



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 (ИиПО)

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 "RentAuto"»

Студент: Хитров Никита Сергеевич

Группа: ИКБО-20-21

Работа представлена к защите 8.12.23 (дата) [подпись] / Хитров Н.С./
(подпись и ф.и.о. студента)

Руководитель: старший преподаватель Сеницын Анатолий Васильевич

Работа допущена к защите 25.12.23 (дата) [подпись] / Сеницын А.В./
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: отлично

[подпись] 25.12.23 / Куликов А.В., доцент /
[подпись] 25.12.23 / Сеницын А.В., ст. преподаватель /
(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

Студент: Хитров Никита Сергеевич

Группа: ИКБО-20-21

Срок представления к защите: 08.12.2023

Руководитель: Синицын Анатолий Васильевич, старший преподаватель

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

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, 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: Болбаков /Р. Г. Болбаков/, «14» сентября 2023 г.

Задание на КР выдал: Синицын /А.В. Синицын/, «14» сентября 2023 г.

Задание на КР получил: Хитров /Н.С. Хитров/, «14» 09 2023 г.

АННОТАЦИЯ

Отчёт 33 с., 15 рис., 1 табл., 21 источн.

SPRING, JAVA, REST, HTTP, ПРИЛОЖЕНИЕ

Объектом разработки является сервис аренды автомобилей.

Цель работы – разработка архитектуры и серверной части приложения сервис аренды автомобилей.

В ходе работы был проведён анализ предметной области и обзор сайтов с аналогичной тематикой. Методом сравнительного анализа определены необходимые сущности и возможности, которые должны быть включены в приложение.

Рассмотрен процесс разработки архитектуры, выбор необходимых технологий для разработки и процесс создания приложения.

Результатом является приложение, отвечающее стандартам RESTful, позиционирующий готовое API для сервис аренды автомобилей.

Report 33 p., 15 fig., 1 table, 21 sources.

SPRING, JAVA, REST, HTTP, APPLICATION.

The object of development is an rent auto service.

The purpose of the work is to develop the architecture and server part of the rent auto service.

In the course of the work, an analysis of the subject area and a review of sites with similar topics were carried out. By the method of comparative analysis, the necessary entities and capabilities that should be included in the application are determined.

The process of architecture development, the choice of necessary technologies for development and the process of creating an application are considered.

The result is an application that meets RESTful standards, positioning a ready-made API for an rent auto service.

СОДЕРЖАНИЕ

| | |
|---|----|
| ВВЕДЕНИЕ | 6 |
| 1 ОБЩИЕ СВЕДЕНИЯ | 7 |
| 1.1 Обозначение и наименование интернет-ресурса | 7 |
| 1.2 Функциональное назначение | 7 |
| 2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ | 8 |
| 3 ВЫБОР И ОБОСНОВАНИЕ ТЕХНОЛОГИЙ | 9 |
| 3.1 Выбор архитектуры | 9 |
| 3.2 Выбор основных технологий | 10 |
| 3.3 Прикладное программное обеспечение, необходимое для разработки и функционирования приложения..... | 11 |
| 4 РАЗРАБОТКА АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ НА ОСНОВЕ ВЫБРАННОГО ПАТТЕРНА..... | 13 |
| 5 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ИНТЕРНЕТ-РЕСУРСА..... | 16 |
| 5.1 Разработка слоя моделей..... | 16 |
| 5.2 Разработка слоя репозитория..... | 16 |
| 5.3 Разработка слоя сервисов..... | 17 |
| 5.4 Разработка слоя контроллеров..... | 17 |
| 6 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ИНТЕРНЕТ-РЕСУРСА | 20 |
| 7 ПРОВЕДЕНИЕ ТЕСТИРОВАНИЯ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ | 21 |
| ЗАКЛЮЧЕНИЕ | 29 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ..... | 30 |
| ПРИЛОЖЕНИЕ А | 32 |

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

Сферы государства, общества и бизнеса сегодня тесно переплетены с информационными технологиями, которые играют ключевую роль в организации различных процессов. Без них трудно представить эффективное функционирование, поскольку современные технологии значительно упрощают, систематизируют и оптимизируют рабочие процессы. Разработка современных приложений представляет собой сложную и многогранную задачу, требующую от разработчика глубокого понимания архитектуры, технологий и инструментов.

Целью данной курсовой работы является разработка серверной части веб-приложения «Сервис аренды автомобилей» с использованием микросервисов, архитектурных паттернов проектирования MVC и REST и современных технологий на базе фреймворка Spring.

Для выполнения поставленной цели курсовой работы необходимо выполнить следующие пункты:

- 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;
- 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 – «Rentauto».

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 основе информации из таблицы 1 можно выделить несколько основных функций, которые должны присутствовать в приложении, такие как, бронирование авто, контроль доступа к забронированной машине, управление тарифами, история поездок и отчетность, служба поддержки и обратная связь. Одним из минусов аналогичных приложений является отсутствие ленты новостей с актуальными товарами, новинками и обзорами на товары.

На основе полученных данных были составлены функциональные требования для приложения. 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 интерфейс администратора для управления списком пользователей и списком автомобилей, предоставленных для аренды.

3 ВЫБОР И ОБОСНОВАНИЕ ТЕХНОЛОГИЙ

3.1 Выбор архитектуры

При выборе технологий для разработки приложения первым этапом является определение его архитектуры. Рассмотрим несколько распространенных подходов.

Микросервисный монолит [1] представляет собой гибридный подход, где функциональность приложения разделена на изолированные модули (микросервисы), но они все равно развертываются и работают в пределах одной кодовой базы. Это сочетание позволяет извлекать преимущества микросервисной архитектуры, не внося дополнительных изменений в инфраструктуру.

REST (Representational State Transfer) [2] – широко применяемый архитектурный стиль, предоставляющий подход к построению распределенных систем на основе взаимодействия между компонентами. REST разделяет функциональность на ресурсы, представления и методы, обеспечивая эффективное взаимодействие между клиентом и сервером. Выбор REST способствует созданию гибких, расширяемых и легко поддерживаемых приложений.

Микросервисная архитектура [3] основана на делении приложения на небольшие автономные модули, называемые микросервисами. Каждый микросервис выполняет конкретную функцию, взаимодействуя с другими через API. Этот подход обеспечивает гибкость разработки, масштабируемость и возможность обновления компонентов, но требует дополнительных усилий для координации.

Монолитная архитектура [4] представляет собой классический метод, при котором все компоненты приложения, включая бизнес-логику, пользовательский интерфейс и базу данных, объединены в единой программе или кодовой базе. Хотя монолит легко развертывается, он может стать проблемой при масштабировании и увеличении сложности проекта.

Чистая архитектура (Clean Architecture) [5] – концепция, предложенная

Робертом Мартином, направлена на создание гибких, модульных и легко поддерживаемых приложений. Она включает в себя принципы SOLID, такие как принцип единственной ответственности, открытости/закрытости, подстановки Барбары Лисков, разделения интерфейса и инверсии зависимостей.

В контексте сервиса аренды автомобилей наиболее подходящей является микросервисная архитектура, которая обеспечивает равномерное распределение нагрузки между различными сервисами.

Для разработки архитектуры ПП был выбран паттерн проектирования MVC и подход к проектированию Clean Architecture.

3.2 Выбор основных технологий

В качестве языка программирования для серверной части веб-приложения был выбран Java, компилируемый объектно-ориентированный язык с широким признанием в сфере разработки серверных компонентов. Решение основывалось на нескольких ключевых критериях: Java, как один из наиболее популярных языков программирования, обеспечивает доступ к обширному сообществу разработчиков, что облегчает поиск поддержки, руководств и библиотек для эффективной разработки.

Основой Java является виртуальная машина Java (JVM), что позволяет выполнять код на различных операционных системах без изменений. Это обеспечивает высокую масштабируемость и переносимость разработанного бэкенда между разными платформами. Java предлагает встроенные средства безопасности, защищая приложение от различных видов атак, таких как переполнение буфера или неправильное освобождение памяти.

Язык обладает обширным инструментарием и подходами для масштабирования приложений, включая поддержку распределенных систем, возможности многопоточности и создание масштабируемых архитектур. Богатство библиотек, фреймворков и инструментов в экосистеме Java упрощает разработку бэкенда, предоставляя множество опций для эффективной реализации функциональности.

Для разработки использовался Spring Framework, предоставляющий готовую инфраструктуру для создания веб-приложений на Java. Spring обеспечивает необходимые инструменты, такие как Dependency Injection (DI) и Inversion of Control (IoC), с упором на принципы чистой архитектуры.

PostgreSQL была выбрана в качестве базы данных, предоставляющей мощные возможности объектно-реляционной системы управления базами данных (СУБД). Этот выбор обусловлен поддержкой PostgreSQL полнотекстового поиска, гео-пространственных запросов, транзакций и других функциональностей. Благодаря открытому исходному коду и активному сообществу разработчиков, PostgreSQL является популярным выбором для проектов разного масштаба в различных отраслях.

Для обмена данными между серверной и клиентской частями приложения использовались протокол HTTP и формат JSON. JSON, как универсальный формат, обеспечивает удобочитаемую передачу данных между различными языками программирования.

Для объектно-реляционного отображения (ORM) применялся Hibernate. Этот фреймворк позволяет работать с базами данных в объектно-ориентированной манере, упрощая взаимодействие с БД и предоставляя различные стратегии отображения данных.

Дополнительно, для уменьшения объема кода использовалась библиотека Lombok. Ее аннотации позволяют генерировать методы и конструкторы автоматически во время компиляции, что способствует повышению чистоты и читаемости кода.

3.3 Прикладное программное обеспечение, необходимое для разработки и функционирования приложения

Для разработки и функционирования приложения использовались несколько прикладных программных средств. Начнем с JDK (Java Development Kit) [12], который представляет собой комплект инструментов для разработки Java-приложений. JDK включает в себя компилятор, отладчик и другие утилиты. Для удобства работы с Java был выбран IntelliJ IDEA,

редактор с функциональностью интегрированной среды разработки (IDE). IntelliJ IDEA обеспечивает разработчиков широким набором инструментов, включая автоматическое завершение кода, рефакторинг, анализ кода и интеграцию с системами контроля версий. Гибкость и настраиваемость редактора позволяют адаптировать его под любые потребности.

Для управления зависимостями и задачами проекта использовалась система автоматической сборки Maven [13]. Maven обеспечивает возможность управления зависимостями, компиляции кода, создания артефактов. Эта система сборки способствует более эффективному управлению проектом и его зависимостями.

Для тестирования API приложения был использован Postman – многофункциональная платформа для тестирования и разработки API. Postman предоставляет интуитивно понятный интерфейс для создания, отправки и отладки запросов к API. Поддерживая разнообразные HTTP-запросы, включая GET, POST, PUT и DELETE, Postman также обеспечивает возможность работы с RESTful API. Этот инструмент предоставляет функциональность для автоматизации тестирования, включая создание коллекций запросов и использование переменных окружения. Использование Postman упрощает взаимодействие с API и предоставляет расширенные возможности для тестирования различных приложений.

4 РАЗРАБОТКА АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ НА ОСНОВЕ ВЫБРАННОГО ПАТТЕРНА

В предыдущей главе для разработки приложения были выбраны микросервисная архитектура и паттерн REST.

На рисунке 1 представлена схема межстраничной навигации, на которой отображены все пути использования веб-сервиса аренды автомобилей. Схема включает как пользовательскую сторону, так и функционал администратора.

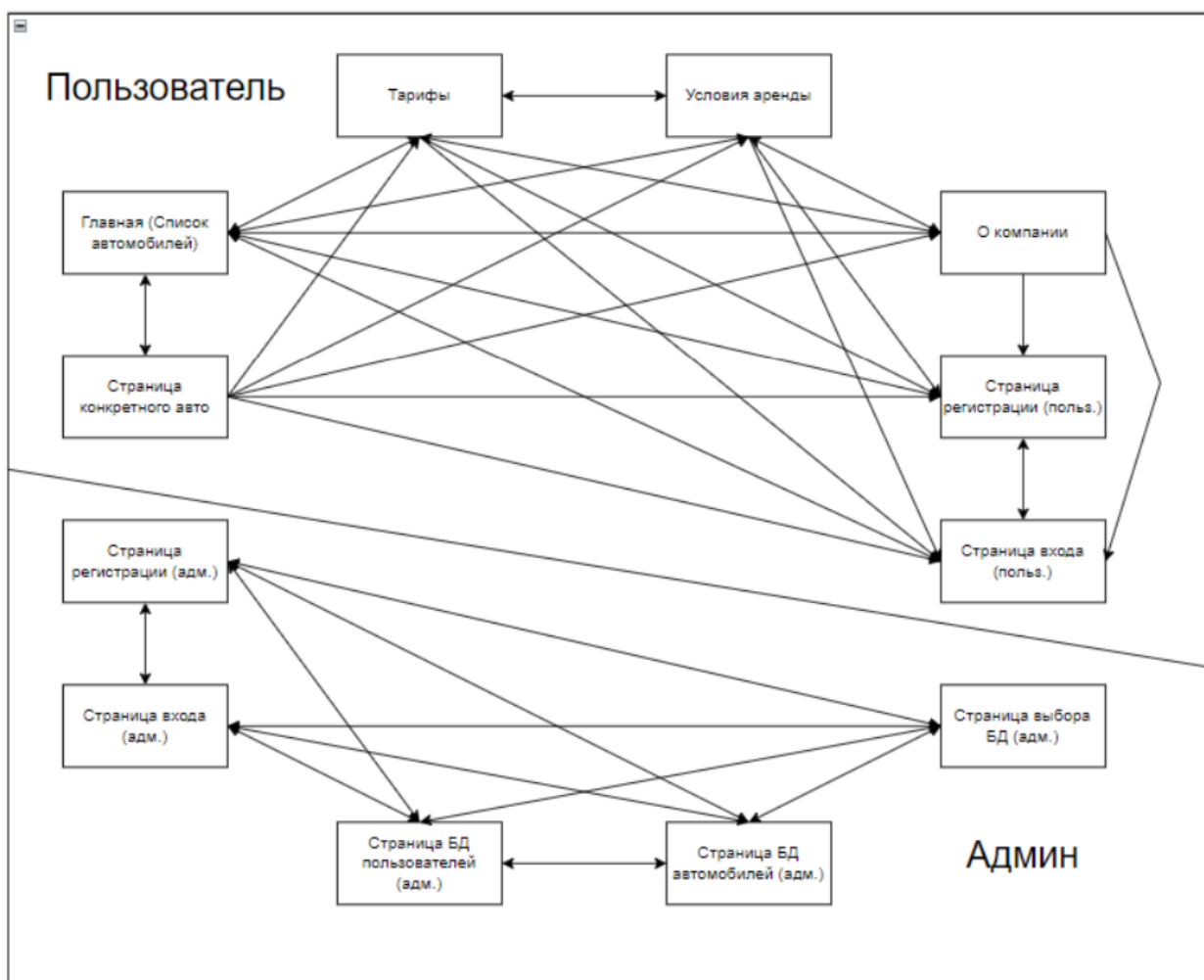


Рисунок 1 – Диаграмма последовательности работы приложения

В архитектурном подходе, используемом при разработке веб-приложения можно выделить следующие слои:

- контроллер (Controller),
- сервис (Service),
- модель (Model),
- репозиторий (Repository).

Слой контроллера выступает в роли посредника между представлением

и другими компонентами системы. Его основная задача - обработка пользовательского ввода и взаимодействие с сервисами для выполнения бизнес-логики. Контроллеры получают запросы от представления, обрабатывают их, вызывают соответствующие сервисы и передают полученные данные обратно в представление.

Сервисный слой содержит бизнес-логику и отвечает за обработку данных и выполнение операций, связанных с бизнес-правилами. Этот слой может использоваться между различными компонентами, включая контроллеры и репозитории, и обеспечивает возможность повторного использования логики.

Модели представляют собой структуры данных, предназначенные для передачи информации между различными слоями и системами. Они включают в себя логику обработки данных и часто служат контейнерами для передачи данных между различными слоями приложения.

Репозиторий отвечает за доступ и взаимодействие с внешними источниками данных, такими как базы данных, API или локальные хранилища. Его функциональность включает в себя предоставление методов для получения, сохранения, обновления и удаления данных. Репозитории абстрагируют доступ к данным от других слоев и обеспечивают независимость от конкретных источников данных. Контейнеры – это исполняемые среды, которые содержат в себе компоненты и предоставляют им ресурсы и средства для работы. В программном продукте были выделены следующие контейнеры:

- контейнер Spring – является сервисом серверной части сервиса аренды автомобилей;

- контейнер PostgreSQL – используется для хранения информации о пользователях и автомобилях;

- контейнер Frontend – является сервисом клиентской части сервиса аренды автомобилей.

Диаграмма развертывания (Deployment diagram) [15] в UML является типом диаграммы, который позволяет представить физическую архитектуру

системы и размещение ее компонентов на аппаратном обеспечении (хостах, серверах) или средах выполнения.

Deployment-диаграмма используется для:

- представления физической конфигурации системы,
- определения взаимодействия между компонентами и устройствами,
- определения требований к сетевой инфраструктуре,
- планирования масштабирования и оптимизации системы.

Разработанная концептуальная диаграмма развертывания представлена на рисунке 2.

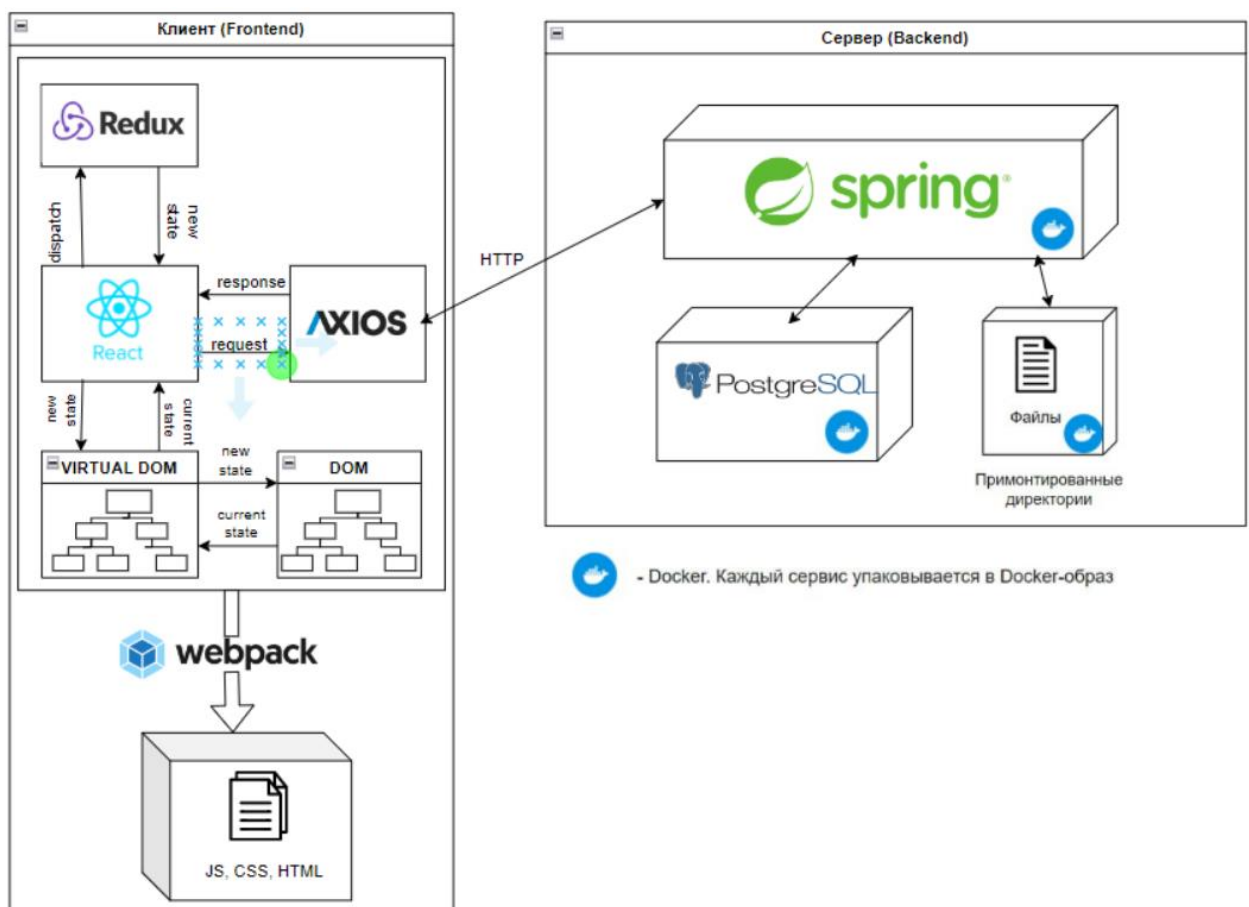


Рисунок 2 – Диаграмма развертывания

5 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ИНТЕРНЕТ-РЕСУРСА

5.1 Разработка слоя моделей

В процессе разработки интернет-проекта были созданы модели данных в виде POJO-объектов в Java, которые описывают основные сущности предметной области проекта. Каждая из этих моделей имеет свои поля и методы, которые описывают ее поведение в системе. В приложении представлены следующие модели: Car, Person и перечисление Role.

Для создания моделей данных использовались такие аннотации как @Entity, которая указывает, что класс является сущностью базы данных., @Data - генерирует геттеры, сеттеры, методы toString(), equals() и hashCode(). С реализацией модели на примере модели автомобиля можно ознакомиться в приложении А на рисунке А.1.

5.2 Разработка слоя репозитория

В разработанном приложении этот слой реализован с помощью интерфейсов репозитория из Spring Data. Репозитории обеспечивают доступ к базам данных и другим источникам данных. Они реализуют CRUD-операции (Create, Read, Update, Delete) и обеспечивают хранение и извлечение данных.

Репозитории позволяют абстрагироваться от конкретных технологий хранения данных и скрыть сложность работы с базой данных за простым интерфейсом. Это позволяет существенно упростить разработку приложений и снизить затраты на поддержку и обновление кода.

Важным аспектом репозитория является возможность использования Spring Data JPA [17], который позволяет создавать репозитории с помощью аннотаций и автоматически генерировать SQL-запросы на основе сигнатур методов. Это позволяет повысить производительность, уменьшить количество кода и сократить время на разработку приложения.

Еще одним важным аспектом репозитория является возможность использования транзакций. Транзакции обеспечивают целостность данных, позволяют избежать ошибок в случае нескольких одновременных обращений к базе данных и гарантируют, что все изменения будут либо применены, либо

откатываются.

Для автоматического определения репозиториев, они были помечены аннотацией `@Repository`.

Для всех сущностей использовались стандартные методы интерфейса репозитория `JpaRepository`.

5.3 Разработка слоя сервисов

Сервисы в Spring Framework – это компоненты приложения, которые выполняют бизнес-логику и обрабатывают запросы от других компонентов. Они предназначены для выделения бизнес-логики в отдельный слой приложения. Сервисы в Spring могут быть созданы с помощью аннотации `@Service`.

Были разработаны классы-сервисы отвечающие за реализацию бизнес-логики каждой отдельной сущности: `CarService`, `PersonService`.

С реализацией сервиса на примере класса сервиса заказов можно ознакомиться в приложении А на рисунке А.2.

5.4 Разработка слоя контроллеров

RESTful [19] контроллеры в Spring используются для создания веб-сервисов, которые предоставляют клиентам API [20] для доступа к данным и функциональности на сервере. Они позволяют обрабатывать запросы через HTTP-протокол и возвращать данные в формате JSON/XML.

Spring Framework предоставляет ряд аннотаций для работы с RESTful контроллерами. Одной из наиболее часто используемых является `@RestController`, которая указывает, что класс является RESTful контроллером и будет обрабатывать входящие HTTP-запрос. Другие полезные аннотации включают `@GetMapping`, `@PostMapping`, `@PutMapping` и `@DeleteMapping`, которые указывают тип запроса, который будет обработан методом контроллера, а также `@RequestBody`, который используется для чтения данных запроса HTTP и преобразования их в объект Java. `@PathVariable` используется для получения параметров URL, а `@RequestParam` – для чтения параметров запроса HTTP. Другие аннотации, такие как `@ResponseStatus`,

@CrossOrigin, @ModelAttribute, @Valid и @ExceptionHandler, также могут использоваться для управления поведением RESTful контроллеров в Spring. На рисунке 3 представлен список разработанных контроллеров для каждой сущности, а также класс AuthenticationController, который обрабатывает HTTP-запросы для аутентификации пользователей. Он содержит два метода POST-запроса: register и authenticate, которые обрабатывают запросы на регистрацию новых пользователей и аутентификацию существующих пользователей соответственно. Класс использует аннотации @CrossOrigin, @RestController и @RequiredArgsConstructor для работы веб-приложения и внедрения сервиса аутентификации.

Метод register принимает POST-запрос, который содержит данные нового пользователя в формате JSON и передает их сервису аутентификации для создания новой учетной записи пользователя. Метод возвращает JWT-токен в виде JSON-объекта.

Метод authenticate принимает POST-запрос, который содержит данные пользователя, такие как имя пользователя и пароль, в формате JSON, и передает их сервису аутентификации для проверки подлинности пользователя. Метод возвращает JWT-токен в виде JSON-объекта.

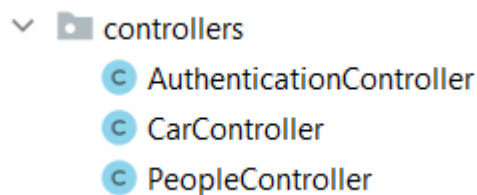


Рисунок 3 – Разработанные контроллеры

Список эндпоинтов представлен на рисунке 4.













| Backend | | |
|--|------------------------------------|--------------------------|
|  /cars [GET] | HTTP Server Spring MVC Controllers | CarController |
|  /cars [POST] | HTTP Server Spring MVC Controllers | CarController |
|  /cars/set_car_to_person/{person_id}/{car_id} [POST] | HTTP Server Spring MVC Controllers | CarController |
|  /cars/set_image_name/{car_id}/{image_name} [POST] | HTTP Server Spring MVC Controllers | CarController |
|  /cars/{id} [GET] | HTTP Server Spring MVC Controllers | CarController |
|  /cars/{id} [DELETE] | HTTP Server Spring MVC Controllers | CarController |
|  /authenticate [POST] | HTTP Server Spring MVC Controllers | AuthenticationController |
|  /register [POST] | HTTP Server Spring MVC Controllers | AuthenticationController |
|  /people [GET] | HTTP Server Spring MVC Controllers | PeopleController |
|  /people [PUT] | HTTP Server Spring MVC Controllers | PeopleController |
|  /people/{id} [GET] | HTTP Server Spring MVC Controllers | PeopleController |
|  /people/{id} [DELETE] | HTTP Server Spring MVC Controllers | PeopleController |

Рисунок 4 – Список эндпоинтов разработанного API

6 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ИНТЕРНЕТ-РЕСУРСА

Frontend-часть разработанного приложения представляет собой динамичный пользовательский интерфейс, созданный с использованием современных технологий. Основной фреймворк, на котором построено приложение, — React. React обеспечивает эффективное управление состоянием компонентов, и это ядро взаимодействия пользователя с интерфейсом.

Redux применяется для управления глобальным состоянием приложения. Состояние хранится централизованно, что обеспечивает легкость отслеживания и управления данными. Взаимодействие с серверной частью приложения реализовано с применением библиотеки axios внутри Redux Toolkit. Axios позволяет осуществлять HTTP-запросы, а Redux Toolkit эффективно интегрирует их в глобальное состояние.

React Router DOM обеспечивает навигацию между различными страницами приложения. Он поддерживает динамический рендеринг компонентов в зависимости от URL, что способствует логичному и удобному взаимодействию пользователя с интерфейсом.

Стили интерфейса реализованы с применением CSS. Этот выбор обеспечивает гибкость в оформлении и структурировании элементов интерфейса, позволяя создавать уникальные и современные дизайнерские решения.

Ключевым моментом в архитектуре приложения является использование Redux Toolkit и axios для обеспечения эффективного взаимодействия с сервером. Redux Toolkit позволяет создавать срезы состояния, асинхронно обновлять их с использованием axios и обрабатывать результаты запросов. Это содействует легкости разработки, поддержки и масштабирования приложения. Благодаря React и React Router DOM пользовательский интерфейс становится отзывчивым, дружелюбным к пользователю и легко расширяемым для будущих изменений и дополнений.

7 ПРОВЕДЕНИЕ ТЕСТИРОВАНИЯ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ

На рисунке 5 изображено тестирование регистрации. Отправляется запрос с почтой, именем пользователя и паролем, при этом возвращается JWT токен.

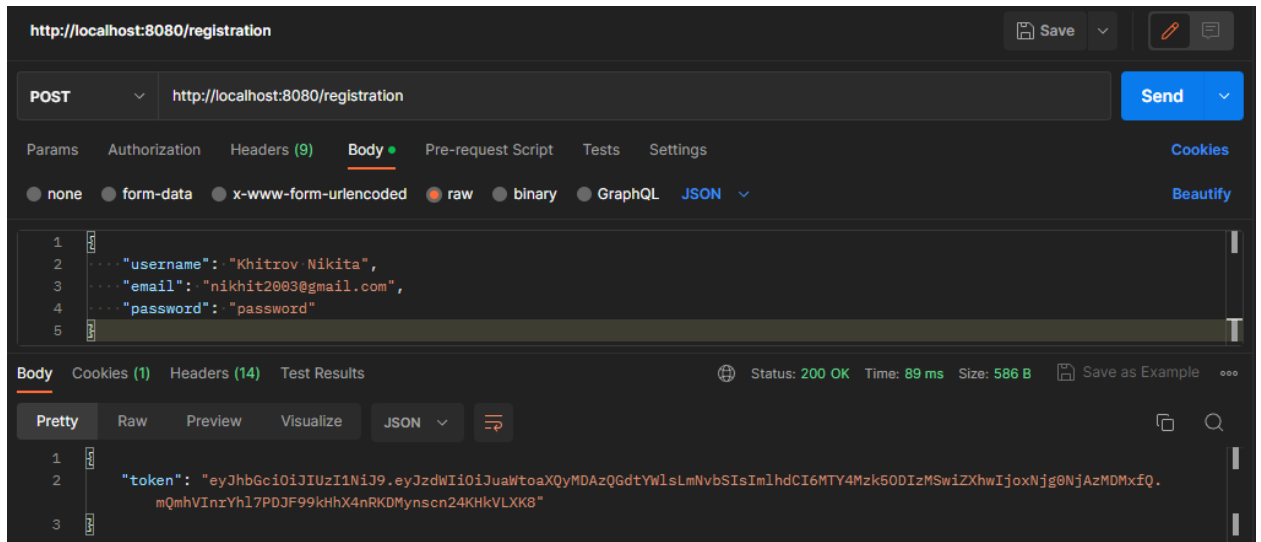


Рисунок 5 – Тестирование регистрации

На рисунке 6 изображено тестирование авторизации. Отправляется запрос с почтой и паролем, при этом возвращается JWT токен.

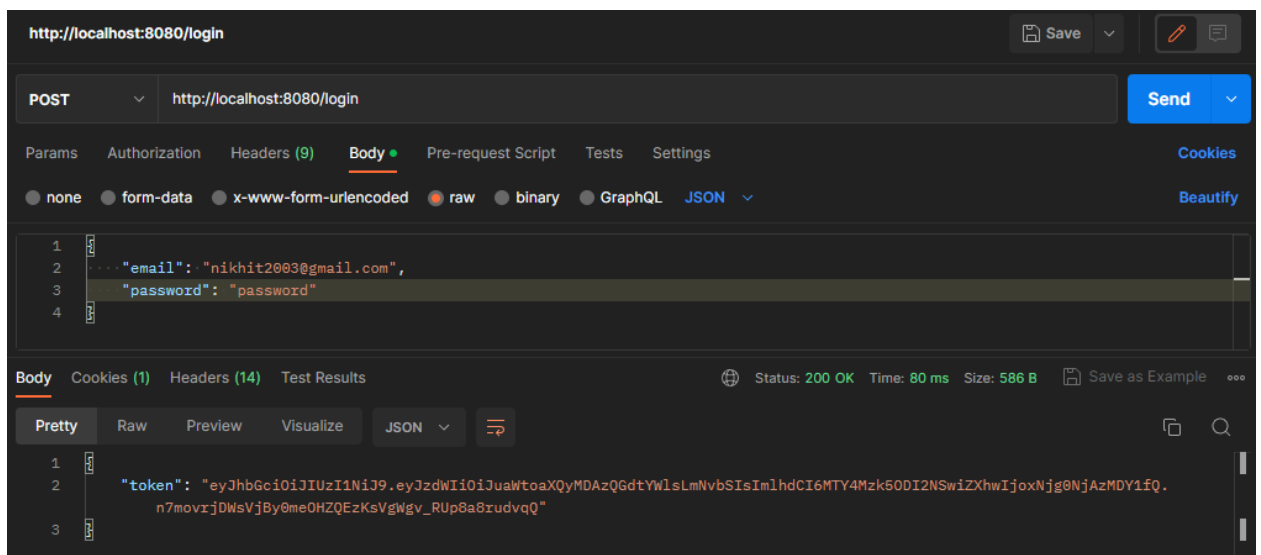


Рисунок 6 – Тестирование авторизации часть 1

На рисунке 7 изображено тестирование авторизации. Отправляется запрос с почтой и паролем, при этом возвращается статус ошибки 403, так как такого пользователя не существует.

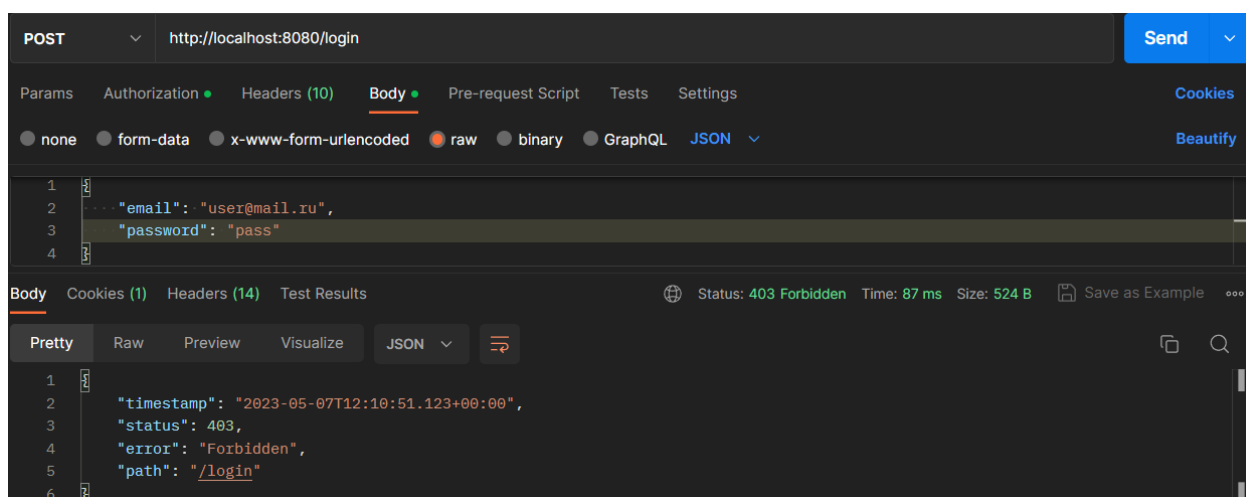


Рисунок 7 – Тестирование авторизации часть 2

На главной странице (рисунок 8) представлено тестирование внешнего вида сайта, включая проверку корректного отображения основного контента, элементов навигации и графических элементов, таких как баннеры и изображения.

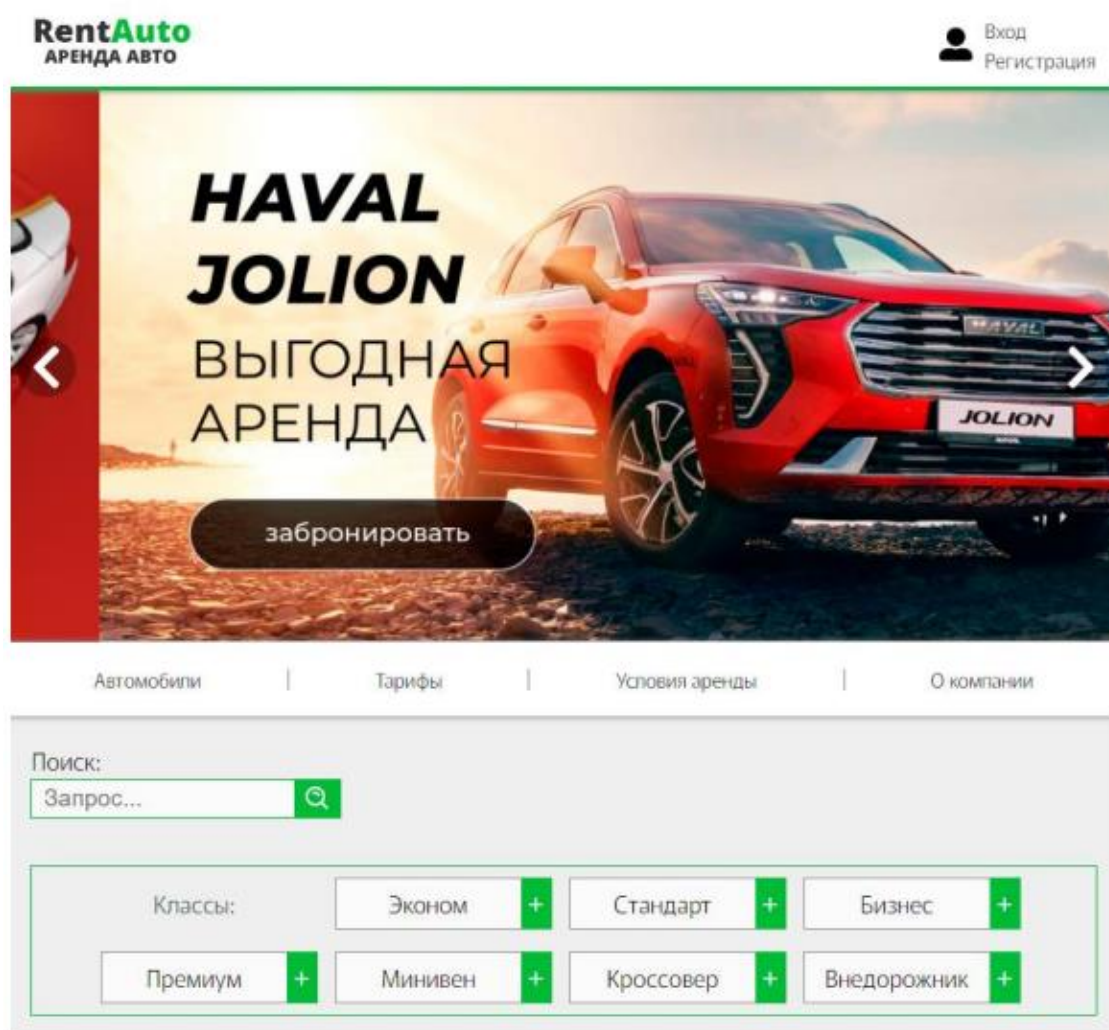


Рисунок 8 – Тестирование главной страницы

На рисунке 9 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.

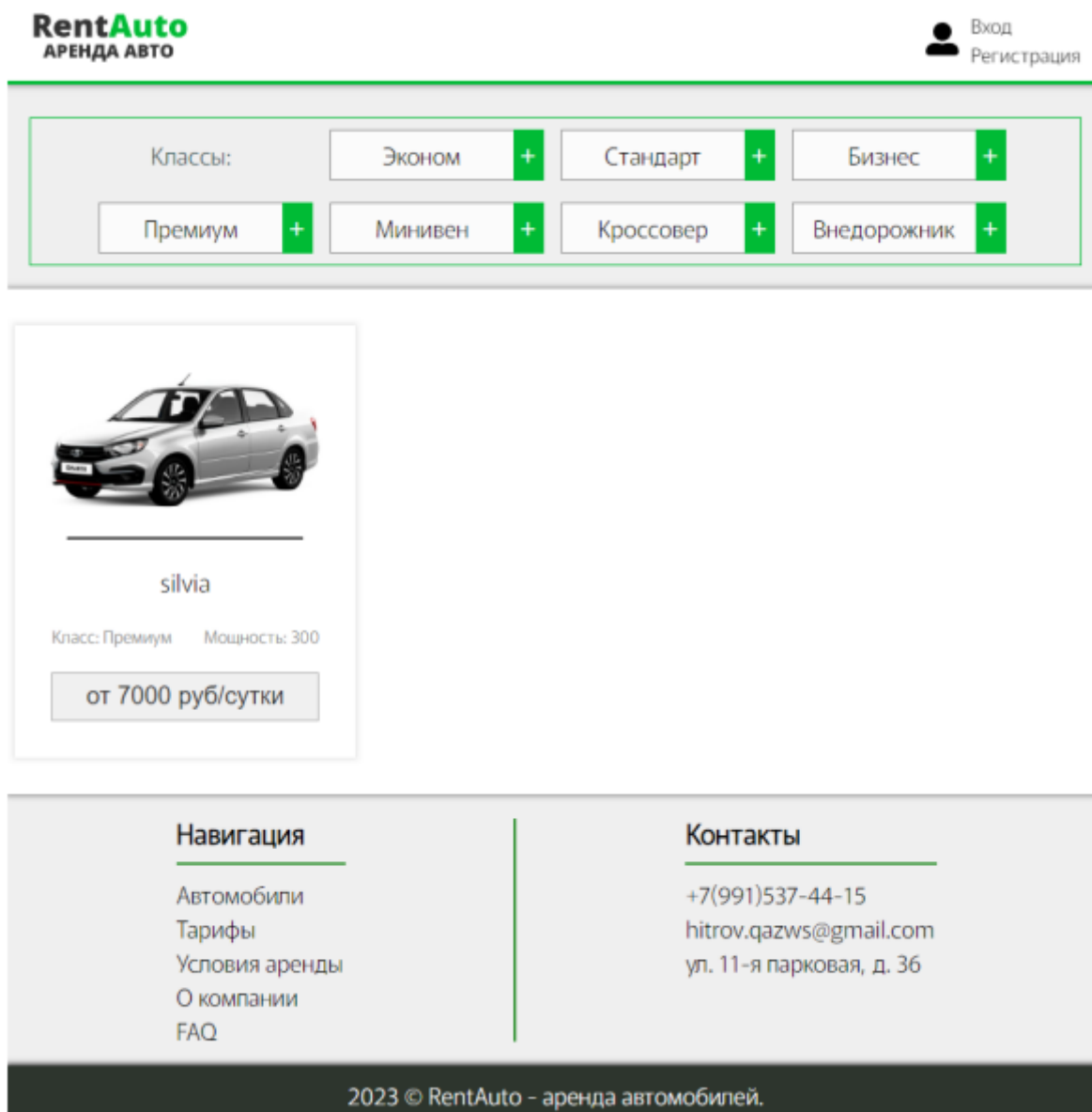


Рисунок 9 – Тестирование получения списка автомобилей

На рисунке 10 отображается тестирование страницы с тарифами на авто. Этот процесс включает в себя проверку корректного отображения тарифов, их описаний.

The screenshot shows the RentAuto website interface. At the top, there is a header with the logo 'RentAuto АРЕНДА АВТО' on the left and 'Вход' and 'Регистрация' on the right. Below the header is a large promotional banner with a blue circle containing '10% НА АРЕНДУ' and the text 'АРЕНДА АВТО НА САЙТЕ ПОЛУЧИ СКИДКУ!'. Below the banner is a navigation bar with links: 'Автомобили', 'Тарифы', 'Условия аренды', and 'О компании'. The main content area is titled 'Список тарифов:' and contains a table with four columns: 'Категория', 'Минимальная цена', 'Максимальная цена', and 'Депозит'. The table lists eight categories: Эконом, Стандарт, Бизнес, Премиум, Минивен, Кроссовер, and Внедорожник, each with its corresponding price range and deposit. At the bottom of the table, a note states 'Все цены указаны в рублях без НДС'.

| Категория | Минимальная цена | Максимальная цена | Депозит |
|-------------|------------------|-------------------|------------|
| Эконом | 1300 руб./сутки | 2300 руб./сутки | 10000 руб. |
| Стандарт | 2300 руб./сутки | 2800 руб./сутки | 11500 руб. |
| Бизнес | 3400 руб./сутки | 4900 руб./сутки | 17300 руб. |
| Премиум | 8300 руб./сутки | 9400 руб./сутки | 22000 руб. |
| Минивен | 4900 руб./сутки | 8300 руб./сутки | 17300 руб. |
| Кроссовер | 2800 руб./сутки | 3400 руб./сутки | 14000 руб. |
| Внедорожник | 9400 руб./сутки | 11200 руб./сутки | 30000 руб. |

Все цены указаны в рублях без НДС

Рисунок 10 – Тестирование страницы списка тарифов

На рисунке 11 представлено тестирование страницы "О компании". В этом случае проводится проверка корректного отображения информации о компании, ее истории, миссии, а также работы взаимосвязей между различными блоками на странице.

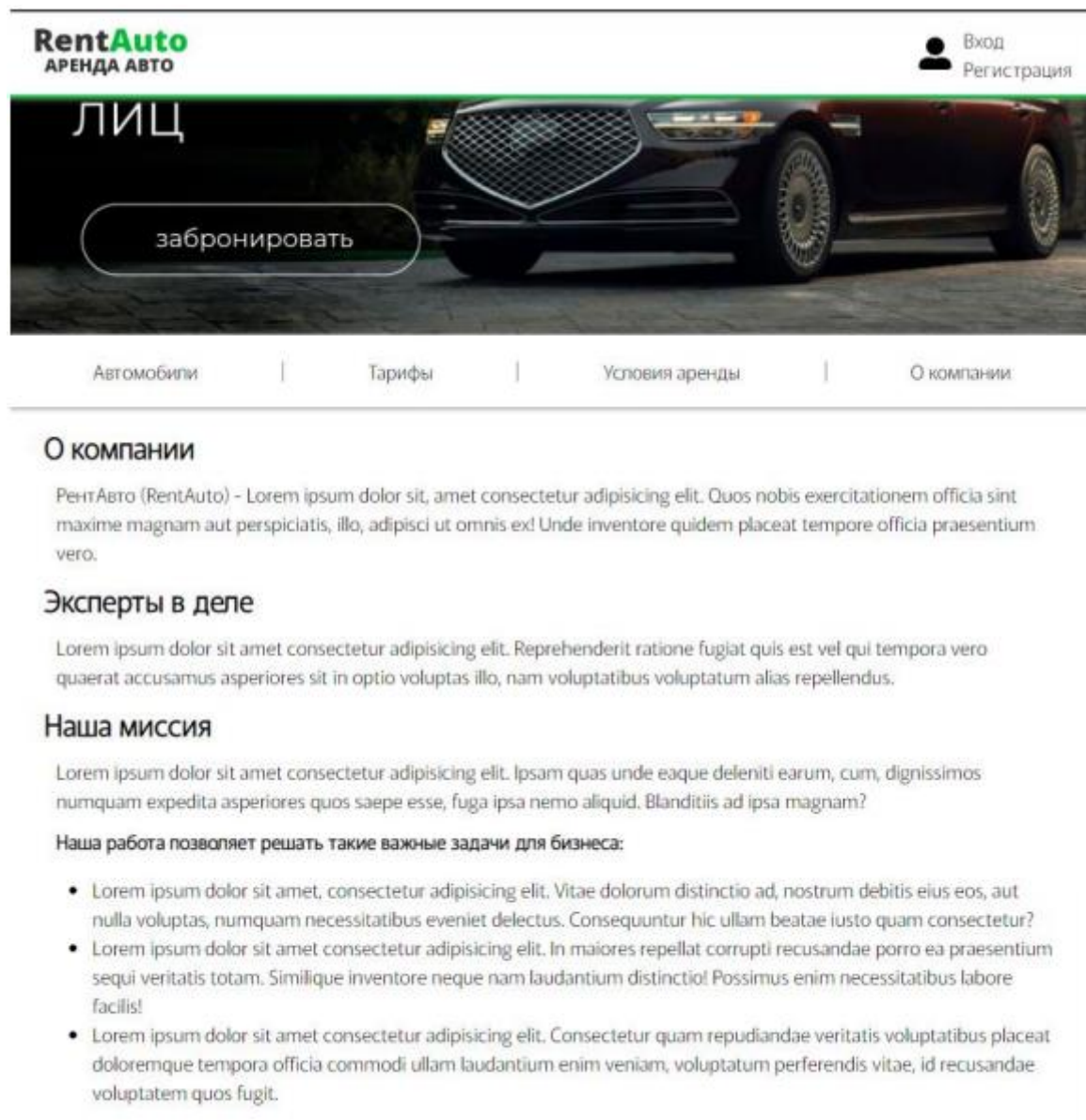
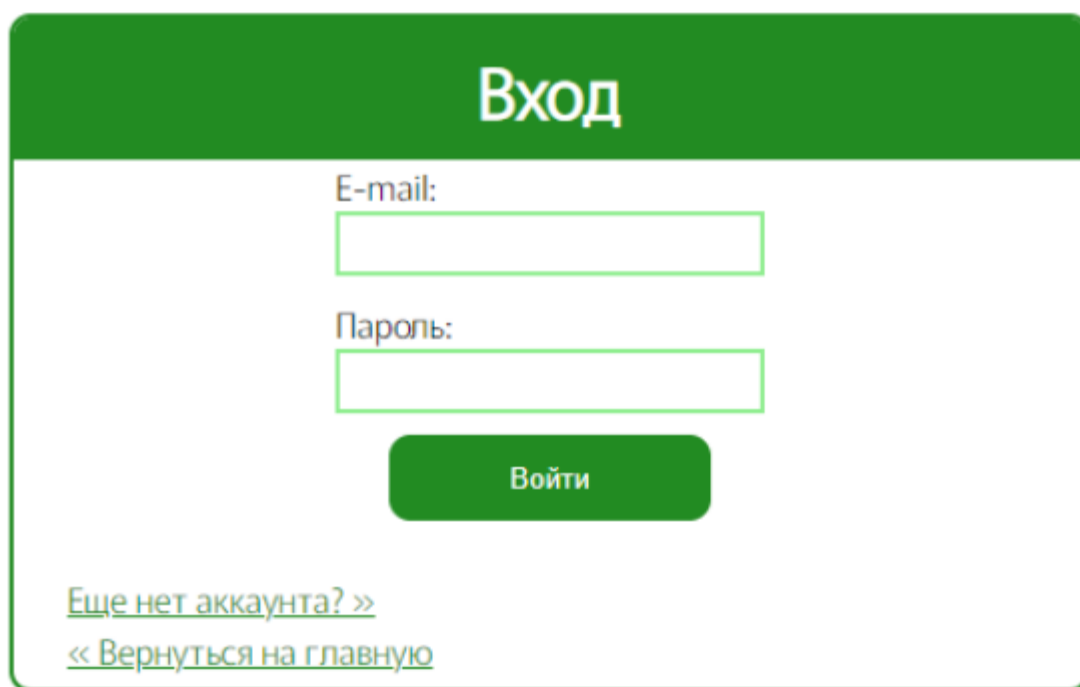


Рисунок 11 – Тестирование страницы «О компании»

Рисунок 12 отражает тестирование страницы авторизации. Это включает в себя проверку корректного функционирования формы ввода логина и пароля, аутентификации пользователя.



Вход

E-mail:

Пароль:

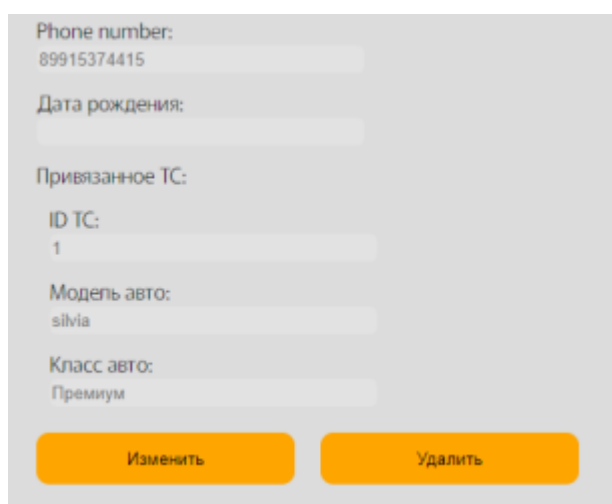
Войти

[Еще нет аккаунта? >>](#)

[<< Вернуться на главную](#)

Рисунок 12 – Тестирование авторизации пользователя

Рисунок 13 демонстрирует тестирование личного кабинета. Это включает проверку корректного отображения персональной информации пользователя, истории поездок, а также функциональности для изменения настроек профиля.



Phone number:
89915374415

Дата рождения:

Привязанное ТС:

ID ТС:
1

Модель авто:
sylvia

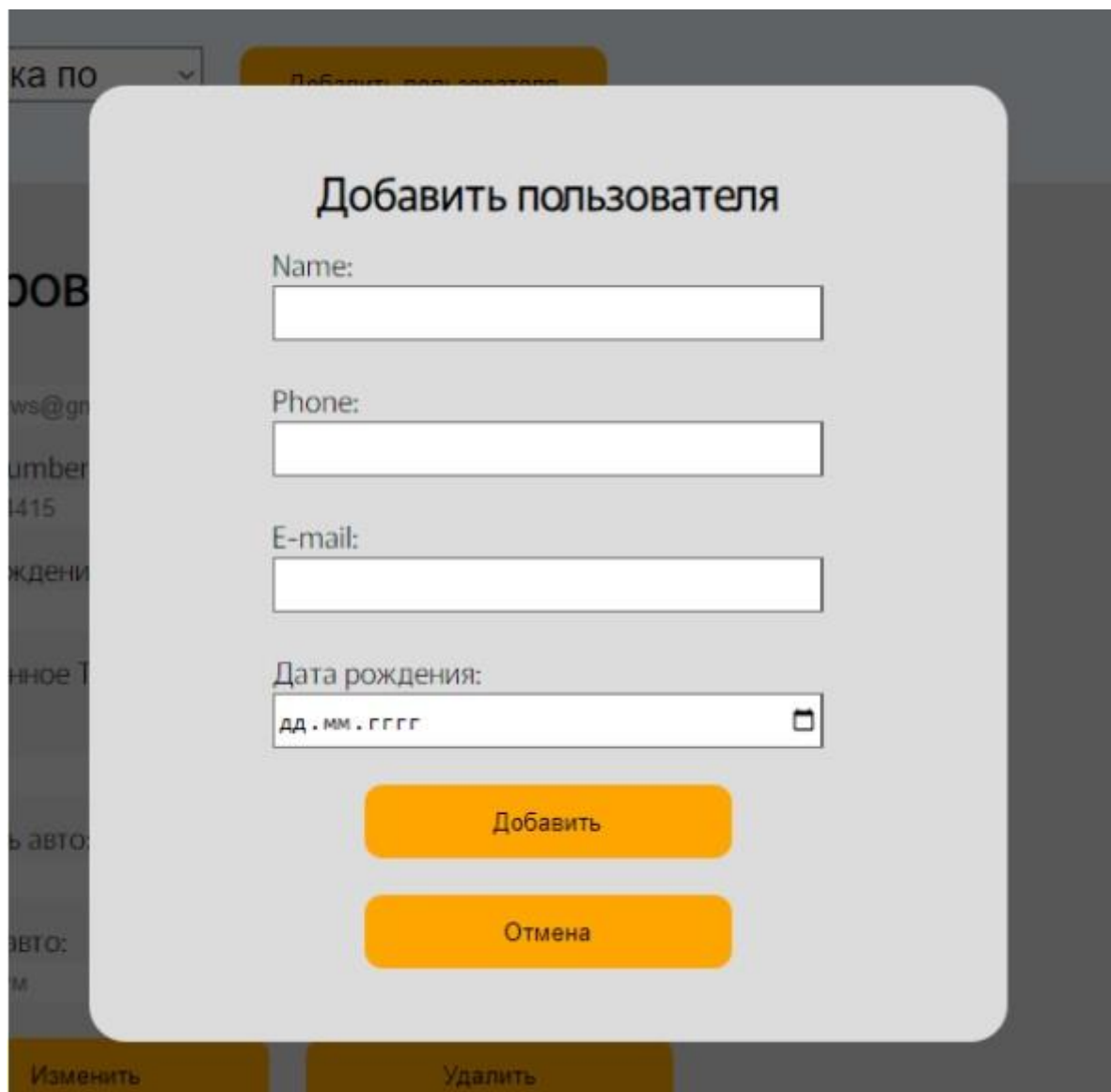
Класс авто:
Премиум

Изменить

Удалить

Рисунок 13 – Тестирование редактирования информации о пользователе

На рисунке 14 видно тестирование страницы добавления и модерирования пользователя для администратора. Это включает в себя проверку корректного функционирования формы добавления пользователя, а также соответствующих механизмов модерации.



The image shows a web application interface for user management. A modal window titled "Добавить пользователя" (Add user) is displayed in the center. It contains four input fields: "Name:", "Phone:", "E-mail:", and "Дата рождения:" (Date of birth:). The "Дата рождения:" field has a calendar icon. Below the fields are two orange buttons: "Добавить" (Add) and "Отмена" (Cancel). The background shows a list of users with buttons like "Изменить" (Edit) and "Удалить" (Delete).

Рисунок 14 – Тестирование добавления пользователя

Рисунок 15 демонстрирует тестирование страницы добавления и модерирования автомобиля для администратора. В этом случае проводится проверка корректного функционирования формы добавления автомобиля, его модерации, и взаимодействия с базой данных.

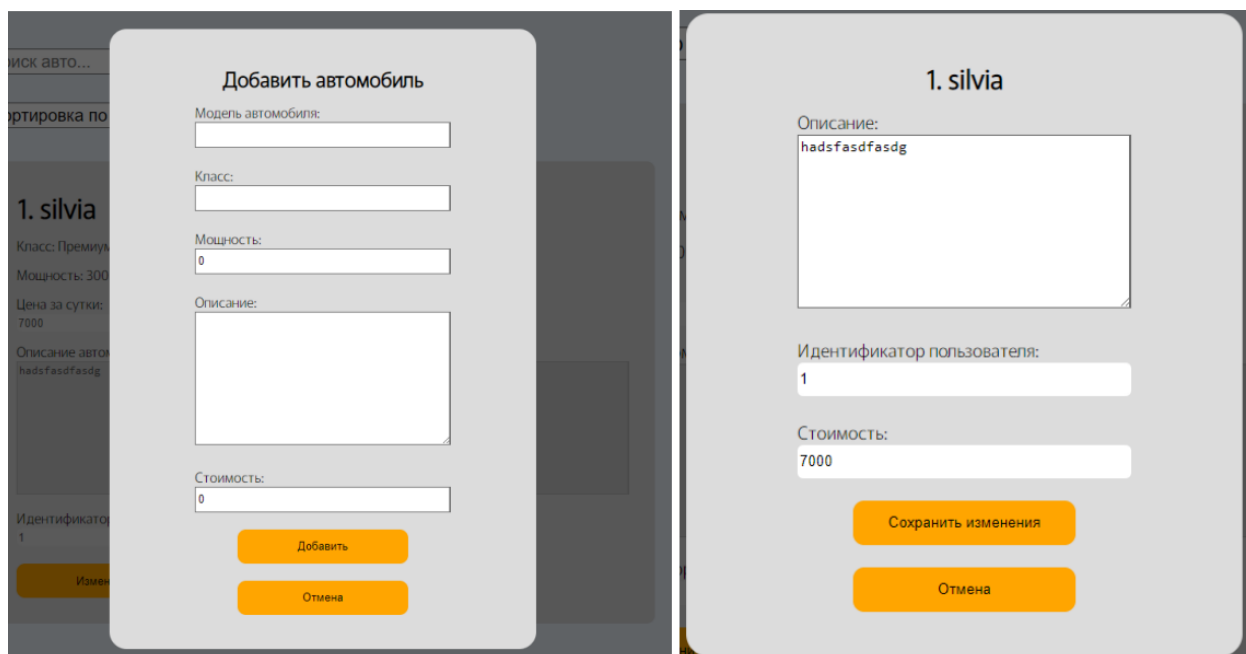


Рисунок 15 – Тестирование добавления и редактирования автомобиля

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.

Исходный код серверной части приложения доступен по ссылке – <https://github.com/matrosovcmtn/car-rental-service>

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

```
12  ~ @Data
13    @Builder
14    @AllArgsConstructor
15    @NoArgsConstructor
16    @Entity
17    @Table(name = "car")
18    public class Car {
19
20    ~    @Id
21        @Column(name = "id")
22        @GeneratedValue(strategy = GenerationType.IDENTITY)
23        private int id;
24
25        @Column(name = "model_name")
26        private String modelName;
27
28        @Column(name = "horse_powers")
29        private int horsePowers;
30
31        private String description;
32
33        private String category;
34
35        private int price;
36
37        private String imageName;
38
39        private boolean isTaken;
40
41    ~    @JsonIgnore
42        @OneToOne
43        @JoinColumn(name = "person_id", referencedColumnName = "id")
44        private Person person;
45
46    }
```

Рисунок А.1 – Фрагмент кода класса автомобиля


```

17  @Service
18  @AllArgsConstructor
19  public class CarService {
20
21      private final CarRepository carRepository;
22
23  >  public void create(Car car) { carRepository.save(car); }
26
27
28  >  public List<Car> findAll() { return carRepository.findAll(); }
31
32  public Car findOne(int id) {
33      Optional<Car> foundCar = carRepository.findById(id);
34      return foundCar.orElseThrow(EntityNotFoundException::new);
35  }
36
37  >  public Car update(Car car) { return carRepository.save(car); }
40
41  >  public void delete(int id) { carRepository.deleteById(id); }
44
45
46
47  }

```

Рисунок А.2 – Фрагмент кода класса сервиса заказов