБЪЯВЛЕНИЕ, КЛИЕНТ-СЕРВЕР, JAVASCRIPT, NODE.JS, POSTGRESQL, ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ VISUAL STUDIO CODE, REACT

Пояснительная записка состоит из введения, шести разделов и заключения.

Целью проекта является разработка приложения для поиска попутчиков.

Во введении приводится практическая значимость и цель дипломного проекта, актуальность выбранной темы на данный момент, а также задачи, которые необходимо выполнить для достижения поставленной цели.

Во введении представлена общая информация о текущем состоянии в сфере рассматриваемой проблемы.

В первом разделе представлена общая теоретическая информация по теме, обзор существующих аналогов разрабатываемой системы и постановка задачи.

Во втором разделе произведено проектирование веб-приложения.

В третьем разделе показан процесс разработки веб-приложения.

В четвертом разделе показан процесс тестирования разработанного вебприложения.

В пятом разделе представлена методика использования разработанного веб-приложения.

В шестом разделе представлены результаты себестоимости и отпускной цены разработанного веб-приложения.

В заключении представлены итоги дипломного проекта и задачи, которые были решены в ходе разработки приложения для поиска попутчиков.

Графическая часть приложения включает в себя:

* структурную схему базы данных;
* диаграмму вариантов использования;
* блок-схему алгоритма создания объявления;
* блок-схему алгоритма созадния заказа;
* блок-схему алгоритма добавления отзыва;
* скриншот работы веб-приложения.

**Abstract**

*ФИО*

*Подпись*

*Дата*

Лист

1

5

*БГТУ 00.00.ПЗ*

Разраб.

*Сивак М.Н.*

Провер.

.

*Сухорукова И.Г.*

Консульт.

*Сухорукова И.Г.*

Н. контр.

*Макарчук М.В*

Утв.

*Пацей Н.В.*

*я*

Abstract

Лит.

Листов

1

74417021, 2022

The explanatory note of the diploma project contains 87 pages of explanatory note, 43 illustrations, 12 tables, 11 sources used, 8 appendices.

DRIVER, PASSENGER, ORDER, ANNOUNCEMENT, CLIENT-SERVER, JAVASCRIPT, NODE.JS, POSTGRESQL, VISUAL STUDIO CODE IDE WEB, REACT

The explanatory note consists of an introduction, six sections and a conclusion.

The purpose of the project is to develop an application to search for fellow travelers.

In the introduction provides the practical relevance and purpose of the diploma project, the relevance of the chosen topic at the moment, as well as the tasks that need to be performed to achieve the goal.

The introduction provides general information about the current state in the field of the problem in question.

The first section presents general theoretical information about the topic, an overview of existing analogues of the developed system and the problem statement.

In the second section, the design of the web application is made.

The third section shows the development process of the web application.

The fourth section shows the process of testing the developed web application.

The fifth section presents the methodology of using the developed web application.

The sixth section presents the cost and selling price results of the developed web application.

The conclusion presents the results of the thesis project and the tasks that were solved during the development of the companion search application.

The graphical part of the application includes:

* a structural diagram of the database;
* the diagram of usage options;
* the block diagram of the ad creation algorithm;
* a flowchart of the algorithm for creating an order;
* a flowchart of the algorithm for adding feedback;
* screenshot of the web application.

Содержание

*ФИО*

*Подпись*

*Дата*

Лист

1

6

*БГТУ 00.00.ПЗ*

Разраб.

*Сивак М.Н.*

Провер.

.

*Сухорукова И.Г.*

Консульт.

*Сухорукова И.Г.*

Н. контр.

*Макарчук М.В*

Утв.

*Пацей Н.В.*

*я*

Содержание

Лит.

Листов

2

74417021, 2022

[Введение 7](#_Toc105460819)

[1 Аналитический обзор литературы и постановка задачи 8](#_Toc105460820)

[1.1 Основные понятия 8](#_Toc105460821)

[1.2 Обзор существующих прототипов 8](#_Toc105460822)

[1.3 Постановка задачи 14](#_Toc105460823)

[1.4 Выводы по разделу 14](#_Toc105460824)

[2 Проектирование веб-приложения 15](#_Toc105460825)

[2.1 Используемые средства разработки 15](#_Toc105460826)

[2.3 Выбор средств реализации программного обеспечения 16](#_Toc105460827)

[2.4 Проектирование базы данных 18](#_Toc105460828)

[2.5 Иерархическая система ролей 21](#_Toc105460829)

[2.5.1 Основные возможности роли «гость» 21](#_Toc105460830)

[2.5.2 Основные возможности роли «пользователь» 22](#_Toc105460831)

[2.5.3 Основные возможности роли «администратор» 22](#_Toc105460832)

[2.6 Проектирование алгоритмов веб-приложения 22](#_Toc105460833)

[2.6.1 Алгоритм создания объявления 23](#_Toc105460834)

[2.6.2 Алгоритм создания заказа 24](#_Toc105460835)

[2.6.3 Алгоритм добавления отзыва 26](#_Toc105460836)

[2.7 Выводы по разделу 26](#_Toc105460837)

[3 Разработка веб-приложения 27](#_Toc105460838)

[3.1 Разработка серверной части 27](#_Toc105460839)

[3.2 Разработка клиентской части 33](#_Toc105460840)

[3.3 Развертывание веб-приложения 35](#_Toc105460841)

[3.4 Выводы по разделу 36](#_Toc105460842)

[4 Тестирование веб-приложения 37](#_Toc105460843)

[4.1 Тестирование основного функционала веб-приложения 37](#_Toc105460844)

[4.2 Тестирование безопасности базы данных и приложения 40](#_Toc105460845)

[4.3 Выводы по разделу 45](#_Toc105460846)

[5 Методика установки и использования 46](#_Toc105460847)

[5.1 Развертывание системы 46](#_Toc105460848)

[5.2 Руководство пользователя 46](#_Toc105460849)

[5.2.1 Описание процессов авторизации и регистрации 46](#_Toc105460850)

[5.2.2 Описание процесса бронирования поездки 48](#_Toc105460851)

[5.2.3 Описание процесса создания объявления 49](#_Toc105460852)

[5.3 Вывод по разделу 52](#_Toc105460853)

[6 Технико-экономическое обоснование проекта 53](#_Toc105460854)

[6.1 Общая характеристика разрабатываемого программного средства 53](#_Toc105460855)

[6.2 Исходные данные для проведения расчетов 53](#_Toc105460856)

[6.3 Методика обоснования цены 54](#_Toc105460857)

[6.3.1 Определение объема программного средства 55](#_Toc105460858)

[6.3.2 Расчет трудоемкости выполненной работы 57](#_Toc105460859)

[6.3.3 Расчет основной заработной платы 57](#_Toc105460860)

[6.3.4 Расчет дополнительной заработной платы 58](#_Toc105460861)

[6.3.4 Расчет отчислений в фонд социальной защиты населения 58](#_Toc105460862)

[6.3.5 Расчет расходов на материалы 59](#_Toc105460863)

[6.3.6 Расчет расходов на оплату машинного времени 59](#_Toc105460864)

[6.3.7 Расчет прочих прямых затрат 59](#_Toc105460865)

[6.3.8 Расчет общепроизводственных и общехозяйственных расходов 60](#_Toc105460866)

[6.3.9 Расчет суммы расходов на разработку программного средства 60](#_Toc105460867)

[6.3.10 Расходы на сопровождение и адаптацию 60](#_Toc105460868)

[6.3.11 Расчет общей суммы расходов 61](#_Toc105460869)

[6.3.12 Определение цены, оценка эффективности 61](#_Toc105460870)

[6.4 Вывод по разделу 64](#_Toc105460871)

[Заключение 65](#_Toc105460872)

[Список использованных источников 66](#_Toc105460873)

[ПРИЛОЖЕНИЕ А. Структурная схема базы данных 67](#_Toc105460874)

[ПРИЛОЖЕНИЕ Б. Диаграмма вариантов использования 68](#_Toc105460875)

[ПРИЛОЖЕНИЕ В. Блок-схема алгоритмма создания объявления 69](#_Toc105460876)

[ПРИЛОЖЕНИЕ Г. Блок-схема алгоритма создания заказа 70](#_Toc105460877)

[ПРИЛОЖЕНИЕ Д. Диаграмма последовательности заказа 71](#_Toc105460878)

[ПРИЛОЖЕНИЕ Е. Скриншот работы веб-приложения 72](#_Toc105460879)

[ПРИЛОЖЕНИЕ Ж. Листинг контроллера пользователя 73](#_Toc105460880)

[ПРИЛОЖЕНИЕ Д. Листинг *React*-компонента *NewAd* 78](#_Toc105460881)

# **ВВЕДЕНИЕ**

*ФИО*

*Подпись*

*Дата*

Лист

1

8

*БГТУ 00.00.ПЗ*

Разраб.

*Сивак М.Н.*

Провер.

.

*Сухорукова И.Г.*

Консульт.

*Сухорукова И.Г.*

Н. контр.

*Макарчук М.В*

Утв.

*Пацей Н.В.*

*я*

Введение

Лит.

Листов

1

74417021, 2022

В наше время информационно-коммуникационная сеть и всемирная система объединенных компьютерных сетей играет очень важную роль в жизни почти каждого человека. Интернет используют не только в качестве развлечений, но и в качестве поставщика полезных сервисов и приложений, которые значительно облегчают жизнь и экономят время людей. В сети существует огромное количество различных веб-сайтов и веб-приложений, отвечающих за разные услуги: доставка одежды, цифровой и прочей техники, доставка продуктов питания, медикаментов, заказ талонов в поликлинике, оформление заказов в кафе, ресторанах и в прочих развлекательных заведениях, создание объявления по продаже автомобилей или других вещей, подача заявок на вакансии, а также оформление поездок как в рамках одного города, так и между городами и даже странами.

С помощью веб-приложений можно привлекать новых клиентов и поддерживать существующих, а если оно выполнено по всем современным стандартам, то его использование будет привлекать также аудиторию пользователей с мобильных устройств.

Актуальность темы: что касается актуальности, то определенная группа людей, работающая далеко от дома, где отсутствует хорошая коммуникация транпортных средств, нуждается в поиске более удобного и экономного способа передвижения до места их работы. Это приложение охватывает более широкую целевую аудиторию. Им можно воспользоваться также для поездки в любое место города, как и при заказе такси. Приложение направлено на аудиторию с низким и выше финансовым достатком.

Цель дипломного проекта: разработка веб-приложения для предоставления водителям и пассажирам возможности создания объявлений и оформления заказов на поездки.

Для достижения цели сформулированы следующие задачи:

* выбор и обоснование средств разработки;
* проектирование базы данных;
* разработка структуры серверной части;
* разработка функций серверной части;
* проектирование веб-интерфейса сайта;
* разработка пользовательского интерфейса сайта;
* тестирование серверной части и пользовательского интерфейса.

Практическая значимость: позволяет любому пользователю сети интернет создавать объявления как в качестве водителя, так и в качестве пассажира, а также осуществлять заказы на поездки.

# **1 Аналитический обзор литературы и постановка задачи**

*ФИО*

*Подпись*

*Дата*

Лист

1

9

*БГТУ 01.00.ПЗ*

Разраб.

*Сивак М.Н.*

Провер.

.

*Сухорукова И.Г.*

Консульт.

*Сухорукова И.Г.*

Н. контр.

*Макарчук М.В*

Утв.

*Пацей Н.В.*

*я*

1 Аналитический обзор литературы и постановка задачи

Лит.

Листов

7

74417021, 2022

Онлайн платформы для поиска попутчиков и бронирования поездок стали быстро набирать популярность в последние несколько лет ввиду увеличения количества заказов и необходимиости оптимизировать и ускорить этот процесс для снижения нагрузки на операторов центров звонков огранизаций, предоставляющих услуги поездок. Раньше оформить поездку в другой город можно было только осуществив звонок в организацию, но теперь есть возможность сделать это проще и быстрее, используя веб-сайт или мобильное приложение, предоставленное от данной организации. Вместе с тем появилась возможность оставлять отзывы и люди, которые желают знать качество работы водителей, теперь легко могут посмотреть отзывы о водителях прямо перед заказом.

## **1.1 Основные понятия**

Онлайн-бронирование – это процесс осуществления бронирования через Интернет, в интерактивном режиме, то есть используя при этом веб-сайт или веб-приложение, предназначенное для бронирования целевой услуги.

Веб-приложение – это клиент-серверное приложение, в котором клиент взаимодействует с сервером при помощи специального приложения – браузера. Логика веб-приложения разделена между клиентом и сервером, хранение данных происходит, преимущественно, на сервере, а обмен информацией между клиентом и сервером осуществляется посредством сети [1].

## **1.2 Обзор существующих прототипов**

На текущий момент существует уже некоторое количество сервисов для обеспечения бронирования поездок или поиска попутчиков, которые постоянно развиваются и стремяться удовлетворить все желания клиентов. Но тем не менее немногие из такого рода веб-приложений являются удобными для использования, как для водителей, так и для пассажиров.

Из открытых источников были найдены следующие веб-приложения для осуществления поездок:

* *blablacar.com*;
* *smilebus.by*;
* *dovezu.ru*.

*Blablacar.com* – ведущий мировой туристический сервис с активным сообществом путешественников [2].

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.

Благодаря удобному поиску, можно легко найти поездку по вашему маршруту, а также выбрать способ поездки: с попутчиками или на автобусе.

Данное веб-приложение является одним из самых удобных в сети Интернет. Оно имеет большое количество пользователей по всему миру. Так же пользователи могут оставлять объявления в качестве водителя и ожидать заявки от попутчиков. Главная страница представлена на рисунке 1.1.

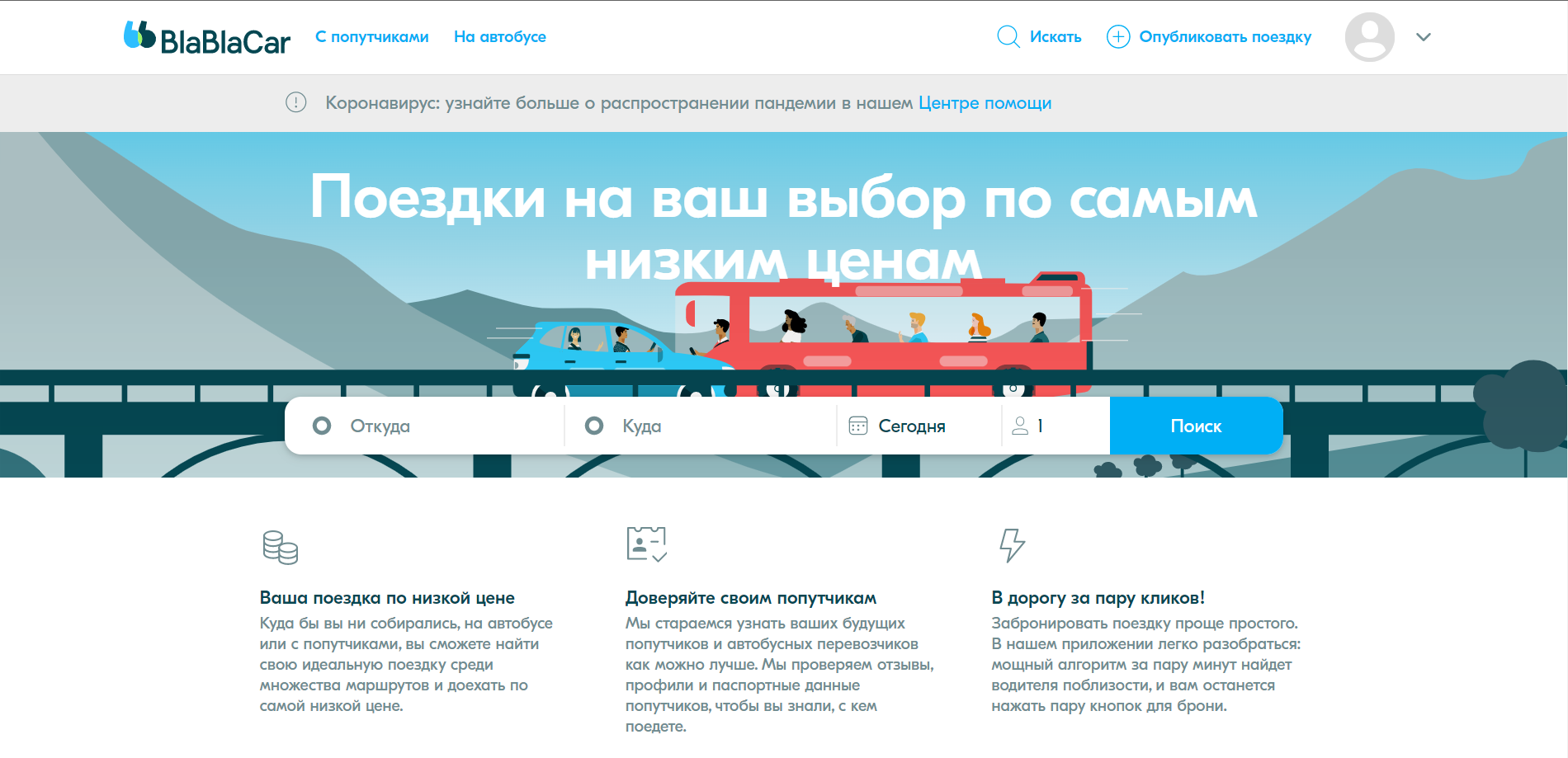


Рисунок 1.1 – Главная страница *Blablacar.com*

После заполнения всех полей и нажатии на кнопку «Поиск», откроется страница со всеми объявлениями, которые удовлетворяют результатам поиска. Страница с результатами поиска представлена на рисунке 1.2.

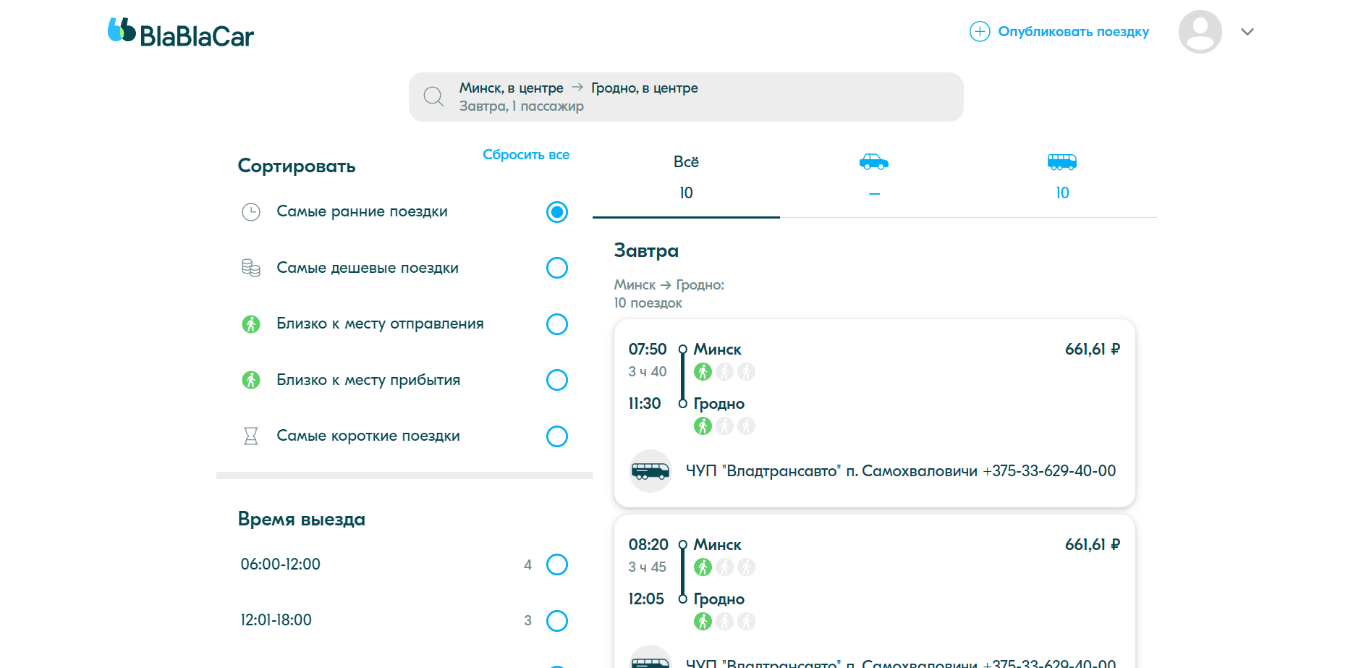


Рисунок 1.2 – Поиск объявлений на *Blablacar.com*

На данной странице можно также провести сортировку по различным фильтрам, расположенным на панели слева от списка объявлений, либо нажать на поле поиска и поменять какие-либо параметры. Здесь можно также перейти на другие вкладки для просмотра объявлений водителей частных автомобилей или автобусов. При нажатии на объявление, пользователь попадет на страницу с подробным описанием поездки. Страница с подробным описанием поездки представлена на рисунке 1.3.

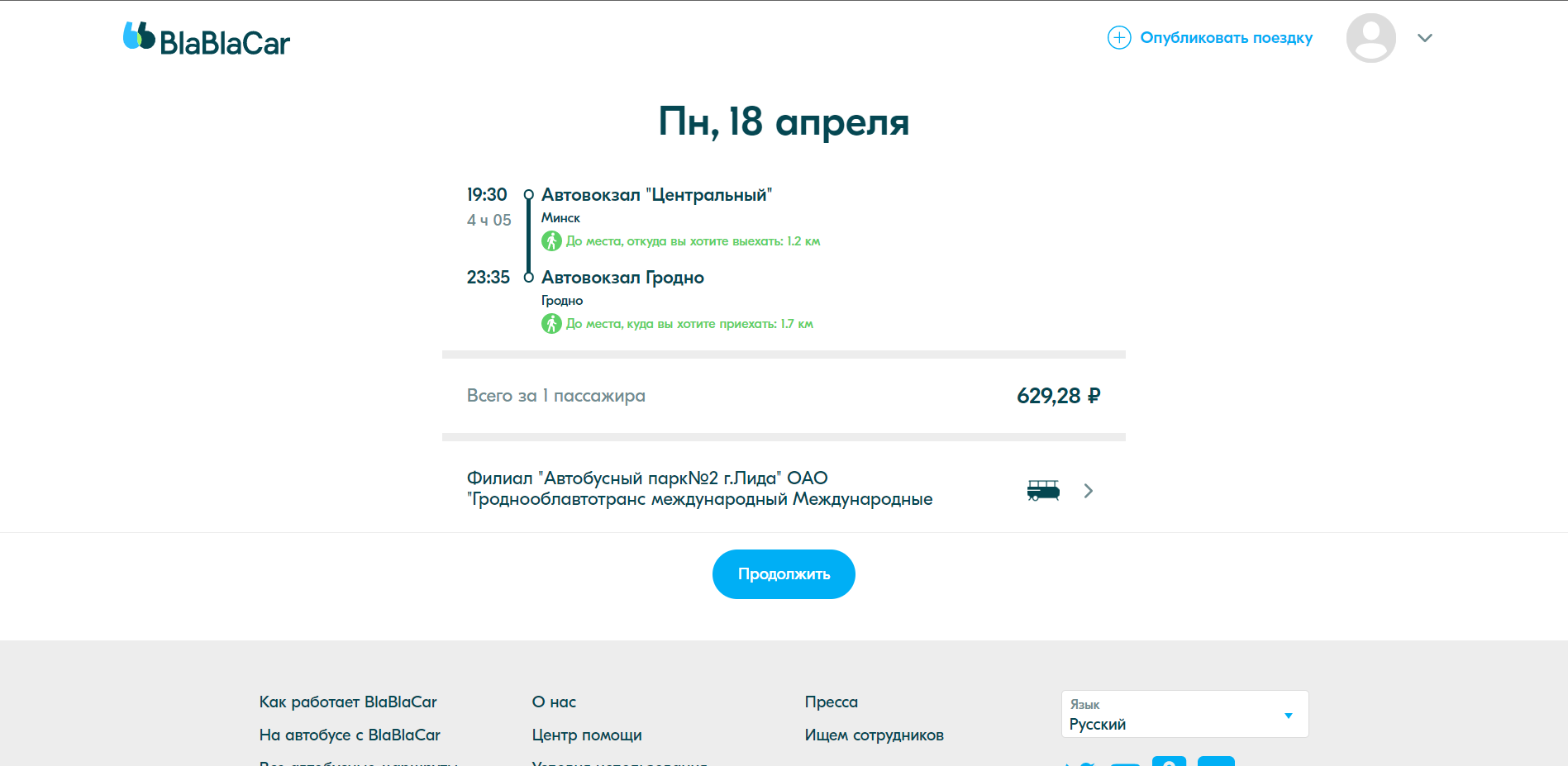


Рисунок 1.3 – Подробное описание поездки на *Blablacar.com*

При переходе на следующую страницу предлагается выбрать желаемое место в транспортном средстве, как показано на рисунке 1.4.

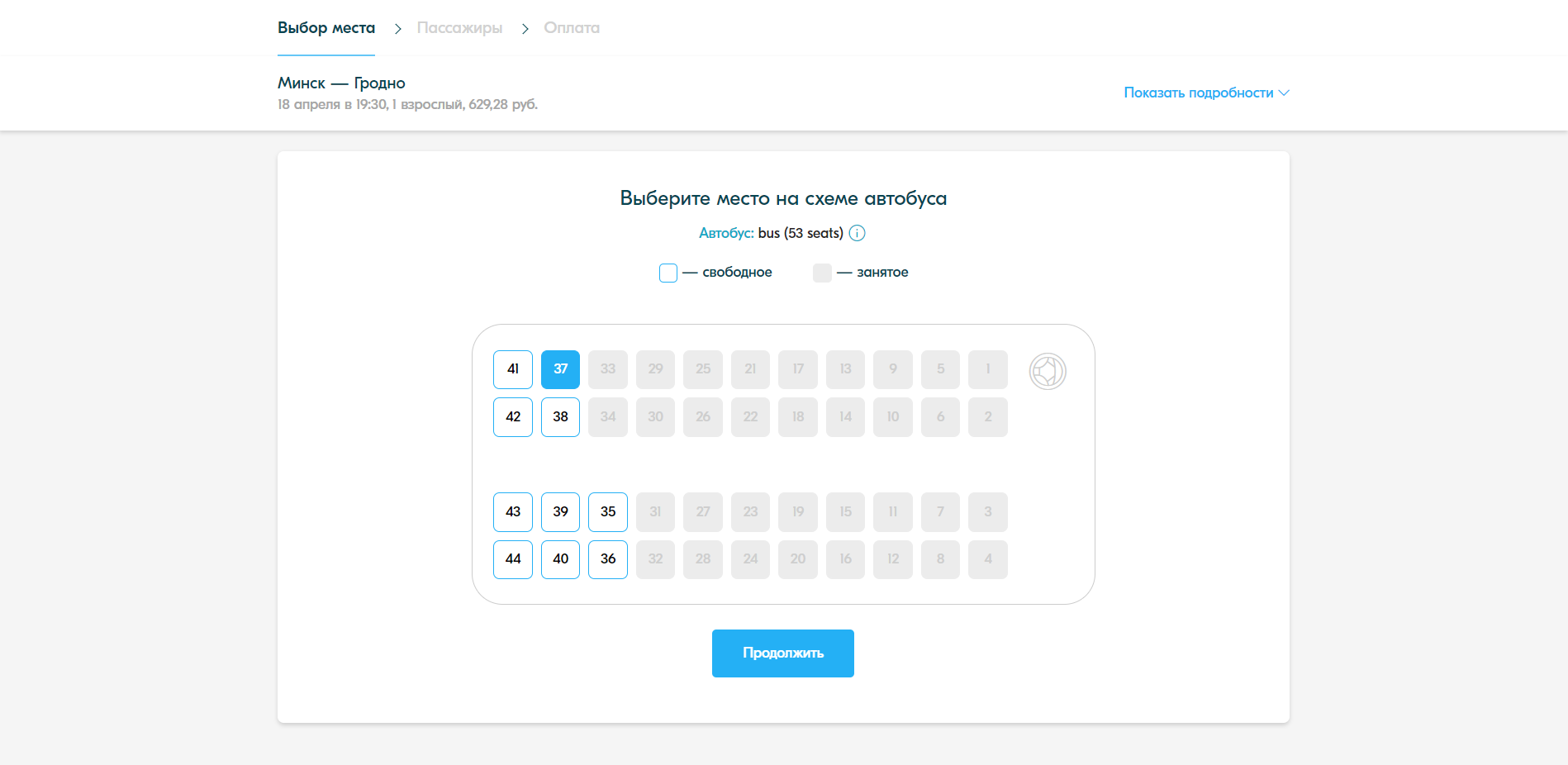


Рисунок 1.4 – Выбор места в транспорте на *Blablacar.com*

После данного шага пользователь окажется на странице, где ему следует заполнить поля с именем, фамилией, почтой и номером телефона, а также оплатить поездку картой. Так же пользователь может ознакомиться с политикой возврата денежных средств в случае отмены поездки.

Функционал приложения включает в себя:

* фильтр оъявлений;
* выбор конкретных маршрутов;
* выбор количества мест;
* публикация собственных объявлений;
* оплата картой онлайн.

Однако данный сервис для поездок не лишен недостатков. В результате исследования данного сайта, были найдены следующие недостатки *Blablacar.com*:

* довольно долгий процесс оформления заказа;
* невозможность посмотреть отзывы;
* при переходе к оформлению заказа, все дальнейшие шаги выполняются в новой вкладке, что не совсем удобно.

Сдедующее приложение *Smilebus.by* – это ведущий сервис по заказу маршруток между городами в рамках Беларуси [3]. На главной странице сайта можно вести поиск объявлений, что очень удобно для быстрого доступа к списку объявлений. Так же немного ниже есть карточки-ссылки с самыми популярными маршрутами и блок с отзывами о водителях маршруток, где, перейдя по ссылке, можно открыть саписок всех отзывов пассажиров за длительный период времени. Главная страница представлена на рисунке 1.5.

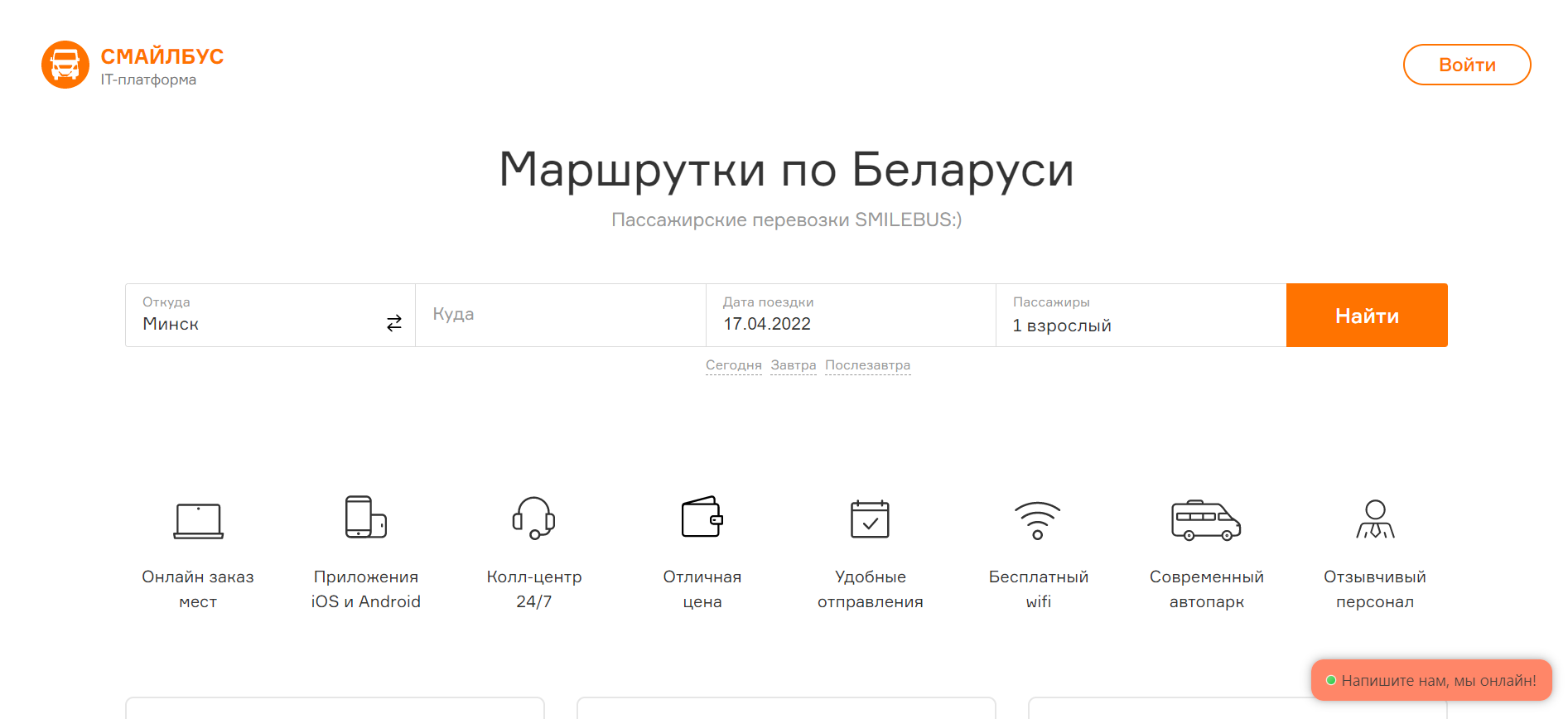


Рисунок 1.5 – Главная страница *Smilebus.by*

После заполнения всех полей поиска и нажатии на кнопку «Поиск» на этой же странице будет отображен список всех объявлений о поездках между выбранными городами, временем и датой отправления, а также отобразяться дополнительные поля с выбором остановок посадки и высадки. Так же существует возможность выбора мест посадки и высадки на карте, что дает возможность посмотреть остановки на карте. В поле поиска можно указать количество мест как для взрослого пассажира, так и для детей. Цена поездки для детей в 2 раза ниже, чем для взрослого. Так же можно указать информацию о дополнительном багаже. Страница с результатами поиска объявлений представлена на рисунке 1.6.

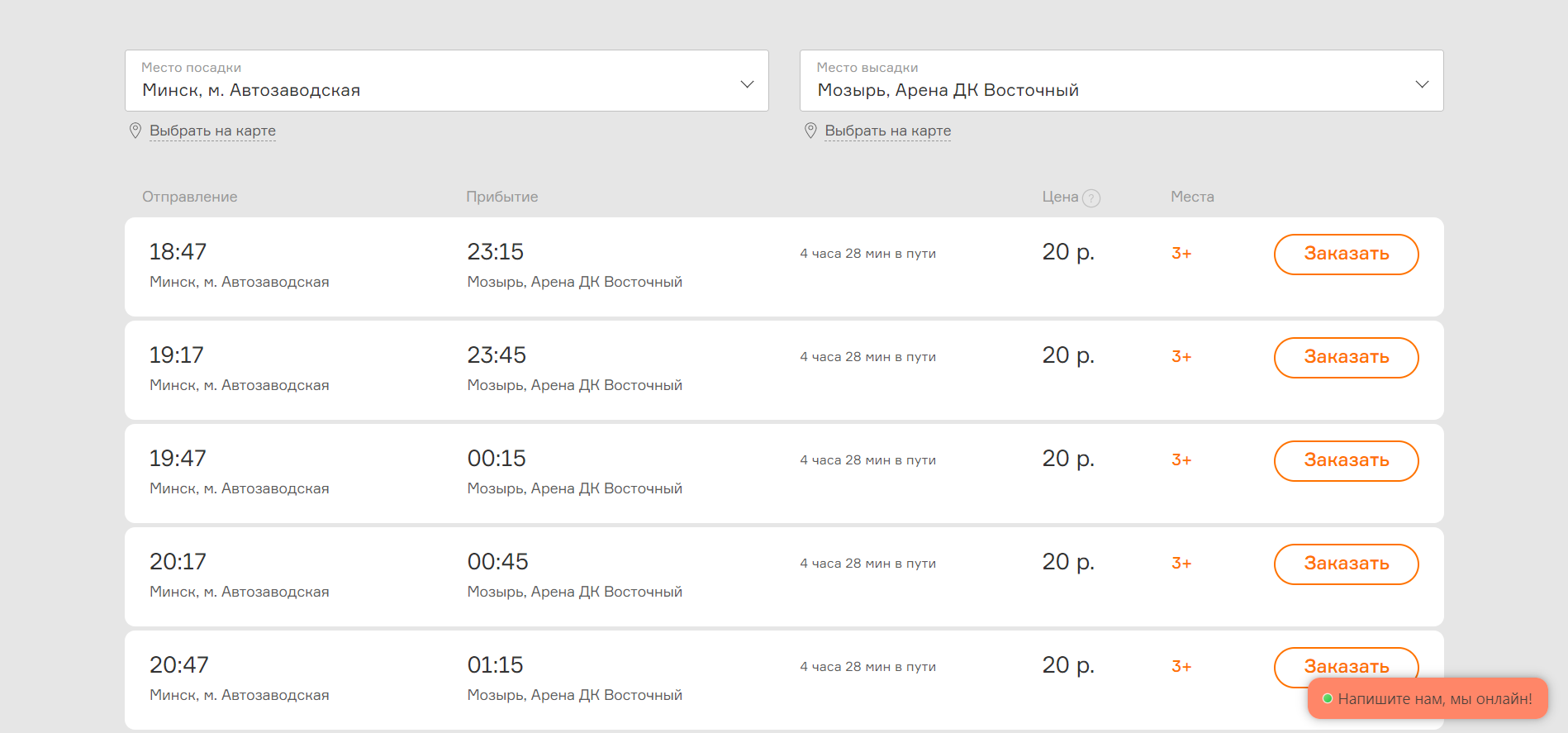


Рисунок 1.6 – Результаты поиска объявлений на *Smilebus.by*

После нажатия на кнопку «Заказать», пользователь будет переведен на другую страницу с более подробной информацией о поездке, на которой так же будет возможность подкорректировать некоторые детали заказа. Данный шаг представлен на рисунке 1.7.

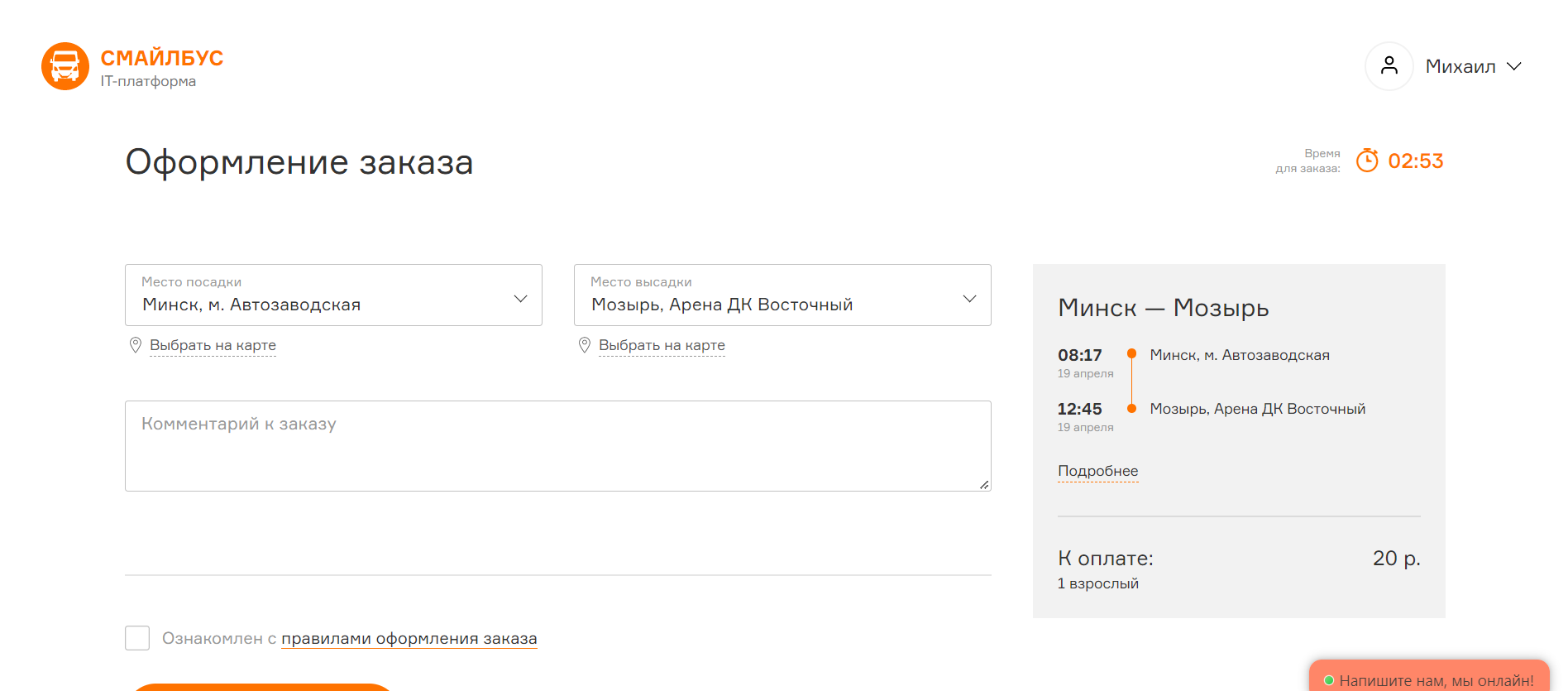


Рисунок 1.7 – Оформление заказа на *Smilebus.by*

После ознакомления со всеми деталями заказа можно приступить к финальному шагу, нажав на кнопку «Оформить заказ». После чего на экране отобразиться сообщение, что заказ был оформлен и на номер телефона пользователя придет сообщение с информацией о заказе.

Функционально приложение включает в себя:

* фильтр оъявлений;
* выбор маршрутов между городами;
* выбор количества заказываемых мест;
* оставление комментариев к заказу;
* возможность оставлять отзывы о водителях после поездки.

В данном приложении минусов обнаружено не было.

*Dovezu.ru* – сервис для поисков попутчиков по городу. Здесь можно пройти регистрацию и создавать свои объявления в качестве водителя, а также можно оформлять заказы на поездку в рамках города по определенному адресу [4]. Главная страница сайта представлена на рисунке 1.8.

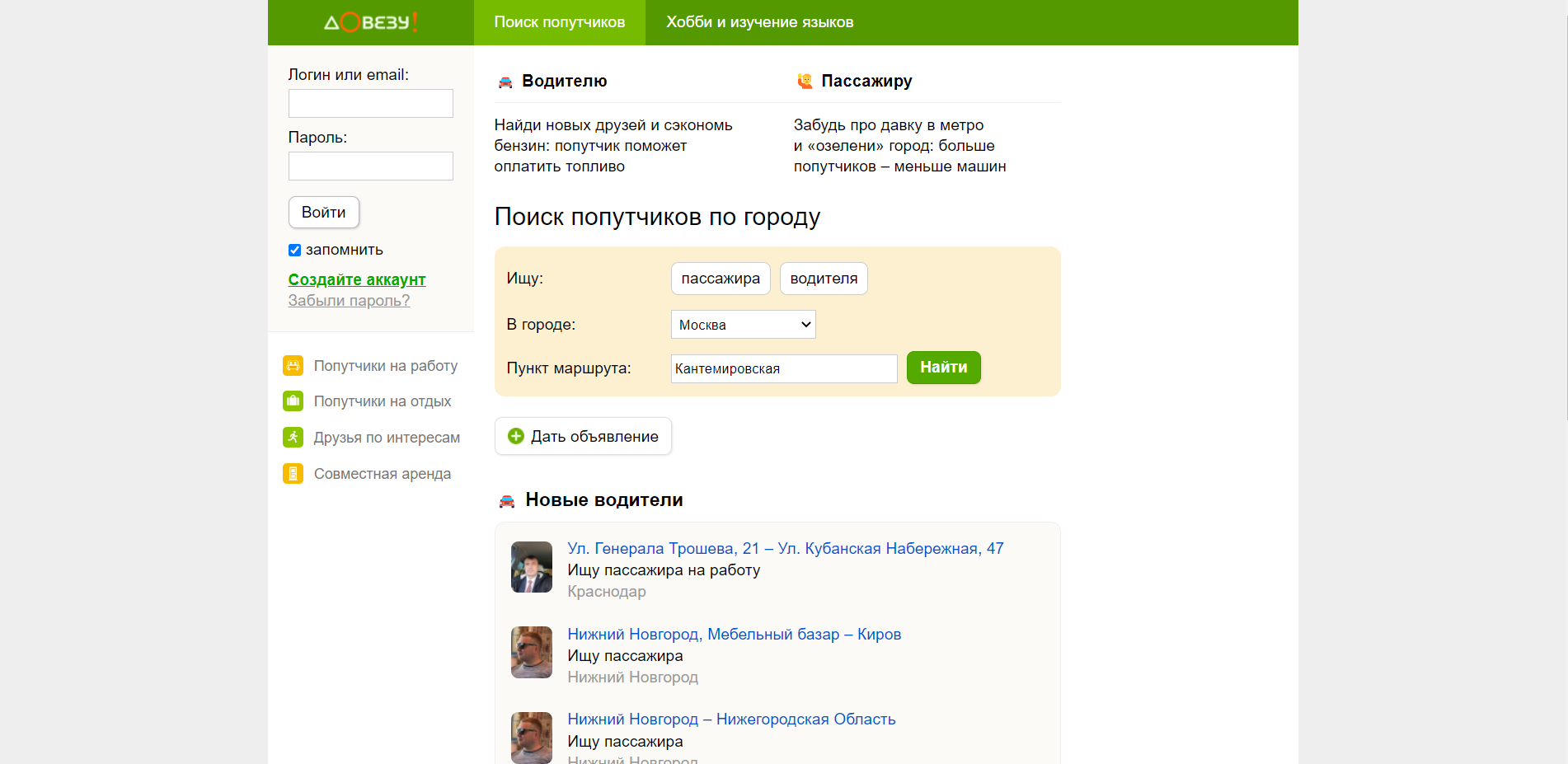


Рисунок 1.8 – Главная страница *Dovezu.ru*

После создания профиля можно осуществлять заказы на поездки, а также вести диалог с водителями и пассажирами.

Функциональные возможности приложения:

* фильтр объявлений;
* выбор маршрутов в рамках выбранного города;
* создание объявлений в качестве водителя;
* создание объявлений в качестве пассажира;
* поиск водителей;
* поиск пассажиров;
* оформление заказов.

Данное приложение имеет ряд существенных минусов:

* не внушает абсолютно никакого доверия;
* нет возможности посмотреть отзывы о пользователях;
* для передачи информации между клиентом и сервером используется протокол *HTTP* [5];
* неудобный интерфейс для современных пользователей;
* устаревший дизайн.

Таким образом, рассмотренные недостатки и достоинства аналогов должны быть учтены при разработке программного средства.

## **1.3 Постановка задачи**

Разрабатываемое программное средство целесообразно организовать как вебприложение, которое подразумевается использовать в сети интернет. Проектирование программного средства будет включать в себя:

* проектирование структуры базы данных;
* проектирование классов моделей, контроллеров и представлений, сервисов и репозиториев, интерфейса.

Для достижения выше обозначенных целей можно выделить основные задачи, которые необходимо реализовать в рамках дипломного проектирования:

* реализовать механизм регистрации и авторизации пользователей;
* реализовать механизм создания объявлений в качестве водителей и пассажиров;
* реализовать вывод списка объявлений пассажиров и водителей;
* разработать функционал бронирования поездок;
* разработать функционал для редактирования информации профиля пользователя;
* разработать функционал для возможности составления отзывов о водителях;
* реализовать поиск объявлений по адресу отправления, прибытия, дате и времени;
* разработать функционал для добавления и удаления автомобилей профиля.

## **1.4 Выводы по разделу**

В данном разделе был проведен аналитический обзор существующих онлайн-систем для бронирования поездок и поиска попутчиков. Выявлены их достоинства и недостатки. Основываясь на рассмотренных онлайн-системах, было принято решение создания веб-приложения, которое учтет все достоинства рассмотренных приложений и устранить выявленные недостатки. Было принято решение огранизовать программное средство как веб-приложение, а также были выделены основные задачи, которые необходимо реализовать в рамках дипломного проектирования.

# **2 Проектирование веб-приложения**

*ФИО*

*Подпись*

*Дата*

Лист

1

16

*БГТУ 02.00.ПЗ*

Разраб.

*Сивак М.Н.*

Провер.

.

*Сухорукова И.Г.*

Консульт.

*Сухорукова И.Г.*

Н. контр.

*Макарчук М.В*

Утв.

*Пацей Н.В.*

*я*

2 Проектирование  
веб-приложения

Лит.

Листов

12

74417021, 2022

## **2.1 Используемые средства разработки**

Для разработки программного средства был выбран редактор исходного кода *Microsoft Visual Studio Code*, функционирование которого гарантирует необходимый уровень качества используемого кода на протяжении всего циклического процесса разработок программного обеспечения – от проектирования до непосредственной разработки.

*Microsoft Visual Studio Code* – редактор исходного кода, позволяющий разрабатывать кроссплатформенные веб-приложения на различных языках программирования, разработанный компанией *Microsoft* [6].

**2.2 Архитектура веб-приложения**

Архитектура проекта – один из важнейших этапов создания приложения. Правильно выбранная и реализованная архитектура даст возможность быстро разрабатывать, модифицировать и расширять функционал проекта.

Архитектура веб-приложения представляет собой классический Клиент – Сервер, схема которого представлена на рисунке 2.1.

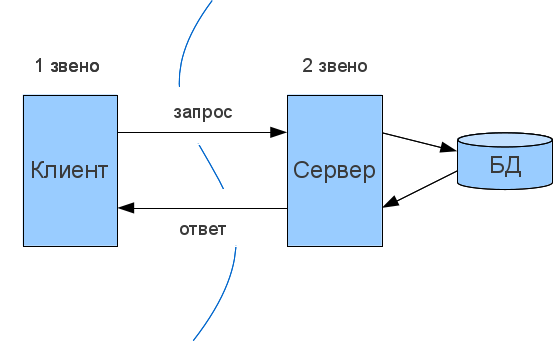
****

Рисунок 2.1 – Клиент-сервер

Клиент предоставлен браузерным приложением, которое посылает запросы (*request*) по сети с использованием *HTTP* протокола, а сервер должен, принимать запросы от клиента и давать ответ (response). В нашем случае приложение разделено на две части. Это *frond-end* (клиентская) и *back-end* (серверная) части.

При разработке приложения использовался *MVC* шаблон. Модель предоставляет остальным компонентам приложения объектно-ориентированное отображение данных. Объекты модели могут осуществлять загрузку и сохранение данных в реляционной базе данных, а также реализуют бизнес-логику.

Контроллер является обработчиком запроса и чаще всего является связующим звеном между представлением и моделью. Его главная задача – достать данные из модели и передать его представлению.

Представление же в свою очередь должно отрисовывать страницы на основе данных, полученных с контроллера. Выбор представления для отрисовки – задача контроллера.

На рисунке 2.2 показано взаимодействие между слоями в *MVC*. Шаблон *Model-View-Controller* для веб-приложений обеспечивает интеграцию с вебсервером и сервером баз данных.

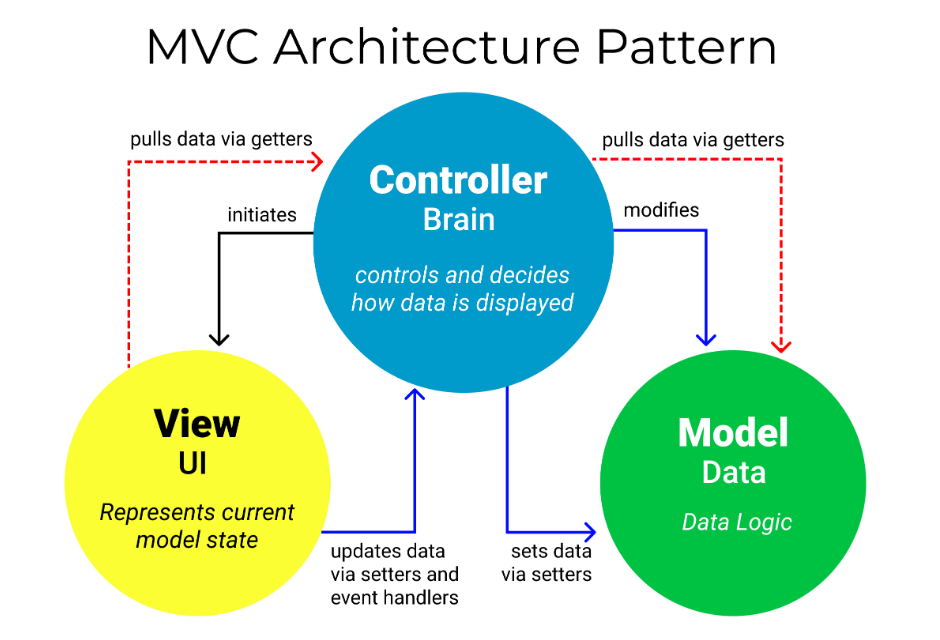


Рисунок 2.2 – Взаимодействия между слоями в *MVC* архитектуре

Объекты модели могут осуществлять загрузку и сохранение данных в реляционной базе данных, а также реализуют бизнес-логику.

## **2.3 Выбор средств реализации программного обеспечения**

В качестве языка программирования был выбран язык *Javascript*, который отличается своей простотой, гибкостью в использовании. Вместо того чтобы писать большое количество строк кода, можно просто использовать готовую конструкцию, а компилятор сделает все остальное. Но некоторые такие конструкции являются не самыми оптимальными с точки зрения производительности. Все эти недостатки перекрываются за счет удобочитаемости кода и высокой скорости разработки [7].

Серверная часть написана на платформе *Node.js*, язык – *Javascipt* c использованием фреймворка *ExpressJS* и *ORM Sequelize*.

*Node.js* – это кроссплатформенная среда исполнения Javascript с открытым исходным кодом, которая выполняет код *Javascript* вне веб-браузера [8]. *Node.js* позволяет разработчикам использовать *Javascript* для написания инструментов командной строки и для сценариев на стороне сервера – запуск сценариев на стороне сервера для создания динамического содержимого веб-страницы перед ее отправкой в веб-браузер пользователя. Следовательно, *Node.js* представляет собой парадигму «*Javascript* везде», объединяющую разработку веб-приложений вокруг единого языка программирования, а не разных языков для серверных и клиентских сценариев.

*Express.js*, или просто *Express*, – это фреймворк веб-приложений для *Node.js*, выпускаемый как бесплатное программное обеспечение с открытым исходным кодом под лицензией *MIT* [9]. Он предназначен для создания веб-приложений и *API*. Его называют де-факто стандартным серверным фреймворком для *Node.js*.

*Sequelize* – это *ORM*-инструмент *Node.js* на основе promise для *Postgres, MySQL, MariaDB, SQLite, Microsoft SQL Server, Amazon Redshift* и облака данных *Snowflake* [10].

Для разработки клиентской части был также выбран язык *Javascript* и библиотека *React.js*. *React.js* представляет собой *JavaScript*-библиотеку с открытым исходным кодом для разработки пользовательских интерфейсов. Прежде всего он нацелен на разработку одностраничных приложений. *React* разрабатывается и поддерживается *Facebook, Instagram* и сообществом отдельных разработчиков и корпораций. Его цель – предоставить высокую скорость, простоту и масштабируемость [11].

*React* может использоваться для разработки одностраничных и мобильных приложений. В качестве библиотеки для разработки пользовательских интерфейсов *React* часто используется с другими библиотеками, такими как *MobX, Redux* и *GraphQL*.

Можно выделить следующие особенности библиотеки *React.js*:

* однонаправленная передача данных;
* виртуальный *DOM*;
* использование *JSX (JavaScript XML)*;
* наличие методов жизненного цикла;
* не только отрисовка *HTML* в браузере.

Однонаправленная передача данных подразумевает собой передачу свойств от родительских компонентов к дочерним. Компоненты получают свойства как множество неизменяемых значений, поэтому компонент не может напрямую изменять свойства, но может вызывать изменения через *callback*-функции. Такой механизм называют «свойства вниз, события наверх».

Виртуальный *DOM* – это кэш-структура в памяти, которая позволяет вычислять разницу между предыдущим и текущим состояниями интерфейса для оптимального обновления *DOM* браузера. Таким образом, программист может работать со страницей, считая, что она обновляется вся, но библиотека самостоятельно решает, какие компоненты страницы необходимо обновить.

*JavaScript XML (JSX)* – это расширение синтаксиса *JavaScript*, которое позволяет использовать *HTML*-подобный синтаксис для описания структуры интерфейса. Как правило, компоненты написаны с использованием *JSX*, но также есть возможность использования обычного *JavaScript*.

Компоненты *React* реализуют ри, который принимает входные данные и возвращает данные для отображения. *React* использует *XML*-подобный синтаксис *JSX*. Входные данные, которые передаются в компонент, могут быть доступны в render через *this.props*.

Помимо ввода входных данных (доступных через свойство класса *this.props*), компонент может поддерживать и изменять внутреннее состояние (доступное через свойство класса *this.state*).

Методы жизненного цикла позволяют разработчику запускать код на разных стадиях жизненного цикла компонента. Например:

* *shouldComponentUpdate* – позволяет предотвратить перерисовку компонента с помощью возврата false, если перерисовка не нужна;
* *componentDidMount* – вызывается после первой отрисовки компонента. Часто используется для запуска получения данных с удаленного источника через *API*;
* *render* – важнейший метод жизненного цикла. Каждый компонент должен иметь этот метод. Обычно вызывается при изменении данных компонента для перерисовки данных в интерфейсе.

## **2.4 Проектирование базы данных**

База данных – совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ.

Система управления базами данных – совокупность программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия ее с прикладными программами.

Для проектирования и разработки базы данных, необходимой для корректной работы разрабатываемой онлайн-системы, использовалась реляционная база данных *PostgreSQL*.

*PostgreSQL* – объектно-реляционная СУБД. Это дает ему некоторые преимущества над другими *SQL* базами данных с открытым исходным кодом, такими как *MySQL*. Фундаментальная характеристика объектно-реляционной базы данны – это поддержка пользовательских объектов и их поведения, включая типы данных, функции, операции, домены и индексы. Это делает *Postgres* невероятно гибким и надежным. Среди прочего он умеет создавать, хранить и извлекать сложные структуры данных [12].

*PostgreSQL* создана на основе некоммерческой СУБД *Postgres*, разработанной как проект с открытым исходным кодом в Калифорнийском университете в Беркли в 1986 году.

Сильными сторонами *PostgreSQL* считаются:

* высокопроизводительные механизмы транзакций и репликации;
* расширяемая система встроенных языков программирования: в стандартной поставке поддерживаются *PL/pgSQL, PL/Perl, PL/Python* и *PL/Tcl*;
* можно использовать *PL/Java, PL/PHP, PL/Py, PL/R, PL/Ruby, PL/Scheme*;
* имеется поддержка загрузки модулей расширения на языке *C*;
* наследование;
* возможность индексирования геометрических объектов и наличие базирующегося на ней расширения *PostGIS*;
* встроенная поддержка слабоструктурированных данных в формате *JSON* с возможностью их индексации;
* возможность создавать новые типы данных;
* возможность создавать типы индексов, языки программирования, модули расширения, подключать любые внешние источники данных.

В результате работы была разработана база данных, содержащая 7 таблиц. Далее будет приведено описание каждой таблицы.

Таблица *Users* хранит в себе информацию о пользователях системы, в частности – почтовый адрес, пароль, имя, пол, номер телефона и т.д. Более подробное описание полей, их тип приведено в таблице 2.1.

Таблица 2.1 – Структура таблицы «*Users*»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| *id* | *int* | Идетнификатор таблицы |
| *roleId* | *int* | Идентификатор роли |
| *userName* | *string* | Имя пользователя |
| *userPhone* | *string* | Телефон пользователя |
| *userEmail* | *string* | Email адрес пользователя |
| *userPassword* | *string* | Пароль пользователя |
| *gender* | *string* | Пол пользователя |
| *userPhoto* | *string* | Ссылка на фото профиля |
| *userStatus* | *int* | Статус пользователя |
| *rate* | *double* | Оценка пользователя |

Таблица *Roles* хранит в себе роли пользователя. Описание полей приводится в таблице 2.2.

Таблица 2.2 – Структура таблицы «*Roles*»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| *id* | *int* | Идентификатор таблицы |
| *roleName* | *string* | Имя роли |

Таблица *Cars* хранит в себе информацию об автомобилях пользователей. Подробное описание полей можно увидеть в таблице 2.3.

Таблица 2.3 – Структура таблицы «*Cars*»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| *id* | *int* | Идентификатор таблицы |
| *carBrand* | *string* | Марка автомобиля |
| *carModel* | *string* | Модель автомобиля |
| *carNumber* | *string* | Номер автомобиля |
| *carPhotoLink* | *string* | Ссылка на фото автомобиля |
| *userId* | *int* | Идентификатор пользователя-владельца |

Таблица *Ads* содержит информацию объявлений, созданных пользователями. В таблице 2.4 перечислены все поля с подробным описанием каждого.

Таблица 2.4 – Структура таблицы «*Ads*»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| *id* | *int* | Идентификатор таблицы |
| *userId* | *int* | Идентификатор пользователя |
| *role* | *string* | Роль пользователя относительно текущей поездки (водитель/пассажир) |
| *carId* | *int* | Идентификатор автомобиля |
| *startAddress* | *int* | Адрес отправления |
| *finishAddress* | *int* | Адрес прибытия |
| *startDate* | *Date* | Дата отправления |
| *startTime* | *Datetime* | Время отправления |
| *finishTime* | *Datetime* | Время прибытия |
| *seatsCount* | *int* | Количество мест |

Таблица *Addresses* содержит информацию об адресах. Более подробное описание приведено в таблице 2.5.

Таблица 2.5 – Структура таблицы «*Addresses*»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| *id* | *int* | Идентификатор таблицы |
| *country* | *string* | Страна |
| *city* | *string* | Город |
| *street* | *string* | Улица |
| *streetNum* | *string* | Номер помещения |

Таблица *Orders* содержит информацию о заказах пользователей. Описание полей можно видеть в таблице 2.6.

Таблица 2.6 – Структура таблицы «*Orders*»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| *id* | *integer* | Идентификатор таблицы |
| *driverId* | *integer* | Идентификатор водителя |
| *passengerId* | *integer* | Идентификатор пассажира |
| *adId* | *integer* | Идентификатор объявления |
| *seatsCount* | *integer* | Количество забронированных мест |

Таблица *Reviews* хранит в себе данные отзывов о пользователях системы как о водителях. Подробное описание полей представлено в таблице 2.7.

Таблица 2.7 – Структура таблицы «*Reviews*»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| *id* | *integer* | Идентификатор таблицы |
| *driverId* | *integer* | Идентификатор водителя |
| *passengerId* | *integer* | Идентификатор пассажира |
| *rate* | *integer* | Оценка водителя |
| *comment* | *string* | Комментарий к поездке |

Для отображения данных с БД в объектное представление была использована *Sequelize ORM*. Структурную схему базы данных можно найти в приложении А.

## **2.5 Иерархическая система ролей**

Реализованное приложение имеет три типа пользователей (администратор, пользователь, гость). Функционал у данных типов пользователей имеет иерархическую структуру – это значит, что роль «пользователь» имеет функционал гостя и функционал, предназначенный для роли «пользователь», а роль «администратор» – функционал роли «пользователь», функционал роли «гость» и «администратор».

Графическое описание функциональных возможностей по ролям в виде диаграммы вариантов использования представлено в приложении Б.

### **2.5.1 Основные возможности роли «гость»**

Роль «гость» предназначена для неавторизованных пользователей. Данный тип пользователей может использовать следующие возможности системы:

* регистрация;
* авторизация;
* просмотр списка объявлений;
* поиск объявлений.

### **2.5.2 Основные возможности роли «пользователь»**

Роль «пользователь» предназначена для авторизованных пользователей. Данный тип пользователей, помимо всех возможностей системы роли «гость», имеет возможность:

* просматривать комментарии и отзывы о себе;
* смотреть список объявлений и вести их поиск;
* оформлять заказ на поездку;
* создавать объявления как попутчик;
* создавать объявления как водитель;
* оставлять комментарии и отзывы к водителям;
* просматривать список своих заявок;
* просматривать список заявок пассажиров;
* редактировать профиль;
* закрывать заказы;
* удалить свой профиль.

### **2.5.3 Основные возможности роли «администратор»**

Роль «администратор» предназначена для авторизованных пользователей. Данный тип пользователей, помимо всех возможностей системы роли «гость» и роли «пользователь», может выполнять следующие действия:

* просматривать список всех пользователей (водителей/попутчиков);
* блокировать пользователей;
* просматривать список всех комметариев;
* удалять комментарии с нежелательным содержанием.

## **2.6 Проектирование алгоритмов веб-приложения**

В данном разделе приведено описание основных алгоритмов разработанного веб-приложения, чтобы более детально разобраться, как работают определенные бизнес-функции и каким образом осуществляется выполнение конкретных действий, являющихся важной составляющей данных алгоритмов.

Для начала была выполнена формулировка алгоритмов в объемные команды для общего представления назначения рассматриваемого функционала веб-приложения.

Затем на каждом последующем этапе отдельные детали алгоритмов уточнялись. Данный процесс продолжался, пока алгоритмы не состояли из команд, которые были бы понятны базовому исполнителю.

Такой подход к проектированию алгоритмов значительно повышает качество и надежность разрабатываемых функциональных модулей в проекте и называется методом последовательного уточнения.

Для реализации алгоритмов были построены блок-схемы, особенностью которых является представление графической интерпретации действий алгоритмов, тем самым – упрощение их понимания и более наглядное представление.

### **2.6.1 Алгоритм создания объявления**

Когда пользователь решает создать объявление в качестве пассажира для ожидания принятия его заявки другими пользователями, выступающими в качестве водителя, он может перейти на страницу создания объявлений и заполнить все необходимые поля, по которым будут понятны его требования для осуществления поездки.

Когда пользователь желает создать объявление в качестве водителя, ему также нужно перейти на страницу создания объявлений, но при этом в поле выбора роли в рамках осуществляемой поездки, ему нужно указать в выпадающем списке роль «Водитель».

На рисунке 2.3 представлена графическая схема алгоритма создания объявления.

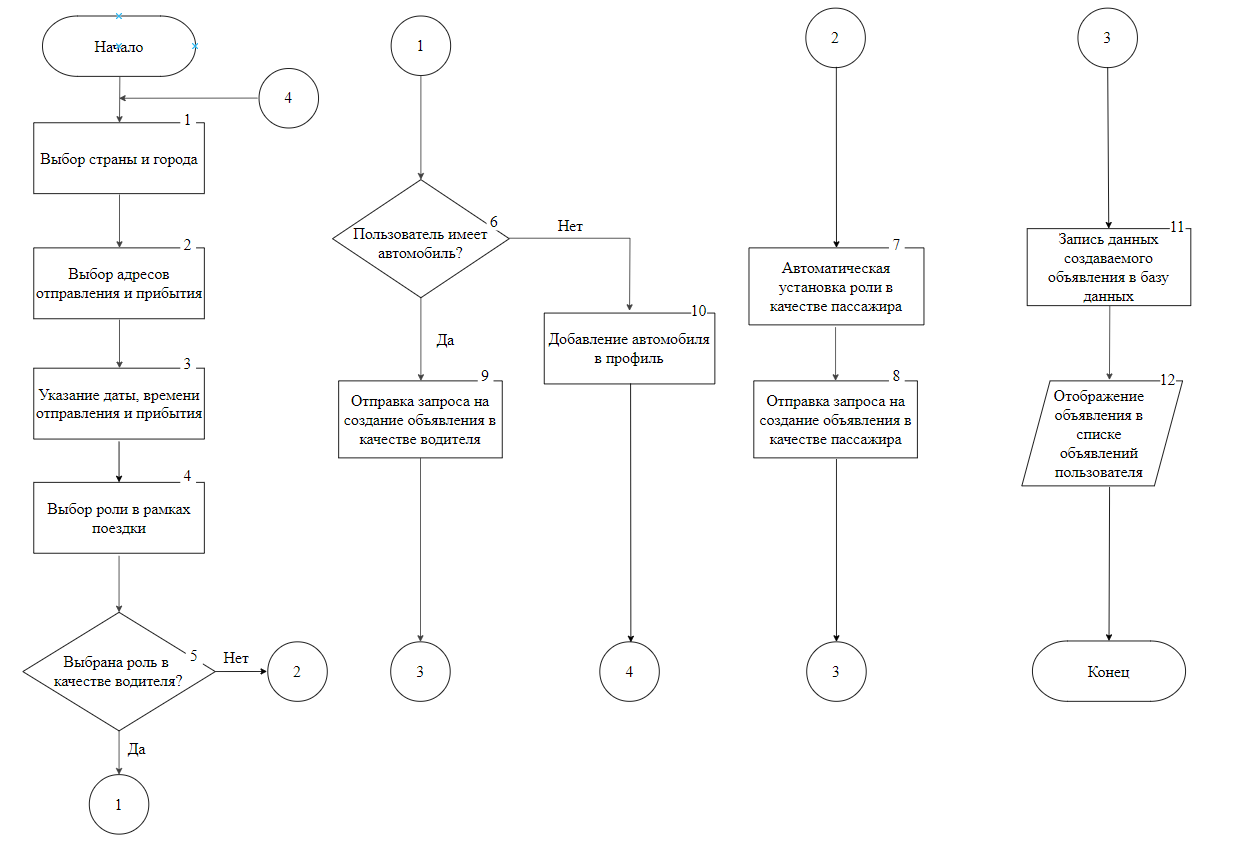


Рисунок 2.3 – Блок-схема алгоритма создания объявления

В полном размере блок-схему алгоритма создания объявления можно увидеть в приложении В.

После того, как пользователь выбрал роль водителя, система произведет проверку на наличие у данного пользователя автомобиля в профиле. Если пользователь предварительно не добавил автомомбиль, при выборе роли «Водитель» он сразу же получит уведомление о том, что перед тем, как создавать объявления в качестве водителя, ему необходимо для начала добавить автомобиль в свой профиль. Также вместе с пояснением в уведомлении отобразится ссылка для перехода в личный профиль пользователя. В случае, если пользователь продолжит создание объявления без добавления автомобиля в своем профиле, роль в соответствующем поле по умолчанию установиться в значение «Пассажир». Таким образом будет создано объявление в качестве пассажира.

Если же пользователь добавил автомобиль в профиль, то при повторении предыдущих шагов, после выбора роли «Водитель», ему отобразится еще одно поле с выпадающим списком, где будет предложено выбрать один из автомобилей, имеющихся у пользователя. Из этого можно сделать вывод, что каждый пользователь может иметь несколько автомобилей.

### **2.6.2 Алгоритм создания заказа**

Алгоритм осуществления заказа имеет схожесть с алгоритмом создания объявления. Схожесть заключается в проверке выбора объявления. Когда пользователь хочет выбрать попутчика для поездки, ему необходимо выбрать объявление, в котором указана роль поездки в качестве пассажира. В обратном случае – он выбирает объявление с ролью в качестве водителя.

Блок-схему алгоритма создания заказа можно увидеть на рисунке 2.4 и в приложении Г.

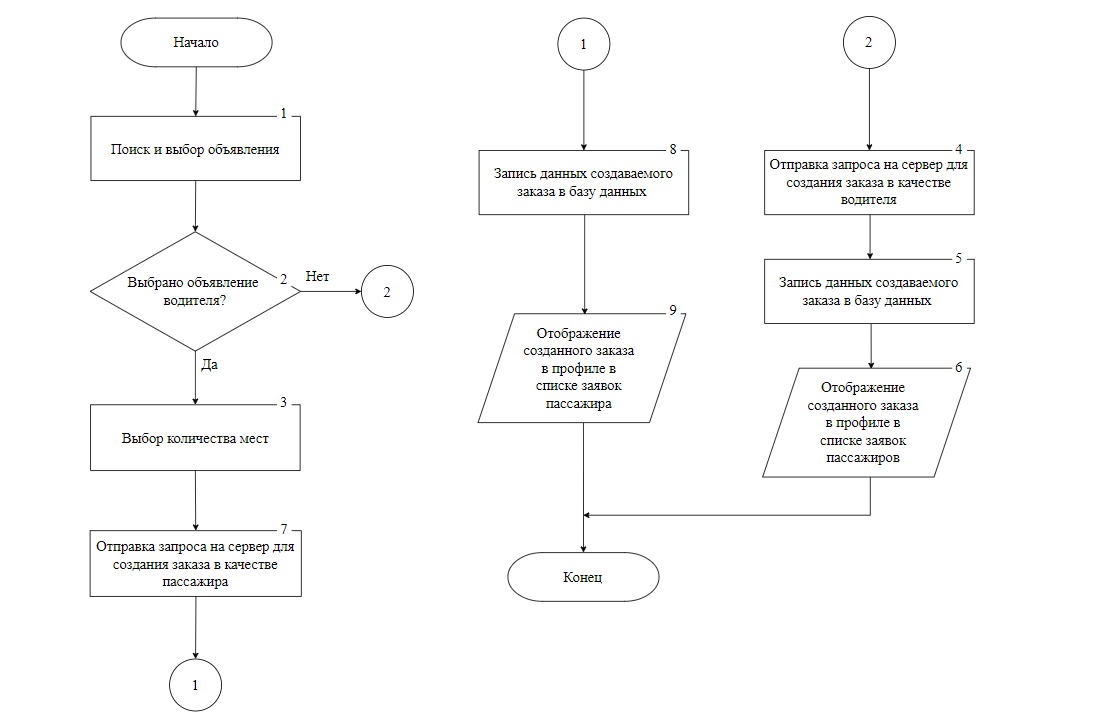


Рисунок 2.4 – Блок-схема алгоритма созадния заказа

Начало алгоритма подразумевает собой открытую страницу с объявлениями пользователей, где расположены объявления водителей и пассажиров.

Следующий шаг алгоритма – выбор объявления.

Если пользователь выбрал объявления водителя, то для осуществления бронирования поездки ему достаточно указать количество бронируемых мест и нажать на кнопку «Поехали!». После осуществления приведенных действий, пользователь попадет на страницу своего профиля, где во вкладке «Заказы» в списке заказов пользователя отобразится карточка с описанием только что осуществленного заказа. После данного действия он может оставить отзыв о водителе по окончанию поездки.

Если пользователь выберет объявление пассажира и нажмет на кнопку «Поехали!», то, при наличии в своем профиле автомомиля, он успешно примет заявку от пассажира и попадет на вкладку «Заказы», где в списке заявок пассажиров увидит недавно принятый заказ. В обратном случае – при отсутствии в своем профиле автомобиля, пользователь получит уведомление о том, что для начала должен добавить авто в свой профиль, где по ссылке под описанием причины невозможности принять заявку пассажира он сможет перейти в свой профиль и выполнить необходимое требование для осуществления заказа.

Также в дополнение к блок-схеме алгоритма создания заказа была добавлена диаграмма последовательности заказа, которую можно увидеть на рисунке 2.5 и в приложении Д.

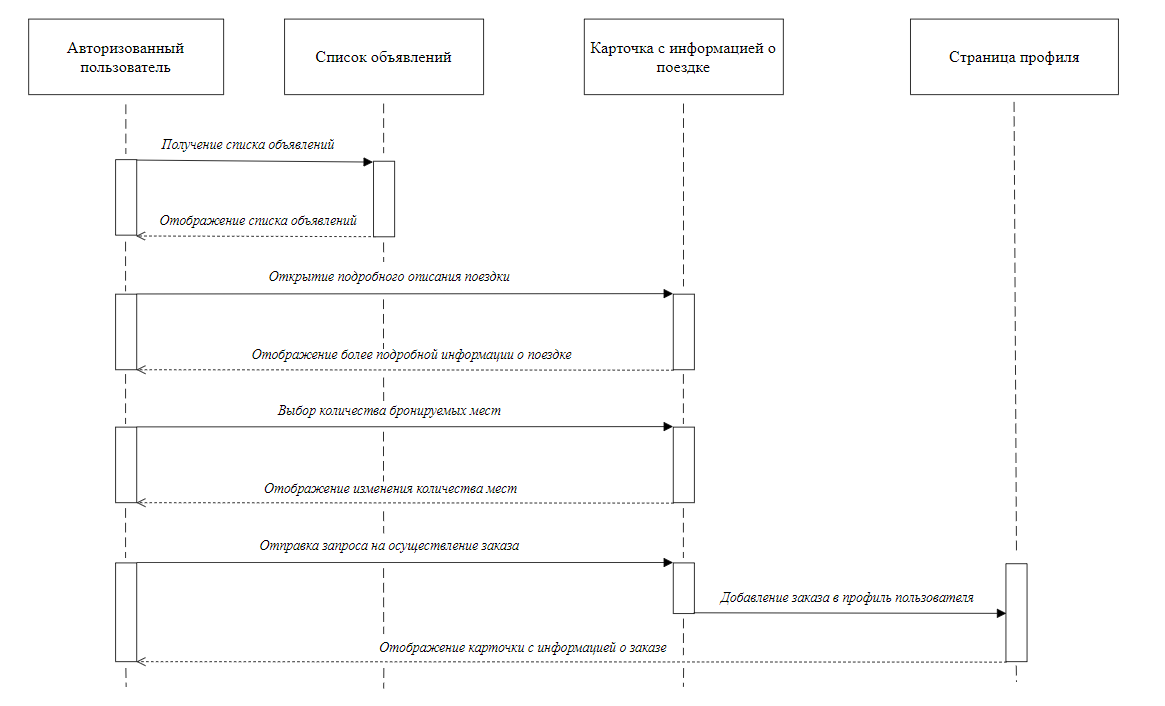


Рисунок 2.5 – Диаграмма последовательности заказа

### **2.6.3 Алгоритм добавления отзыва**

Для того, чтобы видеть больше информации о пользователях, которая поможет сделать выбор в сторону какой-либо поездки, в веб-приложении была добавлена возможность оставлять отзывы. Также стоит отметить, что оставлять отзывы о пользователях можно только после осуществления поездки. Для этого достаточно просто перейти к списку своих заявок на поездки на странице профиля, выбрать поездку, к которой оставляется отзыв, и дать оценку с комментарием.

Алгоритм добавления отзывов представляет собой довольно стандартный набор действий, при котором пользователь для начала выбирает объявление, которое удовлетворяет его требованиям к поездке, после чего осуществляет заказ.

Далее пользователь после осуществления заказа может перейти на страницу своего профиля, открыть вкладку «Заказы», выбрать пункт меню «Список ваших заявок» и выбрать карточку с информацией о поездке, заказ на которую был им осуществлён.

После этого пользователю необходимо нажать на кнопку «Оставить отзыв», расположенную в нижней части карточки с информацией о заказе. Далее можно указать оценку к поездке, а также оставить комментарий к ней. Можно так же оставить отзыв без комментария, давая только оценку.

После указания оценки и оставления комментария к поездке, нужно нажать на кнопку «Отправить», после чего информация отзыва появится на странице профиля пользователя, которому он адресован, на вкладке «Отзывы».

## **2.7 Выводы по разделу**

Для разработки веб-приложения был выбран язык программирования *Javascript*, платформа *Node.js*, так как они обеспечивают наибольший выигрыш в скорости и гибкости разработки, обладают развитой базой готовых решений для типичных и ряда специфических задач, а также хорошей документированностью и развитым сообществом, которое позволяет на порядок быстрее искать решения на возникающие в процессе изучения, в том числе глубокого, платформы и разработки приложений вопросы и трудности.

Для клиентской части веб-приложения был выбран язык *Javascript* и библиотеки *React.js* и *MUI*, так как они имеют гибкую архитектуру и позволяют в короткие сроки разработать веб-приложение [13].

Также были рассмотренны алгоритмы осуществления заказа и создания объявления.

# **3 Разработка веб-приложения**

*ФИО*

*Подпись*

*Дата*

Лист

1

28

*БГТУ 03.00.ПЗ*

Разраб.

*Сивак М.Н.*

Провер.

.

*Сухорукова И.Г.*

Консульт.

*Сухорукова И.Г.*

Н. контр.

*Макарчук М.В*

Утв.

*Пацей Н.В.*

*я*

3 Разработка веб-приложения

Лит.

Листов

10

74417021, 2022

На первом этапе разработки веб-приложения и серверной части необходимо установить программное обеспечение, добиться стабильной работы серверов и всех служб. Эти действия можно охарактеризовать, как настройка среды.

Далее необходимо разработать *REST-API* сервер [14].

И на последнем этапе необходимо разработать клиентское веб-приложение.

Согласно техническому заданию, программное средство состоит из следующих сервисов:

* база данных;
* серверное приложение;
* клиентское веб-приложение.

## **3.1 Разработка серверной части**

Разрабатываемое веб-приложение представляет собой сервис для поиска попутчиков, где главная цель – это упростить поиск попутчков для тех, кто выступает в роли водителей, и поиска водителей для тех, кто ищет поездку, а также возможность поиска объявлений по нужным критериям. В результате была создана база данных, которая удовлетворяет поставленным задачам. Данная реализация достаточно проста и универсальна, с ее помощью в дальнейшем можно изменять и дополнять структуру базы данных. Функционально программное средство должно выполнять следующие задачи:

* аутентификация пользователей;
* регистрация пользователей;
* создание объявлений в качестве водителя;
* создание объявлений в качестве попутчика;
* поиск объявлений;
* бронирование поездкок;
* принятие заявок попутчиков.

Во время разработки *RESTful API* для данного проекта возникла необходимость реализовать аутентификацию пользователя. Одним из принципов *REST* является независимость от состояния. Это значит, что клиент должен сам позаботиться о своей аутентификации при каждом запросе. Эту задачу выполняет *JSON WebToken* или просто *JWT*. *JWT* – это открытый стандарт для создания токенов доступа, основанный на формате *JSON* [15]. Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях.

*JWT* имеет некоторые преимущества – он самодостаточен, все необходимые для аутентификации данные можно хранить в самом токене.

Последовательно рассмотрим устройство токена. *JWT* состоит из трех основных частей: заголовка (*header*), нагрузки (*payload*) и подписи (*signatur*e). Заголовок и нагрузка формируются отдельно, а затем на их основе вычисляется подпись. После первого логина, клиенту возвращается сгенерированный сервером *JWT*.

При каждом запросе клиент должен передавать *JWT* установленным *API* способом. В нашем случае через заголовок *x-access-token*. Сервер декодирует *header* и *payload* и проверяет зарезервированные поля. Если все в порядке, по указанному в *header* алгоритму составляется подпись. Если полученная подпись совпадает с исходной, пользователь проходит авторизацию.

Реализация *JWT* аутентификации в нашем приложении осуществляется с использованием библиотеки *jsonwebtoken*. Для этого надо написать логику проверки токена, отправляемого пользователем при каждом запросе к защищенному ресурсу. Также в нашем случае при запросе пользователя мы получаем заголовок *x-access-token*, проверяем, существует ли он, и если он был получен благополучно то мы вызываем метод *jwt.verify*, в коллбэке которого при успешной верификации извлекаем идентификатор пользователя для дальнейших действий с ним.

*JWT*-токен будет передаваться в каждом запросе и контроллер, при наличии пкрикрепленного к маршруту *middleware authJwt*. Описанная выше логика аутентификации будет расшифровывать токен и в случае ошибки возвращать 401 статус ошибки.

Общая структура серверной части проекта представлена на рисунке 3.1

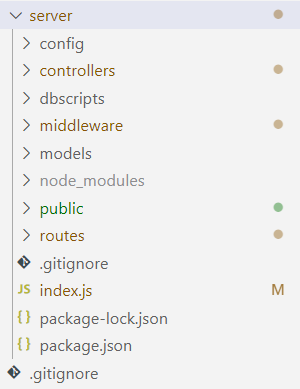


Рисунок 3.1 – Общая структура серверной части проекта

Корневая директория server содержит следующие поддиректории: директория *config*, где находятся файлы кофигурации для соединения с базой данных, а также секретный ключ для проверки токена при аутентификации; *controllers* – папка, где хранятся контроллеры, обрабатывающие запросы со стороны клиента и возвращающие результат в формате *JSON*; *middleware* – папка, в которой находятся промежуточные обработчики запросов, которые отлавливают запрос до его поступленя в обработку контроллеру; директория *models*, где находятся файлы-модели с описанием сущностей базы данных с помощью *Sequelize ORM*; *node\_modules* – папка со всеми зависимостями проекта; *public* – каталог, где хранятся изображения добавленных автомомбилей, а также фотографии профилей пользователей; *routes* – директория, содержащая в себе фалы с маршрутизацией запросов.

Есть базовые компоненты, без которых минимальная работа невозможно, а есть компоненты необходимые для определенных задач.

Файлы *package.json* и *package-lock.json* – это файлы проекта, которые содержат все важные данные о структуре зависимостей проекта и их версий.

Файл *.gitignore* – необходим для системы контроля версий *Git*.

Файл *index.js* – это файл, содержащий в себе все конструкции кода, необходимые для запуска приложения, соединения с базой данных, а также инициализацию других компонентов, необходимых для корректной работы приложения.

Далее рассмотрим структуру папки *controllers*, представленной на рисунке 3.2.

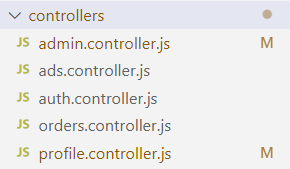


Рисунок 3.2 – Структура папки *controllers*

В папке находятся файлы, отвечающие за обработку запросов, поступающих от клиента с ерверу. Рассмотрим файл *auth.controller.js*, один из методов которого представлен в листинге 3.1.

|  |
| --- |
| // Save User to Database – signUp method  User.create({  userName: userName,  userPhone: userPhone,  userEmail: userEmail,  userPassword: bcrypt.hashSync(userPassword, 8),  roleId: 2, //user  gender: gender,  userStatus: 1,  })  .then(() => {  res.send({ message: "User was registered successfully!" });  })  .catch(err => {  res.status(500).send({ message: err.message }); |

Листинг 3.1 – Файл *auth.controller.js*

Данный файл является контроллером, обрабатывающим запросы на регистрацию и авторизацию пользователей.

Метод *signUp* получает на вход данные, введенные пользователем в поля модального окна регистрации и проверяет их корректность. При успешной проверке данных, создается новый пользователь в системе посредством добавления его в базу данных.

Также в данном файле находится метод *signIn*, который используется для входа пользователей в систему. После получения данных на вход, он хеширует пароль, отправляемый пользователем при запросе на авторизацию, и сверяет его с хешем пароля, хранящемуся в базе данных, при успешном выполнении этой функции, пользователю вернется ответ со статусом 200 *OK*, в котором также будет отправлен сгенерированный *jwt*-токен для дальнейшей аутентификации.

Далее рассмотрим структуру директории *models*, представленной на рисунке 3.3.

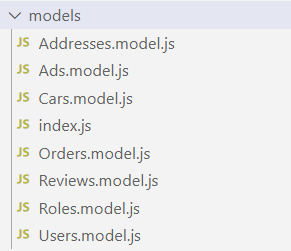


Рисунок 3.3 – Структура папки *models*

В данной папке расположены файлы с описанием моделей сущностей базы данных, для чего была использована библиотека *Sequelize ORM*. Каждая модель соответствует таблице в базе данных, а связи между моделями, описанные в файле *index.js* каталога *models*, определяют отношения таблиц между собой.

Благодаря синхронизации с базой данных при первом запуске приложения создаются все необходимые таблицы и связи между ними, что экономит массу времени по сравнению с ручным написанием скриптов или при запуске приложения на другом устройстве. Достаточно будет изменить только файл конфигурации для базы данных.

В качесве примера определения модели на основе синтаксиса библиотеки *Sequelize ORM* в соответствии с таблицой в базе данных будет выступать часть описания схемы для таблицы «*Users*».

Пример участка кода определения данной модели можно увидеть на листинге 3.2.

|  |
| --- |
| const { Roles } = require(".");  module.exports = (sequelize, Sequelize) => {  const Users = sequelize.define('Users', {  id: {  type: Sequelize.INTEGER,  autoIncrement: true,  primaryKey: true,  allowNull: false,  field: 'id'  },  roleId: {  type: Sequelize.INTEGER,  allowNull: false,  references: Roles,  referencesKey: 'id',  field: 'roleId',  onDelete: 'CASCADE'  },  userName: {  type: Sequelize.STRING,  allowNull: true,  field: 'userName'  },  userPhone: {  type: Sequelize.STRING,  allowNull: false,  field: 'userPhone'  },  userEmail: {  type: Sequelize.STRING,  allowNull: false,  field: 'userEmail'  }, |

Листинг 3.2 – Описание модели «*Users*»

Далее перейдем к просмотру структуры директории *routes*, изображение которой представлено на рисунке 3.4.

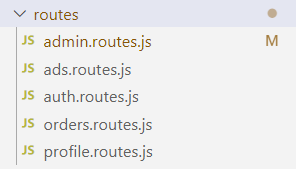


Рисунок 3.4 – структура директории *routes*

В данной директории находятся файлы с определением маршрутов, по которым клиент будет выполнять запросы к серверу. Все маршруты разделены по названию и по логике в зависимости от того, к каким сущностям производится запрос от клиента. Инициализация всех маршрутов находится в файле *index.js* корневого директория приложения.

Создавая архитектуру приложения таким образом, время, затраченное на создание какого-либо нового функционального модуля, значительно сокращается, так как гораздо проще ориентироваться в проекте, где все каталоги и файлы имееют названия, соответствующие названиям других сущностей, с которыми они связаны.

Рассмотрим файл *auth.routes.js*, предназначенный для маршрутизации запросов на регистрацию и авторизацию пользователей. Содержимое файла представлено на листинге 3.3.

|  |
| --- |
| const { verifySignUp } = require("../middleware");  const controller = require("../controllers/auth.controller");  module.exports = function(app) {  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);  }; |

Листинг 3.3 – Файл маршрутизации *auth.routes.js*

При каждом запросе на маршрут авторизации или аутентификации в объект ответа будут прикреплятся заголовки *Access-Control-Allow-Headers* и *x-access-token, Origin, Content-Type, Accept* для того, чтобы указать, какие заголовки могут использоваться во время фактического запроса от клиента, что позволяет нам использовать заголовок *x-access-token* для осуществления аутентификации по технологии *JWT*. После отправке по маршрутам, маршрутизатор передает полномочия на дальнейшую обработку запроса соответствующему методу контроллера *auth.controller.js*.

## **3.2 Разработка клиентской части**

В рамках создания клиентской части веб-приложения для поиска попутчиков, был создан веб-интерфейс, который позволяет использовать весь необходимый функционал из разработанного серверного приложения.

Для разработки клиентской части использовались следующие технологии:

* язык гипертекстовой разметки *HTML*;
* каскадные таблицы стилей *CSS*;
* библиотека *React.js*;
* библиотека *Material UI*;
* система контроля версий *Git*;
* менеджер пакетов *NPM*.

На рисунке 3.5 предоставлена общая структура проекта клиентской части разрабатываемого веб-приложения.

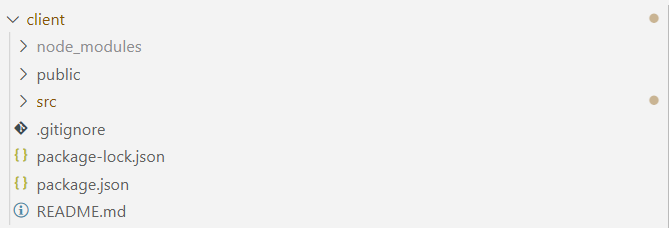
****

Рисунок 3.5 – Общая структура проекта клиентской части

Разработка была начата с разработки основной структуры проекта клиентской части веб-приложения.

Описание файлов проекта, приставленных на рисунке 3.5:

* *src* – основная директория всего исходного кода клиентской части разработанного веб-приложения;
* *.gitignore* – необходим для системы контроля версий *Git*;
* *package.json* – файл формата *JSON*, содержащий описание проекта, его зависимостей и их версий;
* *README.md* – кратко описывающий проект, используемые им технологии и шаги, необходимые для его развертывания;
* *public* – папка, в которой находятся все изображения, используемые на сайте.

Подробная структура компонентов показана на рисунке 3.6.

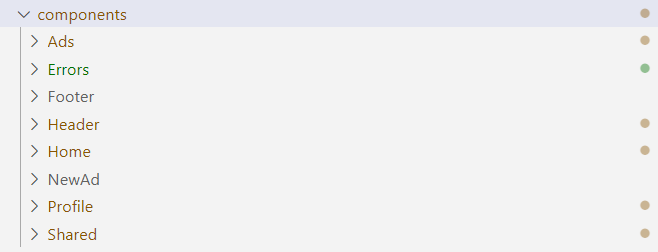


Рисунок 3.6 – Подробная структура компонентов проекта

Директории *Ads, Errors, Footer, Header, Home, NewAd, Profile, Shared* содержат в себе *React*-компоненты, которые отвечают за внешний вид веб-приложения.

Использование компонентной архитектуры приложения значительно упрощает и ускоряет разработку программного обеспечения.

Существует множество разных подходов при разработке компонентной архитектуры, но при применении каждой из них рекоммендуется разбивать большие компоненты на несколько, чтобы было проще ориентироваться в коде и быстро искать причину возникновения различных проблем, а также это помогает структурировать компоненты относительно функциональной логики, которую они выполняют.

Также часто используемые участки кода зачастую выносят в так называемые Shared-компоненты. Вместо того, чтобы вставлять реализацию в каждый шаблонный компонент, его просто выносят в отдельный директорий и с помощью импорта используют в виде тега.

Пример начала общего компонента для поиска объявлений представлен на листинге 3.4.

|  |
| --- |
| <Box style={{ backgroundColor: "rgb(34, 34, 34)" }}>  <Container style={stylesHome.mainSearchContainer} maxWidth='lg'>  <Grid container xs={12} spacing={2}>  <Grid item xs>  <FormControl sx={{ m: 1, maxWidth: 800, minWidth: 200 }} fullWidth>  <InputLabel id="demo-simple-select-label">Город</InputLabel>  <Select  labelId="demo-simple-select-label"  id="demo-simple-select"  value={city}  label="Город"  onChange={handleCityChange}  > |

Листинг 3.4 – Начало компонента *Search*

Основная часть компонента представлена на листинге 3.5.

|  |
| --- |
| <Box maxWidth='lg' style={{ background: 'white', padding: '2em', maxHeight: '600px', marginTop: '10px', margin: '0px auto', backgroundColor: 'white', overflowY: 'scroll', borderRadius: '10px', display: searchMenu }}>  <Grid container xs={12} style={{ textAlign: '-webkit-center' }}>  {ads.length == 0 ? <Typography style={{ margin: '0 auto', fontSize: '1.6em' }}>По результатам поиска ничего не найдено :(</Typography> : ''}  {ads.map((ad) => (  ad['User.id'] != localStorage.getItem('x-user-id')  ?  <>  <Grid  item |

Листинг 3.5 – Основная часть компонента *Search*

Далее на листинге 3.6 представлен участок компонента-контейнера, где используется общий компонент *Search*, приведенный в листинге 3.5 выше.

|  |
| --- |
| <Box style={stylesHome.bannerBoxImage}>  <Search />  </Box>  <Box style={{ backgroundColor: '#19181C', padding: '5em 0' }}>  <Box>  <Typography variant='h3' color='white'>Наши преимущества</Typography>  </Box>  <Container style={{ marginTop: '80px' }} maxWidth='lg'>  <Grid container xs={12}>  <Grid item xs sx={{ m: 1 }}> |

Листинг 3.6 – Компонент-контейнер *Home*

Представленный компонент использует только тег с названием общего компонента *Search*, что значительно упрощает читаемость кода и предоставляет больший комфорт для разработки приложения.

## **3.3 Развертывание веб-приложения**

Для того, чтобы развернуть данное веб-приложение, для начала нужно установить следующие компоненты на удаленный сервер, либо на локальный компьютер:

* актуальная версия *NodeJS*;
* менеджер пакетов *NPM*;
* СУБД *PostgreSQL*.

После установки перечисленных компонентов, необходимо открыть директорию с названием проекта. В данном приложении сервер и клиент запускаются отдельно, поэтому для начала нужно перейти в консоль серверной части приложения и выполнить следующие команды:

* *npm i;*
* *npm run build;*
* *npm start.*

Далее необходимо убедится, что сервер запущен и прослушивает указанный в настройках проекта порт. Пример результата запуска сервера представлен на рисунке 3.7.

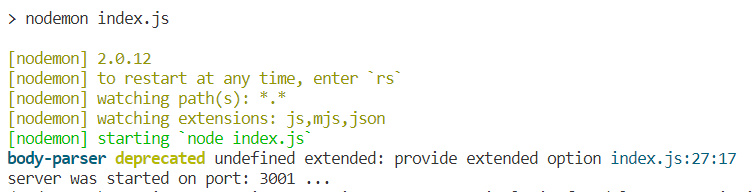


Рисунок 3.7 – Успешный запуск сервера

Далее точно такие шаги нужно проделать для запуска клиентской части. После запуска клиентской части приложения, автоматически откроется браузер, где отобразится интерфейс веб-приложения после прогрузки компонентов.

После того, как были выполнены все шаги по установке необходимого программного обеспесчения, пакетов и необходимых зависимостей для запуска и корректной работы клиента и сервера, приложение будет полностью готово к использованию конечными пользователями.

## **3.4 Выводы по разделу**

В данном разделе были подробно рассмотрены этапы разработки серверной и клиентской частей веб-приложения. На первом этапе было установлено и настроено необходимое программное обеспечение, произведена надстройка среды разработки. В процессе разработки были подробно рассмотрены реализация некоторых компонентов серверной части приложения, а также реализация компонентов предствлений и компонентов-контейнров клиентской частей веб-приложения.

После этапа разработки было произведено ручное развертывание веб-приложения.

# **4 Тестирование веб-приложения**

*ФИО*

*Подпись*

*Дата*

Лист

1

38

*БГТУ 04.00.ПЗ*

Разраб.

*Сивак М.Н.*

Провер.

.

*Сухорукова И.Г.*

Консульт.

*Сухорукова И.Г.*

Н. контр.

*Макарчук М.В*

Утв.

*Пацей Н.В.*

*я*

4 Тестирование веб-приложения

Лит.

Листов

9

74417021, 2022

Тестирование – очень важный и трудоемкий этап процесса разработки программного обеспечения, так как правильное тестирование позволяет выявить подавляющее большинство ошибок, допущенных при составлении программ.

Тестирование программного средства проводится с целью обеспечить качество разрабатываемого продукта. Исходя из стандарта *ISO/IEC* 12207-2003, посвященного описанию систем обеспечения качества программного обеспечения, под качеством мы будем понимает совокупность характеристик программного продукта, относящихся к его способности удовлетворять установленные и предполагаемые потребности клиента.

Пользователь ожидает, что используемое им веб-приложение очень просто, интуитивно понятно и работает без сбоев. Поэтому качество приложения играет очень большую роль в его популярности.

Тестирование в области приложения – это процесс оценки того, что все части приложения ведут себя как надо.

Основным параметром качества программы является надежность.

Надежность – способность программного обеспечения выполнять требуемые задачи в обозначенных условиях на протяжении заданного промежутка времени или указанное количество операций. Атрибуты данной характеристики – это завершенность и целостность всей системы, способность самостоятельно и корректно восстанавливаться после сбоев в работе, отказоустойчивость.

Тестирование программного средства проводилось на основании функциональных требований, представленных в разделе 2. При этом проверялось соответствие функциональности реализованных страниц и описаний требований.

## **4.1 Тестирование основного функционала веб-приложения**

Тестирование программного средства проводилось на основании функциональных требований к программному продукту, представленных во втором разделе. При этом проверялось соответствие функциональности реализованных страниц и описаний требований.

Было произведено тестирование основного функционала приложения. Помимо теста соответствия функционала описанию, было произведено тестирование приложения на наличие уязвимостей.

Также были проведены тесты по типу ввода неправильных данных, переход напрямую по ссылкам, не предназначеным для текущих типов пользователей. Некоторые из тестов основного функционала представлены в таблице 4.1.

Таблица 4.1 – Результат тестирования основного функционала

|  |  |  |  |
| --- | --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат | Результат теста |
| Регистрация пользователя | Пользователь создан в системе, данные записаны в базу. | Пользователь создан в системе, данные записаны в базу. | Прошел |
| Авторизация пользователя | Пользователь получает доступ к функционалу приложения. | Пользователь получает доступ к функционалу приложения. | Прошел |
| Просмотр страницы профиля | Пользователь видит информацию о своем профиле. | Пользователь видит информацию о своем профиле. | Прошел |
| Редактирование личной информации о профиле | Пользователь меняет личную информацию в своем профиле, вся информация отображается корректно после изменений. | Пользователь меняет информацию в своем профиле, вся информация отображается корректно после изменений. | Прошел |
| Добавление автомобиля в профиль | Пользователь добавляет автомобиль в свой профиль, список добавленных автомобилей отображается корректно. | Пользователь добавляет автомобиль в свой профиль, список добавленных автомобилей отображается корректно. | Прошел |
| Удаление автомобиля из профиля | Пользователь удаляет автомобиль из профиля. Удаленный автомобиль больше не отображается в списке автомобилей профиля. | Пользователь удаляет автомобиль из профиля. Удаленный автомобиль больше не отображается в списке автомобилей профиля. | Прошел |
| Изменения информации об автомобиле | Пользователь меняет информацию о своем автомобиле. Обновленные данные корректно отображаются в списке автомобилей. | Пользователь меняет информацию о своем автомобиле. Обновленные данные корректно отображаются в списке автомобилей. | Прошел |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат | Результат теста |
| Оставление отзывов о водителях | Пользователь осталяет комментарий водителю. На вкладке с отзывами водитель видет оставленный пассажиром комментарий | Пользователь осталяет комментарий водителю. На вкладке с отзывами водитель видет оставленный пассажиром комментарий | Прошел |
| Закрытие заказа | После нажатия на кнопку «Закрыть заказ» на вкладке «Зкакзы» на странице пользователя, этот заказ помечается, как закрытый и объявление, к которому создан заказ, удаляется. | После нажатия на кнопку «Закрыть заказ» на вкладке «Зкакзы» на странице пользователя, этот заказ помечается, как закрытый и объявление, к которому создан заказ, удаляется. | Прошел |
| Просмотр страницы объявлений | Пользователю отображаются все объявления, кроме тех, которые он сам создавал. | Пользователю отображаются все объявления, кроме тех, которые он сам создавал. | Прошел |
| Поиск объявлений | После заполнения фильтров для поиска, пользователю отображаются объявления, удовлетворяющие фильтрам. | После заполнения фильтров для поиска, пользователю отображаются объявления, удовлетворяющие фильтрам. | Прошел |
| Создание объявления | После создания объявление должно отображаться на странице объявлений. | После создания объявление должно отображаться на странице объявлений. | Прошел |
| Выход из системы пользователем | Пользователь выходит из системы | Пользователь вышел из системы | Прошел |

Окончание таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат | Результат теста |
| Бронирование поездки | После бронирования поездки, в профиле пользователя, осуществившего заказ, во вкладке «Заказы» должна отображаться карточка с кратким описанием заказа. | После бронирования поездки, в профиле пользователя, осуществившего заказ, во вкладке «Заказы» должна отображаться карточка с кратким описанием заказа. | Прошел |

Таким образом, тестирование основного функционала приложения успешно пройдено. Весь заявленный функционал отвечает поставленным функциональным требованиям.

## **4.2 Тестирование безопасности базы данных и приложения**

В приложении практически предусмотрена проверка корректности введенных данных, с помощью ограничения целостности и на уровне клиентской части. На рисунке 4.1 продемонстрировано, что будет если нарушено одно из условий ограничения целостности при регистрации. В данном случае должен быть указан *email* и пароль.

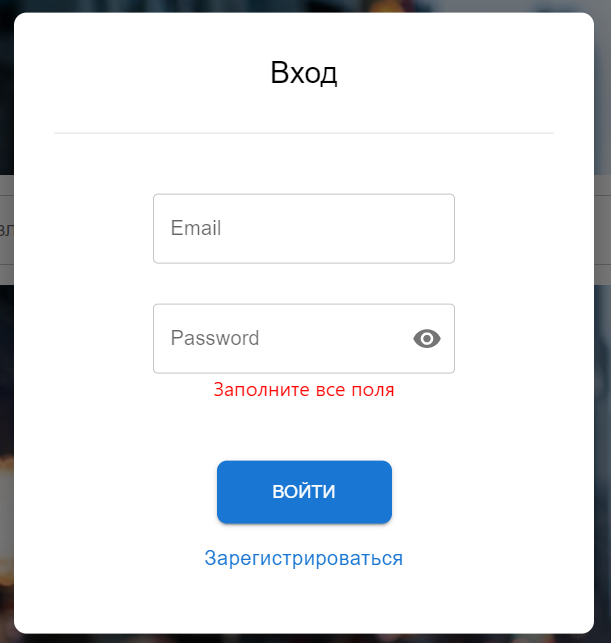


Рисунок 4.1 – Модальное окно входа в профиль

Приложение предназначено только для демонстрации ограничения доступных действий следующих пользователей:

* неавторизованный пользователь (*guest*);
* авторизованный пользователь (*user*);
* администратор (*admin*).

Далее рассмотрим, что происходит при попытке регистрации пользователя с уже существующей почтой в базе данных. Для этого введем в поле почты уже зарегистрированного пользователя *vini@gmail.com.* Результат проверки представлен на рисунке 4.2.

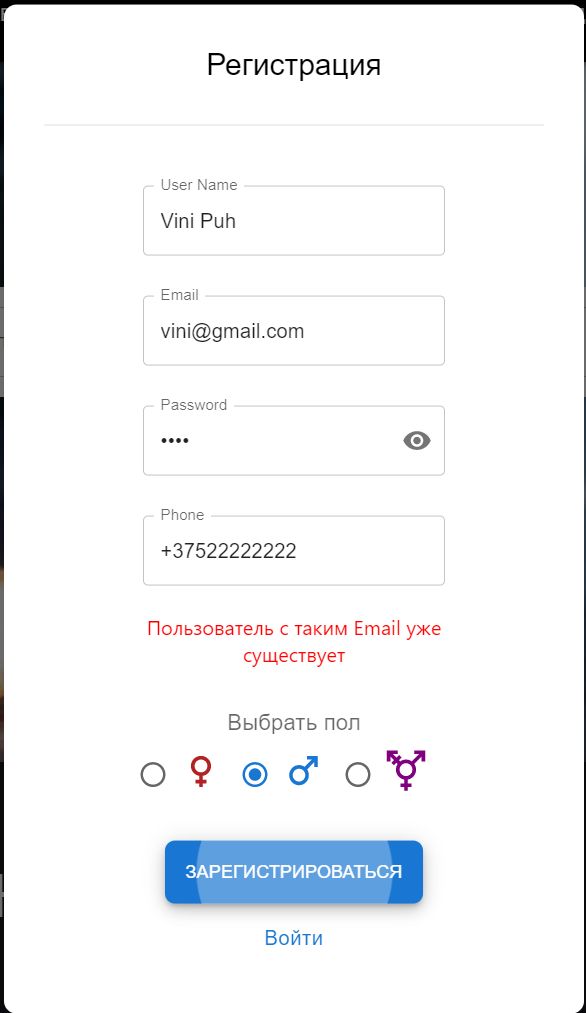


Рисунок 4.2 – Ошибка: пользователь уже зарегестрирован

Пользователи c ролями *user* и *admin* могут бронировать поездки, однако такой возможности нет у пользователя с ролью *guest*.

Если пользователь перейдет на страницу с объявлениями, либо воспользуется поиском и после получения списка объявлений попробует забронировать место, либо принять заявку от попутчика, то система перенаправит его на страницу с ошибкой 401 – *Unauthorized*. На рисунке 4.3 представлен результат, который получит неавторизованный пользователь (*guest*), если попробует забронировать поездку.

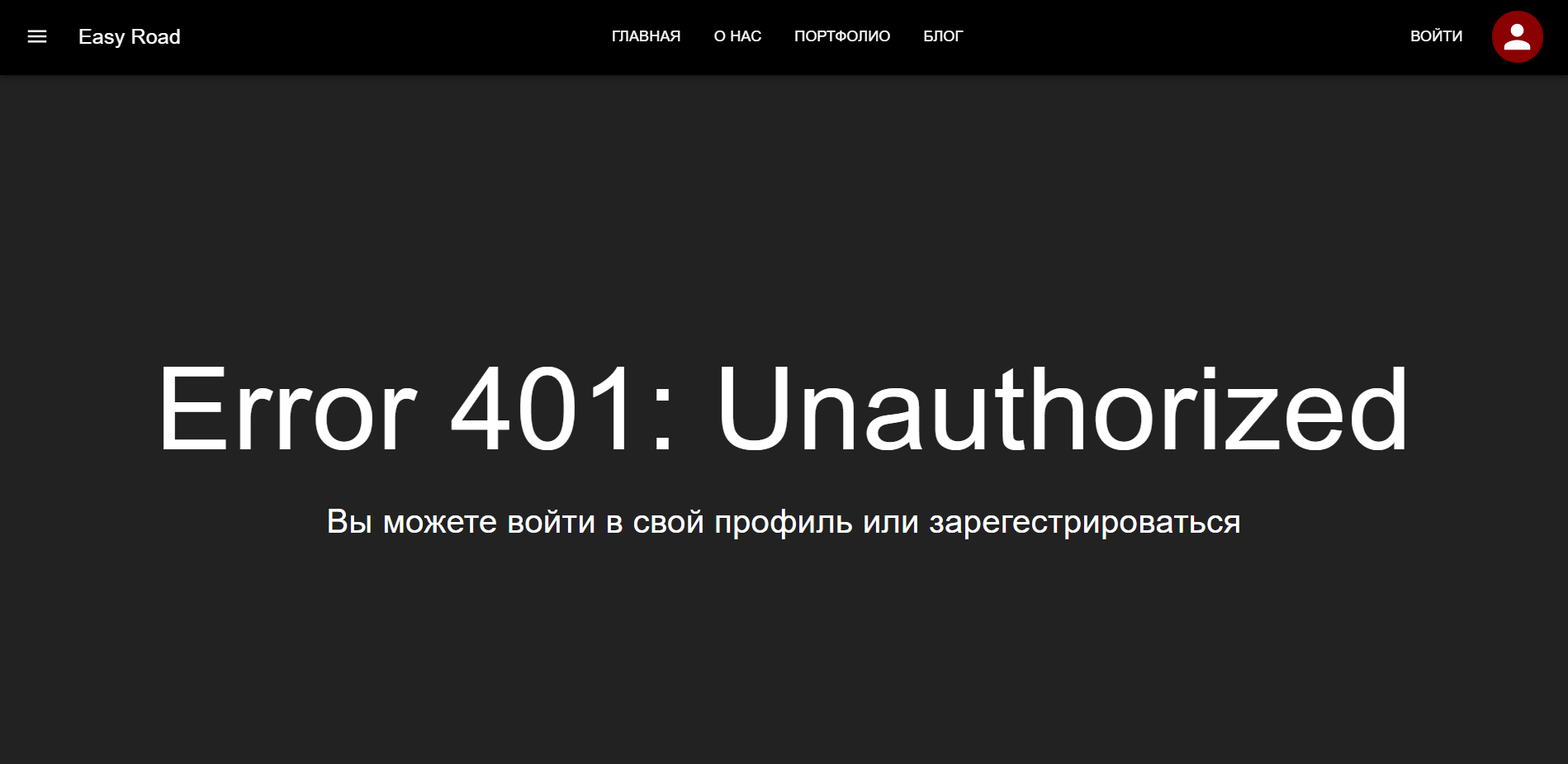


Рисунок 4.3 – Бронирование поездки пользователем с ролью *guest*

Как вино из изображения на рисунке 4.3, пользователь, как и ожидалось, был перенаправлен на страницу с ошибкой, к которой приводится описание причины, почему так произошло.

Далее, для подтверждения того, что каждый пользователь имеет уникальную идентификацию для своего профиля и получает корректную и достоверную информацию, связанную только с его проофилем, предоставлены изображения с личными кабинетами разных пользователей. На рисунках 4.4 и 4.5 отображется наличие разделения контекста для разных пользователей.

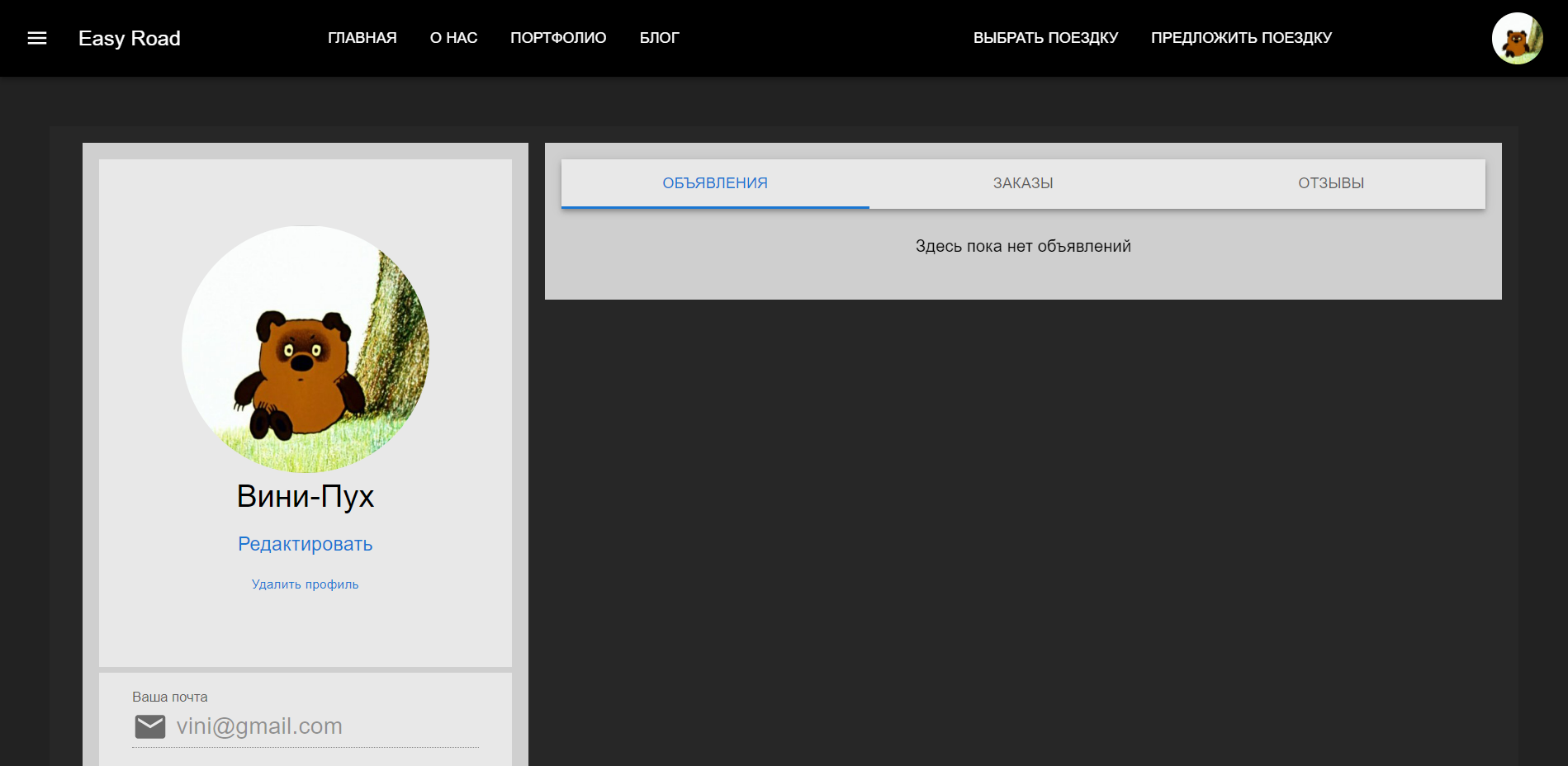


Рисунок 4.4 – Страница профиля пользователя *vini@gmail.com*

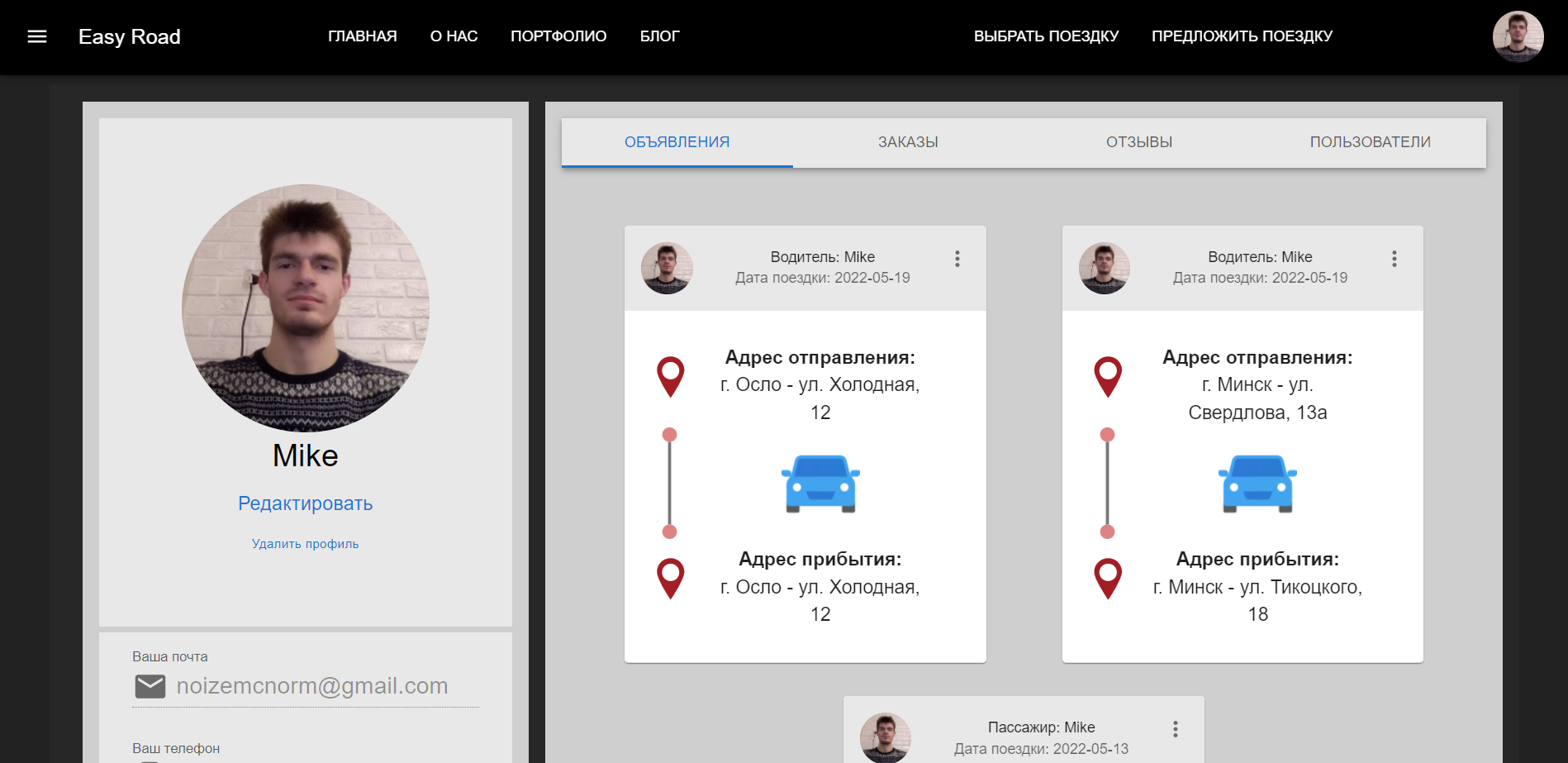


Рисунок 4.5 – Страница профиля пользователя *noizemcnorm@gmail.com*

В результате мы видим, что страница двух пользователя имеет только те данные, которые соответствуют профилю каждого из них.

Для предотвращения непонимания со стороны пользователей при правильном наборе доменного имени веб-приложения, но некорректного маршрута, которого нет в определении серверной части приложения, был создан механизм перенаправления на страницу с ошибкой 404 – *Not Found*.

Если пользователь попытается перейти по несуществующему пути, приложение перенаправит его на страницу 404, которая представлена на рисунке 4.6.

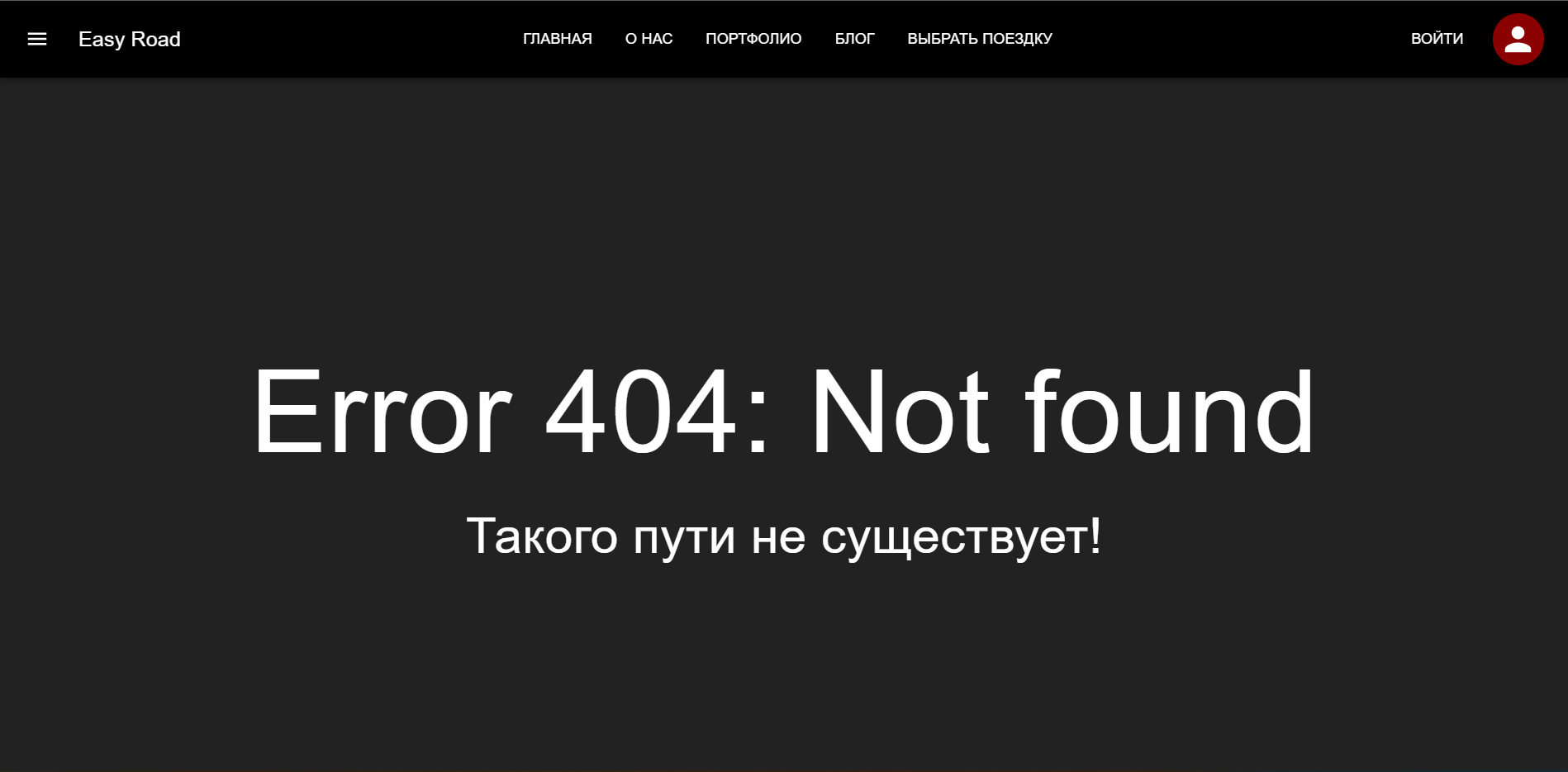


Рисунок 4.6 – Страница 404 (*Not found*)

Далее рассмотрим случай, когда пользователь, уже авторизованный в системе, пытается принять заявку от попутчика, предварительно не добавив в свой профиль автомобиль.

Из соображений безопасности и надежности системы, были созданы механизмы, предотвращающие проблемные ситуации, связанные с мошенничеством, а также с доверием возможных потенциальных клиентов данного приложения.

При описанных действиях пользователя, приложение отобразит модальное окно с сообщением об ошибке, что пользователь не может принимать заявки от попутчиков, пока не добавит автомобиль в свой профиль. Изображение модельного окна с сообщением об ошибке представлено на рисунке 4.7.

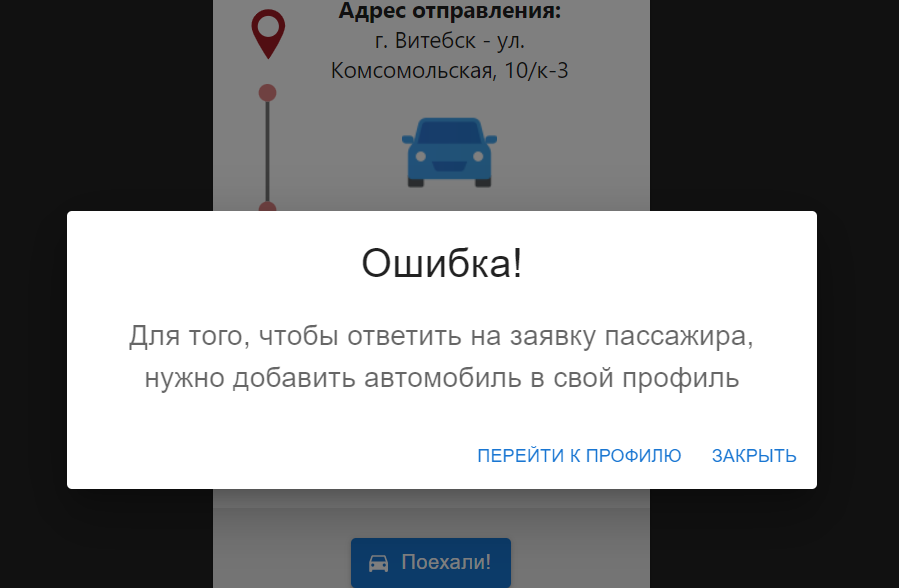


Рисунок 4.7 – Модальное окно с ошибкой

При создании объявлений как в качестве попутчика, как и в качестве водителя, в форме для создания записи о планируемой поездке заполняется ряд полей, которые и составляют полную информацию о поездке в объявлениях. Рассмотрим такой параметр, как количество мест. При создании объявления в качесвте водителя, пользователь указывает, сколько мест он предоставляет для попутчиков в рамках одной поездки. Количество мест выбирается добровольно на усмотрение автора объявления. Также и пользователи, выступающие в качестве попутчиков, могут указать количество бронируемых мест при создании объявления. Любой авторизованный пользователь может зайти на страницу объявлений и выбрать поездку у водителя, указав при этом в элементе управления количество бронируемых мест. Исходя из описанного функционала следует предусмотреть случай, когда в каком-либо объявлении водителя заканчиваются свободные места для попутчиков. Для этого был реализован функционал, который помечает объявления со всеми забронированными местами более темными тонами, меняя их внешний вид, чтобы сделать их отличными от активных объявлений, чтобы пользователям было проще ориентироваться, тем самым экономя их время при поиске желаемых маршрутов для поездки. Так же объявления, помеченные как неактивные, не предоставляют возможности нажимать на кнопки для откытия подробного описания поездки или осуществления брони.

Пример с неактивными объявлениями представлен на рисунке 4.8.

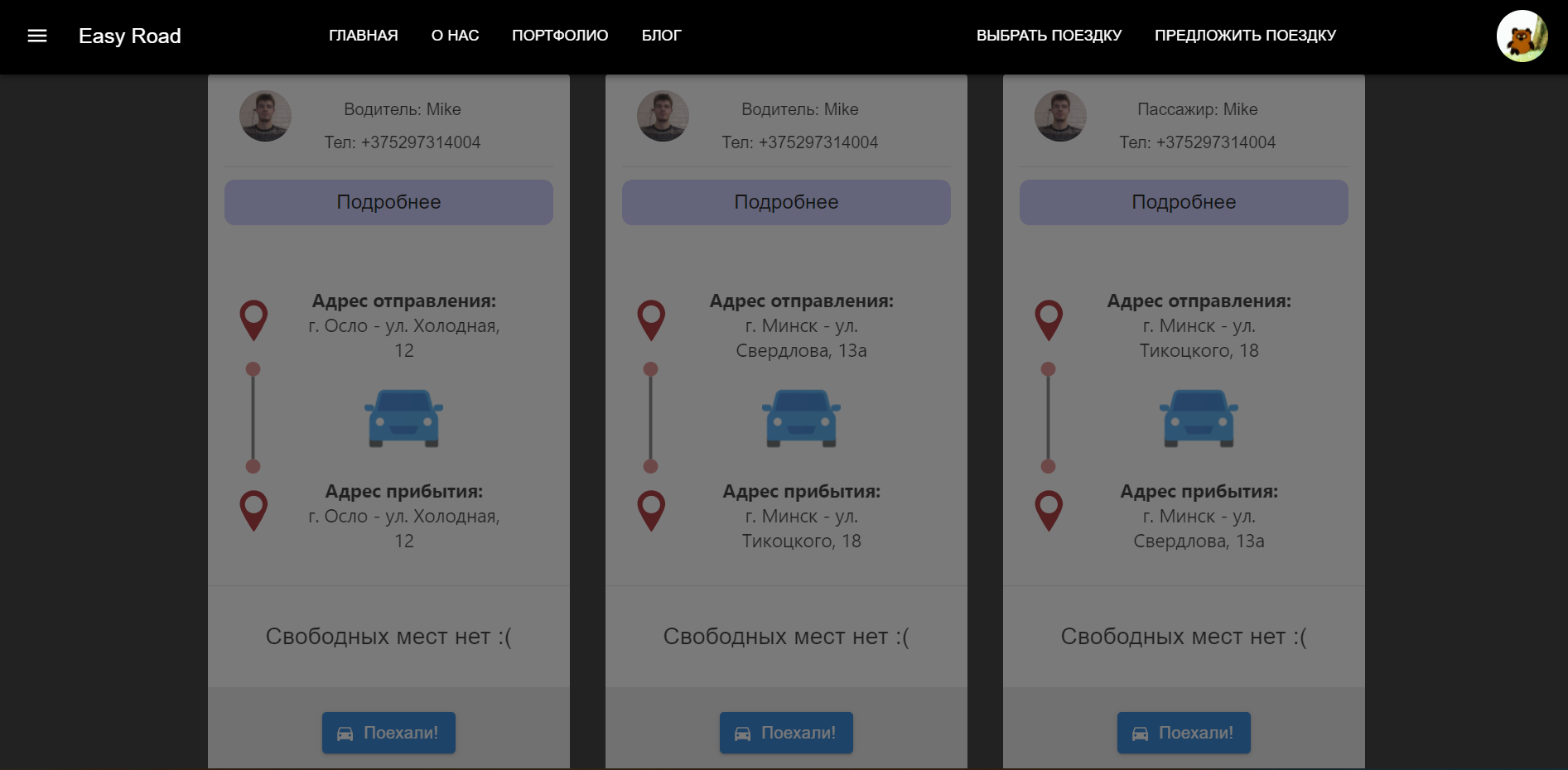


Рисунок 4.8 – Неактивные объявления

## **4.3 Выводы по разделу**

В данном разделе было проведено четырнадцать ручных тестов, в которых были рассмотрены все возможные случаи поведения конечного пользователя при использовании веб-приложения. Из представленных в данном разделе тестов можно сделать вывод, что разработанное программное средство показывает стабильные результаты работы, а также весь функционал соответствует требованиям, работает корректно и выполняет поставленные задачи.

# **5 Методика установки и использования**

*ФИО*

*Подпись*

*Дата*

Лист

1

47

*БГТУ 05.00.ПЗ*

Разраб.

*Сивак М.Н.*

Провер.

.

*Сухорукова И.Г.*

Консульт.

*Сухорукова И.Г.*

Н. контр.

*Макарчук М.В*

Утв.

*Пацей Н.В.*

*я*

5 Методика установки и использования

Лит.

Листов

7

74417021, 2022

## **5.1 Развертывание системы**

Так как серверная часть приложения была написана с использованием платформм *Node.js*, то оно является кроссплатформенным и может запускаться на различных операционных системах (*MacOS/Windows/Linux*). Также следует отметить, что приложение кроссбраузерное и корректно работает в том числе на мобильных устройствах и планшетах. В качестве СУБД должна использоваться *Postgres*, при этом база данных с минимально необходимым набором данных автоматически создастся при первом запуске приложения.

Для сборки клиентской части необходимо запустить команду *npm run build*, которая запускает сборку всех компонентов приложения в один *js*-файл. После этого приложение полностью готово к запуску.

## **5.2 Руководство пользователя**

### **5.2.1 Описание процессов авторизации и регистрации**

Для того чтобы авторизоваться в системе, пользователь должен открыть модальное окно авторизации. Ссылка на открытие окна имеется в верхнем меню под псевдонимом «Войти». Вид модального окна представлен на рисунке 5.1.



Рисунок 5.1 – Окно авторизации

Чтобы пройти процесс авторизации, пользователь должен быть зарегистрирован. Для того чтобы зарегистрироваться, пользователь должен открыть модальное окно регистрации, нажав кнопку «Зарегистрироваться» внизу модального окна авторизации. Изображение окна регистрации представлено на рисунке 5.2.

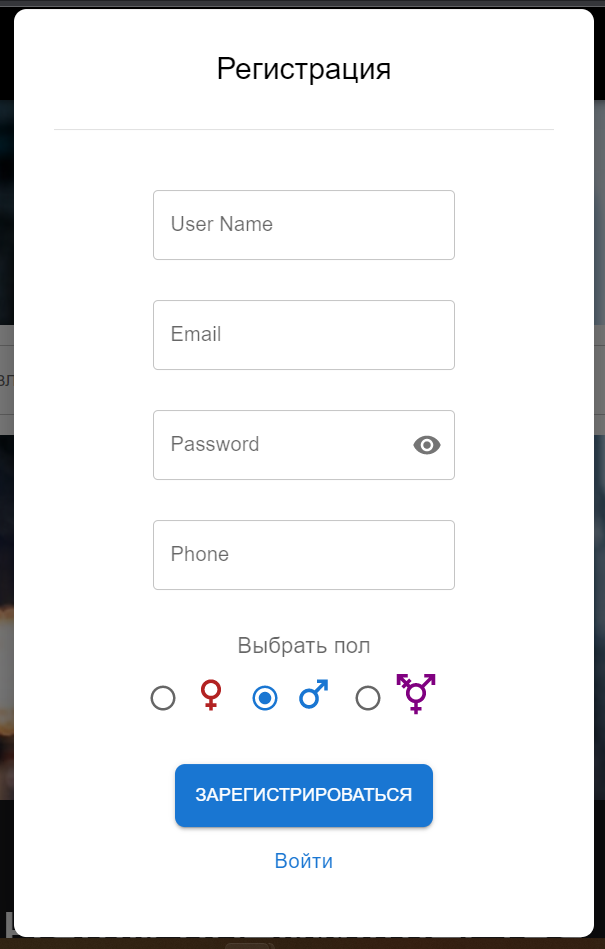


Рисунок 5.2 – Окно регистрации

После осуществления регистрации пользователю нужно будет осуществить процесс авторизации для доступа к защищенным ресурсам веб-приложения.

Точно так же можно вернуться и в окно авторизации при нажатии на ссылку «Войти» под кнопкой «Зарегистрироваться» модального окна регистрации.

Как было указано ранее: роль «гость» может просматривать список объявлений.

Страница для просмотра объявлений представлена на рисунке 5.3.

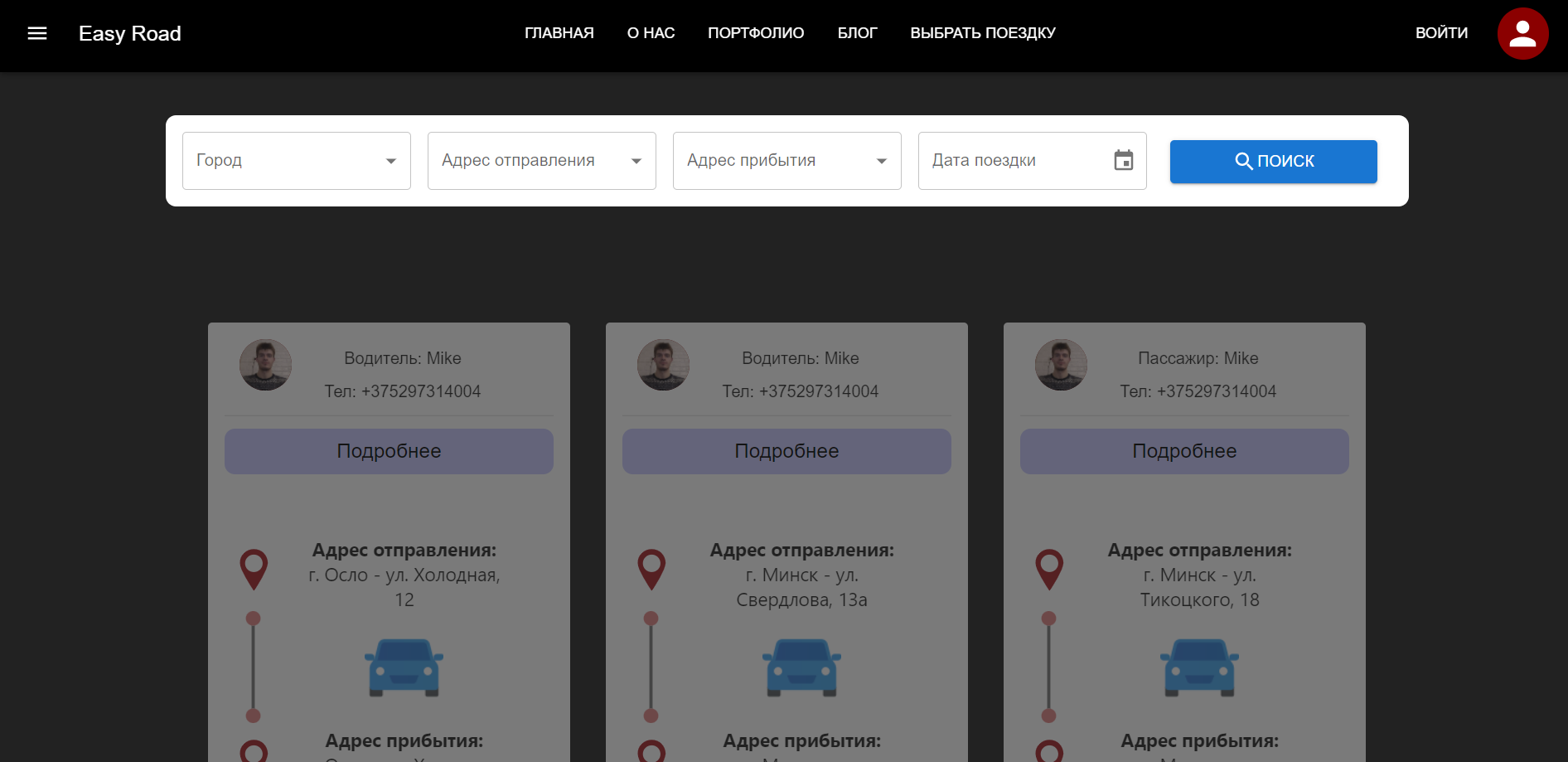


Рисунок 5.3 – Список объявлений

Также роль «гость» может вести поиск объявлений по фильтрам. Данный функционал представлен на рисунке 5.4.

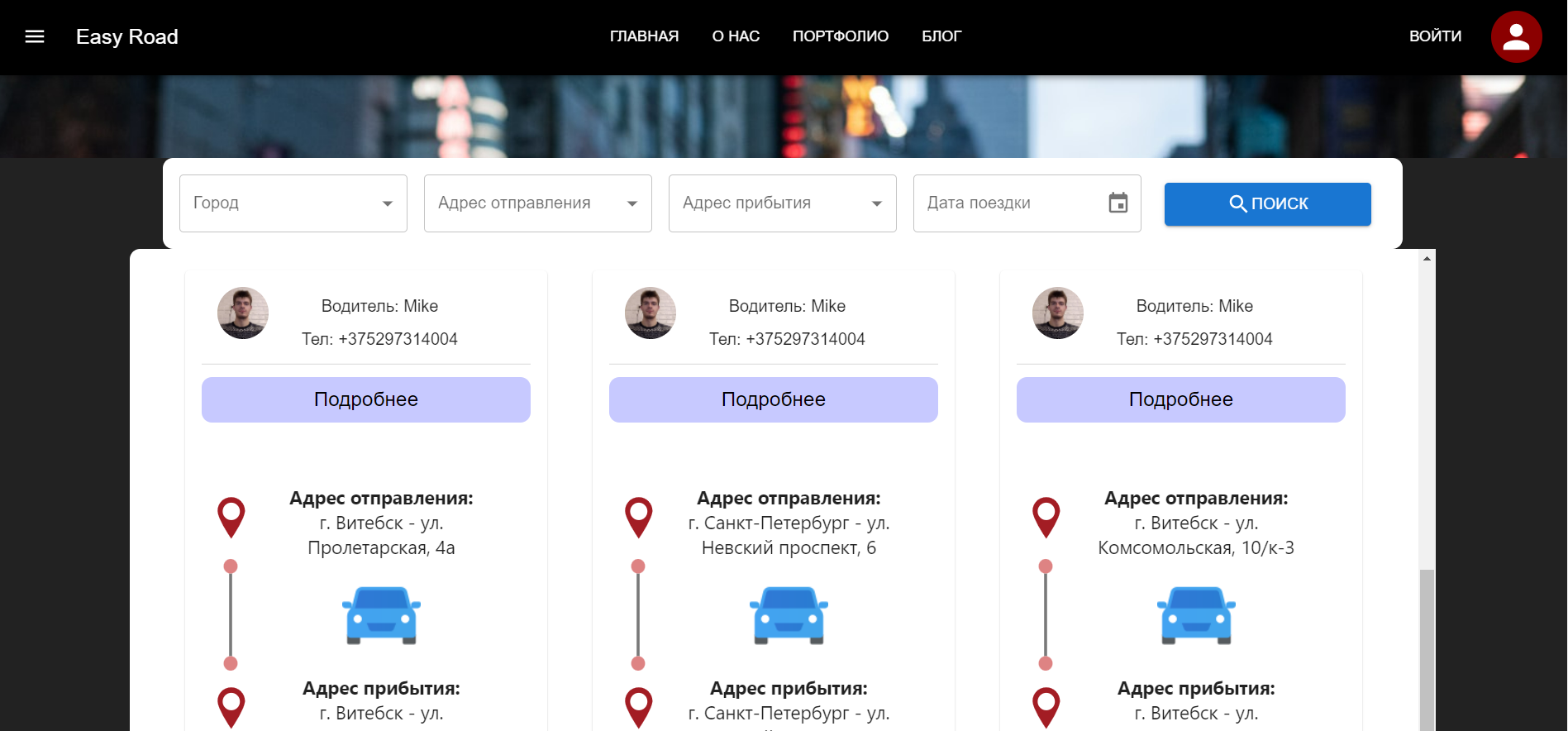


Рисунок 5.4 – Поиск объявлений по фильтрам

### **5.2.2 Описание процесса бронирования поездки**

Для того чтобы забронировать поездку, пользователь сначала должен открыть список объявлений. Далее он проводит поиск по критериям и выбирает нужное объявление. После пользователь может посмотреть подробную информацию о поездке нажав на кнопку «Подробнее» на карточке объявления, где отображается информацию о дате, времени поездки, фото автомобиля, на котором будет осуществляться поездка, а также рейтинг пользователя. На рисунке 5.5 можно увидеть изображение объявления в свернутом и раскрытом вариантах.

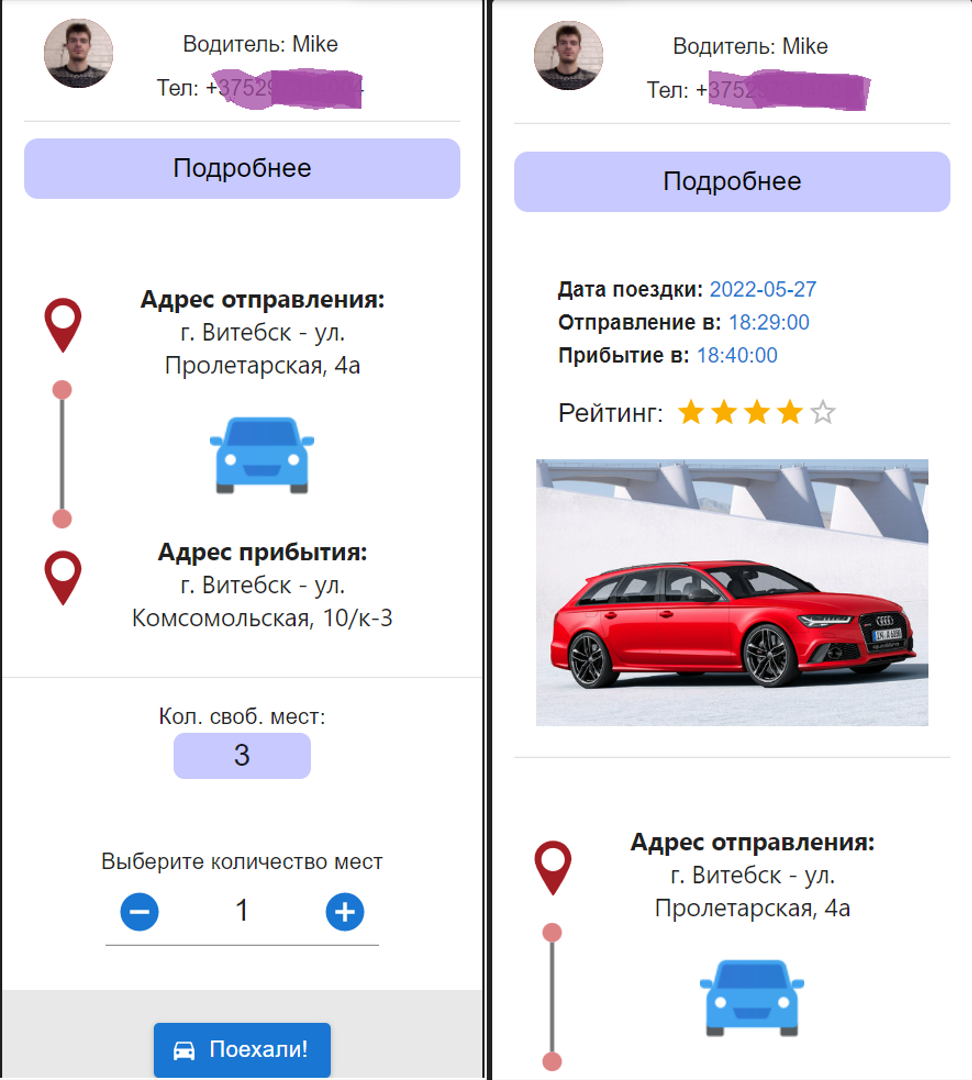


Рисунок 5.5 – Карточка объявления

Далее пользователь может выбрать количество мест и нажать на кнопку «Поехали», после чего создается бронь поездки, информация о которой появляется и у водителя, и у пассажира, осуществившего бронирование места.

Также в данном веб-приложении был реализован поиск объявлений по фильтрам. Поиск объявлений можно вести как по всем полям, так и заполнив только одно из полей. Форма для осуществления поиска находится на двух страницах веб-приложения – на странице списка объявлений и на главной странице.

На рисунке 5.6 представлен результат поиска объявлений.

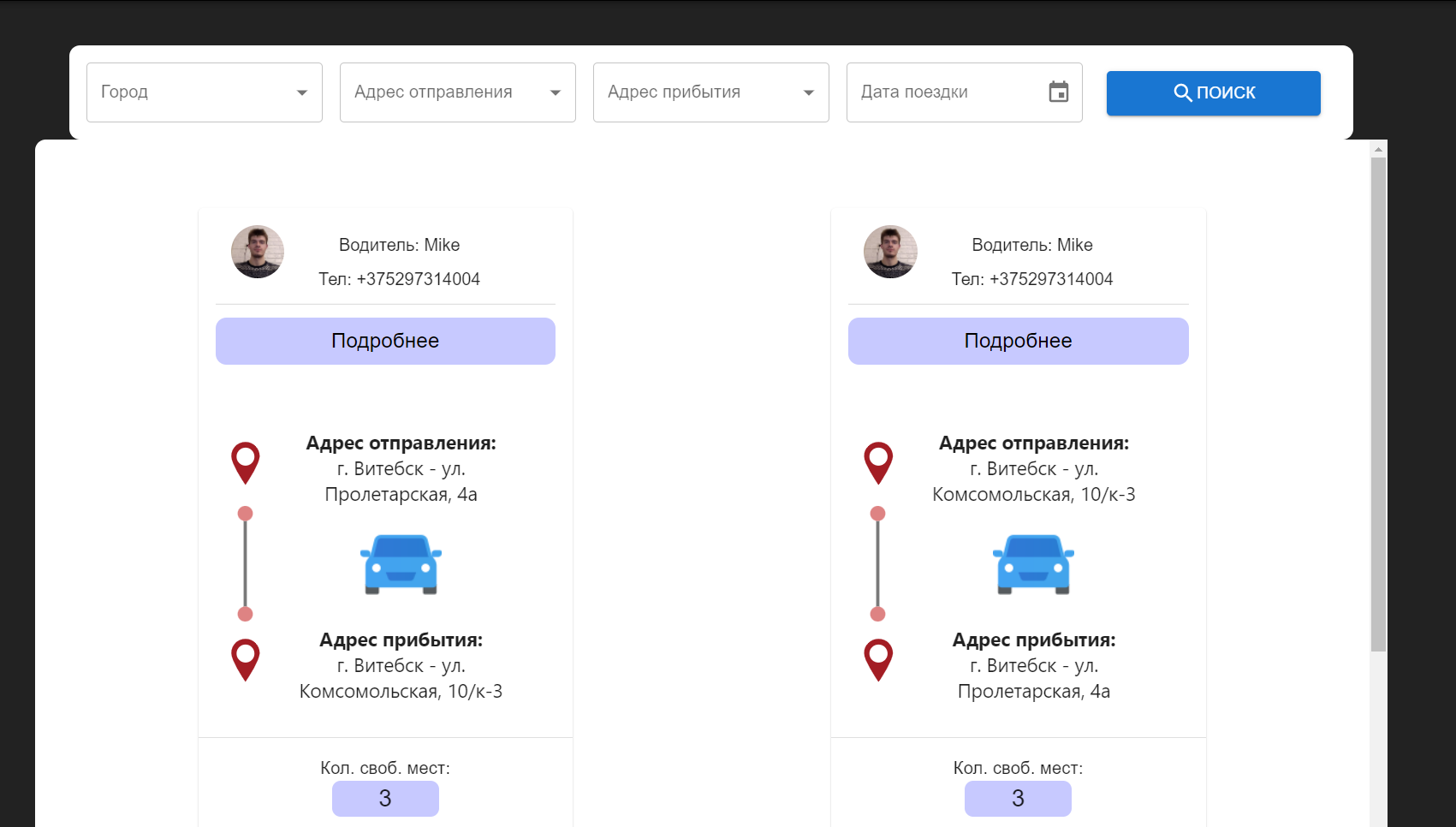


Рисунок 5.6 – Поиск объявлений

### **5.2.3 Описание процесса создания объявления**

Для того, чтобы создать объявление, нужно перейти на страницу создания объявлений, ссылка на которую находится в верхней панели веб-приложения и называется «Предложить поездку». Страница с формой создания объявления представлена на рисунке 5.7.

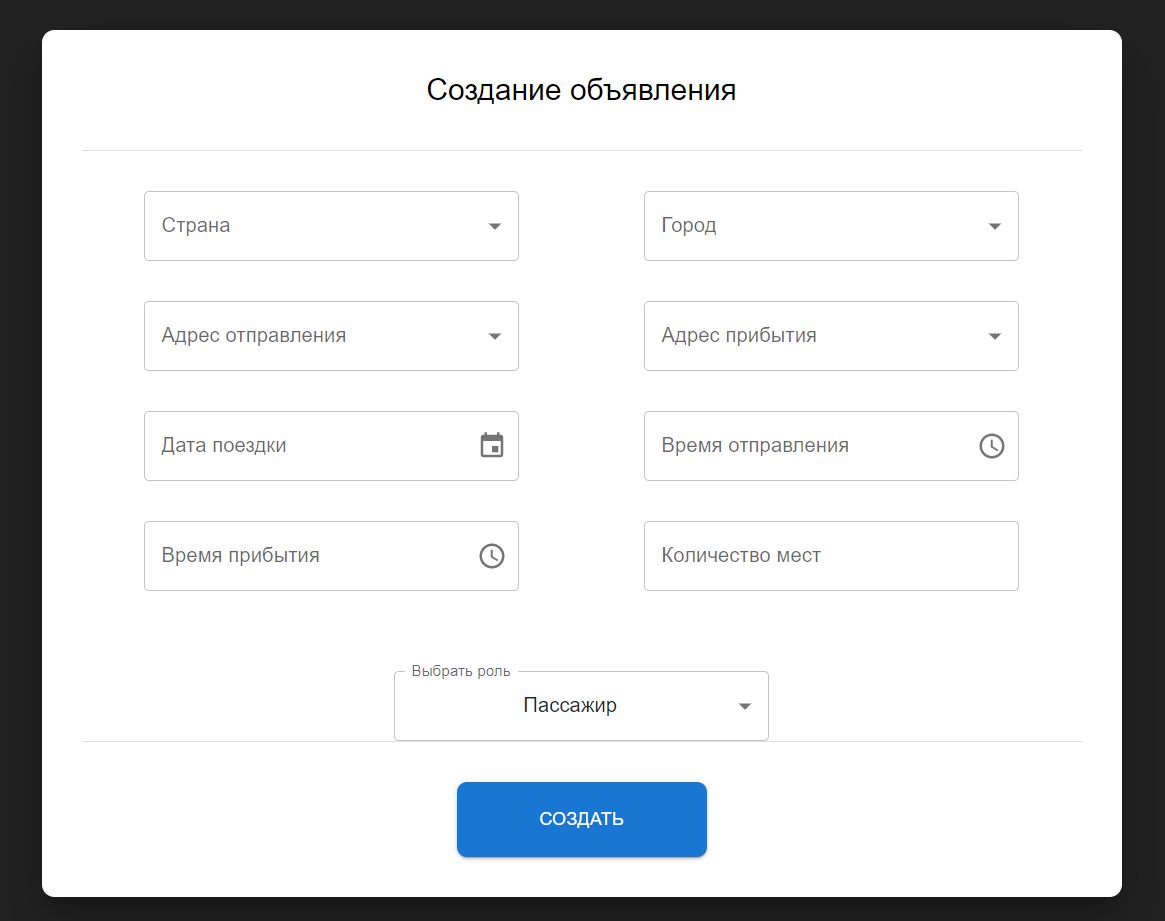


Рисунок 5.7 – Форма создания объявления

Для создания объявления необходимо заполнить все расположенные в форме поля. Чтобы данные можно было ввести корректно, поля были реализованы в виде выпадающего списка для меньшей вероятности совершения ошибок со стороны пользователей.

Также нужно отметить, что для создания объявления в качестве водителя, необходимо добавить в профиль свой автомобил, так как созадвать объявления в качестве водителя можно только при этом условии.

Для создания объявлений в качестве попутчиков достаточно просто заполнить все поля и убедится, что в поле выбора роли для поездки установлена роль пассажира. Роль пассажира будет установлена по умолчанию при переходе на страницу создания объявлений.

В поле «Выбрать роль» пользователь должен выбрать, в качестве кого он создает объявление. Если выбирается роль «Водитель», то появляется еще одно поле для выбора имеющихся у пользователя автомобилей. Пример приведен на рисунке 5.8.

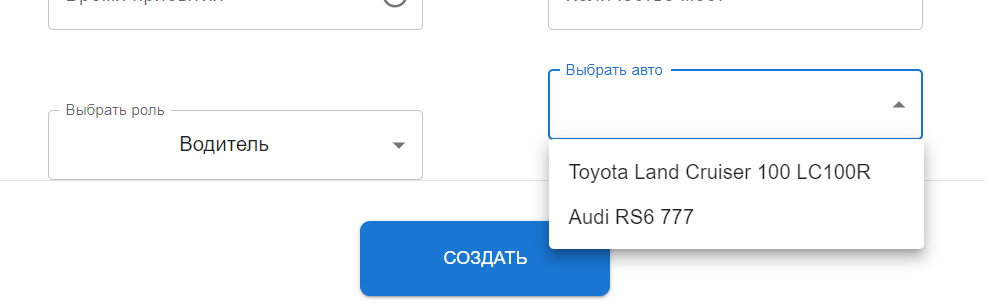


Рисунок 5.8 – Выбор роли при создании объявления

Если пользователь еще не добавил автомобиль в профиль, при выборе роли «Водитель» появится модальное окно с уведомлением, что для начала он должен добавить автомобиль в свой профиль. Содержимое сообщения можно увидеть на рисунке 5.9.

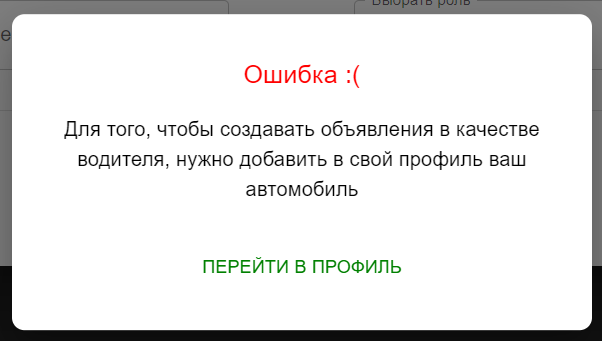


Рисунок 5.9 – Ошибка создания объявления

Также пользователь может просматривать и редактировать информацию своего профиля. Данная возможность представлена на рисунке 5.10.

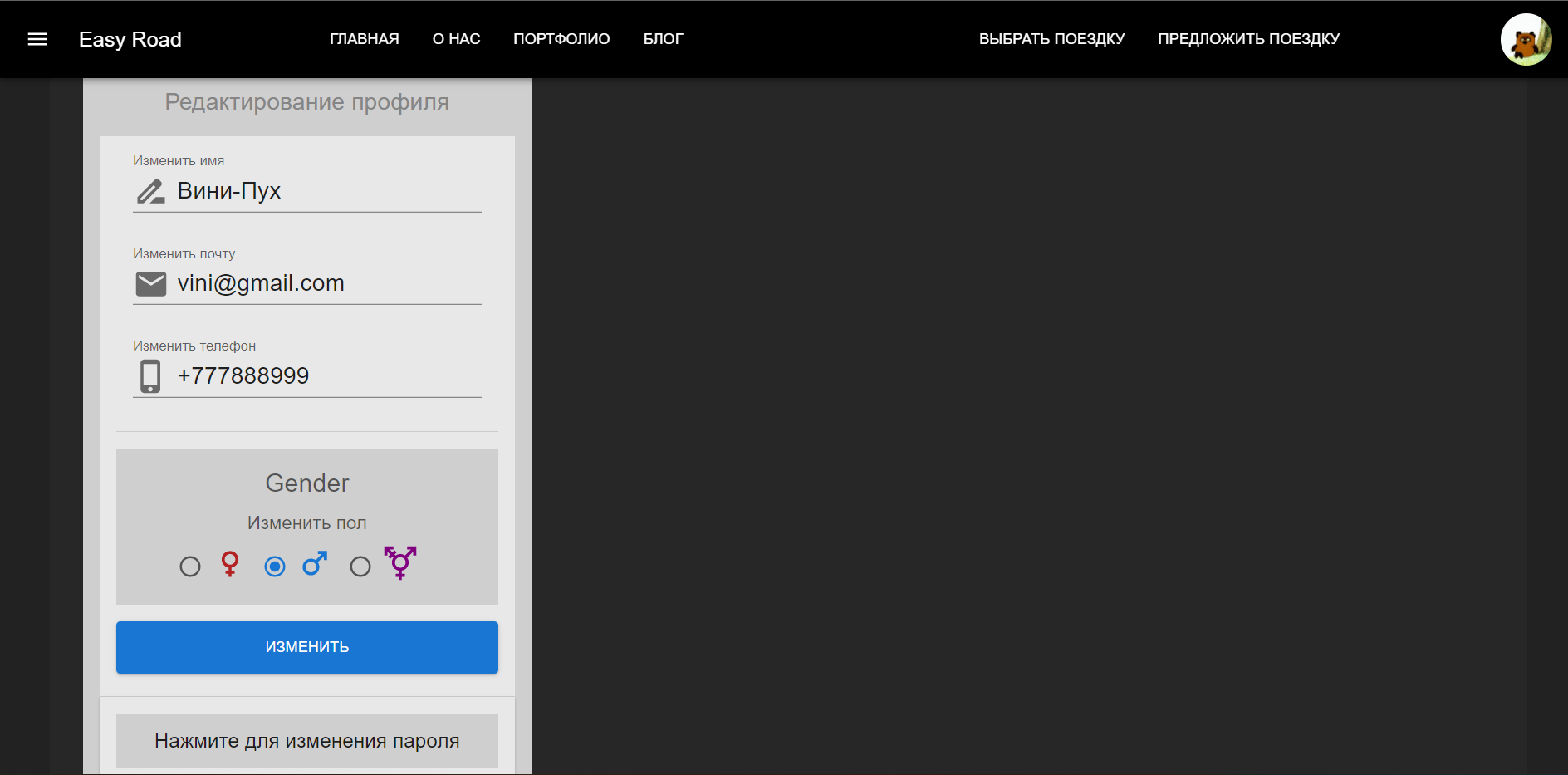


Рисунок 5.10 – Просмотр и редактирование информации профиля

Для того чтобы редактировать информацию профиля, достаточно нажать на кнопку «Редактировать» на странице профиля, после чего поля станут доступны для изменения. Когда все необходимые данные будут изменены, следует нажать на кнопку «Изменить», после чего все обновленные данные сразу же отобразятся на странице профиля.

Также у каждого пользователя, имеющего профиль, есть удалить свой аккаунт. Для этого необходимо перейти на страницу профиля, где под кнопкой «Редактировать» находится кнопка «Удалить профиль», и осуществить нажатие на нее. Изображение с кнопкой для удаления профиля представлено на рисунке 5.11.

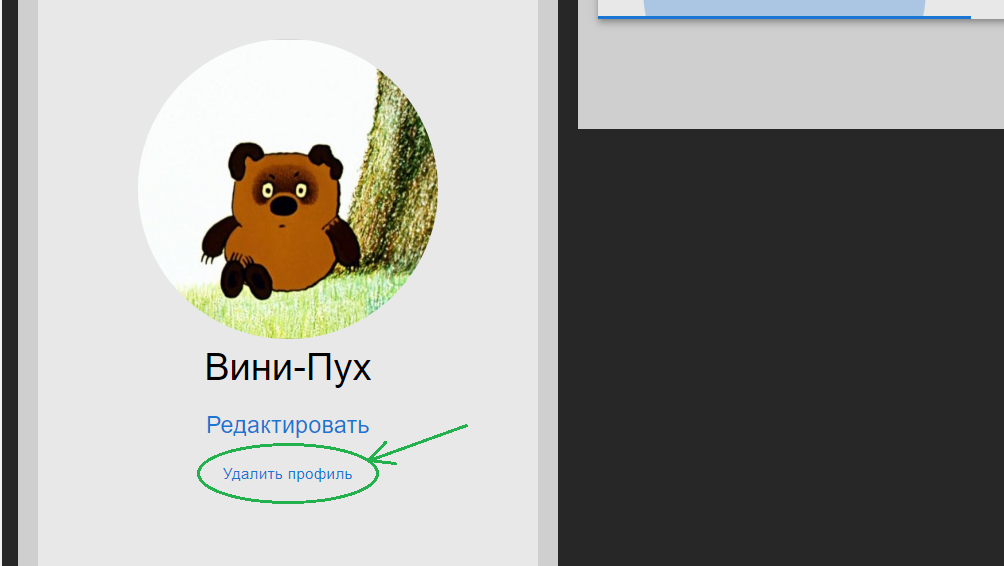


Рисунок 5.11 – Кнопка удаления профиля

**5.2.4 Описание действий администратора**

Как было описано раньше, администратор имеет те же функциональные возможности, что и авторизованный пользователь, но в дополнении к ним, он получает возможность блокировать пользователей, а также просматривать все комментарии в системе, чтобы иметь возможность удалить комментарии с недоброжелательным содержанием.

На рисунке 5.12 представлена вкладка профиля администратора, предназначенная для модерации комментариев пользователей сайта.

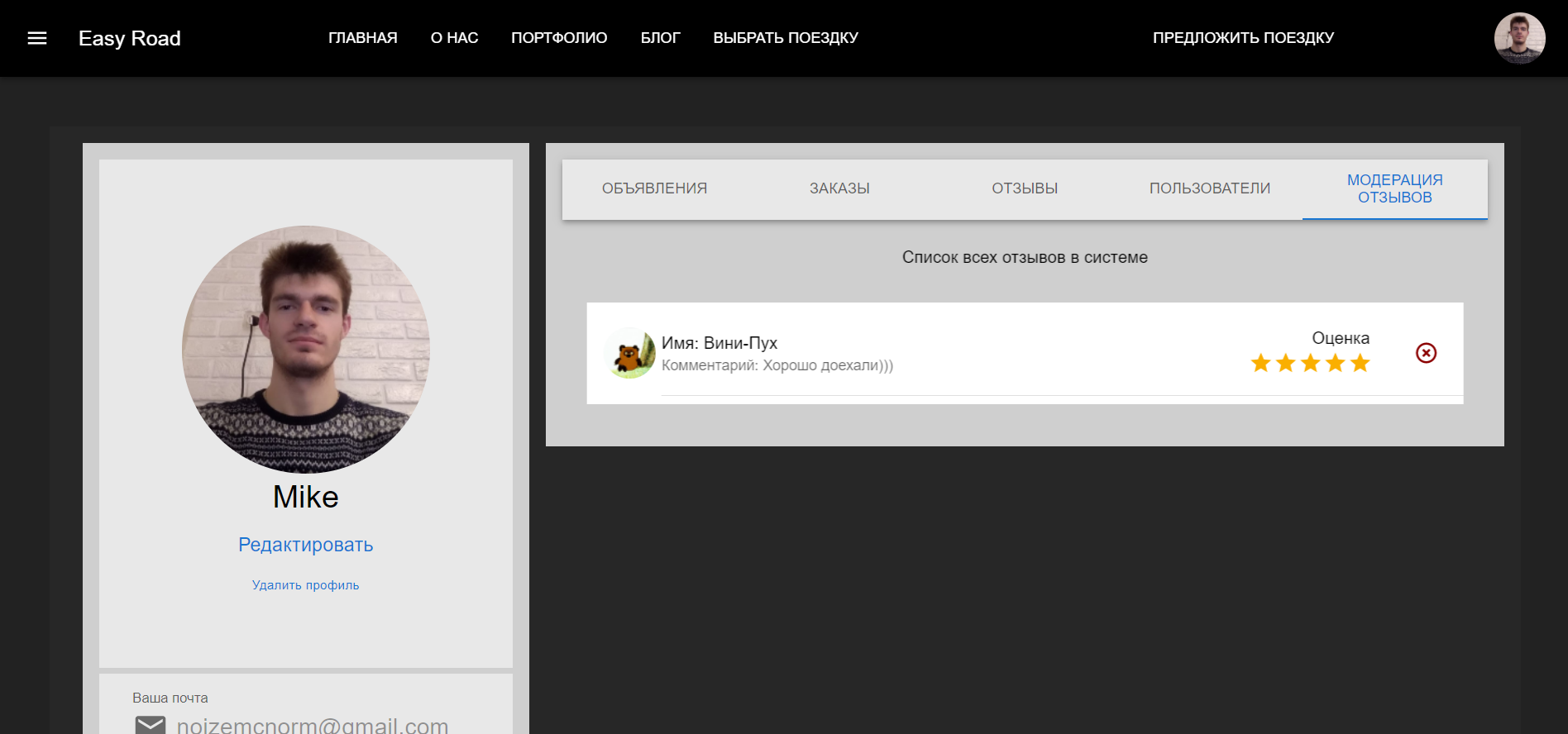
****

Рисунок 5.12 – Вкладка модерации комментариев

## **5.3 Вывод по разделу**

В данном разделе было описано руководство пользователя, где описан весь функционал пользователей всех ролей. Был подробно рассмотрен процесс взаимодействия различных пользователей с системой, рассмотрены возможные пути использования приложения, описан существующий функционал приложения, а также описан процесс запуска проекта с использованием некоторых операционных систем.

В данном разделе был подробно рассмотрен процесс взаимодействия различных пользователей с системой, рассмотрены возможные пути использования приложения, описаны существующий функционал приложения.

# **6 Технико-экономическое обоснование проекта**

Листов

*ФИО*

*Сивак М.Н.*

*Пацей Н.В.*

*Подп.*

*Дата*

Разраб.

Пров.

Утв.

Лит.

Лист

*БГТУ 06.00.ПЗ*

*74417021, 2022*

Н. контр.

*Соболевский А.С.*

1

12

Консульт.

## **6.1 Общая характеристика разрабатываемого программного средства**

Основной целью экономического раздела является экономическое обоснование целесообразности разработки веб-приложения, представленного в дипломном проекте. Для выполнения поставленной цели будет определены экономические показатели, в том числе себестоимость продукта.

Объектом разработки в экономической части является веб-приложение *«Easy Road»*. Приложение разработано для дальнейшей реализации компании, ранее не имевшей приложения подобной направленности. Главная задача данного приложения – упростить поиск попутчиков и водителей для осуществелния поездок.

В рамках данного раздела необходимо определить затраты, произведенные на всех стадиях разработки описанного программного средства. Также необходимо провести расчет экономии основных видов ресурсов в связи с использованием данного программного средства.

## **6.2 Исходные данные для проведения расчетов**

Источниками исходных данных для расчетов выступают действующие законы и нормативно-правовые акты. Исходные данные для расчета стоимости разработки приведены в таблице 6.1.

Таблица 6.1 – Исходные данные для расчета

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателя | Единица измерения | Условные обозначения | Норматив |
| Численность разработчиков | чел. | Чр | 1 |
| Норматив дополнительной заработной платы | % | Ндз | 12 |
| Ставка отчислений в Фонд социальной защиты населения | % | Нфсзн | 34 |
| Ставка отчислений по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний | % | Нбгс | 0,3 |
| Стоимость одного машино-часа | руб. | Смч | 0,06 |
| Норматив прочих затрат | % | Нпз | 15,5 |

*Макарчук М.В*

*Сухорукова И.Г.*

*6 Технико-экономическое обоснование проекта*

Окончание таблицы 6.1

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателя | Единица измерения | Условные обозначения | Норматив |
| Норматив общепроизводственных и общехозяйственных расходов | % | Нобп, обх | 142 |
| Норматив расходов на сопровождение и адаптацию | % | Нрса | 14 |
| Уровень рентабельности | % | Урент | 30 |
| Ставка НДС | % | Нндс | 20 |
| Коэффициент изменения скорости обработки информации | ед. | Кск | 0,6 |
| Тарифная ставка I разряда в организации за месяц | руб. | ОТI | 780 |
| Тарифный коэффициент | ед. | Кт | 3,04 |
| Коэффициент естественных потерь рабочего времени | ед. | Кп | 1,1 |
| Коэффициент премирования | ед. | Кпр | 1 |
| Норматив расхода машинного времени за отладку 100 строк исходного кода | маш. часов | Нмв | 12 |

Данные, указанные в предыдущей таблице, потребуются при дальнейших расчет всех необходимых параметров в ходе экономического анализа веб-приложения.

## **6.3 Методика обоснования цены**

В современных рыночных экономических условиях программное средство выступает преимущественно в виде продукции организаций, представляющей собой функционально завершенные и имеющие товарный вид программное средство, реализуемые покупателям по рыночным отпускным ценам. Разработка проектов программных средств требует разнообразных затрат и зачастую значительных объемов ресурсов (трудовых, материальных, финансовых). В связи с этим, разработка и реализация каждого проекта обосновывается, как технически, так и экономически. Проект стоит разрабатывать, если он дает определенные преимущества по сравнению с известными передовыми аналогами или, по сравнению с существующей практикой. Поэтому, перед разработкой программного средства, специалисты, используя соответствующие методы, находят наиболее рациональные программные решения, обеспечивающее высокий технический уровень программы, дающие существенную экономию ресурсов. У разработчика экономический эффект выступает в виде чистой прибыли от реализации веб-приложения, остающейся в распоряжении организации, а у пользователя – в виде экономии трудовых, материальных и финансовых ресурсов, получаемой за счет следующих показателей:

* снижения трудоемкости расчетов и алгоритмизации программирования и отладки программ;
* снижения расходов на материалы;
* ускорение ввода в эксплуатацию новых систем;
* улучшения показателей основной/дополнительной деятельности в результате использования веб-приложения.
* стоимостная оценка веб-приложения у разработчиков предполагает определение затрат, что включает следующие статьи:
* заработная плата исполнителей – основная и дополнительная;
* отчисления в фонд социальной защиты населения;
* отчисления по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний;
* расходы на материалы и комплектующие;
* расходы на спецоборудование;
* расходы на оплату машинного времени;
* прочие прямые затраты;
* накладные расходы.

На основании затрат рассчитывается себестоимость и отпускная цена разработанного веб-приложения.

### **6.3.1 Определение объема программного средства**

Для общей оценки объема программного средства функции приложения оцениваются с помощью специальной классификационной таблицы, в которой определяется объем каждой функции. На основании информации о функциях, приведенной в таблице 6.3, рассчитывается общий объем программного средства *V*o, условных машинных команд, по формуле 6.1.

|  |  |
| --- | --- |
|  | (6.1) |

где – объем *i*-ой функции ПС, условных машинных команд;

*n* – общее число функций.

Ниже приведена таблица с информацией о затратах рабочего времени на разработку программного средства.

Таблица 6.2 – Содержание и объем функций на разрабатываемое ПС

|  |  |
| --- | --- |
| Содержание работ | Затраты рабочего времени, дней |
| Инициализация клиентской и серверной частей | 1 |
| Разработка моделей базы данных | 2 |
| Заполнение базы данных необходимыми данными | 1 |

Окончание таблицы 6.2

|  |  |
| --- | --- |
| Содержание работ | Затраты рабочего времени, дней |
| Определение архитектуры проекта | 1 |
| Определение библиотек для разработки серверной части | 1 |
| Определение библиотек для разработки клиентской части | 1 |
| Создание логики для аутентификации и регистрации | 3 |
| Создание логики для редактирования профиля | 7 |
| Создание логики для создания объявлений | 2 |
| Создание логики для осуществления бронирования поездок | 3 |
| Создание логики для поиска объявлений | 2 |
| Создание логики для оставления отзывов | 1 |
| Создание логики проверок для всей системы | 9 |
| Тестирование серверной части | 3 |
| Рзработка главной страницы | 3 |
| Разработка страницы с объявлениями | 10 |
| Разработка страницы создания объявления | 2 |
| Разработка страницы профиля пользователя | 17 |
| Тестирование клиентской части | 5 |
| Всего | 74 |

По данным, приведенным в методическом пособии, в разработанном приложении присутствуют функции, приведенные в таблице 6.2.

Таблица 6.3 – Содержание и объем функций на разрабатываемое ПС

|  |  |  |
| --- | --- | --- |
| № функции | Содержание функции | Объем, условных машинных команд |
| 102 | Контроль, предварительная обработка и ввод информации | 450 |
| 204 | Обработка наборов и записей базы данных | 2 670 |
| 208 | Организация поиска и поиск в базе данных | 5 480 |
| 210 | Загрузка базы данных | 2 780 |
| 305 | Обработка файлов | 720 |
| 506 | Обработка ошибочных и сбойных ситуаций | 410 |
| 707 | Графический вывод результатов | 480 |
| Итого | | 12 990 |

Исходя из данных таблицы 6.3, можно рассчитать объем программного сред­ства, разработанного в процессе дипломного проектирования:

450 + 720 + 480 + 2 780 + 5 480 + 410 + 2 670 = 12 990 (маш. команд).

В связи с достаточно быстрым изменением вычислительной техники рекомендуется определить скорректированный объем функций , условных машинных команд.

Вычисляется скорректированный объём функций по формуле 6.2.

|  |  |
| --- | --- |
| , | (6.2) |

где – коэффициент изменения скорости обработки информации.

Коэффициент изменения скорости обработки информации равен 0,6 – по исходным данным, приведенным в таблице 6.1.

*=* 12 990 · 0,6 = 7 794 (маш. команд).

Таким образом, скорректированный объем функций условных машинных командравен 7 794 условных машинных команд.

### **6.3.2 Расчет трудоемкости выполненной работы**

Общая трудоемкость рассчитывается по формуле 6.3.

|  |  |
| --- | --- |
|  | (6.3) |

где – нормативная трудоемкость программного средства, чел-дней;

– дополнительный коэффициент сложности, ед.

Поскольку приложение обеспечивает хранение, ведение и поиск данных в сложных структурах, то дополнительный коэффициент сложности = 0,07, а  = 126 исходя из таблицы в приложении Е.

чел-дней.

### **6.3.3 Расчет основной заработной платы**

Для определения величины основной заработной платы, было проведено исследование величин заработных плат для специалистов в сфере веб-программирования на языке программирования *JavaScript,* а также с использованием платформы *Node.js* и библиотеки *React.js*. Источником данных служили открытые веб-порталы, различные форумы, официальная отчетность, а также общий средний уровень заработка в сфере информационных технологий. Итогом изучения стала информация о том, что средняя месячная заработная плата для позиций *junior/middle* составляет около 1 700 рублей.

Основная заработная плата рассчитывается по формуле 6.4.

|  |  |
| --- | --- |
|  | (6.4) |

где – основная заработная плата, руб.;

– время разработки, месяцев;

– количество разработчиков, человек;

– средняя месячная заработная плата.

Время разработки будет рассчитываться по формуле 6.5.

|  |  |
| --- | --- |
|  | (6.5) |

где – время разработки, месяцев;

– количество дней разработки, дней.

Проект разрабатывался одним человеком на протяжении трех месяцев. Расчеты основной заработной платы и времени разработки представлены ниже:

= 74 / 21 = 3,5 месяцев,

= 3,5 ⋅ 1 ⋅ 1 700 = 5 950 руб.

Далее основная заработная плата, рассчитанная по указанной выше методике, будет использоваться при других расчетах.

### **6.3.4 Расчет дополнительной заработной платы**

Законодательство о труде предусматривает наличие выплат, определяющиеся по нормативу в процентах к основной заработной плате по формуле 6.6. 

|  |  |
| --- | --- |
|  | (6.6) |

где – основная заработная плата, руб.;

– норматив дополнительной заработной платы, %.

Исходя из основной заработной платы, а также норматива дополнительной заработной платы, можно рассчитать сумму дополнительной заработной платы:

 руб.

Дополнительная заработная плата включает выплаты, предусмотренные законодательством о труде, и определяется по нормативу в процентах по отношению к основной заработной плате.

### **6.3.4 Расчет отчислений в фонд социальной защиты населения**

Отчисления в Фонд социальной защиты населения определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей и вычисляются по формуле 6.7.

|  |  |
| --- | --- |
|  | (6.7) |

где – основная заработная плата, руб.;

– дополнительная заработная плата, тыс. руб.;

– норматив отчислений в Фонд социальной защиты населения, %.

Отчисления в БРУСП «Белгосстрах» вычисляются по формуле 6.8.

|  |  |
| --- | --- |
|  | (6.8) |

Исходя из полученных данных высчитаем отчисления в Фонд социальной защиты наcеления (ФСЗН) и в БРУСП «Белгосстрах»:

руб.,

руб.

Полученные выше значения будут использованы при подсчете заработной платы программиста с отчислениями.

### **6.3.5 Расчет расходов на материалы**

Расходы на материалы определяются с учетом действующих нормативов. Сумма затрат на материалы рассчитывается по формуле 6.9.

|  |  |
| --- | --- |
|  | (6.9) |

где – норма расхода материалов в расчете на 100 строк исходного кода программного средства, руб;

– скорректированный объем функций, условных машинных команд.

Так как программное средство решает задачи поиска в базе данных и обработки строк, сопоставимые по ресурсоемкости с задачами расчетного характера, то  = 0,46 руб. – по данным, приведенным в приложении методического пособия для программных средств, решающих задачи расчетного характера.

= 0,46 ⋅ 7 794 / 100 = 35,85 руб.

По данной статье отражаются расходы на бумагу, диски, флэш-карты и другие материалы, необходимые для разработки ПС. Нормы расхода материалов в суммарном выражении определяются в расчете на 100 строк исходного кода.

### **6.3.6 Расчет расходов на оплату машинного времени**

Сумма расходов на оплату машинного времени определяется как произведение стоимости одного машино-часа на уточненный объем программного средства и на норматив расхода машинного времени на отладку ста строк исходного кода и рассчитывается по формуле 6.10.

|  |  |
| --- | --- |
|  | (6.10) |

Учитывая, что норматив машинного времени на отладку ста строк исходного кода равен 0,05, можно определить сумму расходов на оплату машинного времени:

= 0,06 ⋅ 7 794 ⋅ 12 / 100 = 56,12 руб.

Расходы на оплату машинного времени, необходимого для разработки и отладки разрабатываемого веб-приложения,равны 56,12 рублей.

### **6.3.7 Расчет прочих прямых затрат**

Расходы на конкретное программное средство включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы.

Такие расходы рассчитываются по формуле 6.11.

|  |  |
| --- | --- |
|  | (6.11) |

где – норматив прочих затрат в целом по организации, %.

= 5 950 ⋅ 15,5 /100 = 922,25 руб.

Таким образом, прочие прямые затраты при разработке веб-приложениясоставляют 922,25 рублей.

### **6.3.8 Расчет общепроизводственных и общехозяйственных расходов**

Сумма общепроизводственных, общехозяйственных расходов – произведение основной заработной платы исполнителей на конкретное программное средство на норматив накладных расходов в целом по организации по формуле 6.12.

|  |  |
| --- | --- |
|  | (6.12) |

Все данные необходимые для вычисления есть, поэтому можно определить сумму накладных расходов:

= ⋅ 142 / 100 =  руб.

Получим, что накладные расходы, необходимые для разработки веб-приложения,равны 8 449 рублей.

### **6.3.9 Расчет суммы расходов на разработку программного средства**

Сумма расходов на разработку программного средства определяется как сумма основной и дополнительной заработных плат исполнителей на конкретное программное средство, отчислений на социальные нужды, расходов на материалы, расходов на оплату машинного времени, суммы прочих затрат и суммы накладных расходов.

|  |  |
| --- | --- |
|  | (6.13) |

= 5 950 + 714 + 2 265,76 + 19,99 + 35,85 + 56,12 + 922,25 + 8 449 = 18 412,17 руб.

Таким образом, получим сумму всех расходов на разработку веб-приложенияравной 18 412,17 рублей.

### **6.3.10 Расходы на сопровождение и адаптацию**

Сумма расходов на сопровождение и адаптацию программного средства определяется как произведение суммы расходов на разработки программного средства на норматив расходов на сопровождение и адаптацию разработанного программного средства .

Сумма расходов на сопровождение и адаптацию программного средства вычисляется по формуле 6.14.

|  |  |
| --- | --- |
|  | (6.14) |

где – сумма расходов на сопровождение и адаптацию ПС, руб.;

– общая сумма расходов на разработку ПС, руб.;

– норматив расходов на сопровождение и адаптацию, %.

Основываясь на исходные данные, расположенные в таблице 6.1 и формулу 6.14, норматив расходов на сопровождение и адаптацию Нрса равен 14%. Сумма расходов на сопровождение и адаптацию ПС составляет:

руб.

Получим, что расходы, необходимые на сопровождение и адаптацию веб-приложения, равны рублей.

### **6.3.11 Расчет общей суммы расходов**

Общая сумма расходов на разработку с затратами на сопровождение и адаптацию определяется по формуле 6.15.

|  |  |
| --- | --- |
| , | (6.15) |

= 18 412,17 + 2 577,7 = 20 989,87 руб.

Получим, что общая сумма расходов веб-приложения, выполняющего роль магазина компьютерной техники, равна 20 989,87 рублей.

### **6.3.12 Определение цены, оценка эффективности**

Отпускная цена рассчитывается на основании цены разработчика, которая формируется на основе показателя рентабельности продукции. Рентабельность и прибыль по создаваемому программному средству определяются исходя из результатов анализа рыночных условий, переговоров с заказчиком (потребителем) и согласования с ним отпускной цены, включающей дополнительно налог на добавленную стоимость.

Прибыль от реализации программного средства вычисляется по формуле 6.16.

|  |  |
| --- | --- |
|  | (6.16) |

где – уровень рентабельности, %;

– полная себестоимость программного средства, руб.

Цена разработчика программного средства без налогов находится по следующей формуле:

|  |  |
| --- | --- |
|  | (6.17) |

Сумма налога на добавленную стоимость рассчитывается из соотношения:

|  |  |
| --- | --- |
|  | (6.18) |

где – цена разработчика программного средства, руб.;

– ставка НДС, %.

Планируемая отпускная цена с НДС вычисляется по следующей формуле:

|  |  |
| --- | --- |
| . | (6.19) |

Исходя из вышеописанных данных рассчитаем прибыль от реализации программного средства, цену разработчика без налогов, сумму налогов на добавленную стоимость, а также планируемую отпускную цену с учетом НДС.

= 20 989,87 / 100 = 6 296,96 руб.,

= 20 989,87 + 6 296,96 = 27 286,83 руб.,

НДС = 27 286,83  / 100  = 5 457,37 руб.,

= 27 286,83 + 5 457,37 = 32 744,2 руб.

Для анализа стоимости разработки необходимо рассмотреть стоимость разработки приложений, обладающих аналогичным функционалом. Для этого я воспользовался тремя сайтами услуг по разработке программного обеспечения.

Первым является сайт *Estimatemyapp.com*. По итогам расчета, при выборе всех необходимых параметров, можно увидеть, что на разработку аналогичного продукта в среднем будет затрачено 67 дней разработки, а стоимость составит 36 900$. Изображение результатов работы ресурса представлена на рисунке 6.1.

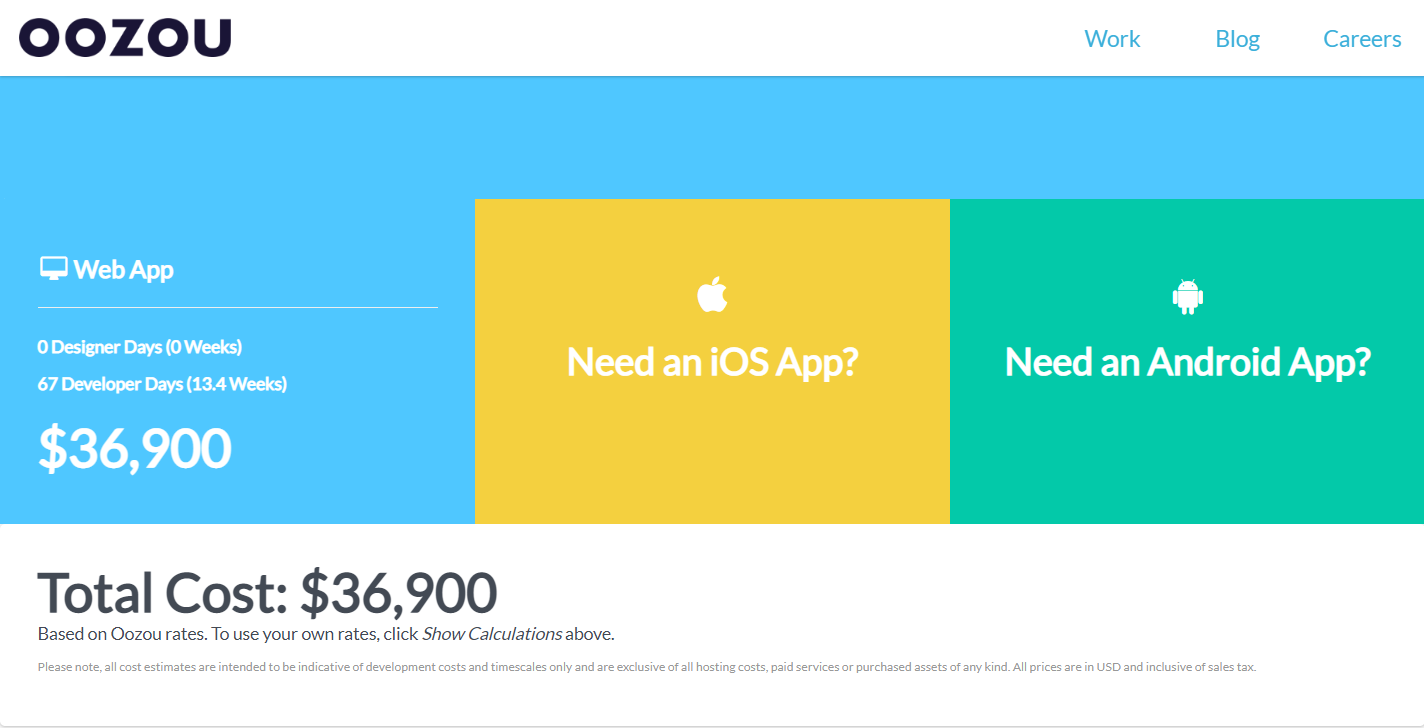


Рисунок 6.1 – Результат работы сайта *Estimatemyapp.com*

Вторым ресурсом является сайт *Thebestapp.ru*. Этот ресурс работает аналогично предыдущему, но итоговая цена будет представлена в российских рублях. На рисунке 6.2 представлен результат работы сайта-агрегатора.



Рисунок 6.2 – Результат работы сайта *Thebestapp.ru*

Исходя из результатов расчета стоимости разработки веб-приложения на данном ресурсе, примерная сумма похожего продукта обойдется в 2 005 000 российских рублей, что в пересчете на белорусские рубли составляет 80 055 рублей.

Последним ресурсом является сайт *Appcraft.pro*. Выбрав все необходимые компоненты системы, стоимость полной версии приложения составляет 1 050 000 российских рублей, что в пересчете на белорусскую валюту составляет 41 924 рублей. Ниже на рисунке 6.3 приведен результат работы сайта.

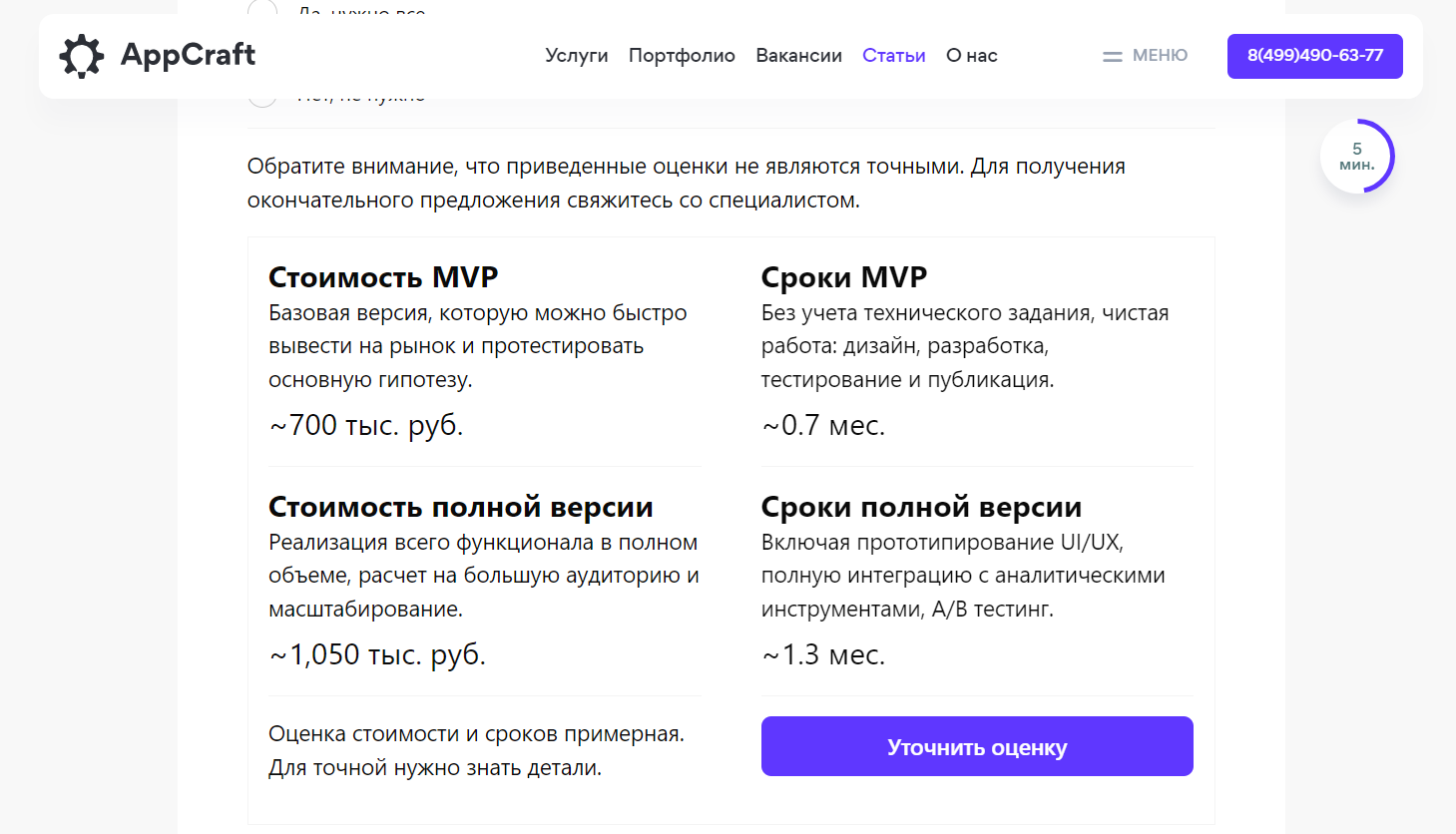


Рисунок 6.3 – Результат работы сайта *Appcraft.pro*

Как видно из результатов расчетов итоговой стоимости разработки продукта на разных платформах, стоимость на первых двух сайтах приблизительно одинакова, но на третьем ресурсе стоимость значительно отличается, что может быть обусловленно рынком, для которого данные платформы предлают свои услуги.

## **6.4 Вывод по разделу**

В таблице 6.4 представлены результаты расчетов для основных показателей данного раздела.

Таблица 6.4 – Результаты расчетов

|  |  |
| --- | --- |
| Наименование показателя | Значение |
| Время разработки, мес. | 3,5 |
| Количество программистов, чел. | 1 |
| Основная заработная плата, руб. | 5 950 |
| Сумма отчислений на социальные цели, руб. | 2 265,76 |
| Сумма расходов на материалы, руб. | 35,85 |
| Расходы на оплату машинного времени, руб. | 56,12 |
| Прочие прямые затраты, руб. | 922,25 |
| Общепроизводственные и общехозяйственные расходы, руб. | 8 449 |
| Сумма расходов на разработку программного средства, руб. | 18 412,17 |
| Расходы на сопровождение и адаптацию ПС, руб. | 2 577,7 |
| Обща сумма расходов, руб. | 20 989,87 |
| Сумма налога на добавленную стоимость, руб. | 5 457,37 |
| Отпускная цена с НДС, руб. | 32 744,2 |
| Прибыль от реализации, руб. | 6 296,96 |

Исходя из совокупности всех вышеперечисленных факторов, определим, что разработка данного веб-приложения несет положительный экономический эффект.

Необходимость разработки программного средства обусловлена востребованностью людей в совершении онлайн-бронирования поездок. Для того, чтобы сделать этот процесс более комфортным, безопасным и быстрым, был проведен анализ аналогов, выявлены лучшие и худшие их стороны, благодаря которым становиться понятно, на что больше стоит сделать упор при разработке подобного продукта.

Разработка веб-приложения осуществлялась с целью наибольшего удовлетворения пользователей сервисов данного сегмента. Использование приложения позволяет упростить бронирование поездок и сделать этот процесс как можно комфортнее, безопаснее и быстрее.

Разработка программного средства, осуществляемая одним программистом в течении трех с половиной месяцев, при заданных условиях обойдется компании в 32 744,2 руб. Реализации данного программного средства принесет прибыль компании на сумму 6 296,96 рублей. Цена за разработанный продукт соответствует рыночным ценам за продукцию, обладающую аналогичным функционалом, и является более низкой чем у вышеперечисленных.

# **ЗАКЛЮЧЕНИЕ**

*ФИО*

*Подпись*

*Дата*

Лист

1

66

*БГТУ 00.00.ПЗ*

Разраб.

*Сивак М.Н.*

Провер.

.

*Сухорукова И.Г.*

Консульт.

*Сухорукова И.Г.*

Н. контр.

*Макарчук М.В*

Утв.

*Пацей Н.В.*

*я*

Заключение

Лит.

Листов

1

74417021, 2022

В ходе выполнения дипломного проектирования была проанализирована предметная область, а также рассмотрены существующие прототипы, выявлены их преимущества и недостатки, которые были учтены на этапе проектирования и разработки данного программного средства.

Также была спроектирована база данных, клиентская и серверная части приложения. Описан принцип взаимодействия между компонентами приложения, а также архитектура частей приложения. Особенности реализации *Node.js* сервера и взаимодействие с клиентом.

Были описаны принципы и подходы, применяемые для взаимодействия серверной части приложения с базой данных.

Было проведено тестирование основных частей программного средства, произведен обзор возможных ошибок и их поиск при работе пользователей и проведены мероприятия по их устранению.

Выполнено технико-экономическое обоснование разработки данного программного средства, рассчитана экономическая эффективность использования данного программного средства.

На основании полученных данных можно сделать вывод, что разработанный проект является экономически выгодным.

Результатом проделанной работы стало веб-приложение для поиска попутчиков.

Разработанным веб-приложением могут пользоваться несколько типов пользователей: администратор, пользователь и гость. Каждый тип пользователей обладает определенным функционалом.

Поставленная цель была выполнена в полном объеме, веб-приложение было разработано с учетом всех требований, работоспособность подтверждена тестированием веб-приложения.

Веб-приложение соответствует цели дипломного проекта, имеет достаточный функционал, а также реализует все поставленные перед ним задачи.

# **Список использованных источников**

*ФИО*

*Подпись*

*Дата*

Лист

1

67

*БГТУ 00.00.ПЗ*

Разраб.

*Сивак М.Н.*

Провер.

.

*Сухорукова И.Г.*

Консульт.

*Сухорукова И.Г.*

Н. контр.

*Макарчук М.В*

Утв.

*Пацей Н.В.*

*я*

Список использованных источников

Лит.

Листов

1

74417021, 2022

1 Веб-приложение [Электронный ресурс] / Википедия. – Режим доступа: <https://ru.wikipedia.org/wiki/web>. – Дата доступа: 05.04.2022

2 Веб-сайт для заказа поездок и поиска попутчиков онлайн. [Электронный ресурс]. – Режим доступа: <https://www.blablacar.ru/>. – Дата доступа: 06.04.2022

3 Веб-сайт для бронирование маршруток между городами Беларуси. [Электронный ресурс]. – Режим доступа: <https://smilebus.by/>. – Дата доступа: 06.04.2022

4 Веб-сайт для поиска попутчиков и водителей. [Электронный ресурс. – Режим доступа: <http://www.dovezu.ru/>. – Дата доступа: 06.04.2022

5  Протокол *HTTP* [Электронный ресурс] / Википедия. – Режим доступа: <https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol>. – Дата доступа: 10.04.2022

6 Официальный сайт документации *Visual Studio Code* [Электронный ресурс]. – Режим доступа: <https://code.visualstudio.com>. – Дата доступа: 17.03.2022

7 Официальный сайт документации *JavaScript* [Электронный ресурс] / developer.mozilla.org. – Режим доступа: <https://developer.mozilla.org/>. – Дата доступа: 12.03.2022

8 Официальный сайт документации *Node.js* [Электронный ресурс] / nodejs.org/en/. – Режим доступа: <https://nodejs.org/en/>. – Дата доступа: 26.03.2022

9 Официальный сайт документации *Express.js* [Электронный ресурс]. – Режим доступа: <https://expressjs.com/>. – Дата доступа: 02.04.2022

10 Официальный сайт документации *Sequelize* [Электронный ресурс] . – Режим доступа: <https://sequelize.org/>. – Дата доступа: 05.04.2022

11 *ReactJS* [Электронный ресурс] / reactjs.org. – Режим доступа: <https://reactjs.org/>. – Дата доступа: 28.03.2022

12 *Postgresql* [Электронный ресурс] / Википедия. – Режим доступа: <https://ru.wikipedia.org/wiki/Postgresql>. – Дата доступа: 18.03.2022

13 Официальная документация *MUI* [Электронный ресурс]. – Режим доступа: <https://mui.com/>. – Дата доступа: 24.03.2022

14  *REST* [Электронный ресурс] / Википедия. – Режим доступа: <https://ru.wikipedia.org/wiki/REST>. – Дата доступа: 15.03.2022

15 *JSON Web Token* [Электронный ресурс] / Википедия. – Режим доступа: <https://en.wikipedia.org/wiki/JSON_Web_Token>. – Дата доступа: 16.03.2022

ПРИЛОЖЕНИЕ А  
Структурная схема базы данных

ПРИЛОЖЕНИЕ Б  
Диаграмма вариантов использования

ПРИЛОЖЕНИЕ В  
Блок-схема алгоритмма создания объявления

ПРИЛОЖЕНИЕ Г  
Блок-схема алгоритма создания заказа

ПРИЛОЖЕНИЕ Д  
Диаграмма последовательности заказа

ПРИЛОЖЕНИЕ Е  
Скриншот работы веб-приложения

ПРИЛОЖЕНИЕ Ж  
Листинг контроллера пользователя

|  |
| --- |
| const db = require('../models');  const jwt = require("jsonwebtoken");  const config = require("../config/auth.config.js");  const Users = db.Users;  const Cars = db.Cars;  const Ads = db.Ads;  const Roles = db.Roles;  const TravelHistory = db.TravelHistory;  const Orders = db.Orders;  const bcrypt = require('bcryptjs');  exports.getProfileInfo = async (req, res) => {  const userId = req.headers['x-user-id'];  console.log('USER ID: ' + userId);  try {  const userCar = await  Cars  .findAll({  where: { userId: userId }  })  await Users  .findOne(  {  where: {  id: userId,  }  },  ).then(user => {  res.status(200).send(  {  data:  {  user: user,  cars: userCar  }  }  )  })  } catch (e) {  res.status(500).json({  message: 'Something went wrong, try again: ' + e.message  })  }  }  exports.deleteCar = async (req, res) => {  const carId = req.params.id;  await Cars.destroy({ where: { id: carId } })  .then(() => {  res.status(202).send("Car successfully deleted :)")  })  .catch((err) => {  throw err.message  })  }  exports.getCars = async (req, res) => {  const userId = req.headers['x-user-id']  await Cars  .findAll({  where: { userId: userId }  })  .then((cars) => {  res.status(200).send(cars)  })  .catch((err) => {  throw err.message  })  }  exports.addCar = async (req, res) => {  const userId = req.body.userId;  const carBrand = req.body.carBrand;  const carModel = req.body.carModel;  const carNumber = req.body.carNumber;  const carPhoto = req.body.carPhotoLink  await Cars  .create({  carBrand: carBrand,  carModel: carModel,  carNumber: carNumber,  carPhotoLink: carPhoto,  userId: userId  })  .then((car) => {  res.send(car)  })  .catch((e) => {  res.status(500).json({  mesage: 'Something went wrong, try again: ' + e.mesage  })  })  }  exports.updateCar = async (req, res) => {  const carId = req.params.id;  const carBrand = req.body.carBrand;  const carModel = req.body.carModel;  const carNumber = req.body.carNumber;  const carPhoto = req.body.carPhotoLink;  let updateData = {  carBrand: carBrand,  carModel: carModel,  carNumber: carNumber,  carPhotoLink: carPhoto,  }  Object.keys(updateData).map((key, index) => {  if(!updateData[key]) {delete updateData[key]}  })  await Cars  .update(  updateData,  {  where: { id: carId }  }  )  .then((car) => {  res.send(car)  })  .catch((e) => {  res.status(500).json({  mesage: 'Something went wrong, try again: ' + e.mesage  })  })  }  exports.updateProfileInfo = async (req, res) => {  const userId = req.body.userId;  const userEmail = req.body.userEmail;  const userPhone = req.body.number;  const userName = req.body.name;  const userGender = req.body.gender;  const user = await Users.findOne({  where: {  id: userId  }  })  if (!user) {  return res.status(404).send({ message: "User Not found." });  }  await Users.update({  userName: userName,  userEmail: userEmail,  userPhone: userPhone,  gender: userGender,  }, {  where: {  id: userId  }  })  .then((user) => {  res.send(user)  })  .catch(err => {  res.status(500).send({ message: err.message });  });  }  exports.changePassword = async (req, res) => {  const userId = req.body.userId;  const oldPassword = req.body.oldPassword;  const newPassword = req.body.newPassword;  if (!oldPassword || !newPassword) {  res.send('Заполните все поля!')  }  if (oldPassword.length < 4 || newPassword.length < 4) {  res.send('Пароль должен иметь не менее 4 символов!');  }  const user = await Users.findOne({  where: {  id: userId  }  })  if (!user) {  return res.status(404).send({ message: "User Not found." });  }  const passwordIsValid = bcrypt.compareSync(  oldPassword,  user.userPassword  );  if (!passwordIsValid) {  return res.status(401).send({  accessToken: null,  message: "Invalid Password!"  });  }  await Users.update({  userPassword: bcrypt.hashSync(newPassword, 8)  }, {  where: {  id: userId  }  })  .then((user) => {  res.send(user)  })  .catch(err => {  res.status(500).send({ message: err.message });  });  }  exports.deleteProfile = async (req, res) => {  let id = req.params.id;  try {  Users  .destroy(  {  where: {  id: id  }  }  )  .then(() => {  res.status(202).json({ message: 'user deleted successfully!' });  })  }  catch (e) {  res.status(500).json({  mesage: 'Something went wrong, try again: ' + e.mesage  })  }  } |

ПРИЛОЖЕНИЕ Д  
Листинг *React*-компонента *NewAd*

|  |
| --- |
| import { Container, Box, Typography, FormControl, TextField, Button, Grid, Modal, Divider, Select, InputLabel, MenuItem } from "@material-ui/core";  import LocalizationProvider from '@mui/lab/LocalizationProvider';  import Stack from '@mui/material/Stack';  import DesktopDatePicker from '@mui/lab/DesktopDatePicker';  import TimePicker from '@mui/lab/TimePicker';  import AdapterDateFns from '@mui/lab/AdapterDateFns';  import { useEffect, useState } from 'react';  import { getHours, set } from "date-fns";  import { ru } from "date-fns/locale";  import axios from "axios";  import { useNavigate } from 'react-router-dom'  let stylesNewAd = {  mainContainer: {  padding: '5em 0',  },  newAddBox: {  maxWidth: 800,  minWidth: 320,  margin: '0 auto',  bgcolor: 'background.paper',  border: 'none',  boxShadow: 24,  borderRadius: '10px',  p: 4,  textAlign: 'center',  }  }  const modalError = {  position: 'absolute',  top: '50%',  left: '50%',  transform: 'translate(-50%, -50%)',  width: 400,  bgcolor: 'background.paper',  border: 'none',  borderRadius: '10px',  boxShadow: 24,  p: 4,  };  export default function NewAd() {  const navigate = useNavigate();  const goToProfile = () => {  navigate('/profile')  }  const [open, setOpen] = useState(false);  const handleOpen = () => setOpen(true);  const handleClose = () => setOpen(false);  const [coutries, setCountries] = useState([]);  const [cities, setCities] = useState([]);  const [addresses, setAddresses] = useState([]);  const [date, setDate] = useState(null);  const [startTime, setStartTime] = useState(null);  const [finishTime, setFinishTime] = useState(null);  const [role, setRole] = useState('');  const [country, setCountry] = useState('');  const [city, setCity] = useState('');  const [startAddress, setStartAddress] = useState('');  const [finishAddress, setFinishAddress] = useState('')  const [price, setPrice] = useState('');  const [seatsCount, setSeatsCount] = useState('');  const [cars, setCars] = useState([]);  const [carField, setCarField] = useState('none');  const [carId, setCarId] = useState(undefined);  const dateChange = (newValue) => {  setDate(newValue);  };  const handleRoleChange = (event) => {  setRole(event.target.value)  if (event.target.value == 'driver') {  if (localStorage.getItem('x-user-cars') == []) {  handleOpen()  setRole('passenger')  }  else {  getCars();  setCarField('block')  }  }  else {  setCarField('none')  }  }  const handleCarIdChange = (event) => {  setCarId(event.target.value);  }  const handleCountryChange = (event) => {  setCountry(event.target.value)  getCities(event.target.value)  }  const handleCityChange = (event) => {  setCity(event.target.value)  getAddresses(event.target.value)  }  const handleStartAddressChange = (event) => {  setStartAddress(event.target.value)  }  const handleFinishAddressChange = (event) => {  setFinishAddress(event.target.value)  }  const handlePriceChange = (event) => {  setPrice(event.target.value)  }  const handleSeatsCountChange = (event) => {  setSeatsCount(event.target.value)  }  const getCountries = () => {  axios.get('/ads/countries', { headers: { 'x-access-token': localStorage.getItem('x-access-token') } })  .then((res) => {  setCountries(res.data);  })  .catch((e) => {  console.log(e)  })  }  const getCities = (param) => {  axios.get(`/ads/cities/${param}`, { headers: { 'x-access-token': localStorage.getItem('x-access-token') } })  .then((res) => {  setCities(res.data);  })  .catch((e) => {  console.log(e)  })  }  const getAddresses = (param) => {  axios.get(`/ads/addresses/${param}`, { headers: { 'x-access-token': localStorage.getItem('x-access-token') } })  .then((res) => {  setAddresses(res.data)  })  .catch((e) => {  console.log(e)  })  }  const getCars = () => {  axios.get(`/ads/cars`, {  headers: {  'x-access-token': localStorage.getItem('x-access-token'),  'x-user-id': localStorage.getItem('x-user-id')  }  })  .then((res) => {  setCars(res.data)  console.log('----- CARS LIST -----')  console.log(res.data)  console.log('---------------------')  })  .catch((e) => {  console.log(e.message)  })  }  const createAd = async () => {  const pad = num => ("0" + num).slice(-2);  const getTimeFromDate = timestamp => {  const date = new Date(timestamp);  let hours = date.getHours(),  minutes = date.getMinutes(),  seconds = date.getSeconds();  return pad(hours) + ":" + pad(minutes) + ":" + pad(seconds)  }  const startT = getTimeFromDate(startTime);  const finishT = getTimeFromDate(finishTime);  await axios.post('/ads/create', {  userId: localStorage.getItem('x-user-id'),  role: role,  carId: carId,  country: country,  city: city,  startAddress: startAddress,  finishAddress: finishAddress,  startDate: date,  startTime: startT,  finishTime: finishT,  seatsCount: seatsCount,  price: price  }, {  headers: { 'x-access-token': localStorage.getItem('x-access-token') }  })  .then((res) => {  console.log("===== CREATE AD RESPONSE =====")  console.log(res.data)  console.log("==============================")  })  .catch((err) => {  console.log("===== CREATE AD ERROR =====")  console.log(err.message)  console.log("===========================")  })  navigate('/profile')  }  useEffect(() => {  getCountries();  }, [])  if (localStorage.getItem('x-access-token')) {  return (  <>  <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>  </>  )  }  else {  window.location = '/'  }  } |