

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения 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 Программное обеспечение информационных технологий
(программирование интернет-приложений)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту на тему:**

Web-приложение «Бронирование билетов для поездки между городами
на маршрутном такси»

Выполнил студент Розель Станислав Александрович
(Ф.И.О.)

Руководитель проекта к.т.н., доц. В. В. Смелов
(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. В. В. Смелов
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой _____

Содержание

Введение	3
1 Постановка задачи и обзор аналогичных решений	4
1.1 Постановка задачи	4
1.2. Обзор аналогичных решений.....	4
1.2.1 Web-приложение «Атлас».....	4
1.2.2 Интернет-ресурс «7588.by»	5
1.3 Постановка задачи	6
2 Проектирование web-приложения	7
2.1 Разработка функциональных требований, определение вариантов использования	7
2.2 Архитектура приложения	10
2.3 Архитектура базы данных.....	12
2.4 Выводы по разделу	17
3 Реализация веб-приложения	18
3.1 Программная платформа Node.js	18
3.2 СУБД PostgreSQL	18
3.3 TypeORM	18
3.4 Программные библиотеки	21
3.5 Разработка серверной части приложения.....	22
3.6 Разработка клиентской части приложения	28
3.7 Web-сервер Nginx	31
3.8 Выводы по разделу	31
4 Тестирования web-приложения.....	32
4.1 Автоматизированное тестирование	32
4.2 Ручное тестирование	33
4.3 Выводы по разделу	35
5 Руководство пользователя	36
5.1 Руководство гостя.....	36
5.2 Руководство пассажира	38
5.3 Руководство администратора	39
5.4 Выводы по разделу	41
Заключение	42
Список используемых источников.....	43
Приложение А	45
Приложение Б.....	47
Приложение В	49
Приложение Г.....	50
Приложение Д	54

Введение

Веб-приложение «Бронирование билетов на межгороднее маршрутное такси» предназначено для автоматизации процесса бронирования билетов на межгородние маршруты. Оно предоставляет пользователям удобный интерфейс для выбора маршрута, времени отправления. Проект направлен на упрощение взаимодействия между пассажирами и транспортной компанией, минимизацию времени на оформление поездки и повышение прозрачности услуг.

Цель проекта — разработать функциональное и интуитивно понятное веб-приложение, которое позволит пользователям оперативно бронировать билеты на межгородние маршрутные такси, а администраторам транспортной компании — эффективно управлять расписанием, местами и данными о пассажирах.

Для достижения поставленной цели решаются следующие задачи:

- анализ требований и обзор аналогичных решений (глава 1);
- проектирование архитектуры веб-приложения (глава 2);
- реализация веб-приложения с использованием TypeScript [1] и Node.js [2] (глава 3);
- тестирование функциональности и производительности приложения (глава 4);
- разработка руководства пользователя (глава 5)

Приложение ориентировано на:

- пассажиров, регулярно пользующихся межгородними маршрутными такси, которым важны удобство и скорость покупки билетов.
- транспортные компании, желающие автоматизировать процесс продажи билетов и управления маршрутами.
- молодежь и активных пользователей интернета, предпочитающих онлайн-сервисы для планирования поездок.

Программная платформа - совокупность программных решений и технологий, позволяющая осуществлять разработку и/или производство программных продуктов в определенной предметной области. Для реализации проекта выбрана платформа Node.js, которая обеспечивает высокую производительность серверной части благодаря асинхронной обработке запросов. Node.js позволяет использовать TypeScript как для серверной, так и для клиентской разработки, что упрощает интеграцию и поддержку кода.

1 Постановка задачи и обзор аналогичных решений

1.1 Постановка задачи

Основная идея представляет собой веб-приложение для удобного поиска, бронирования и просмотр билетов на межгороднее маршрутное такси. Основная цель проекта заключается в том, чтобы создать централизованную платформу, где:

- веб-приложение должно обеспечивать удобный и интуитивно понятный интерфейс для выбора маршрутов, дат, времени отправления и мест, что упростит процесс бронирования билетов для пользователей.
- веб-приложение должно предоставлять администраторам транспортной компании инструменты для гибкого управления расписанием, доступными местами и данными о пассажирах, позволяя оперативно реагировать на изменения спроса и маршрутов.
- веб-приложение должно предусматривать механизмы просмотра пользователем о статусе бронирования, что повысит удовлетворенность клиентов и прозрачность сервиса.
- веб-приложение должно быть адаптивным, обеспечивая комфортное использование на различных устройствах (ПК, планшеты, смартфоны), чтобы охватить широкую аудиторию пользователей.

1.2. Обзор аналогичных решений

1.2.1 Web-приложение «Атлас»

Одним из альтернативных решений приложения для бронирования является интернет-ресурс atlasbus.by [3]. Веб-сайт содержит на главной странице сразу же панель для поиска и популярные маршруты, это позволяет пользователю быстро найти нужный маршрут и забронировать билет. Так же есть личный профиль, в котором можно просмотреть и изменить информацию.

Интерфейс интернет-ресурса «atlasbus.by» представлен на рисунке 1.1.

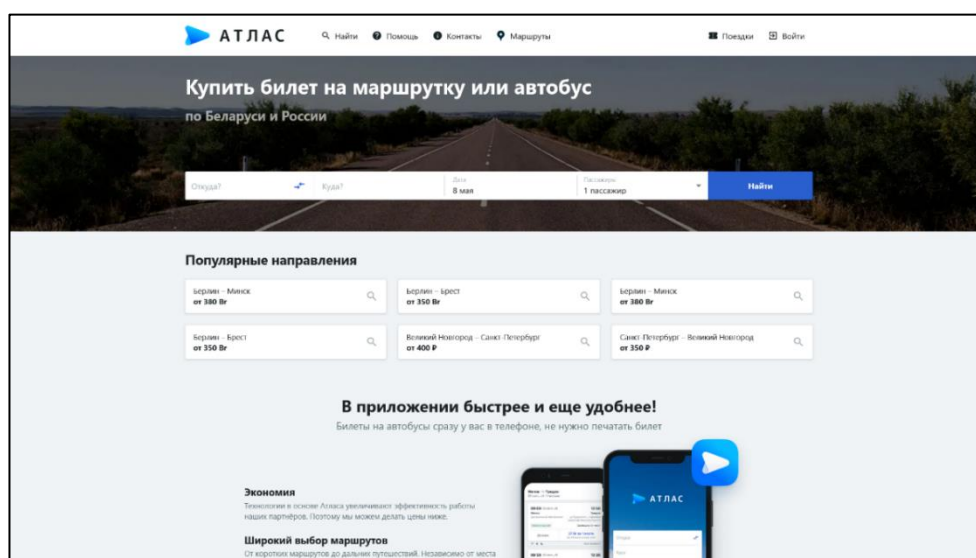


Рисунок 1.1 – Интерфейс «atlasbus.by»

К преимуществам можно отнести поиск на главной странице, чтобы пользователю не пришлось искать его по веб-сайту. Так же можно отнести к преимуществу то, что есть вывод популярных маршрутов, возможно кол-во действий по бронированию может ещё уменьшиться. В результате поиска есть много дополнительных настроек для фильтрации и сортировки, которые помогут найти нужный билет на маршрут ещё быстрее.

К недостаткам можно отнести, что нет полноценного расписание на день отдельной вкладкой, так как пользователь возможно будет рассматривать несколько маршрутов на день.

1.2.2 Интернет-ресурс «7588.by»

Веб-сайт «7588.by» [4] с достаточно простым, но приятным дизайном, нет ничего лишнего, что могло бы отвлекать пользователя или мешать ему. По наполнению сайта видно, что он используется очень маленькой региональной фирмой, но это не значит, что у этого сайта есть достоинства.

Интерфейс интернет-ресурса «7588.by» представлен на рисунке 1.2.

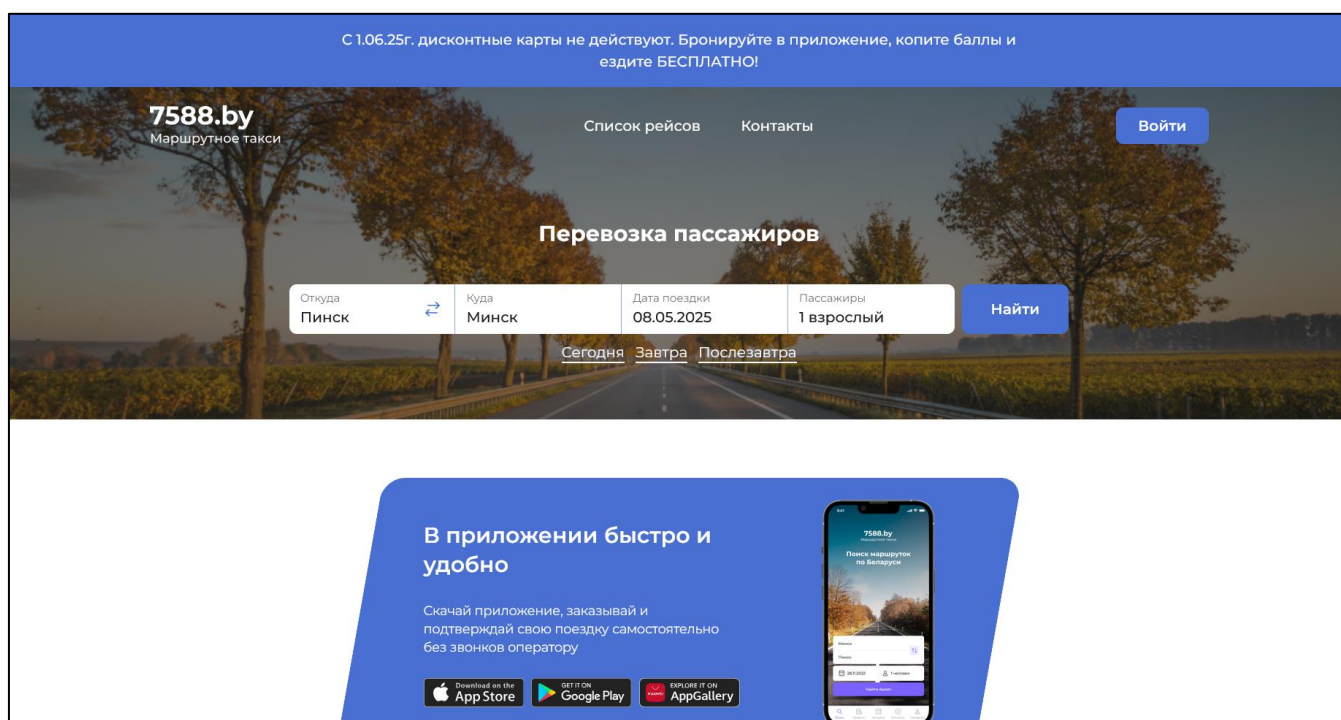


Рисунок 1.2 – Интерфейс «7588.by»

К преимуществам веб-сайта можно отнести то что у него собраны на главной странице популярные вопросы, так пользователь столкнувшись с каким-то простым вопрос может получить на него ответ просто зайдя на сайт и посмотрев на главной странице. Так же в результате поиска мы можем увидеть список билетов и для каждого билета будет видно сколько мест ещё осталось, если пользователю нужно забронировать не только себе.

Недостатками веб-сайта является отсутствие функции изменения ингредиентов пиццы и отсутствие поиска по наименованию товара.

1.3 Постановка задачи

На сегодняшний день актуальность разработки программного средства для управления сетью пиццерий велика, так как многие компании стремятся улучшить свои процессы производства, управления заказами и доставки, а также повысить удовлетворенность клиентов.

Обзор аналогов позволяет проанализировать все преимущества и недостатки альтернативных возможностей и сформулировать список требований, предъявляемых к разрабатываемому в данном курсовом проекте программному средству. Функционально web-приложение должно поддерживать роли «Гость», «Пассажир», «Администратор».

Функции пользователя с ролью «Гость»:

- регистрация;
- аутентификация;
- просмотр расписания движения маршруток;
- поиск билетов;
- сортировка билетов.

Функции пользователя с ролью «Пассажир»:

- авторизация;
- просмотр расписания движения маршруток;
- поиск билетов;
- сортировка билетов;
- бронирование билетов на выбранные рейсы;
- отображение истории бронирования билетов;
- возможность отмены;
- просмотр профиля.

Функции пользователя с ролью «Администратор»:

- авторизация;
- работа с расписанием движения маршруток (добавление, изменение, удаление рейсов);
- работа с информацией о маршрутах (добавление, изменение, удаление маршрутов);
- работа с данными пользователей (просмотр информации пользователя, разблокировка/блокировка).

В данной главе были сформулированы основные требования к приложению на основе преимуществ и недостатков некоторых аналогов.

2 Проектирование web-приложения

2.1 Разработка функциональных требований, определение вариантов использования

«Гость» – это неавторизованный пользователь, который может посетить открытую часть приложения. Данная роль нужна для того, чтобы каждый пользователь мог сразу же ознакомиться с списком доступных билетов и дальнейшем зарегистрироваться для использования веб-приложения.

В таблице 2.1 представлено описание доступных функций для роли «Гость».

Таблица 2.1 – Описание доступных функций для роли «Гость»

Функция	Описание	Номер варианта использования
Регистрация	Регистрация пользователя происходит посредством заполнения полей имени, фамилии, отчества, email, и пароля. После удачной регистрации пользователь оказывается на главной странице	21
Аутентификация	Вход в аккаунт с проверкой введенных email и пароля для наделения соответствующей роли	18
Просмотр расписания движения маршруток	Гость не регистрируясь может ознакомиться с списком доступных билетов.	15
Поиск билетов	Поиск представляет из себя форму с четырьмя полями по которым он может найти билет который ему нужен	14
Сортировка билетов.	Гость может так же сортировать билеты по параметрам отправки, пребывания, стоимость и количества мест	13

«Пассажир» – это авторизованный пользователь, у которого есть возможность полностью управлять данными своего аккаунта, а также может проверять статус билетов. Роль предназначена на самостоятельное бронирование и отказ билетов пользователем.

В таблице 2.2 представлено описание доступных функций для роли «Пассажир».

Таблица 2.2 – Описание доступных функций для роли «Пассажир»

Функция	Описание	Номер варианта использования
Авторизация	Процесс контроля доступа пользователя к определённым ресурсам или действиям в системе. Права доступа определяются на основе роли пользователя.	9

Окончание таблицы 2.2

Функция	Описание	Номер варианта использования
Сортировка билетов	Пассажир может так же сортировать билеты по параметрам отправки, пребывания, стоимость и количества мест	13
Поиск билетов	Поиск представляет из себя форму с четырьмя полями по которым он может найти билет который ему нужен	14
Просмотр расписания движения маршруток	Пассажир может ознакомиться с доступными для бронирования билетами	15
Бронирование билетов на выбранные рейсы	Пассажир может забронировать билет на выбранный рейс	8
Отображение истории бронирования билетов	В профиле пользователь может увидеть полный список всех билетов который он бронировал, отменял и выполнял.	6
Возможность отмены билета	Пользователь может отменить конкретный билет, а так же отменить всю бронь с билетами которые у нее есть	3
Просмотр профиля	Пользователь может просмотреть свои данные, а так же отредактировать их.	1

«Администратор» – пользователь с полными правами доступа к настройкам данных. Администратор может изменять данные о маршрутах, расписании, транспорте, так же может просматривать список всех пользователей.

В таблице 2.3 представлено описание доступных функций для роли «Администратор».

Таблица 2.3 – Описание доступных функций для роли «Пассажир»

Функция	Описание	Номер варианта использования
Авторизация	Процесс контроля доступа пользователя к определённым ресурсам или действиям в системе. Права доступа определяются на основе роли пользователя.	9
Работа с расписанием движения маршруток (добавление, изменение, удаление рейсов)	Администратор может добавить новый рейс для маршрутки, так же изменить данные о рейсе маршрутки или удалить рейс.	12

Окончание таблицы 2.3

Работа с информацией о маршрутах (добавление, изменение, удаление маршрутов)	Администратор может добавить новый маршрут, изменить уже существующий, а так же удалить маршрут.	5
Работа с данными пользователей (просмотр информации пользователя, разблокировка/блокировка).	Администратор видит базовую информацию о пользователе и может заблокировать.	20

На рисунке 2.1 представлена диаграмма вариантов использования



Рисунок 2.1 – Диаграмма вариантов использования

Диаграмма вариантов использования отображает взаимодействие ролей («Гость», «Пассажир», «Администратор») с системой через заданные функции.

Выполненная по стандартам UML, она включает актёров и варианты использования, визуализируя функциональные требования и обеспечивая основу для проектирования архитектуры приложения.

2.2 Архитектура приложения

Для стабильной, безопасной и масштабируемой работы веб-приложения «Бронирование билетов на маршрутное такси для междугородних поездок» подобраны современные технологии, программные платформы и сетевые протоколы.

В данном разделе будут описаны операционная система, сетевые протоколы, фреймворки, применяемые для разработки серверной и клиентской частей приложения, а также для обеспечения безопасности и взаимодействия с базой данных.

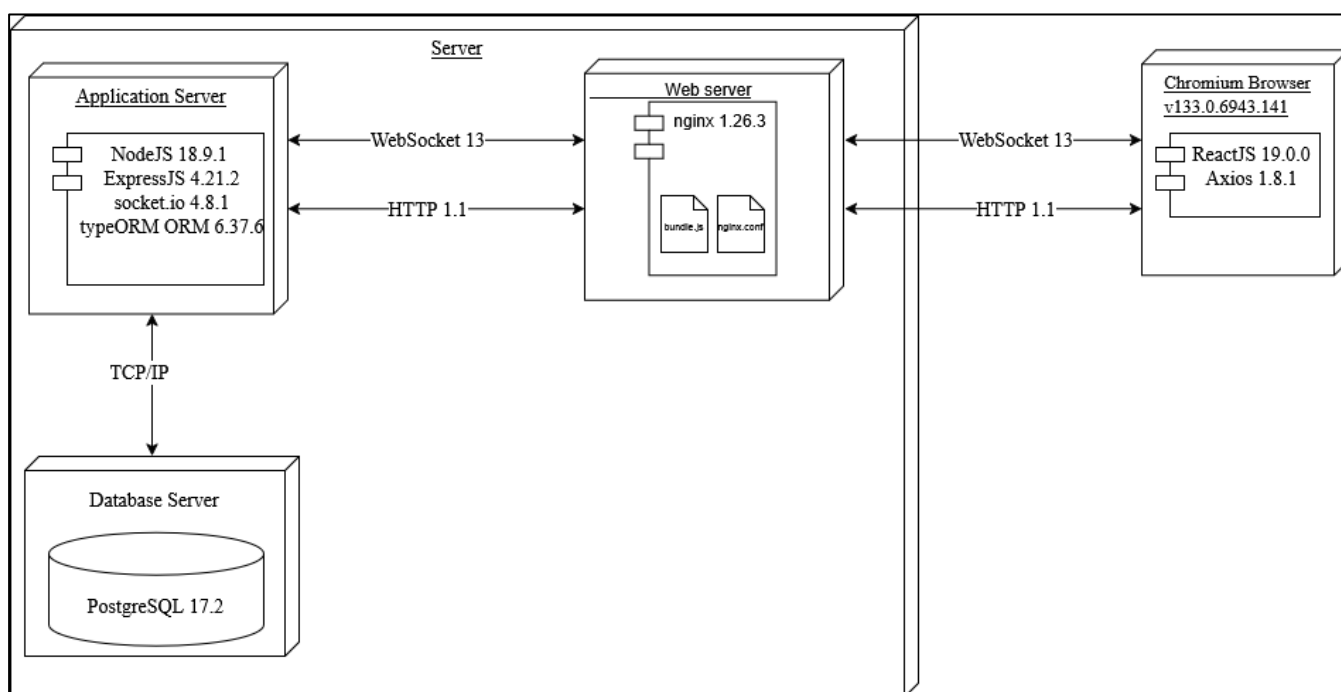


Рисунок 2.2 – Диаграмма развёртывания приложения

Для серверной части приложения выбраны операционная система и веб-сервер, обеспечивающие высокую стабильность, безопасность и производительность:

- Ubuntu Server 24.04 LTS[5] — операционная система, выбранная для развёртывания серверной части приложения (backend) и базы данных веб-приложения «Бронирование билетов на маршрутное такси для междугородних поездок». Эта версия ОС обеспечивает долгосрочную поддержку (Long Term Support), гарантируя регулярные обновления безопасности и совместимость с современными программными компонентами. Ubuntu Server характеризуется простотой настройки, широкой поддержкой серверного ПО и оптимизацией для работы с контейнерами и облачными сервисами;

- Nginx (версия 1.26)[6] — веб-сервер и обратный прокси, применяемый в веб-приложении «Бронирование билетов на маршрутное такси для междугородних поездок» для обработки входящих HTTP/HTTPS-запросов, маршрутизации их к

серверному приложению на Node.js и доставки статических файлов клиентской части. Nginx обслуживает сборку фронтенд-приложения (HTML, CSS, JavaScript файлы, сгенерированные CRA для React), обеспечивая быструю загрузку статических ресурсов. Кроме того, Nginx выполняет функции балансировки нагрузки, кэширования статических файлов для снижения нагрузки на сервер и ускорения работы приложения, а также поддерживает протокол HTTPS (версия 1.3) для безопасного обмена данными.

Для обеспечения надёжной, безопасной и эффективной передачи данных между клиентской и серверной частями приложения используются следующие сетевые протоколы:

- HTTP (версия 1.1) [7]: применяется исключительно на этапе разработки и отладки для упрощения тестирования API и взаимодействия между компонентами системы. В продакшен-среде HTTP заменяется на HTTPS для обеспечения безопасности;

Серверная часть (Backend):

- Node.js (версия 22.13): среда выполнения JavaScript/TypeScript, которая обеспечивает асинхронную обработку запросов и высокую производительность благодаря событийно-ориентированной модели, Node.js поддерживает масштабируемость приложения и позволяет эффективно обрабатывать большое количество одновременных подключений;

- Express.js (версия 4.21) [8] — минималистичный и гибкий веб-фреймворк для Node.js, применяемый в веб-приложении «Бронирование билетов на маршрутное такси для междугородних поездок» для разработки RESTful API. Фреймворк обеспечивает удобную маршрутизацию запросов, обработку HTTP-методов и интеграцию с middleware-компонентами, упрощая создание серверной логики и взаимодействие с клиентской частью приложения.

- WebSocket (RFC 6455) [9] — протокол, используемый в веб-приложении «Бронирование билетов на маршрутное такси для междугородних поездок» для реализации функций в реальном времени. Он обеспечивает блокировку пользователей при нарушении правил использования платформы. WebSocket поддерживает постоянное соединение между клиентом и сервером, что минимизирует задержки и обеспечивает оперативное обновление данных. Для разработки серверной и клиентской частей приложения применены современные фреймворки и инструменты, поддерживающие асинхронную архитектуру, высокую производительность и удобство сопровождения.

Клиентская часть (Frontend):

- React (версия 19.0) [10]: библиотека JavaScript для построения динамических пользовательских интерфейсов, React используется для создания компонентного интерфейса, обеспечивающего модульность, переиспользуемость и высокую скорость отрисовки. Поддержка виртуального DOM позволяет минимизировать затраты на обновление интерфейса при изменении данных;

- Create React App (CRA) [11] – это инструмент командной строки, созданный Facebook и используемый для упрощения создания и настройки одностраничных приложений на React. Он автоматически выполняет настройку конфигурации проекта, включая компиляцию, сборку и другие важные процессы, позволяя

разработчикам сосредоточиться на кодировании и логике приложения, а не на настройке инструментов.

Для реализации функциональности базы данных, аутентификации и безопасности используются следующие сторонние инструменты:

- PostgreSQL (версия 17) [12]: мощная реляционная система управления базами данных (СУБД), выбранная для хранения данных пользователей, медицинских записей, расписаний и услуг, PostgreSQL обеспечивает высокую надёжность, поддержку сложных запросов и транзакций, а также совместимость с асинхронными операциями, что соответствует требованиям независимости слоёв приложения;

- TypeORM (версия 0.3.20) [13] — объектно-реляционное отображение (ORM) для Node.js, используемое для упрощения взаимодействия с базой данных PostgreSQL в веб-приложении «Бронирование билетов на маршрутное такси для междугородних поездок». TypeORM предоставляет высокоуровневый API для работы с моделями данных, позволяя определять структуру таблиц, выполнять запросы и управлять связями (например, один-к-одному, один-ко-многим) без необходимости написания сложных SQL-запросов. Поддержка асинхронных операций, встроенные механизмы валидации данных и автоматическая генерация миграций обеспечивают удобство разработки и безопасность при работе с данными бронирования билетов.

- bcrypt (версия 5.1) [14]: библиотека для хеширования паролей пользователей, используется для безопасного хранения паролей в базе данных, применяя алгоритм хеширования с солью для защиты от атак перебора (brute force);

- jsonwebtoken (версия 9) [15]: библиотека для реализации JSON Web Tokens (JWT), используемых для аутентификации и авторизации пользователей, JWT позволяет безопасно передавать информацию о ролях и правах доступа между клиентом и сервером, обеспечивая разграничение функционала для ролей «Гость», «Пассажир» и «Администратор».

2.3 Архитектура базы данных

В данном подразделе описывается база данных веб-приложения «Бронирование билетов на маршрутное такси для междугородних поездок», разработанной на основе СУБД PostgreSQL.

При проектировании учтены требования к надежности, масштабируемости и эффективности хранения данных. База данных построена на нормализованной реляционной модели, что снижает избыточность данных и гарантирует их согласованность. Модель включает сущности, обеспечивающие выполнение CRUD-операций над пользователями (пассажирами и администраторами), маршрутами, расписанием поездок, билетами и данными о бронированиях.

Отдельное внимание уделено структуре таблиц, связям между таблицами с помощью внешних ключей и ключевых ограничений, которые обеспечивают целостность данных при добавлении, изменении и удалении информации.

На рисунке 2.2 представлена диаграмма, иллюстрирующая логическую схему базы данных и взаимосвязи между основными сущностями веб-приложения «Бронирование билетов».

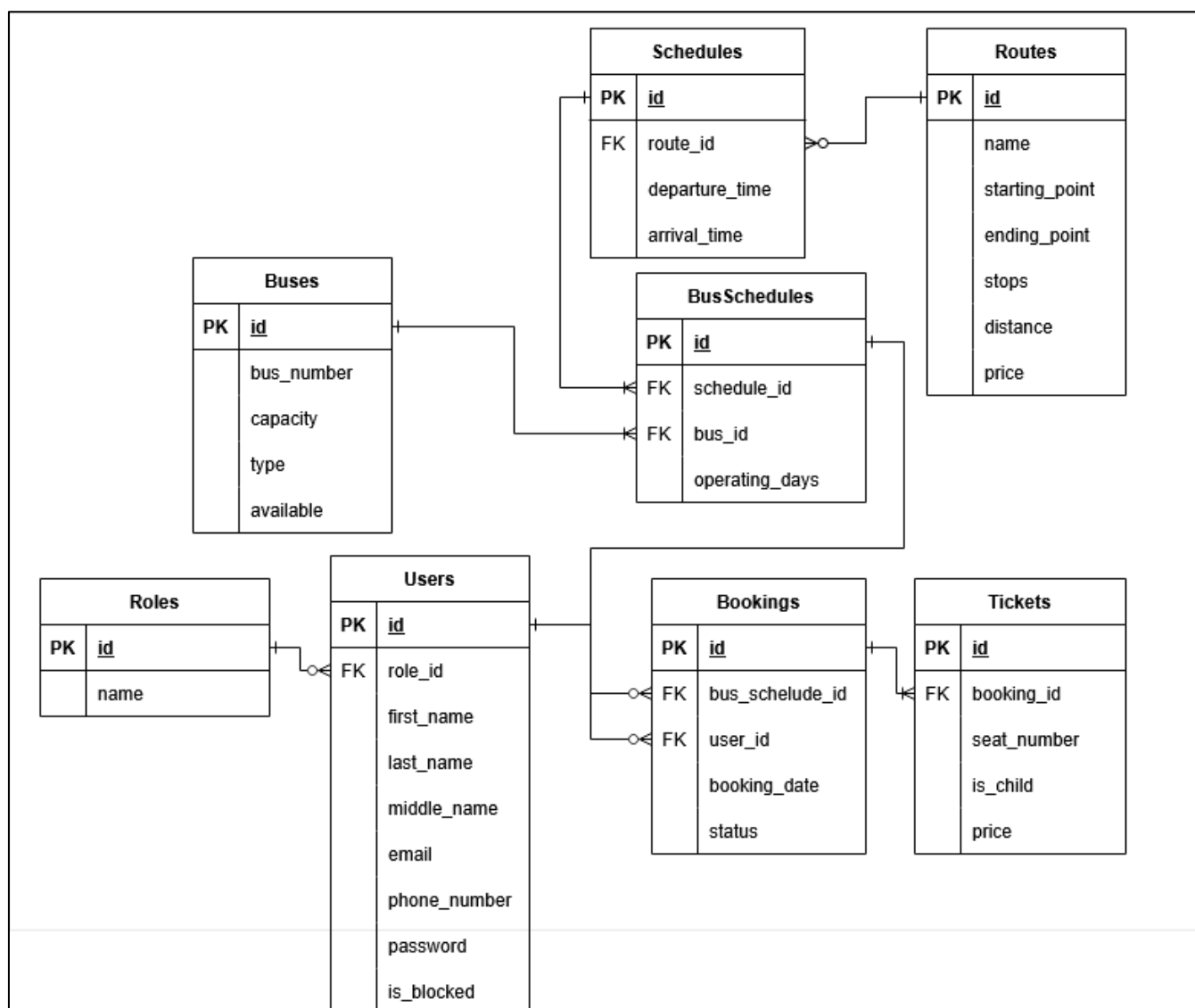


Рисунок 2.2 – Логическая базы данных

В таблице 2.5 представлено назначение всех таблиц базы данных. В приложении А представлен скрипт создания таблиц

Таблица 2.4 – Назначение таблиц в базе данных

Название	Назначение
roles	Роли пользователей
users	Зарегистрированные пользователи
routes	Доступные маршруты
tickets	Забронированные билеты
bookings	Совокупность забронированных билетов пользователей
buses	Доступный транспорт
schedules	Расписание маршрута
busschedules	Совокупность транспорта, который ходит по определенным маршрутам

Для гарантии целостности данных и бесперебойной работы системы между таблицами настроены связи с использованием внешних ключей (FOREIGN KEY). В таблице 2.5 представлено описание всех взаимосвязей между таблицами базы данных.

Таблица 2.5 – Описание связей между таблицами базы данных

Таблица-источник	Связанная таблица	Тип связи	Описание
roles	users	Один ко многим с обязательной связью	Пользователь роль, которая может быть и у других
users	bookings	Один ко многим с необязательной связью	У пользователя могут быть брони, а может и не быть
bookings	tickets	Один ко многим с необязательной связью	Бронь содержит все билеты, которые забронировал пользователь
routes	schedule	Один ко многим с необязательной связью	Расписание маршрута может быть с разным временем
schedule	busschedule	Один ко многим с необязательной связью	По разным путям с разными расписанием могут ходить разный транспорт
bus	busschedule	Один ко многим с необязательной связью	Разный транспорт может ходить по разным маршрутам с разным расписанием
busschedule	bookings	Один ко многим с необязательной связью	Бронь на конкретный маршрут и расписанием с конкретным транспортом

Таблица «roles» хранит возможные в приложении роли. Описание полей представлено в таблице 2.7.

Таблица 2.7 – Описание полей таблицы «roles»

Поле	Описание
id	Уникальный идентификатор роли
name	Название роли (уникальное)

Таблица «users» хранит всех зарегистрированных пользователей, которые будут пользоваться нашим веб-приложением – бронировать билеты или редактировать данные о маршрутах и транспорте. Описание полей представлено в таблице 2.8.

Таблица 2.8 – Описание полей таблицы «users»

Поле	Описание
id	Уникальный идентификатор пользователя
role_id	Роль пользователя
first_name	Имя
last_name	Фамилия
middle_name	Отчество
email	Электронная почта
phone_number	Номер телефона
password	Пароль
is_blocked	Флаг блокировки пользователя

Таблица «routes» хранит информацию о всех маршрутах. Описание полей представлено в таблице 2.9.

Таблица 2.9 – Описание полей таблицы « routes»

Поле	Описание
id	Уникальный идентификатор маршрута
name	Имя маршрута
starting_point	Начальная точка
ending_point	Конечная точка
stops	Промежуточные остановки
distance	Расстояние
price	Цена по маршруту

Таблица «schedules» хранит информацию о времени в которое транспорт будет ходить по маршруту. Описание полей представлено в таблице 2.9.

Таблица 2.9 – Описание полей таблицы «schedules»

Поле	Описание
id	Уникальный идентификатор расписания
route_id	Идентификатор маршрута
departure_time	Время отправления
arrival_time	Время прибытия

Таблица «buses» хранит информацию о времени в которое транспорт будет ходить по маршруту. Описание полей представлено в таблице 2.10.

Таблица 2.10 – Описание полей таблицы «buses»

Поле	Описание
id	Уникальный идентификатор транспорта
bus_number	Номер транспорта
capacity	Количество мест
type	Тип транспорта
available	Доступность

Таблица «busschedules» хранит комбинацию транспорта и расписания маршрута, данная таблица позволяет более гибко распределять транспорт по маршрутам хождения и в разные дни.

Описание полей представлено в таблице 2.11.

Таблица 2.11 – Описание полей таблицы «busschedule»

Поле	Описание
id	Уникальный идентификатор расписания транспорта
bus_id	Идентификатор транспорта
schedule_id	Идентификатор расписания
operating_days	Даты в которые будет работать эта совокупность транспорта и маршрута

Таблица «bookings» хранит информацию о бронированиях пользователей. Описание полей представлено в таблице 2.12.

Таблица 2.12 – Описание полей таблицы «bookings»

Поле	Описание
id	Уникальный идентификатор брони
bus_schedule_id	Идентификатор расписания транспорта
user_id	Идентификатор пользователя
booking_date	Время бронирования
status	Статус бронирования

Таблица «tickets» хранит информацию о забронированных билетах. В таблице будут храниться не только забронированные, но и отмененные билеты. Описание полей представлено в таблице 2.13.

Таблица 2.13 – Описание полей таблицы «tickets»

Поле	Описание
id	Уникальный идентификатор брони
booking_id	Идентификатор брони
seat_number	Номер места
is_child	Тип билета
price	Цена билета

Описанная структура базы данных предоставляет всестороннее решение для управления системой бронирования билетов на маршрутное такси для междугородних поездок, включая все главные аспекты функционирования: от регистрации и авторизации пользователей и маршрутов до организации расписания и ведения записей о бронированиях. Такая всеобъемлющая организация данных позволяет эффективно поддерживать операционную деятельность системы, обеспечивая удобство как для пассажиров, так и для администраторов. Нормализованная реляционная модель с тщательно спроектированными связями между таблицами обеспечивает целостность данных и высокую эффективность их обработки.

2.4 Выводы по разделу

Разработанная структура базы данных для веб-приложения «Бронирование билетов на маршрутное такси для междугородних поездок» представляет собой тщательно спроектированное решение, обеспечивающее эффективное управление всеми ключевыми аспектами системы. Она охватывает основные процессы, включая учет пользователей (пассажиров и администраторов), управление маршрутами, расписанием поездок и бронированиями билетов. Нормализованная реляционная модель, лежащая в основе базы данных, минимизирует избыточность данных, обеспечивает их согласованность и повышает производительность при обработке запросов.

Структура базы данных состоит из 7 взаимосвязанных таблиц, каждая из которых отвечает за определенные сущности системы, такие как пользователи, маршруты, расписание и бронирования. Тщательно разработанные связи между таблицами, реализованные через внешние ключи (FOREIGN KEY), гарантируют целостность данных, предотвращая некорректные изменения или удаление записей, которые могут нарушить логику работы приложения. В общей сложности определено 7 отношений между таблицами, что позволяет эффективно управлять зависимостями, например, между бронированиями и соответствующими маршрутами или пользователями.

Для обработки операций удаления применяются различные стратегии ограничений внешних ключей, такие как CASCADE. Этот механизм обеспечивает гибкость в управлении данными, позволяя адаптировать поведение системы к различным сценариям, например, при аннулировании маршрута или удалении пользовательских данных.

Безопасность данных усилена за счет хеширования паролей пользователей с использованием современных криптографических алгоритмов, что защищает учетные записи от компрометации. Структура базы данных оптимизирована для высокой производительности благодаря продуманной организации таблиц, что ускоряет выполнение запросов даже при большом объеме данных. Кроме того, нормализованная модель обеспечивает масштабируемость, позволяя системе адаптироваться к росту числа пользователей и бронирований.

Разработанная архитектура базы данных поддерживает удобство использования для пассажиров (например, простой процесс бронирования билетов) и администраторов (управление маршрутами и расписанием), а также предоставляет гибкость для дальнейшего расширения функциональности. Например, в будущем возможно добавление интеграции с платежными системами, уведомлений в реальном времени через WebSocket или аналитических инструментов для оптимизации работы маршрутных такси.

Предложенная структура базы данных полностью соответствует функциональным и нефункциональным требованиям проекта, обеспечивая надежную основу для реализации и дальнейшего развития приложения. Подробное описание всех связей и структуры таблиц приведено в таблице 2.6, которая служит ориентиром для разработчиков и администраторов при внедрении и сопровождении системы.

3 Реализация веб-приложения

3.1 Программная платформа Node.js

Для серверной части веб-приложения «Бронирование билетов на маршрутное такси для междугородних поездок» выбрана платформа Node.js — среда выполнения JavaScript/TypeScript на серверной стороне. Этот выбор обусловлен рядом преимуществ: асинхронной событийно-ориентированной архитектурой, обеспечивающей высокую производительность при обработке множества одновременных соединений, и возможностью использования единого языка программирования (JavaScript/TypeScript) как для серверной, так и для клиентской части приложения.

Node.js эффективно обрабатывает интенсивные операции ввода-вывода, что особенно важно для системы бронирования с потенциально большим количеством запросов. Платформа поддерживает быструю разработку благодаря встроенному пакетному менеджеру (npm) и отличной совместимости с современными веб-технологиями. Для обработки HTTP-запросов и создания API используется фреймворк Express.js, который предоставляет компактный, но сильный и удобный набор инструментов для маршрутизации запросов, реализации промежуточной обработки и интеграции с другими компонентами системы.

3.2 СУБД PostgreSQL

Для управления базой данных веб-приложения «Бронирование билетов на маршрутное такси для междугородних поездок» выбрана PostgreSQL — мощная объектно-реляционная СУБД с открытым исходным кодом. PostgreSQL обеспечивает надежное хранение данных, поддержку сложных запросов и транзакций, а также высокий уровень безопасности, что крайне важно для приложения, работающего с конфиденциальной информацией о бронированиях и пользователях. СУБД предоставляет полный набор функций реляционных баз данных, включая внешние ключи, триггеры, хранимые процедуры и представления.

Скрипт создания базы данных представлен в приложении А.

3.3 TypeORM

Для взаимодействия с базой данных PostgreSQL в веб-приложении «Бронирование билетов на маршрутное такси для междугородних поездок» используется TypeORM — мощный ORM (Object-Relational Mapping) для Node.js. TypeORM предоставляет высокоуровневый API для работы с реляционными базами данных, позволяя разработчикам манипулировать JavaScript-объектами вместо написания сложных SQL-запросов вручную. Библиотека поддерживает все основные операции CRUD (Create, Read, Update, Delete), транзакции, миграции, ассоциации между моделями (один-к-одному, один-ко-многим, многие-ко-многим) и выполнение сложных запросов.

TypeORM особенно эффективен для системы бронирования, так как обеспечивает валидацию данных на уровне моделей, автоматическое преобразование типов и встроенную защиту от SQL-инъекций, что критически

важно для безопасной работы с данными пользователей и бронирований. Интеграция с Node.js осуществляется естественно благодаря поддержке асинхронных операций, что гармонично сочетается с событийно-ориентированной архитектурой платформы.

ORM предоставляет гибкие механизмы для определения связей между таблицами, что соответствует реляционной структуре базы данных приложения, включающей сущности, такие как пользователи, маршруты и бронирования. Кроме того, библиотека предлагает автоматическую генерацию миграций, что упрощает управление изменениями в структуре базы данных в процессе разработки и сопровождения приложения.

Инициализация TypeORM выполняется с параметрами из переменных окружения. В листинге 3.1 представлен код инициализации.

```
export const AppDataSource = new DataSource({
  type: 'postgres',
  host: process.env.HOST,
  port: Number(process.env.DB_PORT),
  username: process.env.DB_USERNAME,
  password: String(process.env.DB_PASSWORD),
  database: process.env.DB_NAME,
  synchronize: false,
  entities: ["src/modules/**/*.entities/*.ts"],
  migrations: ["src/migrations/*.ts"],
  logging: false,
})
```

Листинг 3.1 – Инициализация TypeORM

Для работы с базой данных через TypeORM необходимо создать модели – классы, которые представляют таблицы базы данных в объектно-ориентированном стиле. Модели служат прослойкой между приложением и СУБД. Соответствие созданных моделей и таблиц базы данных представлено в таблице 3.1.

Таблица 3.1 – Соответствие таблиц и моделей

Название таблицы	Модель
users	User
routes	Route
tickets	Ticket
bookings	Booking
buses	Bus
schedules	Schedule
busschedules	BusSchedule

Реализация моделей Route и Schedule представлены в листингах 3.2 и 3.3.

```
@Entity("routes")
export class Route {
  @PrimaryGeneratedColumn()
  id: number;
```

```

@Column({ type: "varchar", length: 255 })
name: string;

@Column({ type: "varchar", length: 255 })
starting_point: string;

@Column({ type: "varchar", length: 255 })
ending_point: string;

@Column({ type: "text", nullable: true })
stops: string;

@Column({ type: "float" })
distance: number;

@Column({ type: "float" })
price: number;

@OneToMany(() => Schedule, (schedule) => schedule.route)
schedules: Schedule[];
}

```

Листинг 3.2 – Модель Route

Между моделями, как и между таблицами базы данных были установлены соответствующие связи, из-за особенности TypeORM все связи указываются сразу в модели.

```

@Entity("schedules")
export class Schedule {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  route_id: number;

  @Column({ type: "time" })
  departure_time: Date;

  @Column({ type: "time" })
  arrival_time: Date;

  @ManyToOne(() => Route, (route) => route.schedules)
  @JoinColumn({ name: "route_id" })
  route: Route;

  @OneToMany(() => BusSchedule, busSchedule => busSchedule.schedule)
  busSchedules: BusSchedule[];
}

```

Листинг 3.3 – Модель Schedule

3.4 Программные библиотеки

Для реализации серверной и клиентской частей приложения был выбран современный стек технологий.

Программные библиотеки серверной части приложения описаны в таблице 3.2.

Таблица 3.2 – Библиотеки серверной части приложения

Библиотека	Версия	Описание
Express	4.21.2	Веб-фреймворк для Node.js, упрощающий создание серверов и API с минималистичным синтаксисом и мощной маршрутизацией.
TypeORM	0.3.21	ORM-библиотека для работы с базами данных через TypeScript
jsonwebtoken	9.0.2	Библиотека для создания и проверки JSON Web Tokens (JWT) для аутентификации и авторизации в приложениях.
pg [16]	8.13.3	Драйвер для работы с PostgreSQL в Node.js, предоставляет API для выполнения SQL-запросов и управления подключениями.
bcryptjs	3.0.2	Библиотека для хеширования паролей и их проверки, использует алгоритм bcrypt для обеспечения безопасности.
cors [17]	2.8.5	Middleware для Express, позволяет настраивать Cross-Origin Resource Sharing, разрешая/ограничивая доступ к API с других доменов.
socket.io [18]	4.8.1	Библиотека для реализации real-time коммуникации через WebSocket с поддержкой событий.
routing-controllers [19]	0.11.2	Фреймворк для создания контроллеров и маршрутов в Express-приложениях с использованием декораторов TypeScript.
body-parser [20]	1.20.3	Middleware для Express, парсит тело HTTP-запросов для удобной работы с данными.
dotenv [21]	16.4.7	Модуль для загрузки переменных окружения из файла .env в процесс Node.js, упрощает управление конфигурацией.
jest [22]	29.7.0	Фреймворк для тестирования JavaScript/TypeScript-кода, поддерживает модульные тесты, моки и снапшоты.

Программные библиотеки клиентской части приложения описаны в таблице 3.3

Таблица 3.3 – Программные библиотеки клиентской части

Библиотека	Версия	Описание
React	19.0.0	Библиотека JavaScript направленная на более гибкое создание пользовательских интерфейсов, основанная на компонентах, с фокусом на декларативный рендеринг и управление состоянием.

Окончание таблицы 3.3

Библиотека	Версия	Описание
React-router-dom [23]	7.4.0	Библиотека для маршрутизации в React-приложениях, позволяет создавать навигацию и управлять URL в браузере.
Sass [24]	1.87.0	Препроцессор CSS, добавляющий переменные, вложенные селекторы и миксины для упрощения стилизации.
Zod [25]	3.24.3	Библиотека для валидации данных в TypeScript/JavaScript, позволяет создавать схемы для проверки типов и структуры данных.
socket.io-client [26]	4.8.1	Клиентская часть Socket.io для взаимодействия с сервером через WebSocket, используется для real-time коммуникации.
react-dom [27]	19.0.0	Библиотека для рендеринга React-компонентов в DOM, предоставляет методы для интеграции React с браузером.

Выбранные библиотеки обеспечивают баланс функциональности, надежности и удобства, создавая надежную основу для развития приложения. Активная поддержка сообществом гарантирует стабильность и своевременные обновления компонентов.

3.5 Разработка серверной части приложения

Серверная часть приложения реализована на Node.js с использованием фреймворка Express.js.

В приложении используется REST (Representational State Transfer) [28] архитектурный стиль.

Для структурирования всех маршрутов в серверной части использовалась библиотека routing-controllers, которая позволяет удобно структурировать все маршруты в классах

В листинге 3.4 представлен код разделения маршрутов на модули

```

useExpressServer(app, {
  controllers: [RouteController, ScheduleController, AuthController,
                BusController, BusScheduleController,
                UserController, BookingController, TicketController],
  authorizationChecker: async (action) => {
    const openPaths = [
      '/auth/refresh/',
      '/auth/login/',
      '/auth/register/'
    ];
    if (openPaths.some(path
action.request.path.startsWith(path))) {
      return true;
    }
  }
}) =>

```

```

    const token = action.request.headers.authorization?.split('
')[1];
    if (!token) return false;

    try {
      await authMiddleware(action.request, action.response, () => {
    });
      return true;
    } catch (error) {
      return false;
    }
  },
});
});

```

Листинг 3.5 – Группировка маршрутов по классовым модулям

В таблице 3.4 приведены описания маршрутов для модуля «/auth» и соответствующих им методов контроллеров.

Таблица 3.4 – Маршруты модуля «/auth» и соответствующие им методы

Метод	Маршрут	Метод контроллера	Описание	Номер варианта использования
POST	/register	AuthController.register	Маршрут для регистрации нового пользователя	
POST	/login	AuthController.login	Маршрут для авторизации и аутентификации существующего пользователя	

В таблице 3.5 приведены описания маршрутов для модуля «/users» и соответствующих им методов контроллеров.

Таблица 3.5 – Маршруты модуля «/users» и соответствующие им методы

Метод	Маршрут	Метод контроллера	Описание	Номер варианта использования
GET	/	UserController.getAll	Маршрут для получения всех пользователей	
GET	/:id	UserController.getUserById	Маршрут для получения одного пользователя по id	

Окончание таблицы 3.5

Метод	Маршрут	Метод контроллера	Описание	Номер варианта использования
PATCH	/update/:id	UserController.update	Маршрут для обновления данных о пользователе	
PATCH	/blocked/:id	UserController.updateUserById	Маршрут для блокировки пользователя	

В таблице 3.6 приведены описания маршрутов для модуля «/booking» и соответствующих им методов контроллеров.

Таблица 3.6 – Маршруты модуля «/booking» и соответствующие им методы

Метод	Маршрут	Метод контроллера	Описание	Номер варианта использования
GET	/	BookingController.getAll	Маршрут для получения всех бронирований	
GET	/:user_id	BookingController.getBookingById	Маршрут для получения одной брони по id	
POST	/create	BookingController.createBooking	Маршрут для создания брони	
PATCH	/update/:id	BookingController.updateBookingById	Маршрут для обновления брони	
PATCH	/cancel/:id	BookingController.cancelBookingById	Маршрут для отмены брони	
DELETE	/delete/:id	BookingController.deleteBookingById	Маршрут для удаления брони	

В таблице 3.7 приведены описания маршрутов для модуля «/buses» и соответствующих им методов контроллеров.

Таблица 3.7 – Маршруты модуля «/buses» и соответствующие им методы

Метод	Маршрут	Метод контроллера	Описание	Номер варианта использования
GET	/	BusController.getAll	Маршрут для получения транспорта	

Окончание таблицы 3.7

GET	/:id	BusController.getBusById	Маршрут для получения одного транспорта по id	
POST	/create	BusController.createBus	Маршрут для создания транспорта	
PATCH	/update/:id	BusController.updateBusById	Маршрут для обновления транспорта по id	
DELETE	/delete/:id	BusController.deleteBusById	Маршрут для удаления транспорта	

В таблице 3.8 приведены описания маршрутов для модуля «» и соответствующих им методов контроллеров.

Таблица 3.8 – Маршруты модуля «/bus-schedules» и соответствующие им методы

Метод	Маршрут	Метод контроллера	Описание	Номер варианта использования
GET	/	BusScheduleController.getAll	Маршрут для получения всего расписания транспорта	
GET	/:id	BusScheduleController.getBusScheduleById	Маршрут для получения одного расписания транспорта по id	
POST	/create	BusScheduleController.createBusSchedule	Маршрут для создания расписания транспорта	
PATCH	/update/:id	BusScheduleController.updateBusScheduleById	Маршрут для обновления расписания транспорта по id	
DELETE	/delete/:id	BusScheduleController.deleteBusScheduleById	Маршрут для удаления расписания транспорта	

В таблице 3.9 приведены описания маршрутов для модуля «/routes» и соответствующих им методов контроллеров.

Таблица 3.9 – Маршруты модуля «/routes» и соответствующие им методы

Метод	Маршрут	Метод контроллера	Описание	Номер варианта использования
GET	/	RouteController.getAll	Маршрут для получения всех маршрутов	
GET	/:id	RouteController.getRouteById	Маршрут для получения одного маршрута по id	
GET	/price/:id	RouteController.getPriceById	Маршрут для получения цены маршрута	
POST	/create	RouteController.createRoute	Маршрут для создания маршрута	
PATCH	/update/:id	RouteController.updateRouteById	Маршрут для обновления маршрута по id	
DELETE	/delete/:id	RouteController.deleteRouteById	Маршрут для удаления маршрута	

В таблице 3.10 приведены описания маршрутов для модуля «/schedules» и соответствующих им методов контроллеров.

Таблица 3.10 – Маршруты модуля «/schedules» и соответствующие им методы

Метод	Маршрут	Метод контроллера	Описание	Номер варианта использования
GET	/	ScheduleController.getAll	Маршрут для получения всех расписаний	
GET	/:id	ScheduleController.getScheduleById	Маршрут для получения одного расписания по id	
POST	/create	ScheduleController.createSchedule	Маршрут для создания расписания	
PATCH	/update/:id	ScheduleController.updateScheduleById	Маршрут для обновления расписания по id	

Окончание таблицы 3.10

DELETE	/delete/:id	ScheduleController.deleteScheduleById	Маршрут для удаления расписания	
--------	-------------	---------------------------------------	---------------------------------	--

В таблице 3.11 приведены описания маршрутов для модуля «/tickets» и соответствующих им методов контроллеров.

Таблица 3.11 – Маршруты модуля «/tickets» и соответствующие им методы

Метод	Маршрут	Метод контроллера	Описание	Номер варианта использования
GET	/	TicketController.getAll	Маршрут для получения всех билетов	
GET	/:id	TicketController.getTicketById	Маршрут для получения одного билета по id	
POST	/create	TicketController.createTicket	Маршрут для создания билета	
PATCH	/update/:id	TicketController.updateTicketById	Маршрут для обновления билета по id	
PATCH	/cancel/:id	TicketController.cancelTicketById	Маршрут для отмены билета по id	
DELETE	/delete/:id	TicketController.deleteTicketById	Маршрут для удаления билета	

Файл index.ts является точкой входа для серверной части приложения «Медицинский центр» и выполняет несколько ключевых функций:

- создает HTTP сервер на базе Express-приложения;
- устанавливает порт подключения из переменных окружения (по умолчанию 3001 для HTTP);
- инициализирует Socket.IO сервер поверх HTTP;
- запускает сервера HTTP;
- Подключается к базе данных.

В приложении Б представлено содержимое файла index.ts.

В приложении также были разработаны middleware-обработчики [29].

Middleware – это промежуточное ПО, которое обрабатывает HTTP-запросы и ответы в веб-приложении до их попадания в конечные обработчики (контроллеры).

В листинге 3.5 представлен код middleware предназначенного для проверки подлинности пользователей в приложении.

```
export const authMiddleware = (req: Request, res: Response, next: NextFunction) => {
```

```

const token = req.headers.authorization?.split(' ')[1];
if (!token) {
    return res.status(401).json({ message: 'No token provided' });
}

try {
    const decoded = verifyAccessToken(token);
    (req as any).user = decoded;
    next();
} catch (error) {
    res.status(401).json({ message: 'Invalid token' });
}
};

```

Листинг 3.5 – Код middleware для авторизации

Middleware проверяет подлинность пользователей, анализируя заголовок Authorization с JWT-токеном (схема Bearer). При отсутствии токена возвращает статус 401 (Unauthorized) с ошибкой. Для валидного токена верифицирует его с помощью JWT_SECRET из переменных окружения, извлекает userId и userRole, добавляя их в объект запроса (req) для последующих обработчиков. При недействительном или просроченном токене возвращает 401 с сообщением «Неверный токен». Обеспечивает централизованную защиту API, гарантируя безопасность конечных точек.

При разработке серверной части была реализована паттерн Repository, который выполняет несколько ключевых функций:

- инкапсуляция логики доступа к данным (все SQL-запросы и взаимодействие с БД сосредоточены в одном месте);
- разделение ответственности (контроллеры работают с бизнес-логикой, репозитории - с данными);
- упрощение тестирования (можно легко мокать репозитории при тестировании контроллеров);
- повторное использование кода (общие запросы к БД доступны из разных частей приложения).

На каждый класс-контроллер был разработан репозиторий. В приложении В представлен код «ScheduleRepository».

3.6 Разработка клиентской части приложения

Клиентская часть приложения реализована как одностраничное приложение (Single Page Application, SPA) на базе React, что обеспечивает быструю и плавную работу интерфейса без необходимости полной перезагрузки страниц. В разработке использованы современные подходы к созданию веб-приложений, включая декларативное управление компонентами, маршрутизацию и управление состоянием, что позволяет создать масштабируемую и поддерживаемую кодовую базу. Основной акцент сделан на разработке интуитивно понятного, отзывчивого и доступного пользовательского интерфейса (UI), адаптированного под потребности

двух ключевых групп пользователей: пассажиров и администраторов. Приложение использует React Router v7 для навигации между страницами.

В таблице 3.9 приведено описание все страниц приложения

Таблица 3.9 – Описание всех страниц веб-приложения

Путь(URL)	Роли, которым доступна страница	Описание
/	Все	Главная страница
/login	Гость	Авторизация и аутентификация пользователя
/register	Гость	Регистрация пользователя
/about	Все	Просмотр информации о предоставляемых услугах
/contacts	Все	Контакты, как можно связаться
/dashboard/routes	Администратор	CRUD-интерфейс для маршрута
/dashboard/schedules	Администратор	CRUD-интерфейс для расписания
/dashboard/users	Администратор	Просмотр и блокировка пользователей
/dashboard/buses	Администратор	CRUD-интерфейс для транспорта
/dashboard/bus-schedules	Администратор	CRUD-интерфейс для расписания транспорта
/profile	Пассажир, Администратор	Просмотр информации о пользователе, редактирование информации и просмотр бронирования
/pending-bookings	Пассажир, Администратор	Список броней ожидающих подтверждения

В листинге 3.7 представлена компонента AppContent, которая содержит все страницы приложения

```
const AppContent: React.FC = () => {
  const { refreshToken, logout } = useAuth();

  useEffect(() => {
    setupAxiosInterceptors(refreshAccessToken, logout);
  }, [refreshAccessToken, logout]);
  //protected route для ролей
  return (
    <Router>
```

```

<Routes>
    <Route path="/login" element={<Login />} />
    <Route path="/register" element={<Register />} />
    <Route path="/" element={<HomeProvider><Home
/></HomeProvider>} />
    <Route path="/about" element={<AboutUs />} />
    <Route path="/contacts" element={<Contacts />} />
    <Route path="/401" element={<ErrorPage
statusCode={401} />} />
    <Route path="/403" element={<ErrorPage
statusCode={403} />} />
    <Route path="*" element={<ErrorPage statusCode={404}
/>} />
    <Route element={<AuthProtectedRoute />}>
        <Route element={<AdminProtectedRoute />}>
            <Route path="/dashboard/*"
element={<DashboardProvider><Dashboard /></DashboardProvider>} />
            </Route>
            <Route path="/profile"
element={<ProfileProvider><Profile /></ProfileProvider>} />
            <Route path="/pending-bookings"
element={<ProfileProvider> <PendingBookings /></ProfileProvider>} />
            </Route>
        </Routes>
    </Router>
);
};

```

Листинг 3.7 – Компонента AppContent

Компоненты `AuthProtectedRoute` и `AdminProtectedRoute` являются ключевыми элементами клиентской части приложения, обеспечивающими защиту маршрутов на основе статуса аутентификации и роли пользователя. Они реализованы как обертки для React-компонентов, что позволяет ограничивать доступ к определенным страницам приложения, усиливая безопасность и персонализацию пользовательского опыта. Эти компоненты интегрированы с системой маршрутизации на базе `React Router v7` и взаимодействуют с механизмом авторизации, использующим JWT-токены.

На листинге 3.8 представлен код компоненты `AdminProtected`

```

const AdminProtectedRoute: React.FC = () => {
    const { roleId, getRoleId } = useAuth();
    useEffect(() => {
        const checkAccess = async () => {
            await getRoleId();
        };
        checkAccess();
    }, []);
    if (roleId == 2) { return <Navigate to="/403" replace />; }
    return <Outlet />;
};

```

Листинг 3.8 – Компонента AdminProtectedRoute

В приложении Г представлен листинг кода компоненты PadingBooking, предназначенной для отображения не подтвержденных броней.

3.7 Web-сервер Nginx

Nginx — это мощный веб-сервер, широко применяемый как обратный прокси, балансировщик нагрузки или сервер для доставки статического контента. В приложении Д приведено содержимое конфигурационного файла Nginx.

В данном случае он выполняет несколько ключевых задач:

- обслуживание фронтенда: nginx раздаёт статические файлы фронтенда, которые находятся в `/var/www/tickets_booking/`, статические ресурсы (CSS, изображения) кешируются на год, что ускоряет повторные за-грузки страниц;
- проксирование API-запросов: запросы к `/api/` перенаправляются на бэкенд, работающий на `localhost:3001`, с корректной передачей заголовков;
- WebSocket-подключения: отдельный блок `/socket.io/` обеспечивает проксирование веб-сокетов, с длительным таймаутом (86400 секунд) и поддержкой обновления соединения;

3.8 Выводы по разделу

В ходе разработки web-приложения «Медицинский центр» был реализован комплексный технологический стек, охватывающий как серверную, так и клиентскую части приложения. Серверная часть построена на платформе Node.js с использованием фреймворка Express.js, что обеспечило высокую производительность и масштабируемость при обработке запросов. Выбор PostgreSQL в качестве СУБД позволил гарантировать надежное хранение конфиденциальных медицинских данных при поддержке сложных запросов и транзакций. Интеграция TypeORM значительно упростила взаимодействие с базой данных, предоставив удобный объектно-ориентированный интерфейс для работы с сущностями системы.

Для надежного взаимодействия клиентской и серверной частей внедрен веб-сервер Nginx, кеширование статических ресурсов, проксирование API-запросов и поддержку WebSocket для блокировки в реальном времени. REST API спроектирован с учетом бизнес-процессов медицинского центра, с логичной группировкой маршрутов по модулям. Механизм JWT-аутентификации с автообновлением токенов гарантирует безопасный доступ, а тщательная обработка ошибок и валидация данных повышают надежность. Клиентская часть на React с React Router 7v обеспечивает интуитивный интерфейс и ролевую маршрутизацию. Паттерн Repository поддерживает чистоту кода и разделение ответственности. Socket.io поддерживает real-time взаимодействие.

Выбранные технологии и архитектура, с Nginx как связующим звеном, создают масштабируемую основу для приложения, сочетая производительность, безопасность и удобство, что делает веб-приложение эффективным инструментом для автоматизации.

4 Тестирования web-приложения

4.1 Автоматизированное тестирование

В рамках курсового проектирования было проведено автоматизированное тестирование работы API с использованием фреймворка для тестирования jest (v29.7.0). Было создано 106 модульных тестов [30]. На рисунке представлен результат выполнения модульных тестов

```

PASS src/modules/auth/repository/repository.test.ts (5.876 s)

PASS src/modules/schedules/repository/repository.test.ts (5.907 s)
PASS src/modules/bookings/repository/repository.test.ts (5.947 s)
PASS src/modules/busschedules/repository/repository.test.ts (6.066 s)
PASS src/modules/auth/services/service.test.ts (6.111 s)
PASS src/modules/schedules/controller/controller.test.ts (6.638 s)
PASS src/modules/routes/controller/controller.test.ts (6.668 s)
PASS src/modules/tickets/controller/controller.test.ts (6.703 s)
PASS src/modules/buses/controller/controller.test.ts (6.706 s)
PASS src/modules/bookings/controller/controller.test.ts (6.713 s)
PASS src/modules/auth/conroller/authController.test.ts (6.729 s)
PASS src/modules/busschedules/controller/controller.test.ts (6.846 s)

Test Suites: 15 passed, 15 total
Tests:       106 passed, 106 total
Snapshots:   0 total
Time:        7.61 s, estimated 9 s
Ran all test suites.

```

Рисунок 4.1 – Результат выполнения всех тестов

Отчет работы тестов показан на рисунке 4.2

All files 80.16% Statements 475/595 46.34% Branches 38/82 76.11% Functions 382/504 80.74% Lines 336/546 Press n or j to go to the next uncovered block, b, p or k for the previous block. Filter: <input type="text"/>											
File	Statements	Branches	Functions	Lines							
config	100%	3/3	100%	0/0	100%	0/0	100%	0/0	100%	3/3	
modules/auth/controller	93.33%	28/30	50%	8/16	100%	4/4	92.85%	28/28			
modules/auth/entities	86.2%	25/29	100%	0/0	0%	0/4	91.3%	21/23			
modules/auth/repository	100%	19/19	100%	1/1	100%	9/9	100%	19/19			
modules/auth/services	100%	60/60	100%	11/11	100%	11/11	100%	51/51			
modules/bookings/controller	100%	19/19	100%	0/0	100%	6/6	100%	17/17			
modules/bookings/entities	76.47%	13/17	100%	0/0	0%	0/4	84.61%	11/13			
modules/bookings/repository	69.44%	25/36	90%	9/10	80%	8/10	71.42%	25/35			
modules/buses/controller	100%	17/17	100%	0/0	100%	5/5	100%	15/15			
modules/buses/entities	84.61%	11/13	100%	0/0	0%	0/2	90%	9/10			
modules/buses/repository	73.91%	17/23	20%	1/5	87.5%	7/8	76.19%	16/21			
modules/busschedules/controller	57.14%	16/28	0%	0/5	60%	3/5	53.84%	14/26			
modules/busschedules/entities	75%	12/16	100%	0/0	0%	0/4	83.33%	10/12			
modules/busschedules/repository	100%	18/18	100%	1/1	100%	6/6	100%	18/18			
modules/busschedules/service	18.18%	8/44	0%	0/12	25%	1/4	18.18%	8/44			
modules/routes/controller	94.73%	18/19	100%	0/0	83.33%	5/6	94.11%	18/17			
modules/routes/entities	86.86%	13/15	100%	0/0	0%	0/2	91.66%	11/12			
modules/routes/repository	100%	17/17	100%	1/1	100%	7/7	100%	17/17			
modules/schedules/controller	91.66%	22/24	0%	0/2	100%	5/5	90.9%	20/22			
modules/schedules/entities	75%	12/16	100%	0/0	0%	0/4	83.33%	10/12			
modules/schedules/repository	100%	18/18	100%	1/1	100%	7/7	100%	18/18			
modules/schedules/service	64.7%	22/34	0%	0/7	100%	4/4	64.7%	22/34			
modules/tickets/controller	100%	19/19	100%	0/0	100%	6/6	100%	17/17			

Рисунок 4.2 –Отчет работы всех тестов

Из рисунка можно увидеть, что почти все модули одинаково протестированы, общее покрытие составляет 80%. В приложении Е представлена реализация тестов для ScheduleController.

4.2 Ручное тестирование

В рамках курсового проекта также было проведено было проведено комплексное ручное тестирование функционала, доступного пользователям с разными ролями: «Гость», «Пассажир» и «Администратор». Тестирование охватило ключевые сценарии взаимодействия с приложением, включая регистрацию, авторизацию, бронирование билета, добавление нового маршрута, блокировка пользователя. Тестирование проводилось для проверки соответствия функциональным требованиям, правильности обработки данных, корректности отображения интерфейса и функционирования уведомлений. Для каждого тестового сценария были заданы предусловия, шаги выполнения, ожидаемый и фактический результаты. Итоги тестирования приведены в таблице 4.1, где подробно описаны все проверенные сценарии и их результаты.

Таблица 4.1 – Описание проведенных тестов

Функция	Предусловие	Порядок действий	Ожидаемый результат	Полученный результат
Регистрация	Пользователь не авторизован, находясь на странице регистрации.	Ввести данные в поля формы: <ul style="list-style-type: none"> – в поле Фамилия: Розель – в поле Имя: Станислав – в поле Отчество: Александрович – в поле Email: example@gmail.com – в поле Пароль: 123456 – в поле Подтвердите пароль: 123456 	Редирект на главную страницу, красная кнопка войти сменилась на выйти	Редирект на главную страницу, красная кнопка войти сменилась на выйти
Авторизация	Пользователь не авторизован, находясь на странице авторизации	Ввести данные в поля формы: <ul style="list-style-type: none"> – в поле Email: example@gmail.com – в поле Пароль: 123456 	Редирект на главную (/) страницу, красная кнопка войти сменилась на выйти	Редирект на главную страницу, красная кнопка войти сменилась на выйти

Окончание таблицы 4.1

Функция	Предусловие	Порядок действий	Ожидаемый результат	Полученный результат
Добавление нового маршрута	Пользователь авторизован с ролью Администратор и находится на странице с работой маршрутов	Нажать на кнопку добавить маршрут. Появляется форма и в нее вводим следующие данные: <ul style="list-style-type: none"> – в поле Имя маршрута: Минск – Витебск – в поле Пункт отправления: Минск – в поле Пункт назначения: Витебск – в поле Промежуточные остановки: Борисов, Орша – в поле Расстояние: 290 – в поле Цена: 40 	В списке маршрутов появился новый маршрут	В списке маршрутов появился новый маршрут
Бронирование билета	Пользователь авторизован с ролью Пассажир, находится на главной странице	На главной странице пользователь выбирает билет и нажимает на кнопку выбрать, его редиректит на страницу с бронями для подтверждения, нажимает на кнопку забронировать	Бронь из ожидающих пропадает, теперь она находится в профиле в истории бронирования и с зеленым фоном	Бронь из ожидающих пропадает, теперь она находится в профиле в истории бронирования и с зеленым фоном
Блокировка пользователя	Пользователь авторизован с ролью Администратор и находится на странице просмотра пользователей	Находясь на странице с пользователями, пользователь нажимает на галочку, появляется окно с подтверждением, он подтверждает действия	Около галочки надпись должна смениться на разблокировать, а статус на заблокирован	Около галочки надпись должна смениться на разблокировать, а статус на заблокирован

Комплексное ручное тестирование веб-приложения «Бронирование билетов для поездки между городами на маршрутном такси» подтвердило высокое соответствие реализованного функционала требованиям и техническому заданию. Проверены все основные модули системы, включая регистрацию пользователей, аутентификацию и авторизацию, бронирования билета, добавлением нового маршрута и блокировка пользователя.

4.3 Выводы по разделу

В рамках курсового проекта была проведена комплексная проверка работоспособности веб-приложения «Бронирование билетов для поездки между городами на маршрутном такси», включая ручное и автоматизированное тестирование. Ручное тестирование подтвердило корректность основных сценариев для ролей гостя, пассажира, администратора, включая регистрацию пользователя, аутентификацию и авторизацию, бронирование билета, добавление нового маршрута и блокировку пользователя, с акцентом на взаимодействие между ролями и соблюдение бизнес-логики.

Автоматизированное тестирование API с использованием фреймворка Jest (версия 29.7.0) показало высокую надежность системы: выполнено 106 интеграционных тестов, проверяющих взаимодействие компонентов. Уровень покрытия кода тестами составил 80%, с полным покрытием маршрутов и 74.86% для контроллеров; репозитории тестировались с использованием mock-объектов.

В ходе тестирования не было выявлено ни критических, ни значительных ошибок в работе приложения. Все функциональные модули продемонстрировали стабильную работу в рамках предусмотренных сценариев использования. Полученные результаты позволяют сделать вывод о высокой степени готовности приложения к эксплуатации в реальных условиях веб-приложения для бронирования билетов.

5 Руководство пользователя

5.1 Руководство гостя

При открытии веб-приложения новый пользователь попадает на главную страницу веб-приложения. Вид главной страницы представлен на рисунке 5.1

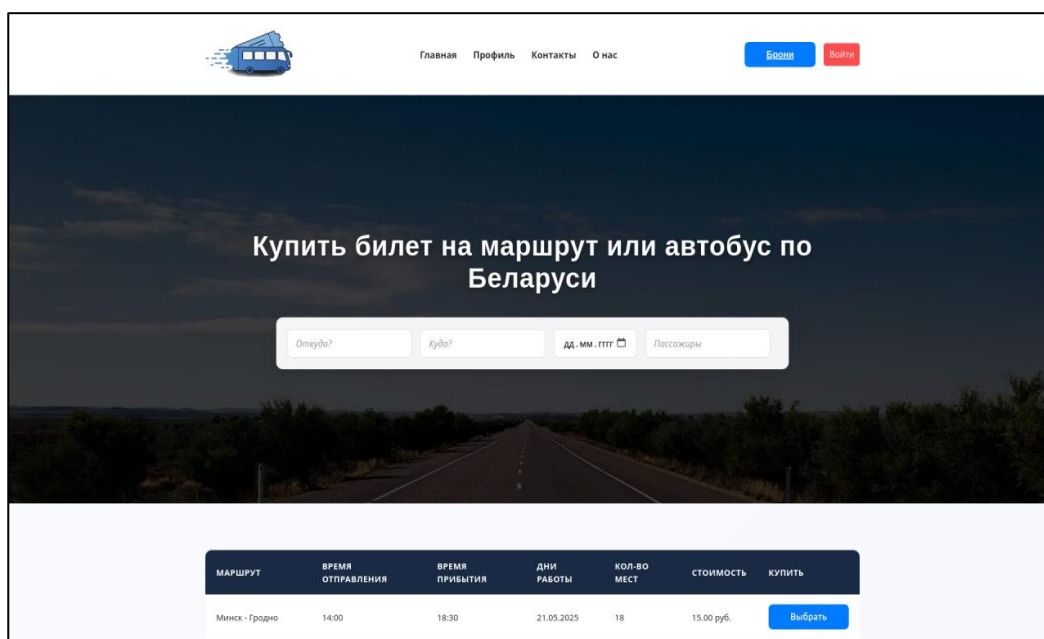


Рисунок 5.1 – Вид главной страницы веб-приложения

Используя верхнюю панель навигации, гость может переходить между страницами приложения. Доступ к разделам «Контакты», «О нас» и «Профиль» открывается только после авторизации. Для входа пользователю нужно заполнить поля «Логин» и «Пароль» в форме и нажать кнопку «Войти», после чего система определит роль пользователя. Интерфейс формы авторизации показан на рисунке 5.2.

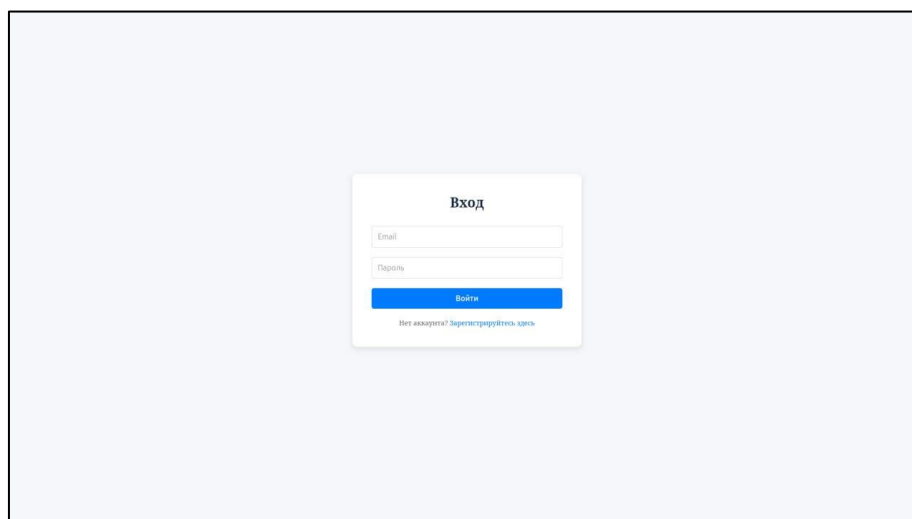
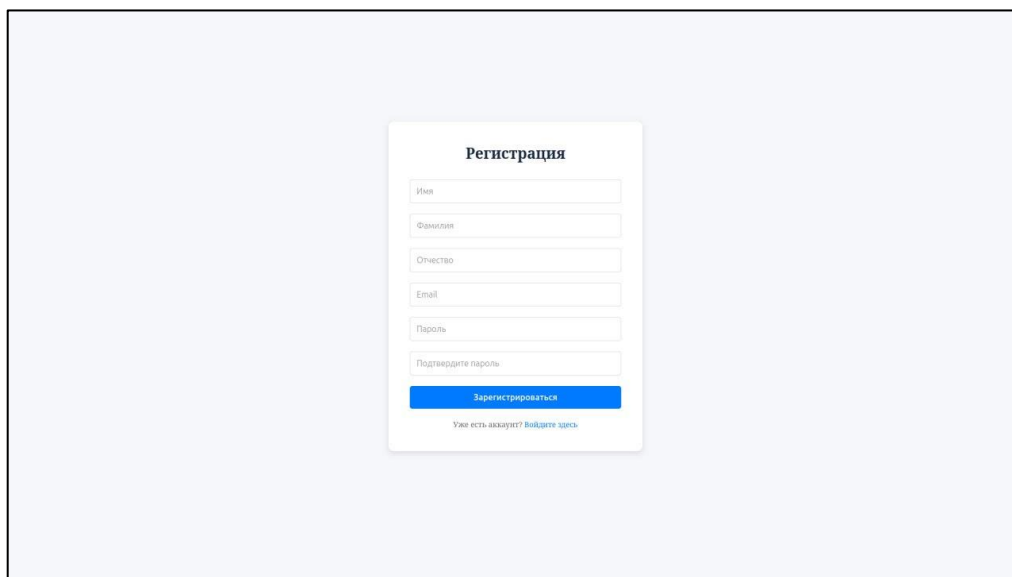


Рисунок 5.2 – Вид страницы для авторизации

Также гостю доступна регистрация. На форму регистрации можно попасть со страницы с формой входа в приложение кликнув по ссылке «Зарегистрироваться». Вид формы регистрации представлен на рисунке 5.3.



Регистрация

Имя

Фамилия

Отчество

Email

Пароль

Подтвердите пароль

[Зарегистрироваться](#)

[Уже есть аккаунт? Войдите здесь](#)

Рисунок 5.3 – Вид страницы для авторизации

Так же пользователь может просмотреть контакты для связи с поставщиком услуг. Вид страницы контактов представлена на рисунке 5.4

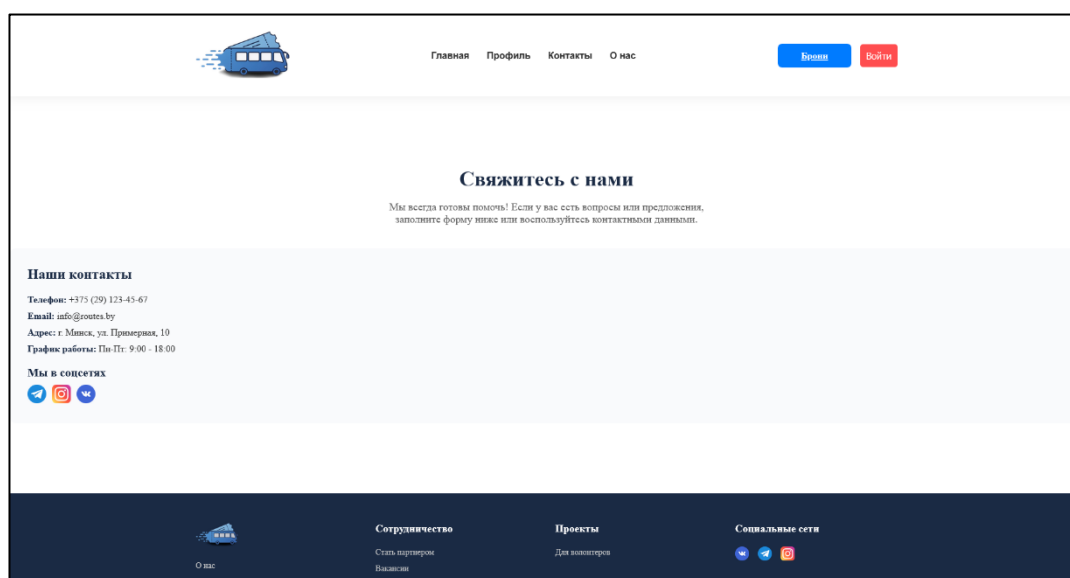


Рисунок 5.4 – Вид страницы контактов

Веб-приложение ещё содержит информационную страницу, на которой описано, на что нацелена компания и общая информация. Вид страницы представлен на рисунке 5.5.

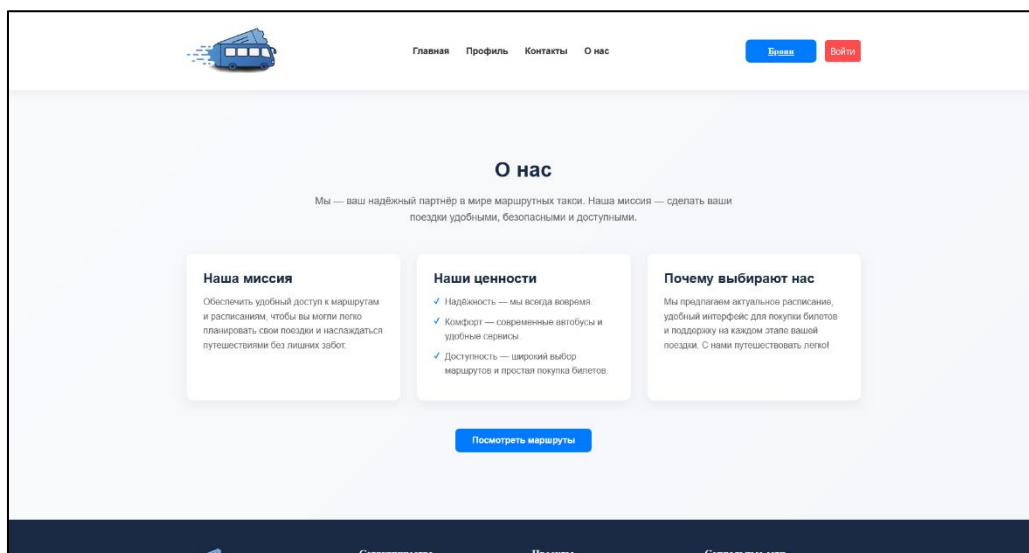


Рисунок 5.5 – Вид страницы с общей информацией

С данной страницы гость может вернуться обратно на главную, нажав на кнопку «Просмотреть маршруты»

5.2 Руководство пассажира

Пассажир так же как и гость может пройти по всем базовым страницам, которые были представлены в руководстве гостя. Пассажир же может просмотреть и редактировать профиль, а так же ознакомиться с своей историей бронирования. Вид страницы профиля представлен на рисунке 5.6.

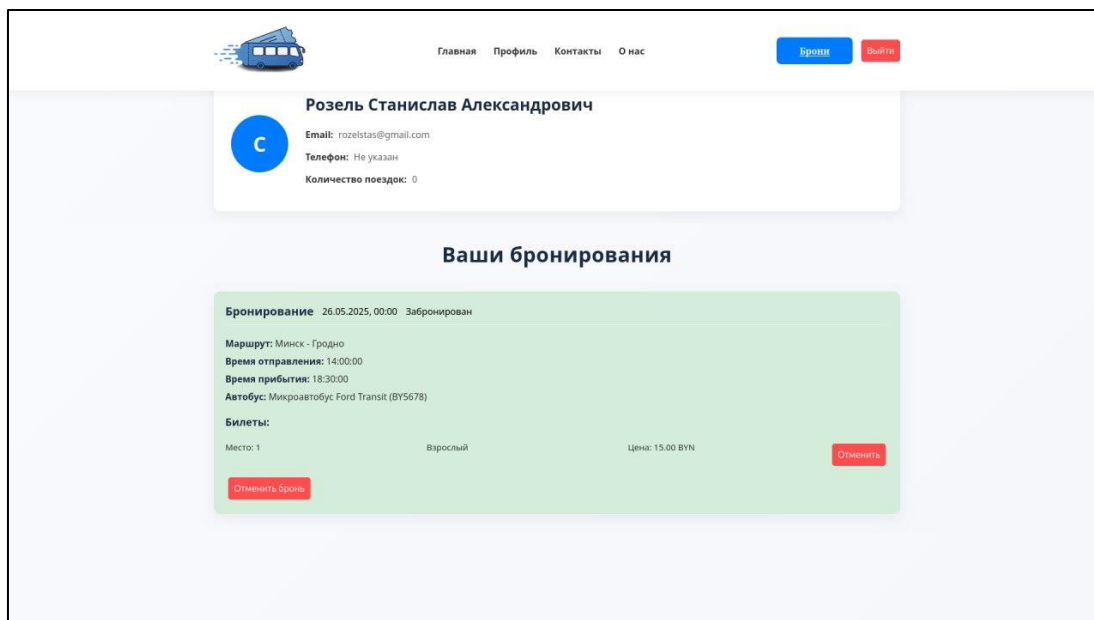
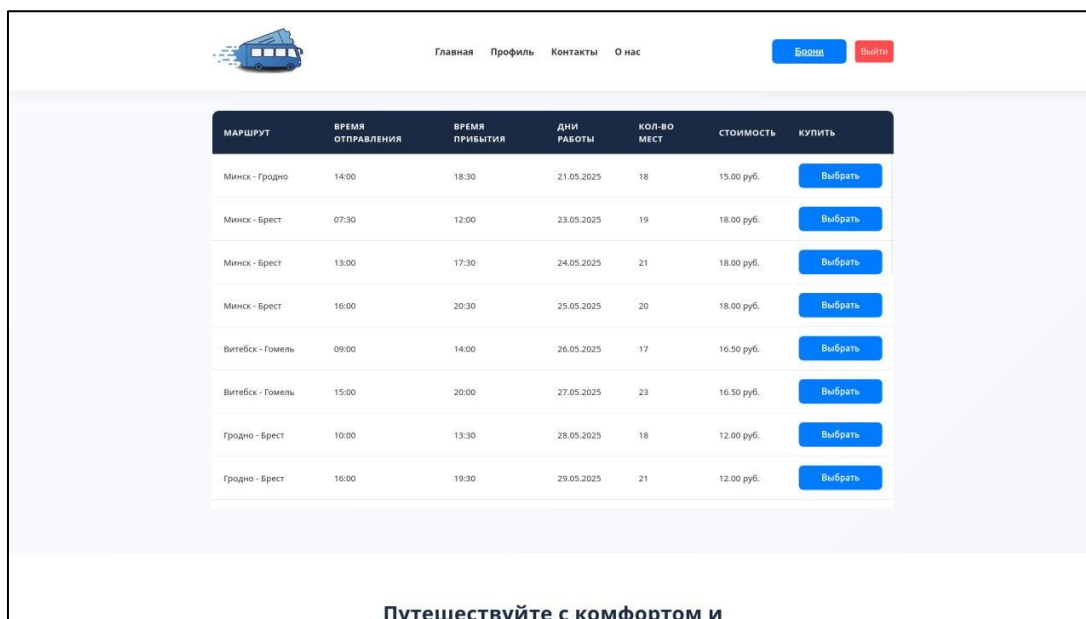


Рисунок 5.6 – Вид страницы профиля

Пассажир может забронировать билет, для этого сначала он смотрит список доступных маршрутов. Таблица маршрутов представлена на рисунке 5.7

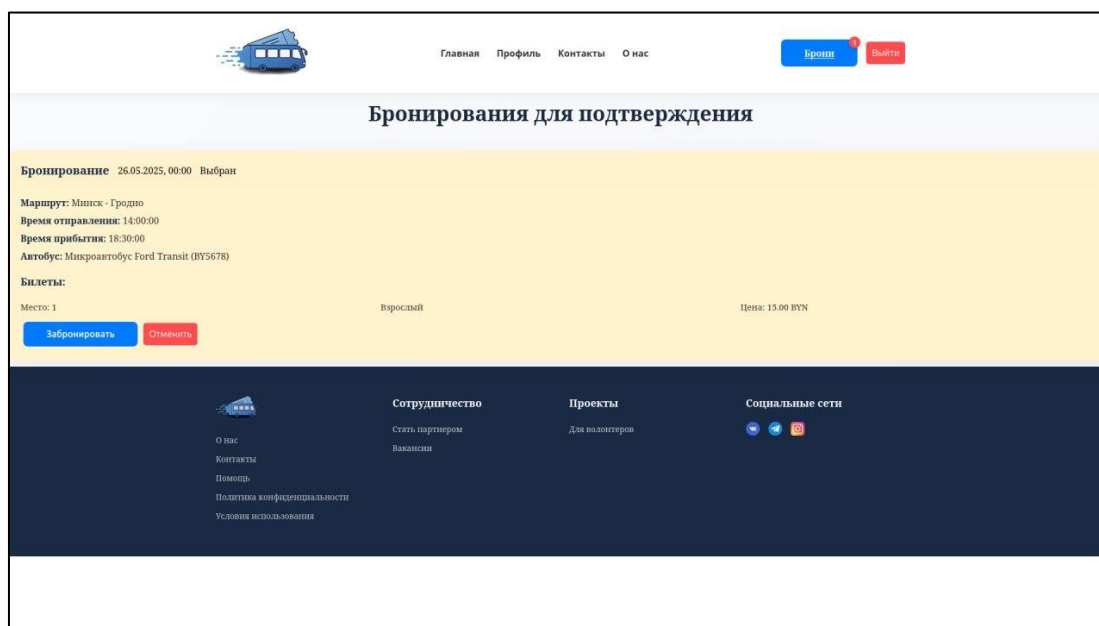


МАРШРУТ	ВРЕМЯ ОТПРАВЛЕНИЯ	ВРЕМЯ ПРИВЫТИЯ	ДНИ РАБОТЫ	КОЛ-ВО МЕСТ	СТОИМОСТЬ	КУПИТЬ
Минск - Гродно	14:00	18:30	21.05.2025	18	15.00 руб.	Выбрать
Минск - Брест	07:30	12:00	23.05.2025	19	18.00 руб.	Выбрать
Минск - Брест	13:00	17:30	24.05.2025	21	18.00 руб.	Выбрать
Минск - Брест	16:00	20:30	25.05.2025	20	18.00 руб.	Выбрать
Витебск - Гомель	09:00	14:00	26.05.2025	17	16.50 руб.	Выбрать
Витебск - Гомель	15:00	20:00	27.05.2025	23	16.50 руб.	Выбрать
Гродно - Брест	10:00	13:30	28.05.2025	18	12.00 руб.	Выбрать
Гродно - Брест	16:00	19:30	29.05.2025	21	12.00 руб.	Выбрать

Путешествуйте с комфортом и

Рисунок 5.7 – Вид таблицы маршрутов

После того как пассажир выбрал билет и нажал на кнопку, его перемещает на страницу с бронированиями, которые ждут подтверждения. Вид страницы с бронированиями представлен на рисунке 5.8.



Бронирования для подтверждения

Бронирование 26.05.2025, 00:00 **Выбран**

Маршрут: Минск - Гродно
Время отправления: 14:00:00
Время прибытия: 18:30:00
Автобус: Микроавтобус Ford Transit (BY3678)

Билеты:
 Место: 1 Взрослый Цена: 15.00 BYN

[Забронировать](#) [Отменить](#)

Сотрудничество
 Стать партнером
 Вакансии

Проекты
 Для волонтеров

Социальные сети
[Facebook](#) [Twitter](#) [Instagram](#)

О нас
 Контакты
 Помощь
 Политика конфиденциальности
 Условия использования















Рисунок 5.8 – Вид страницы с бронированием

Таким образом, роль пассажира играет ключевую роль в бронировании билетов, что позволяет собирать заинтересованную аудиторию.

5.3 Руководство администратора

Администратор так же как и пассажир авторизуется и попадает на страницу с настройкой данных о маршрутах, транспорте, расписании и пользователях.















Администратору доступна возможность добавлять и удалять маршруты, транспорт и расписание. Вид страницы с редактированием маршрутов представлена на рисунке 5.9

ID	Название маршрута	Маршрут	Промежуточные остановки	Расстояние	Стоимость	Действия
1	Минск - Гродно	Минск → Гродно	Лида, Щучин	280.00 км	15.00 руб.	 
2	Минск - Брест	Минск → Брест	Барановичи, Кобрин	340.00 км	18.00 руб.	 
3	Витебск - Гомель	Витебск → Гомель	Орша, Могилёв	320.00 км	16.50 руб.	 
4	Гродно - Брест	Гродно → Брест	Валковский, Пружаны	230.00 км	12.00 руб.	 
5	Минск - Витебск	Минск → Витебск	Борисов, Орша	250.00 км	13.50 руб.	 
6	Минск - Могилёв	Минск → Могилёв	Борисов, Березино	200.00 км	10.50 руб.	 
7	Брест - Пинск	Брест → Пинск	Ивацевичи, Дрогичин	140.00 км	8.00 руб.	 

Добавить маршрут

Рисунок 5.9 – Вид страницы с редактированием маршрутов

Как говорилось ранее администратор может добавлять разные сущности. На рисунке 5.10 представлена форма добавления маршрута.

ID	Название маршрута	Маршрут	Промежуточные остановки	Расстояние	Стоимость	Действия
1	Минск - Гродно	Минск → Гродно	Лида, Щучин	280.00 км	15.00 руб.	 
2				340.00 км	18.00 руб.	 
3				320.00 км	16.50 руб.	 
4				230.00 км	12.00 руб.	 
5				250.00 км	13.50 руб.	 
6				200.00 км	10.50 руб.	 
7				140.00 км	8.00 руб.	 

Добавить маршрут

Рисунок 5.10 – Вид страницы с формой добавления маршрута

Так же администратор может изменять информацию о сущностях. На рисунке 5.11 представлена форма изменения маршрута.

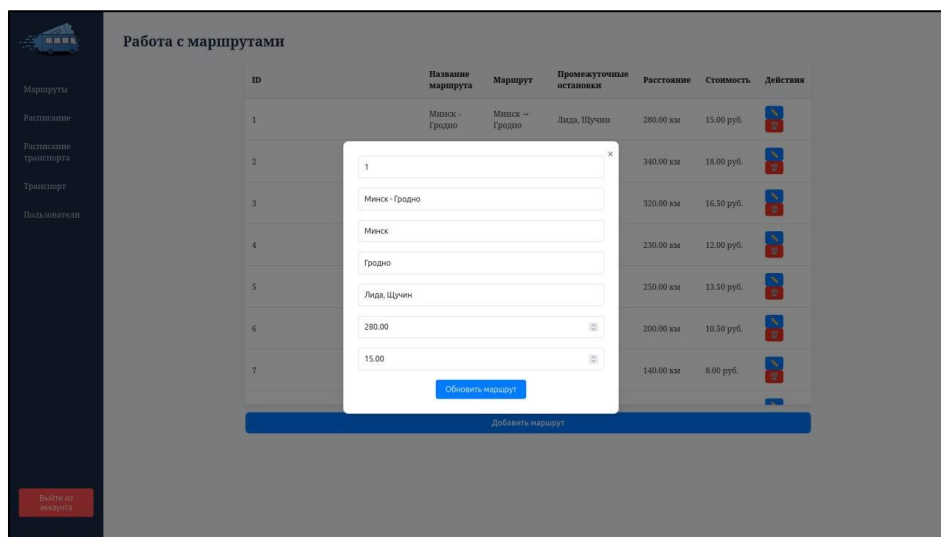


Рисунок 5.10 – Вид страницы с формой изменения маршрута

Ещё одна из возможностей администратора это блокировка и разблокировка пользователя. Вид страницы представлен на рисунке 5.11

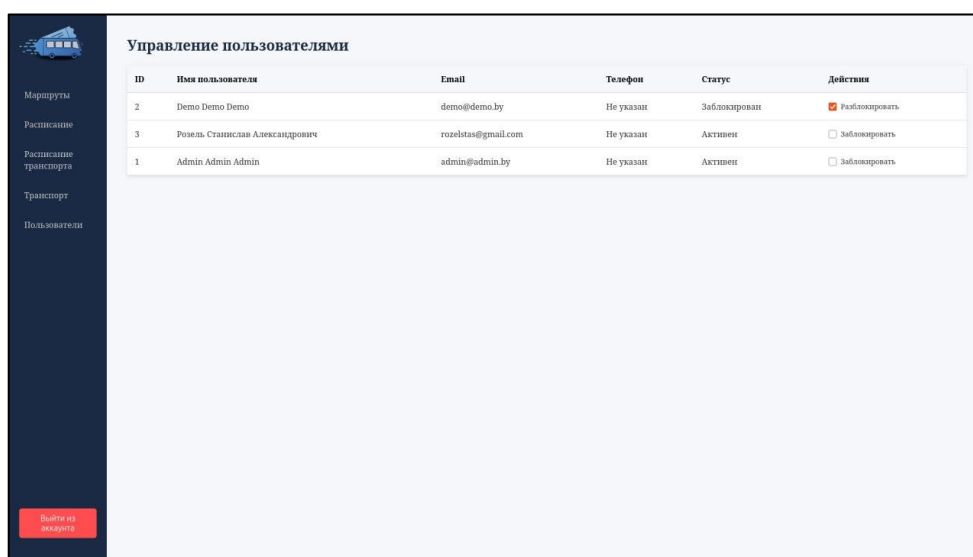


Рисунок 5.11 – Вид страницы с пользователями

Таким образом, было разобраны возможности каждой из ролей и каким образом ими можно воспользоваться.

5.4 Выводы по разделу

Разработанное руководство пользователя представляет собой полное и описание работы с приложения для всех категорий пользователей. Документация охватывает все главные сценарии взаимодействия с системой, начиная от базовых операций для гостей и заканчивая административными функциями управления системой. Руководство структурировано по ролям пользователей, что значительно облегчает навигацию и поиск необходимой информации для каждой категории

Заключение

В рамках курсового проекта было создано полнофункциональное веб-приложение для бронирования билетов на межгороднее маршрутное такси, полностью соответствующее заданным требованиям и техническому заданию. Приложение построено на современном стеке технологий с использованием TypeScript и платформы Node.js, что гарантирует высокую производительность, надежность и масштабируемость системы.

Для управления данными использовалась СУБД PostgreSQL, в рамках которой была спроектирована база данных из 7 таблиц. Для взаимодействия с базой данных применялась ORM-библиотека TypeORM. Было реализовано 46 методов контроллеров для обработки входящих API-запросов и разработан один middleware-обработчик.

Приложение успешно реализует функционал для четырёх категорий пользователей: гостей, пассажиров, водителей и администраторов. Архитектура системы основана на разделении слоёв представления, бизнес-логики и хранения данных, что обеспечивает её гибкость и возможность дальнейшего развития. Особое внимание уделялось асинхронной обработке запросов и разработке удобного пользовательского интерфейса.

В реализации применялись современные подходы к разработке, включая принципы SOLID, паттерны проектирования и асинхронное программирование. Код приложения снабжён подробными комментариями, что упрощает его поддержку и дальнейшее расширение.

Проведённое тестирование (ручное и автоматизированное) подтвердило стабильность всех функциональных модулей. Уровень покрытия кода тестами достиг 80%, при этом ключевые компоненты (маршруты и контроллеры) протестированы более чем на 74%. Разработанное руководство пользователя включает полные инструкции по использованию приложения для всех ролей.

Итогом проекта стало готовое программное решение, пригодное для внедрения в реальных условиях транспортных компаний. Приложение отличается интуитивным интерфейсом, надёжной работой и продуманной архитектурой, позволяющей легко добавлять новые функции. Все цели проекта достигнуты в установленные сроки.

Список используемых источников

1. TypeScript Documentation [Электронный ресурс]. / Режим доступа: <https://www.typescriptlang.org/docs/>. – Дата доступа: 20.02.2025.
2. Node.js Documentation [Электронный ресурс]. / Режим доступа: <https://nodejs.org/en/docs>. – Дата доступа: 20.02.2025.
3. Atlas[Ресурс atlasbus.by]. / Режим доступа: <https://atlasbus.by/>. – Дата доступа: 10.03.2025.
4. 7588.by [Ресурс 7588.by]. / Режим доступа: <https://7588.by/>. – Дата доступа: 10.03.2025.
5. Ubuntu Server Documentation [Электронный ресурс]. / Режим доступа: <https://ubuntu.com/server/docs>. –
6. Nginx Official Documentation [Электронный ресурс]. / Режим доступа: <https://nginx.org/en/docs/>. – Дата доступа: 10.03.2025.
7. RFC 2616: Hypertext Transfer Protocol [Электронный ресурс]. / Режим доступа: <https://datatracker.ietf.org/doc/html/rfc2616>. – Дата доступа: 10.03.2025.
8. RFC 6455: The WebSocket Protocol [Электронный ресурс]. / Режим доступа: <https://datatracker.ietf.org/doc/html/rfc6455>. – Дата доступа: 10.03.2025.
9. Express.js Documentation [Электронный ресурс]. / Режим доступа: <https://expressjs.com/>. – Дата доступа: 10.03.2025.
10. React Documentation [Электронный ресурс]. / Режим доступа: <https://react.dev/>. – Дата доступа: 23.03.2025.
11. React Documentation [Электронный ресурс]. / Режим доступа: <https://react.dev/>. – [Ресурс Create React App (CRA)]. / Режим доступа: <https://create-react-app.dev/>. – Дата доступа: 23.03.2025.
12. PostgreSQL 17 Documentation [Электронный ресурс]. / Режим доступа: <https://www.postgresql.org/docs/17/>. – Дата доступа: 10.03.2025.
13. TypeORM [Ресурс TypeORM]. / Режим доступа: <https://typeorm.io/>. – Дата доступа: 10.03.2025.
14. bcrypt npm Documentation [Электронный ресурс]. / Режим доступа: <https://www.npmjs.com/package/bcrypt>. – Дата доступа: 10.03.2025.
15. JWT Official Documentation [Электронный ресурс]. / Режим доступа: <https://jwt.io/introduction/>. – Дата доступа: 10.03.2025.
16. pg npm Documentation [Электронный ресурс]. / Режим доступа: <https://node-postgres.com/>. – Дата доступа: 10.03.2025.
17. cors npm Documentation [Электронный ресурс]. / Режим доступа: <https://www.npmjs.com/package/cors>. – Дата доступа: 10.03.2025.
18. socket.io Documentation [Электронный ресурс]. / Режим доступа: <https://socket.io/docs/v4/>. – Дата доступа: 23.03.2025.
19. routing-controllers npm Documentation [Электронный ресурс]. / Режим доступа: <https://www.npmjs.com/package/routing-controllers>. – Дата доступа: 23.03.2025.
20. body-parser npm Documentation [Электронный ресурс]. / Режим доступа: <https://www.npmjs.com/package/body-parser>. – Дата доступа: 23.03.2025.
21. dotenv npm Documentation [Электронный ресурс]. / Режим доступа: <https://www.npmjs.com/package/dotenv>. – Дата доступа: 10.03.2025.

22. jest Documentation [Электронный ресурс]. / Режим доступа: <https://jestjs.io/docs/getting-started>. – Дата доступа: 20.04.2025.
23. React Router Documentation [Электронный ресурс]. / Режим доступа: <https://reactrouter.com/en/main>. – Дата доступа: 23.03.2025.
24. Sass Documentation [Электронный ресурс]. / Режим доступа: <https://sass-lang.com/documentation/>. – Дата доступа: 23.03.2025.
25. Zod Documentation [Электронный ресурс]. / Режим доступа: <https://zod.dev/>. – Дата доступа: 23.03.2025.
26. socket.io-client Documentation [Электронный ресурс]. / Режим доступа: <https://socket.io/docs/v4/client-api/>. – Дата доступа: 23.03.2025.
27. react-dom Documentation [Электронный ресурс]. / Режим доступа: <https://react.dev/reference/react-dom>. – Дата доступа: 23.03.2025.
28. REST API Tutorial [Электронный ресурс]. / Режим доступа: <https://restfulapi.net/>. – Дата доступа: 26.02.2025.
29. Express Middleware Documentation [Электронный ресурс]. / Режим доступа: <https://expressjs.com/en/guide/using-middleware.html>. – Дата доступа: 10.03.2025.
30. Integration Testing Guide [Электронный ресурс]. / Режим доступа: <https://www.guru99.com/integration-testing.html>. – Дата доступа: 12.04.2025.

Приложение А

```

CREATE TABLE Roles (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);

CREATE TABLE Routes (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    starting_point VARCHAR(100) NOT NULL,
    ending_point VARCHAR(100) NOT NULL,
    stops TEXT,
    distance DECIMAL(10, 2),
    price DECIMAL(10, 2) NOT NULL
);

CREATE TABLE Buses (
    id SERIAL PRIMARY KEY,
    bus_number VARCHAR(20) NOT NULL,
    capacity INTEGER[] NOT NULL,
    type VARCHAR(50),
    available BOOLEAN DEFAULT TRUE
);

CREATE TABLE Schedules (
    id SERIAL PRIMARY KEY,
    route_id INTEGER NOT NULL,
    departure_time TIME NOT NULL,
    arrival_time TIME NOT NULL,
    FOREIGN KEY (route_id) REFERENCES Routes(id) ON
DELETE CASCADE
);

CREATE TABLE Users (
    id SERIAL PRIMARY KEY,
    role_id INTEGER NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    middle_name VARCHAR(100) NOT NULL,
    phone_number VARCHAR(20) DEFAULT '',
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(100) NOT NULL,
    count_trips INTEGER DEFAULT 0,
    is_blocked BOOLEAN DEFAULT FALSE,
    refresh_token VARCHAR(255),
    FOREIGN KEY (role_id) REFERENCES Roles(id)
);

CREATE TABLE BusSchedules (
    id SERIAL PRIMARY KEY,
    schedule_id INTEGER NOT NULL,
    bus_id INTEGER NOT NULL,
    operating_days DATE,
    FOREIGN KEY (schedule_id) REFERENCES
Schedules(id) ON DELETE CASCADE,
    FOREIGN KEY (bus_id) REFERENCES Buses(id)
);

CREATE TABLE Bookings (

```

```

        id SERIAL PRIMARY KEY,
            bus_schedule_id INTEGER NOT NULL,
            user_id INTEGER NOT NULL,
            booking_date          TIMESTAMP          DEFAULT
CURRENT_TIMESTAMP,
            status VARCHAR(20) NOT NULL,
            FOREIGN KEY (bus_schedule_id) REFERENCES
BusSchedules(id),
            FOREIGN KEY (user_id) REFERENCES Users(id)
        );
CREATE TABLE Tickets (
        id SERIAL PRIMARY KEY,
        booking_id INTEGER NOT NULL,
        seat_number INTEGER NOT NULL,
        is_child BOOLEAN DEFAULT FALSE,
        price DECIMAL(10, 2) NOT NULL,
        FOREIGN KEY (booking_id) REFERENCES Bookings(id)
    );

```

Листинг – Скрипт создания базы данных

Приложение Б

```

import express from 'express';
import { useExpressServer } from 'routing-controllers';
import { AppDataSource } from './src/config/db.config';
import bodyParser from 'body-parser';
import cors from 'cors';
import http from 'http';

import { RouteController } from './src/modules/routes/controller/controller';
import { ScheduleController } from './src/modules/schedules/controller/controller';
import { AuthController } from './src/modules/auth/controller/authController';
import { authMiddleware } from './src/shared/middlewares/auth';
import { BusController } from './src/modules/buses/controller/controller';
import { BusScheduleController } from './src/modules/busschedules/controller/controller';
import { UserController } from './src/modules/auth/controller/userController';
import { BookingController } from './src/modules/bookings/controller/controller';
import { TicketController } from './src/modules/tickets/controller/controller';
import { Server } from 'socket.io';
require('dotenv');

const app = express();
const server = http.createServer(app);

const HTTP_PORT = process.env.HTTP_PORT;
const SOCK_PORT = process.env.SOCK_PORT;

app.use(bodyParser.json());
app.use(cors());

useExpressServer(app, {
  controllers: [RouteController, ScheduleController,
AuthController, BusController, BusScheduleController,
UserController, BookingController, TicketController],
  authorizationChecker: async (action) => {
    const openPaths = [
      '/auth/refresh/',
      '/auth/login/',
      '/auth/register/'
    ];
    if (openPaths.some(path
action.request.path.startsWith(path))) {
      return true;
    }
  }
});

```

```

    const token = action.request.headers.authorization?.split('
')[1];
    if (!token) return false;

    try {
      await authMiddleware(action.request, action.response, ()
=> { });
      return true;
    } catch (error) {
      return false;
    }
  },
  });

  export const io = new Server(server, {
    cors: {
      origin: "*",
    }
  });

  AppDataSource.initialize()
    .then(() => {
      console.log("\x1b[32m", "Database initialize successfully")
    })
    .catch((error) => console.log(error))

  app.listen(HTTP_PORT, () => {
    console.log("\x1b[32m", `Server running on
http://localhost:${HTTP_PORT}`);
  });

  import './src/shared/socketHandlers';

  server.listen(SOCK_PORT, () => {
    console.log(`\x1b[32m`, `Sockets running on port
${SOCK_PORT}`);
  });

```

Листинг – Содержимое файла index.ts

Приложение В

```

import { Repository, DataSource } from "typeorm";
import { AppDataSource } from "../../../../../config/db.config";
import { IRepository } from "../../shared/interfaces/IRepository";
import { Schedule } from "../../entities/Schedule";

export class ScheduleRepository implements IRepository<Schedule>
{
    private repository: Repository<Schedule>;
    constructor(dataSource: DataSource) {
        this.repository = dataSource.getRepository(Schedule);
    }

    async create(data: Partial<Schedule>): Promise<Schedule> {
        const Schedule = this.repository.create(data);
        return await this.repository.save(Schedule);
    }

    async findById(id: number): Promise<Schedule | null> {
        return await this.repository.findOneBy({ id });
    }

    async findAll(): Promise<Schedule[]> {
        return await this.repository.find();
    }

    async update(id: number, data: Partial<Schedule>):
Promise<Schedule | null> {
        const Schedule = await this.findOneById(id);
        if (!Schedule) {
            return null;
        }
        await this.repository.update(id, data);
        return await this.findOneById(id);
    }

    async delete(id: number): Promise<boolean> {
        const result = await this.repository.delete(id);
        return result.affected !== 0;
    }

    async deleteByRouteId(route_id: number): Promise<number> {
        const result = await this.repository.delete({ route: { id:
route_id } });
        return result.affected;
    }
}

export const scheduleRepository = new
ScheduleRepository(AppDataSource);

```

Листинг – Реализация класса ScheduleRepository

Приложение Г

```

import React, { useEffect, useState } from 'react';
import '../styles/css/BookingList.css';
import Footer from '../../shared/components/Footer';
import Header from '../../shared/components/Header';
import { useProfile } from '../context/ProfileContext';
import ConfirmModal from '../../shared/components/ConfirmModal';
import { useModal } from '../context/ModalContext';

const PendingBookings: React.FC = () => {

  const { bookings, tickets, fetchPendingBookings,
    loading, error, formatDate, handleTicketTypeChange,
    trigger, handleCancelBooking, handleBooking } =
useProfile();

  const { modalMessage, isModalOpen, openModal, handleModalClose
} = useModal();

  useEffect(() => {
    fetchPendingBookings();
  }, [trigger]);

  if (loading) {
    return (
      <div className="app">
        <Header />
        <main>
          <div className="booking-
list__loading">Загрузка...</div>
        </main>
        <Footer />
      </div>
    );
  }

  if (error) {
    return (
      <div className="app">
        <Header />
        <main>
          <div className="booking-list__error">{error}</div>
        </main>
        <Footer />
      </div>
    );
  }

  if (bookings.length === 0) {
    return (
      <div className="app">

```



```

{tickets[booking.id]?.length ? (
  <ul>
    {tickets[booking.id].map((ticket) => (
      <li
        key={ticket.id}
className="ticket-item">
        <span>Место:
{ticket.seat_number}</span>
        <span
          className="ticket-type"
          onClick={() =>
handleTicketTypeChange(ticket.id
                           as
                           number,
booking.busSchedule?.schedule?.route_id as number, booking.id,
ticket.is_child)
          }
        >
        {ticket.is_child ? 'Детский' :
'Взрослый' }
        </span>
        <span>Цена: {ticket.price}
BYN</span>
      </li>
    )})
  </ul>
) : (
  <p>Билеты не найдены.</p>
)}
</div>
<div className="booking-item__actions">
  <button
    className="button__action"
    onClick={() =>handleBooking(booking.id)}
  >
    Забронировать
  </button>
  <button
    className="button__cansel"
    onClick={() =>
      openModal(
        'Вы уверены, что хотите отменить это
бронирование?',
        () => handleCanselBooking(booking.id)
      )
    }
  >
    Отменить
  </button>
</div>
</div>
)}}
</div>
</div>
</section>
</main>

```

```
        <Footer/>
        <ConfirmModal
          isOpen={isModalOpen}
          onClose={handleModalClose}
          message={modalMessage}
        />
      </div>
    );
  };

  export default PendingBookings;
```

Листинг – Реализация страницы подтверждение бронирований

Приложение Д

```
server {
    listen 80;
    server_name tickets-booking.by;

    root /var/www/tickets_booking/;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html =404;
    }

    location /api/ {
        proxy_pass http://localhost:3001;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;
    }

    location /socket.io/ {
        proxy_pass http://localhost:4029;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_read_timeout 86400;
    }
}
```

Листинг – Файл конфигурации nginx

Приложение E

```

import express from 'express';
import { useExpressServer } from 'routing-controllers/types';
import request from 'supertest';
import { ScheduleController } from './controller';
import * as scheduleRepo from '../repository/repository';

jest.mock('../repository/repository');

describe('ScheduleController', () => {
  let app: express.Express;

  beforeAll(() => {
    app = express();
    app.use(express.json());
    useExpressServer(app, {
      controllers: [ScheduleController],
    });
  });

  afterEach(() => {
    jest.clearAllMocks();
  });

  it('GET /schedules/ должен возвращать все расписания', async () => {
    const schedules = [{ id: 1, name: 'Schedule1' }, { id: 2, name: 'Schedule2' }];
    (scheduleRepo.scheduleRepository.findAll as
    jest.Mock).mockResolvedValue(schedules);
    const res = await request(app).get('/schedules/');
    expect(res.status).toBe(200);
    expect(res.body).toEqual(schedules);

    expect(scheduleRepo.scheduleRepository.findAll).toHaveBeenCalled();
  });

  it('GET /schedules/:id должен возвращать расписание по id', async
  () => {
    const schedule = { id: 1, name: 'Schedule1' };
    (scheduleRepo.scheduleRepository.findOneById as
    jest.Mock).mockResolvedValue(schedule);
    const res = await request(app).get('/schedules/1');
    expect(res.status).toBe(200);
    expect(res.body).toEqual(schedule);

    expect(scheduleRepo.scheduleRepository.findOneById).toHaveBeenCalled
    With(1);
  });

  it('POST /schedules/create/ должен создавать расписание', async ()
  => {
    const newSchedule = { name: 'Schedule1' };
    const createdSchedule = { ...newSchedule, id: 1 };
  });

```

```

        (scheduleRepo.scheduleRepository.create                                as
jest.Mock).mockResolvedValue(createdSchedule);
        const res = await
request(app).post('/schedules/create/').send(newSchedule);
        expect(res.status).toBe(200);
        expect(res.body).toEqual(createdSchedule);

expect(scheduleRepo.scheduleRepository.create).toHaveBeenCalledWith(
newSchedule);
    });

    it('PATCH /schedules/update/:id должен обновлять расписание', async
    () => {
        const updatedSchedule = { id: 1, name: 'UpdatedSchedule' };
        (scheduleRepo.scheduleRepository.update                                as
jest.Mock).mockResolvedValue(updatedSchedule);
        const res = await
request(app).patch('/schedules/update/1').send(updatedSchedule);
        expect(res.status).toBe(200);
        expect(res.body).toEqual(updatedSchedule);

expect(scheduleRepo.scheduleRepository.update).toHaveBeenCalledWith(
1, updatedSchedule);
    });

    it('DELETE /schedules/delete/:id должен удалять расписание', async
    () => {
        (scheduleRepo.scheduleRepository.delete                                as
jest.Mock).mockResolvedValue(true);
        const res = await request(app).delete('/schedules/delete/1');
        expect(res.status).toBe(200);
        expect(res.body).toBe(true);

expect(scheduleRepo.scheduleRepository.delete).toHaveBeenCalledWith(
1);
    });
});

```

Листинг – Реализация тестов для ScheduleController