МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

КУРСОВАЯ РАБОТА

по дисциплине: Разработка серверных частей интернет-ресурсов
по профилю: Разработка программных продуктов и проектирование информационных систем
направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: Серверная часть веб-приложения «Веб-сервис аренды автомобилей»

Студент: Никитина Валерия Александровна
Группа: ИКБО-20-21
Работа представлена к защите _____(дата) _____ / Никитина В.А./
(подпись и ф.и.о. студента)

Руководитель: Волков М.Ю., старший преподаватель

Работа допущена к защите _____(дата) _____ /Волков М.Ю./
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: _____
_____ / _____ /
_____ / _____ /
(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

2023 г.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине: Разработка серверных частей интернет-ресурсов
по профилю: Разработка программных продуктов и проектирование информационных систем
направления профессиональной подготовки: Программная инженерия (09.03.04)

Студент: Никитина Валерия Александровна
Группа: ИКБО-32-21
Срок представления к защите: 12.12.2023
Руководитель: Волков Михаил Юрьевич, старший преподаватель

Тема: Серверная часть веб-приложения «Веб-сервис аренды автомобилей»

Исходные данные: используемые технологии: JDK (8+), создание Spring web-приложений, Spring ORM и Spring DAO, Maven, gitHub, Jetbrains Intellij IDEA, SQL, PostgreSQL, HTML5, CSS3, JavaScript, ReactJS, Redux-toolkit, Axios, React-router-dom, Microsoft Visual Studio Code, Docker, наличие: межстраничной навигации, внешнего вида страниц, соответствующего современным стандартам веб-разработки, использование архитектурного стиля API REST, нормативный документ: инструкция по организации и проведению курсового проектирования СМКО МИРЭА 7.5.1/04.И.05-18, ГОСТ 7.32-2017.

Перечень вопросов, подлежащих разработке, и обязательного графического материала: 1. Провести анализ предметной области разрабатываемого веб-приложения. 2. Обосновать выбор технологий разработки веб-приложения. 3. Разработать архитектуру веб-приложения на основе выбранного паттерна проектирования. 4. Реализовать слой серверной логики веб-приложения с применением выбранной технологии. 5. Реализовать слой логики базы данных. 6. Разработать слой клиентского представления веб-приложения 7. Создать презентацию по выполненной курсовой работе.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Зав. кафедрой ИиППО: _____/Р. Г. Болбаков/, «25» 11 2023 г.
Задание на КР выдал: _____/М.Ю. Волков/, «25» 11 2023 г.
Задание на КР получил: _____/В.А. Никитина/, «25» 11 2023 г.

АННОТАЦИЯ

Отчёт 35 с., 15 рис., 1 табл., 21 источн.

SPRING, JAVA, REST, HTTP, ПРИЛОЖЕНИЕ

Разрабатываемый проект представляет собой веб-сервис для аренды автомобилей.

Цель работы – создание архитектуры и серверной части приложения для этого сервиса.

В ходе работы проведён анализ предметной области и изучены сайты аналогичной тематики. На основе сравнительного анализа определены ключевые сущности и функции, необходимые для реализации.

Рассмотрены этапы проектирования архитектуры, выбор технологий и процесс разработки приложения.

Результатом стало RESTful-приложение, предоставляющее готовое API для сервиса аренды автомобилей.

Report 35 p., 15 fig., 1 table, 21 sources.

SPRING, JAVA, REST, HTTP, APPLICATION.

The object of development is a car rental service.

The purpose of the work is to create the architecture and server-side components of the car rental service.

During the work, an analysis of the subject area and a review of similar websites were conducted. Using a comparative analysis method, the key entities and functionalities required for the application were identified.

The process of designing the architecture, selecting appropriate technologies, and developing the application was examined.

The result is a RESTful application providing a ready-to-use API for a car rental service.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

API – Application Programming Interface (прикладной программный интерфейс)

CRUD – Create, Read, Update, Delete (Создавать, читать, обновлять, удалять)

DTO – Data Transfer Object (Объект передачи данных)

DI – Dependency Injection (Внедрение зависимости)

IoC – Inversion of Control (Инверсия управления)

IDE – Integrated Development Environment (Интегрированная среда разработки)

JPA – Java Persistence API (Java API хранения данных)

JSON – JavaScript Object Notation (Нотация объектов JavaScript)

ORM – Object-Relational Mapping (Отображение объектно-ориентированных структур на реляционные данные)

HTML – HyperText Markup Language (стандартизированный язык разметки документов для просмотра веб-страниц в браузере)

IoC – Inversion of Control (Инверсия управления)

JDK – Java Development Kit (бесплатно распространяемый компанией Oracle Corporation комплект разработчика приложений на языке Java)

JWT – JSON Web Token (открытый стандарт для создания токенов доступа, основанный на формате JSON)

MVC – Model-View-Controller (архитектура проектирования приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер)

POJO – Plain Old Java Object («старый добрый Java-объект», простой Java-объект, не унаследованный от какого-то специфического объекта и не реализующий никаких служебных интерфейсов сверх тех, которые нужны для бизнес-модели)

REST – Representational State Transfer (Архитектурный стиль передачи данных через HTTP)

XML – eXtensible Markup Language (Расширяемый язык разметки)

ВВЕДЕНИЕ

Сегодня информационные технологии стали неотъемлемой частью всех сфер общества, государства и бизнеса. Они играют важнейшую роль в организации и оптимизации процессов, повышая их эффективность и упрощая выполнение повседневных задач. Разработка современных веб-приложений требует от разработчика владения передовыми технологиями, инструментами и архитектурными подходами, что делает эту задачу сложной и многогранной.

Целью данной курсовой работы является создание серверной части веб-приложения «Сервис аренды автомобилей». Для этого используются микросервисная архитектура, принципы проектирования MVC и REST, а также современные технологии разработки на базе фреймворка Spring.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1) провести анализ предметной области разрабатываемого веб приложения;
2) обосновать выбор технологий разработки веб-приложения;
3) разработать архитектуру веб-приложения на основе выбранного паттерна проектирования;
4) реализовать слой серверной логики веб-приложения с применением выбранной технологии;
5) реализовать слой логики базы данных;
6) разработать слой клиентского представления веб-приложения;
7) создать презентацию по выполненной курсовой работе.

1 ОБЩИЕ СВЕДЕНИЯ

1.1 Обозначение и наименование интернет-ресурса

Темой разработанной серверной части веб-приложения является «Сервис аренды автомобилей». Разработанное приложение получило название – «AutoMobility».

1.2 Функциональное назначение

Разработанное приложение имеет прикладной характер и может послужить серверной частью для настоящего сервиса аренды автомобилей. Приложение для незарегистрированных пользователей включает в себя такие функции как просмотр авто и их фильтрация, регистрация аккаунта. Для зарегистрированных пользователей имеется следующий функционал: авторизация, возможность оставить заявку на обратную связь. Также разработан интерфейс администратора для управления списком пользователей и списком автомобилей, предоставленных для аренды.

2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Для анализа предметной области было проведено исследование в области аналогичных приложений и интернет-ресурсов и были выявлены их преимущества и недостатки, представленные в Таблице 1.

Таблица 1 – Сравнение аналогичных приложений

	Goodokrent	Тройка	bookingcar
Бронирование автомобиля	+	+	+
Комфортный подбор	-	+	+
Акции	+	+	-
Управление тарифами и оплатой	+	+	+
Выбор авто по категориям	-	+	+
Достаточно информативный интерфейс	+	-	-
Служба поддержки	+	+	+

На основании данных, представленных в Таблице 1, можно выделить ключевые функции, которые необходимо реализовать в приложении. Среди них: бронирование автомобилей, управление доступом к забронированным авто, настройка тарифов, ведение истории поездок и отчетности, а также работа службы поддержки и возможность обратной связи. Однако, один из недостатков существующих аналогов — отсутствие новостной ленты, где могли бы публиковаться актуальные предложения, новинки и обзоры.

На основе проведенного анализа были сформулированы функциональные требования для создаваемого приложения. Для незарегистрированных пользователей предусмотрены такие функции, как просмотр доступных автомобилей, возможность их фильтрации и регистрация аккаунта. Для зарегистрированных пользователей добавлены возможности авторизации и подачи заявки на обратную связь. Кроме того, реализован административный интерфейс, позволяющий управлять списком пользователей и автомобилей, доступных для аренды.

3 ВЫБОР И ОБОСНОВАНИЕ ТЕХНОЛОГИЙ

3.1 Выбор архитектуры

При выборе технологий для разработки приложения важным этапом является определение его архитектурного подхода. Рассмотрим несколько популярных вариантов.

Микросервисный монолит [1] представляет собой комбинированный подход, при котором функциональность приложения разбивается на изолированные модули (микросервисы), но они работают в рамках единой кодовой базы. Такой подход позволяет использовать преимущества микросервисной архитектуры без необходимости внесения значительных изменений в инфраструктуру.

REST (Representational State Transfer) [2] — широко используемый архитектурный стиль, предназначенный для построения распределённых систем. Он основан на взаимодействии компонентов через ресурсы, представления и методы. REST упрощает интеграцию клиента и сервера, делая приложения более гибкими, масштабируемыми и удобными в обслуживании.

Микросервисная архитектура [3] предполагает деление приложения на небольшие независимые модули (микросервисы), каждый из которых выполняет строго определённую задачу и взаимодействует с другими через API. Это обеспечивает гибкость, масштабируемость и возможность независимого обновления модулей, хотя требует дополнительных усилий для синхронизации работы сервисов.

Монолитная архитектура [4] — традиционный подход, при котором все компоненты приложения (бизнес-логика, интерфейс, база данных) находятся в одной кодовой базе. Монолит проще развернуть, но с ростом проекта он становится сложнее в поддержке и масштабировании.

Чистая архитектура (Clean Architecture) [5], разработанная Робертом Мартином, направлена на создание модульных, легко тестируемых и поддерживаемых приложений. Она базируется на принципах SOLID, таких

как принцип единственной ответственности, инверсии зависимостей и открытости/закрытости.

Для разработки сервиса аренды автомобилей наиболее подходящим решением является микросервисная архитектура, которая позволяет эффективно распределять нагрузку между компонентами.

При проектировании архитектуры приложения был выбран шаблон MVC и подход Чистой архитектуры (Clean Architecture), что обеспечивает модульность и простоту сопровождения системы.

3.2 Выбор основных технологий

Для разработки серверной части веб-приложения был выбран язык программирования Java — компилируемый объектно-ориентированный язык, который широко используется для создания серверных решений. Этот выбор обусловлен несколькими ключевыми факторами. Во-первых, популярность Java обеспечивает доступ к большому сообществу разработчиков, что значительно облегчает поиск документации, библиотек и решений для различных задач.

Одним из основных преимуществ Java является использование виртуальной машины Java (JVM), что делает приложения кроссплатформенными. Код, написанный на Java, может исполняться на любой операционной системе без необходимости внесения изменений, что обеспечивает высокую переносимость и масштабируемость серверной части. Кроме того, Java обладает встроенными механизмами безопасности, которые защищают приложение от угроз, таких как переполнение буфера или утечка памяти.

Язык предлагает широкий набор инструментов и возможностей для масштабируемой разработки, включая поддержку многопоточности и распределённых систем. Богатая экосистема библиотек и фреймворков упрощает реализацию сложной функциональности, а также сокращает время разработки.

Для создания серверной части был использован Spring Framework, который предоставляет готовую инфраструктуру для разработки веб-приложений. Spring поддерживает ключевые принципы, такие как Dependency Injection (DI) и Inversion of Control (IoC), что способствует чистоте архитектуры и модульности кода.

В качестве базы данных выбрана PostgreSQL, известная как мощная объектно-реляционная СУБД. Она поддерживает такие возможности, как полнотекстовый поиск, геопространственные запросы и транзакции, что делает её подходящей для проектов разного масштаба. Открытый исходный код и активное сообщество PostgreSQL обеспечивают её востребованность среди разработчиков.

Для передачи данных между сервером и клиентом использовались протокол HTTP и формат JSON, который является универсальным и читаемым. JSON обеспечивает удобный способ обмена данными между различными языками программирования.

Для объектно-реляционного отображения (ORM) применялся Hibernate, который позволяет взаимодействовать с базой данных в объектно-ориентированном стиле. Он автоматизирует многие аспекты работы с БД и упрощает написание SQL-запросов благодаря своим стратегиям отображения данных.

Для повышения читаемости и чистоты кода использовалась библиотека Lombok, которая автоматически генерирует конструкторы и методы во время компиляции, существенно сокращая количество шаблонного кода.

Выбор этого технологического стека был сделан для обеспечения высокой производительности, удобства разработки и масштабируемости веб-приложения.

3.3 Прикладное программное обеспечение, необходимое для разработки и функционирования приложения

Для разработки и успешного функционирования приложения было использовано несколько инструментов и программных средств.

Основой разработки стал JDK (Java Development Kit) [12], представляющий собой комплект инструментов для создания приложений на Java. В состав JDK входят компилятор, отладчик и другие утилиты, необходимые для разработки.

Для работы с Java была выбрана интегрированная среда разработки IntelliJ IDEA, которая предоставляет разработчику мощный функционал. Среди ключевых возможностей IntelliJ IDEA — автоматическое завершение кода, рефакторинг, анализ кода и интеграция с системами контроля версий. Высокая гибкость и возможность настройки делают этот инструмент удобным для любых задач.

Управление зависимостями и задачами проекта осуществлялось с помощью системы автоматической сборки Maven [13]. Maven позволяет эффективно управлять зависимостями, выполнять компиляцию кода и создавать артефакты. Этот инструмент упрощает процесс сборки и структуризации проекта, что особенно важно при разработке сложных приложений.

Для тестирования API приложения использовался Postman, многофункциональная платформа для взаимодействия с API. Postman предлагает удобный интерфейс для создания, отправки и отладки запросов. Поддерживая все основные типы HTTP-запросов (GET, POST, PUT, DELETE), Postman идеально подходит для работы с RESTful API. Также инструмент предоставляет возможности для автоматизации тестирования, включая создание коллекций запросов и использование переменных окружения. Это значительно упрощает процесс проверки API и добавляет гибкость в тестировании различных приложений.

Таким образом, сочетание этих инструментов обеспечило удобство разработки, управление проектом и тестирование функциональности API, что сыграло важную роль в создании качественного приложения.

4 РАЗРАБОТКА АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ НА ОСНОВЕ ВЫБРАННОГО ПАТТЕРНА

В предыдущей главе было принято решение использовать микросервисную архитектуру и паттерн REST для реализации приложения.

Схема межстраничной навигации веб-сервиса аренды автомобилей включает следующие связи:

Пользовательская сторона:

1) главная (список автомобилей):
   - переход на страницу конкретного авто;
   - переход к тарифам;
   - переход к условиям аренды;
   - переход на страницу «О компании».
2) Страница конкретного авто:
   - возможность возврата на главную страницу.
3) Тарифы:
   - связь с условиями аренды;
   - возможность возврата на главную страницу.
4) Условия аренды:
   - возможность возврата на главную страницу.
5) О компании:
   - вереход на страницу регистрации пользователя;
   - переход на страницу входа пользователя.
6) Страница регистрации (пользователя):
   - переход на страницу входа пользователя.
7) Страница входа (пользователя):
   - возможность возврата на главную страницу.

Административная сторона:

1) Страница входа (администратора):
   - переход к странице выбора базы данных;
   - возможность входа в управление базой данных пользователей;

– возможность входа в управление базой данных автомобилей.

2) Страница регистрации (администратора):

– переход на страницу входа администратора.

3) Страница выбора базы данных (администратора):

– переход к базе данных пользователей;

– переход к базе данных автомобилей.

4) Страница базы данных пользователей (администратора):

– возможность возврата на страницу выбора базы данных.

5) Страница базы данных автомобилей (администратора):

– возможность возврата на страницу выбора базы данных.

Данная структура навигации позволяет пользователю и администратору удобно взаимодействовать с функционалом веб-сервиса, обеспечивая понятные переходы между разделами.

Архитектура веб-приложения включает в себя несколько ключевых слоев, каждый из которых играет определенную роль:

Слои архитектуры:

1) контроллер (Controller): является посредником между представлением (View) и остальными компонентами системы.

2) Сервис (Service): содержит бизнес-логику приложения.

3) Модель (Model): представляет собой структуры данных.

4) Репозиторий (Repository): отвечает за взаимодействие с источниками данных.

Контейнеры в системе:

1) spring-контейнер: является сервисом серверной части приложения, управляет выполнением бизнес-логики и обработкой запросов.

2) PostgreSQL-контейнер: хранит данные о пользователях, автомобилях и других объектах системы, используется для обеспечения надежного и быстрого доступа к информации.

3) Frontend-контейнер: реализует клиентскую часть приложения, отвечает за взаимодействие с пользователями и отображение данных.

Диаграмма развертывания (Deployment Diagram) [15]:

deployment-диаграмма в нотации UML используется для представления физической архитектуры системы и размещения компонентов.

Основные задачи Deployment-диаграммы:

1) отображение физической конфигурации - показывает, как компоненты размещены на серверах, хостах или других средах.

2) определение взаимодействия между компонентами - демонстрирует связи и взаимодействие между различными элементами системы.

3) Определение сетевых требований - помогает спланировать инфраструктуру, необходимую для взаимодействия компонентов.

4) Масштабирование и оптимизация - используется для оценки возможностей системы в условиях роста нагрузки и последующего масштабирования.

Диаграмма развертывания помогает визуализировать архитектуру системы, определить необходимые ресурсы и оптимизировать её работу. Она также упрощает процесс планирования и обеспечения надежности системы.

5 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ИНТЕРНЕТ-РЕСУРСА

5.1 Разработка слоя моделей

В процессе разработки интернет-проекта были созданы модели данных в виде POJO-объектов на языке Java, которые описывают ключевые сущности предметной области. Эти модели включают поля, представляющие свойства каждой сущности, и методы, определяющие их поведение в системе. Среди реализованных моделей представлены автомобиль (Car), пользователь (Person) и перечисление ролей (Role), которое задает роли пользователей, например клиента или администратора.

Для создания моделей данных использовались аннотации, упрощающие работу с объектами и их взаимодействие с базой данных. Аннотация @Entity указывает, что класс представляет собой сущность базы данных, а @Data автоматически генерирует такие методы, как геттеры, сеттеры, toString(), equals() и hashCode().

Пример - реализации модели автомобиля, включающей описание всех её полей и методов, представлен в приложении А на рисунке А.1. Такие модели обеспечивают удобное представление данных в системе и упрощают взаимодействие между приложением и базой данных.

5.2 Разработка слоя репозиториев

В разработанном приложении слой доступа к данным реализован с использованием интерфейсов репозиториев, предоставляемых Spring Data. Репозитории обеспечивают удобный доступ к базам данных и другим источникам данных, реализуя основные CRUD-операции (создание, чтение, обновление, удаление). Это упрощает процесс хранения и извлечения данных.

Репозитории позволяют разработчикам абстрагироваться от конкретных технологий хранения данных, скрывая сложность взаимодействия с базой данных за интуитивно понятным интерфейсом. Это существенно упрощает процесс разработки, снижает затраты на поддержку кода и облегчает обновление приложения.

Одним из ключевых преимуществ использования репозиториев является поддержка Spring Data JPA, которая позволяет создавать репозитории на основе аннотаций. Методы в таких репозиториях автоматически генерируют SQL-запросы на основе их сигнатур, что значительно уменьшает объем написанного вручную кода и ускоряет разработку приложения.

Еще одним важным аспектом работы репозиториев является возможность управления транзакциями. Транзакции обеспечивают целостность данных, исключают ошибки при параллельных запросах к базе данных и гарантируют, что изменения либо полностью применяются, либо откатываются в случае возникновения ошибок.

Для автоматического определения репозиториев в приложении использовалась аннотация @Repository, что упрощает управление зависимостями в рамках Spring-контекста. Все репозитории строились на основе стандартных методов интерфейса JpaRepository, что позволяет эффективно обрабатывать данные всех сущностей приложения.

5.3 Разработка слоя сервисов

Сервисы в Spring Framework представляют собой ключевые компоненты, которые выполняют бизнес-логику приложения и обрабатывают запросы от других компонентов, таких как контроллеры и репозитории. Они предназначены для организации логики в отдельный слой, обеспечивая модульность и удобство сопровождения кода. В Spring создание сервисов осуществляется с использованием аннотации @Service, которая автоматически регистрирует класс в контексте приложения как компонент.

В разработанном приложении были созданы классы-сервисы для реализации бизнес-логики отдельных сущностей. Например, CarService отвечает за управление бизнес-логикой, связанной с автомобилями, а PersonService – за обработку операций, связанных с пользователями. Эти сервисы взаимодействуют с репозиториями для выполнения операций над данными и обеспечивают их передачу в контроллеры.

Пример реализации сервиса на основе класса, отвечающего за обработку заказов, представлен в приложении А на рисунке А.2. В этом примере демонстрируется, как сервисы обрабатывают бизнес-логику, изолируя её от других компонентов приложения. Такая архитектура упрощает разработку, тестирование и сопровождение приложения.

5.4 Разработка слоя контроллеров

RESTful [19] контроллеры в Spring Framework предназначены для создания веб-сервисов, которые предоставляют API [20] для работы с данными и функциональностью на сервере. Они обеспечивают обработку HTTP-запросов и возвращают ответы в формате JSON или XML, что делает их подходящими для реализации RESTful-приложений.

Ключевая аннотация, используемая для реализации контроллеров, – это @RestController, которая определяет класс как RESTful-компонент для обработки входящих запросов. Для обработки различных типов запросов применяются аннотации, такие как @GetMapping, @PostMapping, @PutMapping и @DeleteMapping, которые связывают методы контроллера с соответствующими HTTP-методами. Дополнительные аннотации, такие как @RequestBody и @PathVariable, позволяют удобно работать с данными запросов, извлекая их из тела запроса или URL соответственно. Аннотация @RequestParam используется для работы с параметрами HTTP-запросов, а такие аннотации, как @ResponseStatus, @CrossOrigin, @ModelAttribute и @ExceptionHandler, предоставляют возможности для тонкой настройки поведения контроллеров.

В рамках проекта был реализован контроллер AuthenticationController, который отвечает за обработку запросов, связанных с аутентификацией пользователей. Этот контроллер содержит два метода POST-запросов: register и authenticate. Метод register принимает данные нового пользователя в формате JSON, передает их в сервис аутентификации для создания учетной записи и возвращает JWT-токен в виде JSON-объекта. Метод authenticate, в свою очередь, обрабатывает данные пользователя, такие как имя и пароль,

проверяет их подлинность через сервис аутентификации и возвращает JWT-токен в случае успешной проверки.

Класс AuthenticationController использует аннотации @CrossOrigin, @RestController и @RequiredArgsConstructor, что упрощает управление зависимостями и позволяет гибко настраивать взаимодействие между клиентом и сервером. Благодаря таким решениям обеспечивается безопасность и удобство работы пользователей с системой через RESTful API.

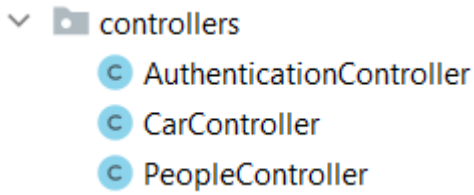

Рисунок 1 – Разработанные контроллеры

Список эндпоинтов представлен на рисунке 2.

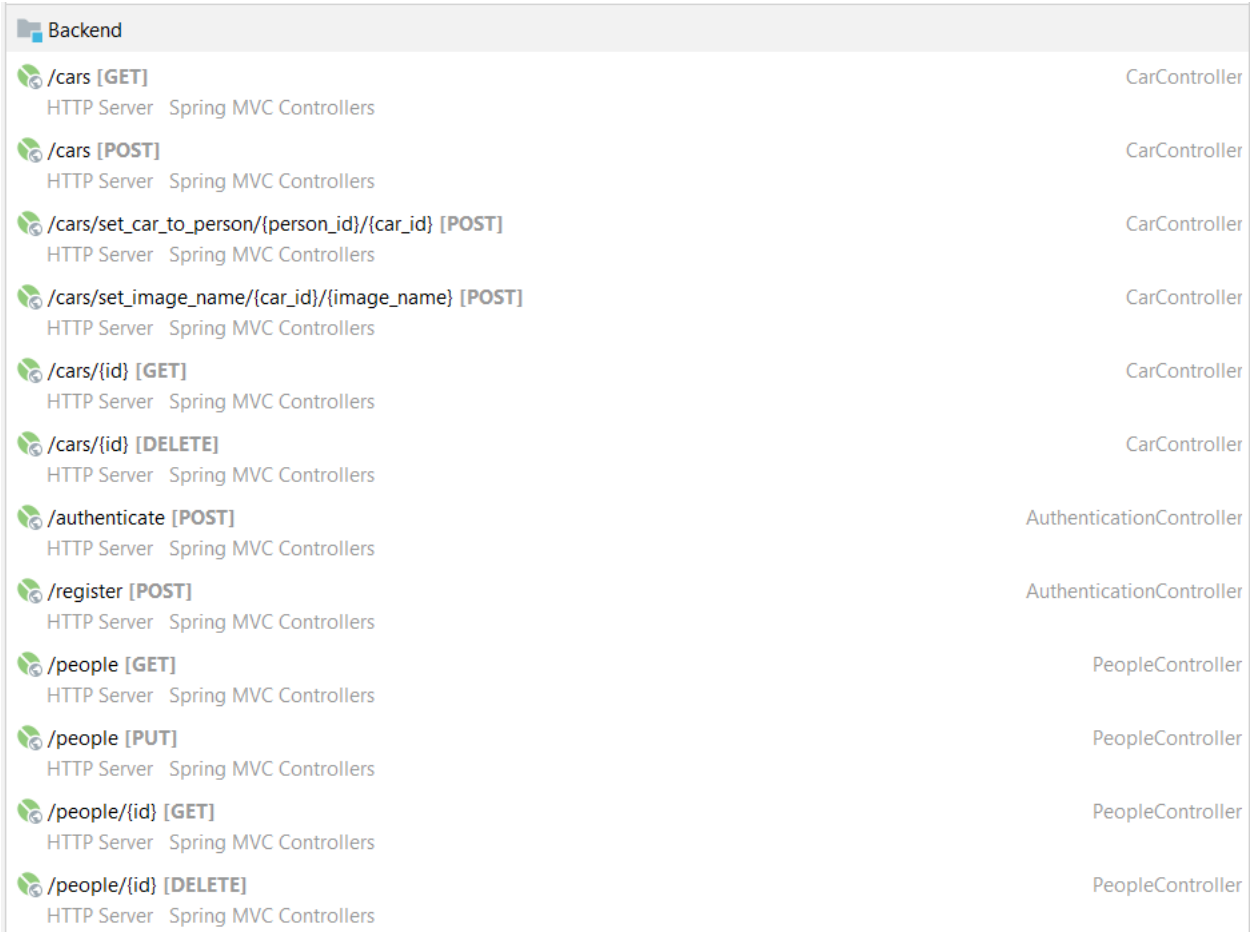

Рисунок 2 – Список эндпоинтов разработанного API

6 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ИНТЕРНЕТ-РЕСУРСА

Frontend-часть разработанного приложения представляет собой современный динамический пользовательский интерфейс, созданный с использованием технологий, обеспечивающих высокую производительность и удобство взаимодействия. В основе архитектуры приложения лежит фреймворк React, который эффективно управляет состоянием компонентов и служит ядром интерфейса. Это позволяет реализовать удобную и отзывчивую работу для пользователя.

Для управления глобальным состоянием приложения используется Redux, что обеспечивает централизованное хранение данных. Такая организация позволяет легко отслеживать и изменять состояние интерфейса. Взаимодействие с серверной частью реализовано с помощью библиотеки axios, интегрированной в Redux Toolkit. Axios обеспечивает выполнение HTTP-запросов, а Redux Toolkit упрощает обработку их результатов и управление состоянием в приложении.

Навигация между страницами осуществляется с помощью React Router DOM. Этот инструмент поддерживает динамическое рендеринг компонентов на основе URL, что делает пользовательский опыт последовательным и интуитивным. Благодаря этому пользователь может легко перемещаться между разделами интерфейса.

Стилизование интерфейса выполнено с применением CSS, что позволяет добиться гибкости в оформлении элементов и создавать современные визуальные решения. Это способствует удобству восприятия и привлекательности интерфейса.

Одной из ключевых особенностей архитектуры является интеграция Redux Toolkit и axios, которая позволяет реализовать эффективное взаимодействие с сервером. Использование срезов состояния, асинхронное обновление данных и обработка запросов делают разработку более простой и масштабируемой.

В результате, благодаря применению React, Redux Toolkit, axios и React Router DOM, пользовательский интерфейс становится не только удобным и отзывчивым, но и легко адаптируемым для будущих улучшений и изменений, что обеспечивает долгосрочную поддержку и развитие приложения.

7 ПРОВЕДЕНИЕ ТЕСТИРОВАНИЯ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ

На Рисунке 3 изображено тестирование регистрации. Отправляется запрос с почтой, именем пользователя и паролем, при этом возвращается JWT токен.

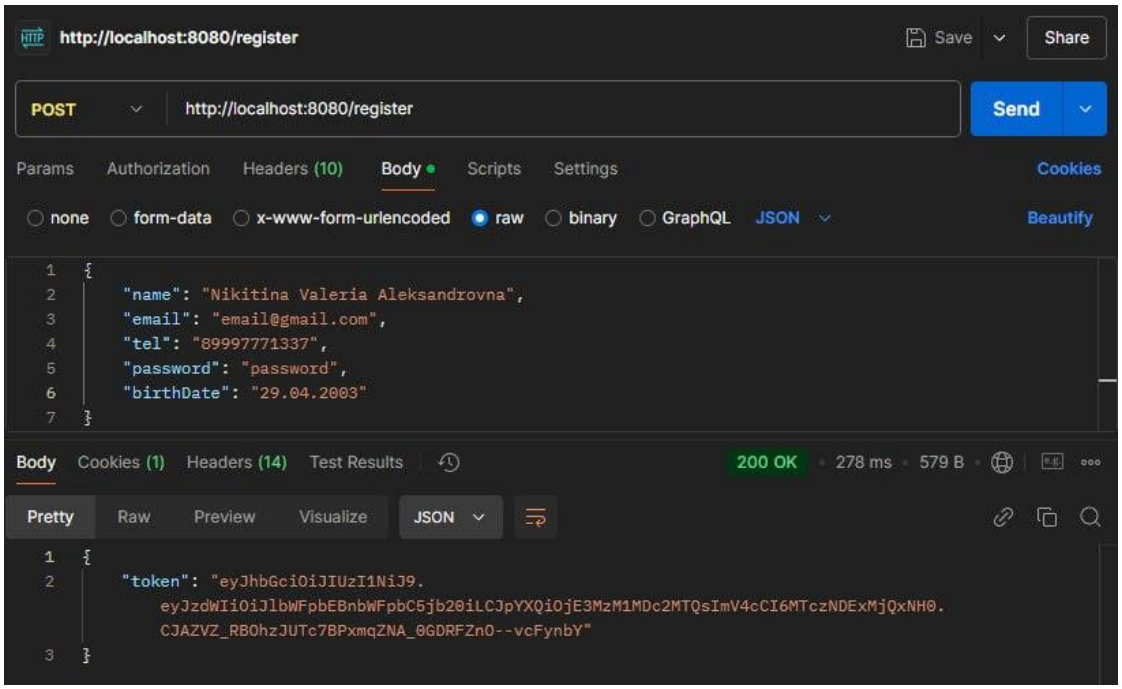

Рисунок 3 – Тестирование регистрации

На Рисунке 4 изображено тестирование авторизации. Отправляется запрос с почтой и паролем, при этом возвращается JWT токен.

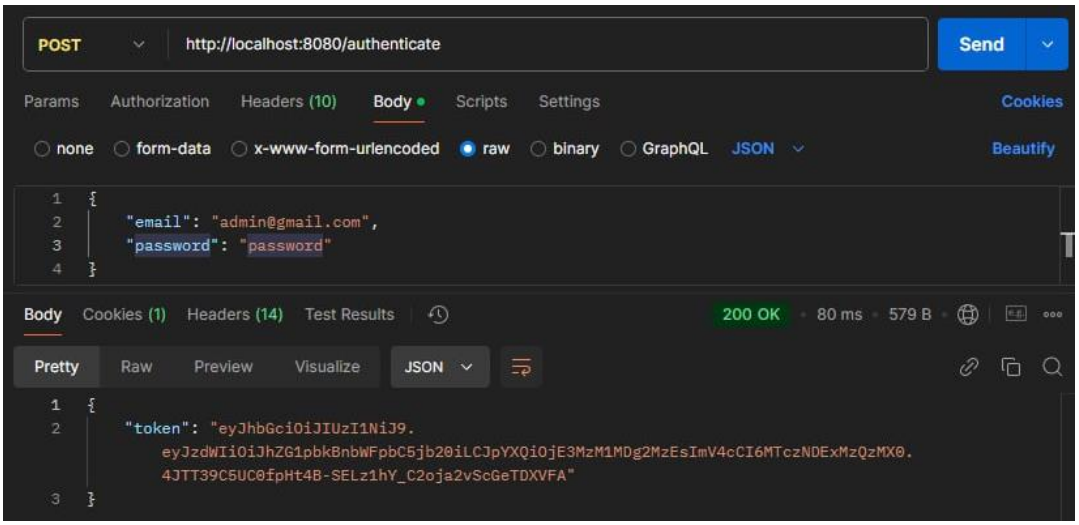

Рисунок 4 – Тестирование авторизации часть 1

На Рисунке 5 изображено тестирование авторизации. Отправляется запрос с почтой и паролем, при этом возвращается статус ошибки 403, так как такого пользователя не существует.

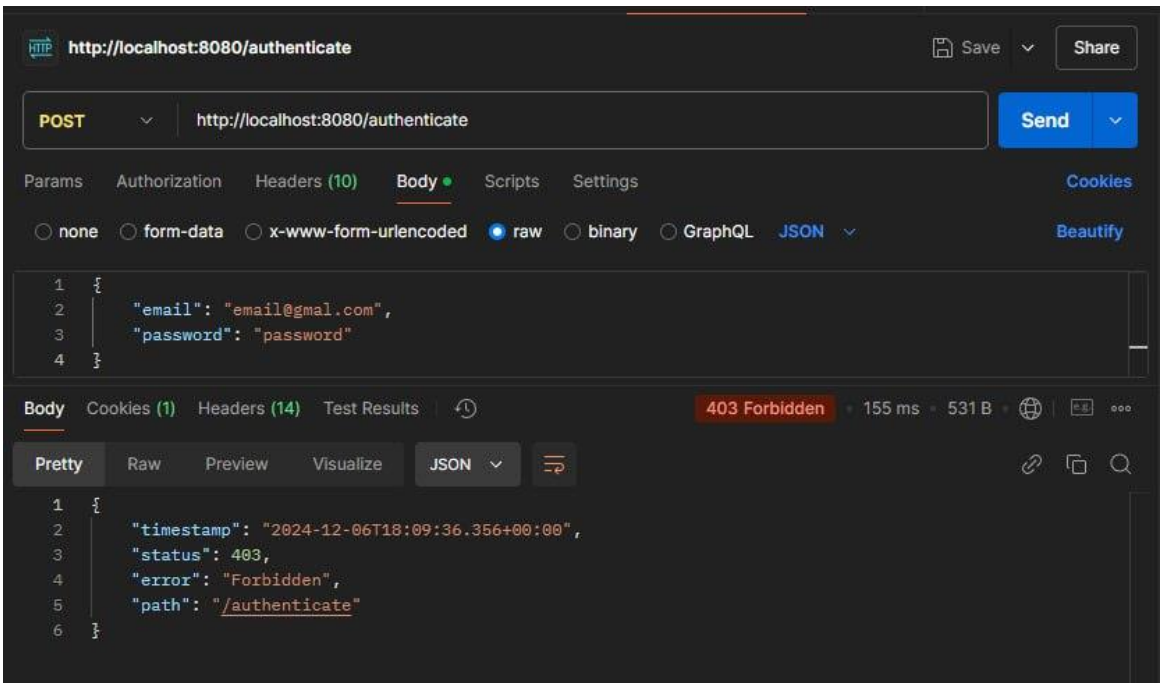

Рисунок 5 – Тестирование авторизации часть 2

На главной странице (Рисунок 6) проведено тестирование внешнего вида сайта, в ходе которого проверялась правильность отображения основного контента, навигационных элементов и графических объектов, таких как баннеры и изображения.

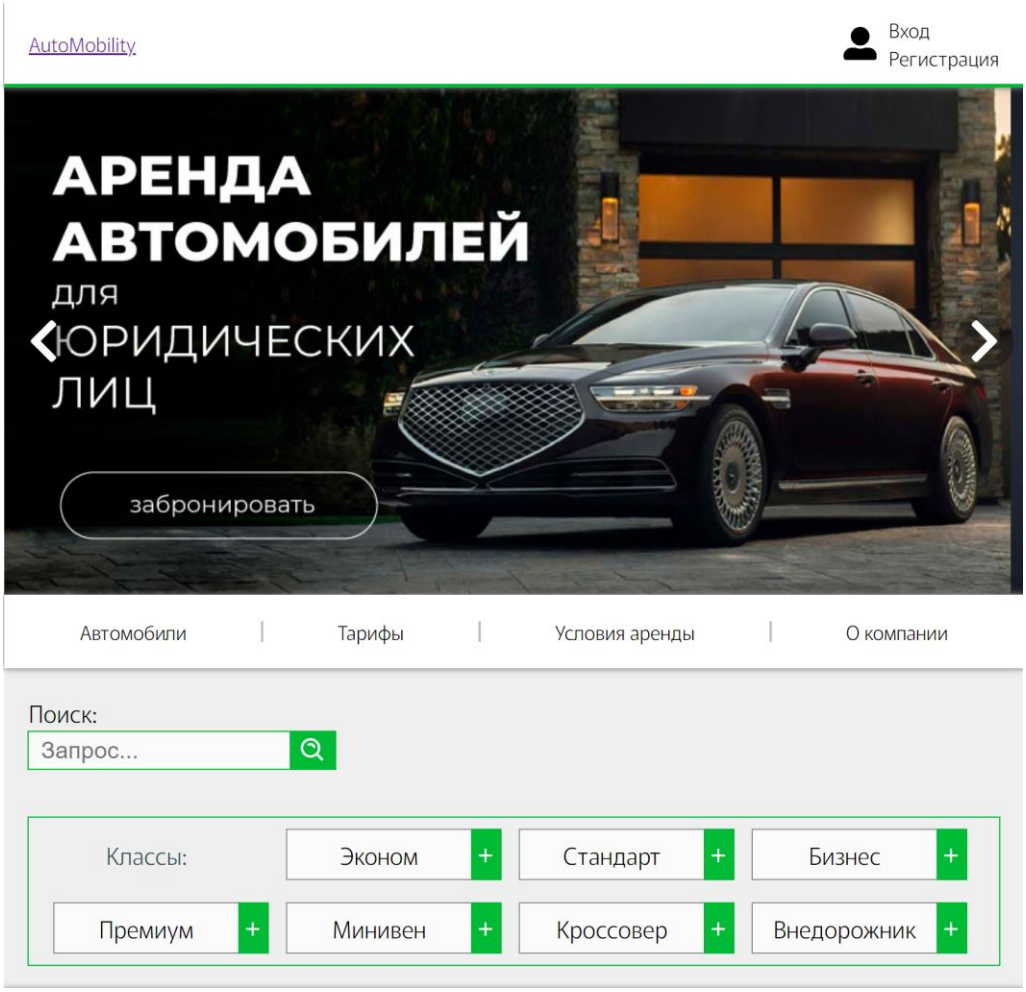

Рисунок 6 – Тестирование главной страницы

На Рисунке 7 демонстрируется тестирование страницы со списком всех машин с фильтрацией. Это включает проверку правильной загрузки списка машин, применение фильтров, а также корректное отображение информации о каждом автомобиле.

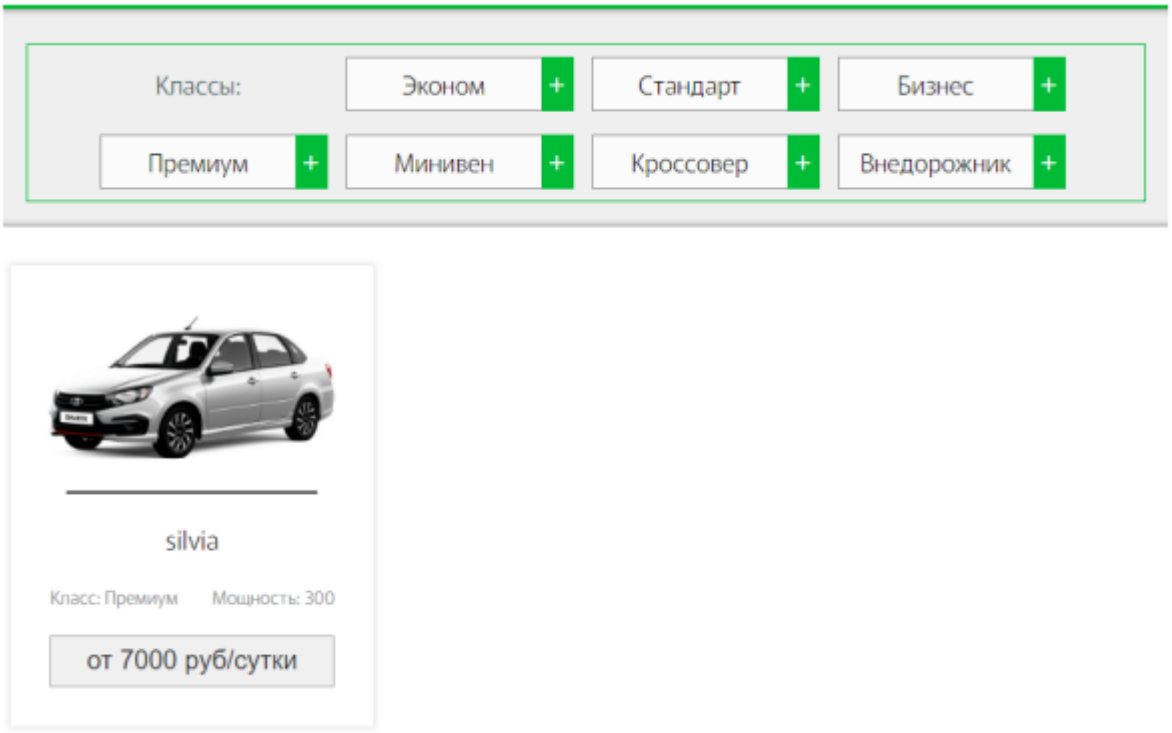

Рисунок 7 – Тестирование получения списка автомобилей

На Рисунке 8 представлено тестирование страницы с тарифами на автомобили. В рамках этого процесса проверяется правильность отображения тарифов и их описаний.

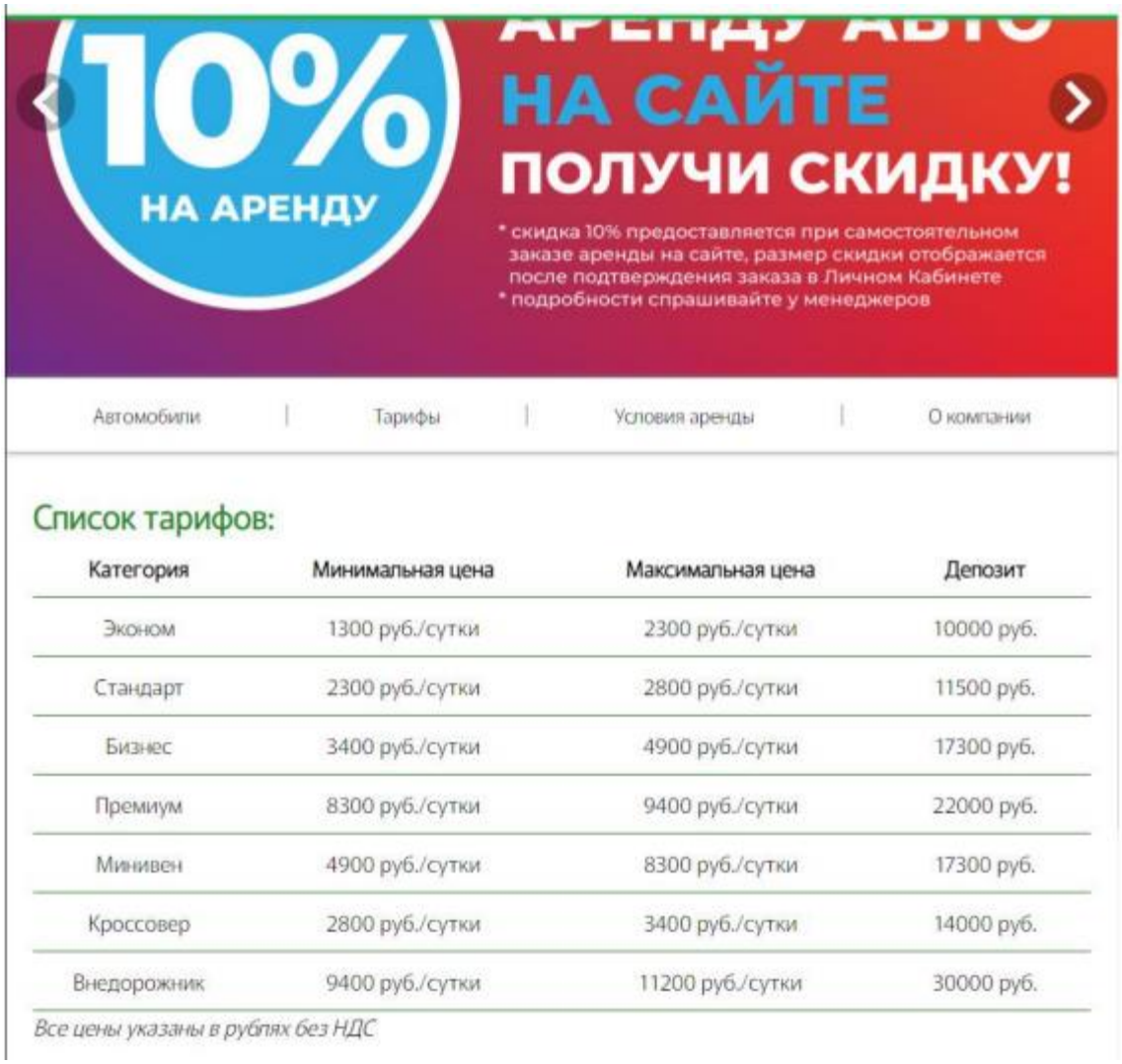

Рисунок 8 – Тестирование страницы списка тарифов

На Рисунке 9 показано тестирование страницы "О компании". Здесь проверяется правильность отображения информации о компании, ее истории, миссии, а также функциональность связи между различными блоками на странице.

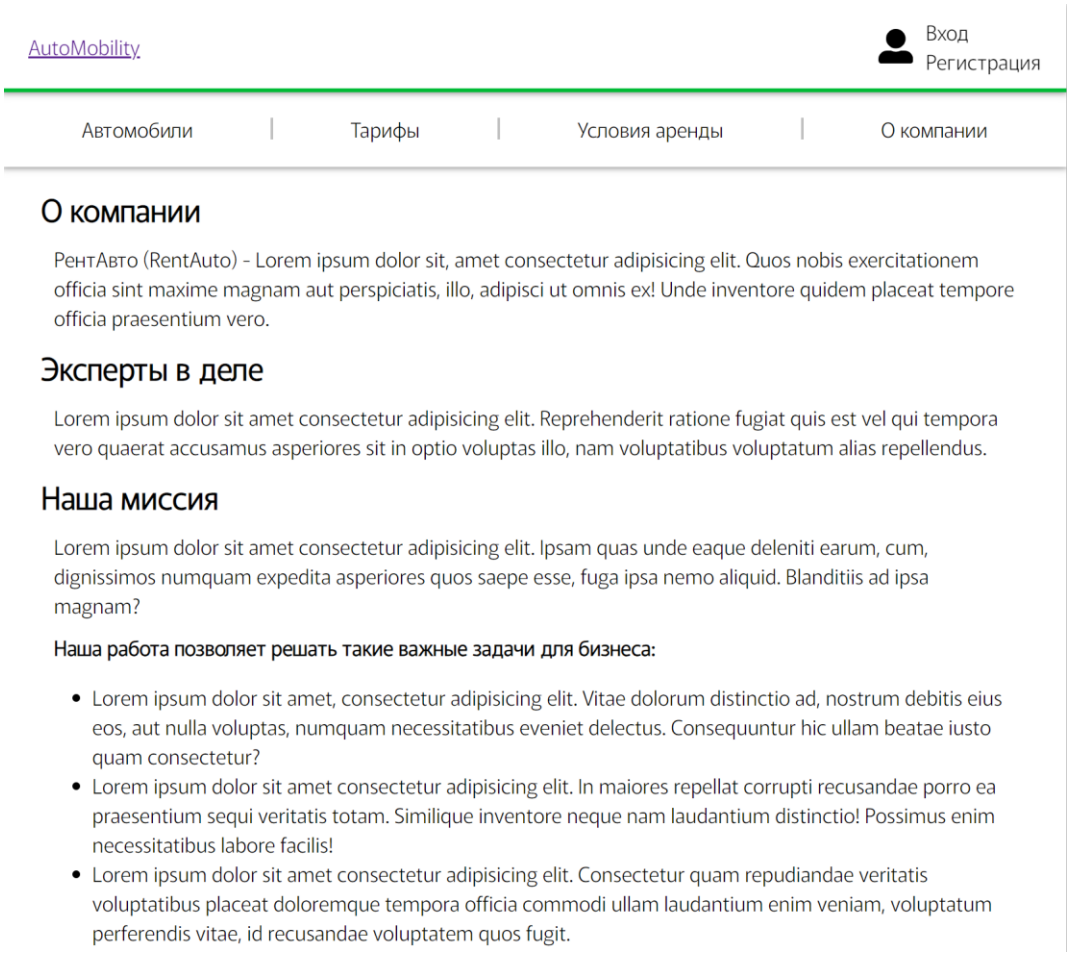

Рисунок 9 – Тестирование страницы «О компании»

На Рисунке 10 показано тестирование страницы авторизации. В ходе теста проверяется правильность работы формы ввода логина и пароля, а также процесс аутентификации пользователя.

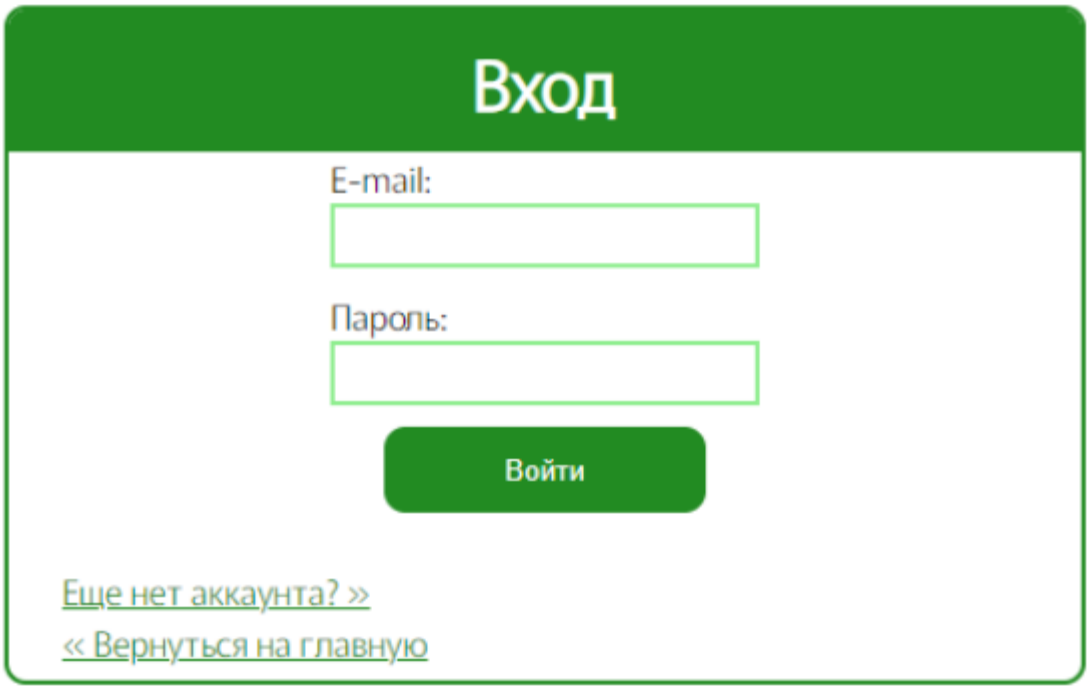

Рисунок 10 – Тестирование авторизации пользователя

На Рисунке 11 показано тестирование личного кабинета. В рамках этого теста проверяется правильность отображения личной информации пользователя, истории поездок, а также работа функций для изменения настроек профиля.

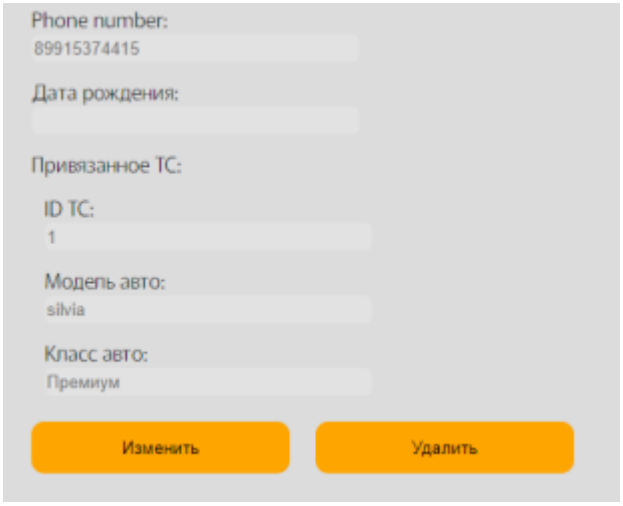

Рисунок 11 – Тестирование редактирования информации о пользователе

На Рисунке 12 показано тестирование страницы добавления и модерации пользователей для администратора. В процессе теста проверяется правильность работы формы добавления пользователя, а также механизмы модерации.

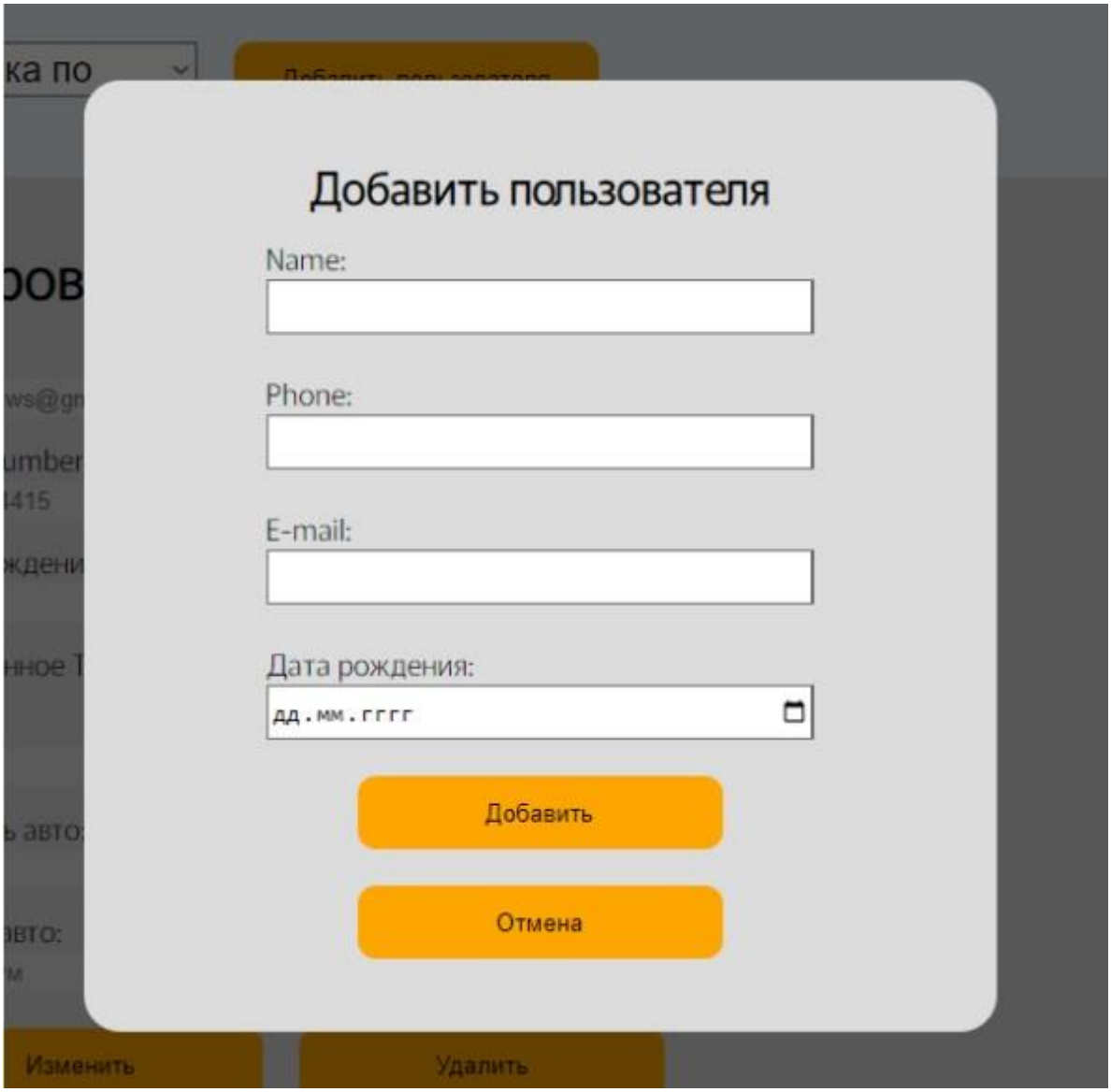

Рисунок 12 – Тестирование добавления пользователя

На Рисунке 13 представлено тестирование страницы добавления и модерации автомобилей для администратора. В рамках теста проверяется правильная работа формы добавления автомобиля, механизмы модерации и корректность взаимодействия с базой данных.

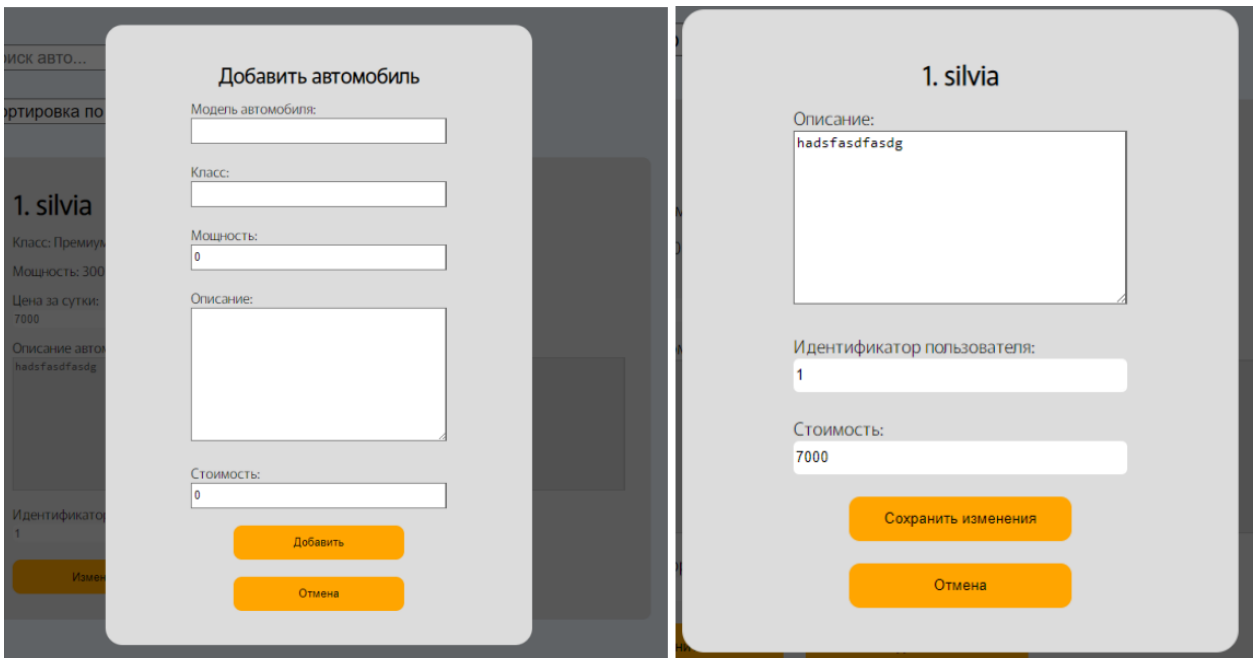

Рисунок 13 – Тестирование добавления и редактирования автомобиля

ЗАКЛЮЧЕНИЕ

Для достижения поставленной цели было необходимо решить следующие задачи:

– провести анализ предметной области разрабатываемого веб-приложения;

– обосновать выбор технологий разработки веб-приложения;

– разработать архитектуру веб-приложения на основе выбранного паттерна проектирования;

– реализовать слой серверной логики веб-приложения с применением выбранной технологии;

– реализовать слой логики базы данных;

– разработать слой клиентского представления веб-приложения;

В результате выполнения данной курсовой работы были выполнены все поставленные задачи, разработано приложение по теме «Сервис аренды автомобилей» с использованием паттерна проектирования REST, произведено мануальное тестирование с помощью инструмента Postman и с помощью веб-интерфейса написанного на ReactJS.

Отчет сформирован согласно инструкции по организации и проведению курсового проектирования и ГОСТ 7.32. Так же был подготовлен демонстрационный материал в виде презентации. Вся проделанная работа была проверена с помощью системы антиплагиат.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Создание микросервисов. 2-е изд. — СПб.: Питер, 2023. — 624 с.: ил. — (Серия «Бестселлеры O’Reilly») ISBN 978-5-9775-6723-7.
2. Ньюмен С. От монолита к микросервисам: Пер. с англ. — СПб.: БХВ-Петербург, 2021. — 272 с.: ил.
3. Ричардсон, С. Шаблоны микросервисов: с примерами на Java. - 1-е издание. - Manning Publications, 2020. - 526 с. - ISBN 978-1617294549.
4. Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. — СПб. : Питер, 2021. — 352 с.
5. Тузовский, А. Ф. Проектирование и разработка web-приложений: учебное пособие для вузов / А. Ф. Тузовский. — Москва: Издательство Юрайт, 2021. — 218 с. — (Высшее образование). — ISBN 978-5-534-00515-8. — Текст : электронный // Образовательная платформа Юрайт [Электронный ресурс]. — URL: https://urait.ru/bcode/469982 (дата обращения: 1.10.2023).
6. Сьерра Кэти, Бэйтс Берт Head First Java / Сьерра Кэти, Бэйтс Берт — СПб. : Эксмо, 2023. — 720 с. — ISBN 978-5-699-54574-2.
7. Раджпут Д. Spring. Все паттерны проектирования. – СПб.: Питер, 2019.
8. PostgreSQL: About [Электронный ресурс] – URL: https://www.postgresql.org/about/ (дата обращения 10.10.2023).
9. Introduction to ORM with Spring [Электронный ресурс] – URL: (дата обращения 11.10.2023).
10. Hibernate Getting Started Guide [Электронный ресурс]. – URL: https://docs.jboss.org/hibernate/orm/6.1/quickstart/html_single (дата обращения 11.10.2023).
11. Project Lombok: Clean, Concise Java Code [Электронный ресурс] – URL: https://www.oracle.com/corporate/features/project-lombok.html (дата обращения 3.11.2023).
12. JAVA Development Kit (JDK) - GeeksforGeeks [Электронный ресурс] – URL: https://www.geeksforgeeks.org/jdk-in-java/ (дата обращения 1.10.2023).

13. What is Gradle? [Электронный ресурс] – URL: https://docs.gradle.org/8.1.1/userguide/what_is_gradle.html (дата обращения 1.11.2023).

14. Модель C4 (C4 model) для визуализации архитектуры программного обеспечения [Электронный ресурс] – URL: https://infostart.ru/pm/1540208/ (дата обращения: 12.10.2023).

15. Фаулер М. UML. Основы, 3-е издание. – Пер. с англ. – СПб: Символ Плюс, 2004. – 192 с. – ил. – ISBN 5-93286-060-X.

16. Кара-Ушанов Владимир Юрьевич МОДЕЛЬ «СУЩНОСТЬ – СВЯЗЬ [Электронный ресурс]: Учебное пособие. – Екатеринбург: УрФУ, 2017 – URL: https://study.urfu.ru/Aid/Publication/13604/1/Kara-Ushanov.pdf (дата обращения 12.10.2023).

17. Spring Data JPA [Электронный ресурс] – URL: https://spring.io/projects/spring-data-jpa (дата обращения 20.10.2023).

18. JPA Criteria Queries | Baeldung [Электронный ресурс] – URL: https://www.baeldung.com/hibernate-criteria-queries (дата обращения 1.03.2023)

19. Hands-On RESTful API Design Patterns and Best Practices / Harihara Subramanian, Pethuru. — Raj Packt Publishing Ltd, 2019. — 378 с. — ISBN 978-1788992664.

20. Меджуи М., Уайлд Э., Митра Р., Амундсен М. Непрерывное развитие API. Правильные решения в изменчивом технологическом ландшафте. — СПб.: Питер, 2020.

21. Хоффман Эндрю X85 Безопасность веб-приложений. — СПб.: Питер, 2021. — 336 с.: ил. — (Серия «Бестселлеры O’Reilly»).

ПРИЛОЖЕНИЕ А

ФРАГМЕНТЫ КОДА РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ

```java
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "car")
public class Car {

    no usages
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    no usages
    @Column(name = "model_name")
    private String modelName;

    no usages
    @Column(name = "horse_powers")
    private int horsePowers;

    no usages
    private String description;

    no usages
    private String category;

    no usages
    private int price;

    no usages
    private String imageName;

    no usages
    private boolean isTaken;
```

Рисунок А.1 – Фрагмент кода класса автомобиля

```java
@Service
@AllArgsConstructor
public class CarService {

    5 usages
    private final CarRepository carRepository;

    3 usages
    public void create(Car car) { carRepository.save(car); }


    1 usage
    public List<Car> findAll() { return carRepository.findAll(); }

    3 usages
    public Car findOne(int id) {
        Optional<Car> foundCar = carRepository.findById(id);
        return foundCar.orElseThrow(EntityNotFoundException::new);
    }

    no usages
    public Car update(Car car) { return carRepository.save(car); }

    1 usage
    public void delete(int id) { carRepository.deleteById(id); }

}
```

Рисунок А.2 – Фрагмент кода класса сервиса заказов