МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования
“Тольяттинский государственный университет”

09.03.03 Прикладная информатика

(код и наименование направления подготовки / специальности)

Разработка программного обеспечения

(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

на тему “Разработка веб-приложения для размещения объявлений о продаже и аренде спецтехники для юридических лиц”

Обучающийся Н.А. Кутьин
(Инициалы Фамилия)

(личная подпись)

Руководитель доцент, канд. пед. наук, Е.А. Ерофеева
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант канд. филол. наук, доцент М.В. Дайнеко
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Тема выпускной квалификационной работы: “Разработка веб-приложения для размещения объявлений о продаже и аренде спецтехники для юридических лиц”.

Выпускная квалификационная работа посвящена разработке веб-приложения для размещения объявлений о продаже и ранде спецтехники для юридических лиц. В результате создано веб-приложение, прошедшее все этапы тестирования и готовое к внедрению в деятельность предприятий, работающих в секторе аренды и продажи спецтехники.

Во введении обоснована актуальность темы, сформированы цели и задачи.

В первой главе проведён подробный обзор рынка спецтехники, выполнен анализ бизнес-процессов и существующих решений, сформулированы функциональные и нефункциональные требования к системе.

Во второй главе разработана архитектура веб приложения, спроектированы база данных и интерфейс пользователя, обоснован выбор стека, реализованы ключевые модули управления объявлениями и пользователями.

В третьей главе выполнено модульное, интеграционное, функциональное и нагрузочное тестирование.

В заключении подведены итоги и описывается полученный результат.

Выпускная квалификационная работа состоит из введения, трех глав, заключения и списка литературы и используемых источников.

Выпускная квалификационная работа состоит из 49 страниц текста, 32 рисунков, 8 таблиц, содержит информацию из 20 источников.

Abstract

The present graduation work is devoted to developing a web application for placing advertisements referred to selling and renting special-purpose equipment for juristic persons.

The graduation work consists of an introduction, 3 parts, 32 figures, 8 tables, a conclusion, and a list of 20 references.

The aim of this research is to develop a fully functional web application for juristic persons to place advertisements related to selling and renting the special-purpose equipment.

The object of the graduation work is the process of placing and managing advertisements related to special-purpose equipment.

The subject of the graduation work is the design and implementation of a web-based system to support businesses in the sphere of selling and renting special-purpose equipment.

The key issue of the graduation work is the creation of a reliable and user-friendly platform that meets both functional and non-functional requirements of the target market.

The graduation work may be divided into several logically connected parts which are: market and business processes analysis, system design and development, as well as its testing.

The first part of the research deals with carrying out a detailed analysis of the special-purpose equipment market. This part also considers the existing solutions, and formulates the requirements for the system. The second part describes the system architecture. In this part, the database and the user interface are designed, the choice of technologies is explained, and the core modules are implemented. The third part presents the results of the module, integration, functional and load testing.

In conclusion, it should be noted that the application is ready for implementation.

Оглавление

Введение

В последние годы наблюдается рост рынка аренды и продаже специализированной техники для строительства, добывающей и промышленной отраслей. Это обусловлено рядом факторов, таких как расширение масштабов инфраструктурных проектов, повышение темпов добычи полезных ископаемых, активизация промышленного производства, стремление компаний оптимизировать собственные ресурсы. При этом многие юридические лица продолжают вести учет техники вручную, формируют договоры через почту и телефон, что может привести к:

- задержкам при обработке заявок;
- высокому риску ошибок из-за человеческого фактора;
- ограниченной прозрачности процессов для заказчика и компании.

Разработка специализированного веб-приложения позволит автоматизировать жизненный цикл заявки: от подачи заявки до ее подписания и оплаты, что повысит скорость, прозрачность процессов, качество обслуживания, снизит операционные издержки, сократит время обработки заявки и укрепит конкурентные позиции компании.

Объект исследования - бизнес-процессы при размещении и обработке объявлений о продаже и аренде спецтехники юридическим лицам.

Предмет исследования – методы и инструменты автоматизации этих бизнес-процессов посредством веб-приложения, включающие разработку архитектуры системы, проектирование базы данных, создание пользовательского интерфейса и обеспечение взаимодействия клиентской и серверной частей.

Целью бакалаврской работы является разработка веб-приложения для размещения объявлений о аренде и продаже спецтехники, оптимизирующего ключевые бизнес-процессы и удовлетворяющее требованиям юридических лиц.

Для достижения цели нужно выполнить следующие задачи:

– проанализировать текущие бизнес-процессы “как есть” и выявить узкие места;
– изучить лучшие практики и существующие решения на рынке;
– сформулировать функциональные и нефункциональные требования к системе;
– разработать архитектуру и информационную модель приложения;
– реализовать клиентскую и серверную части веб-приложения;
– провести тестирование, отладку, оценить производительность.

В работе применяются модели бизнес-процессов, методы системного проектирования, инструменты прототипирования и верстки, практические приемы разработки и тестирования ПО.

Разработанное веб-приложение ориентировано на автоматизацию процессов размещения объявлений о продаже и аренде спецтехники для юридических лиц. Его новизна заключается в подходе к автоматизации B2B процессов аренды и продажи спецтехники, комплексной реализации пользовательского интерфейса и серверной логики на основе современных технологий (React, NestJS, MongoDB). Отличается возможностью гибкой кастомизации под потребности конкретного предприятия.

Глава 1 Анализ предметной области и требований

1.1 Значение программного обеспечения для информационной системы компании

В современных условиях ведения бизнеса эффективность предприятия в значительной степени зависит от уровня автоматизации бизнес-процессов. Один из ключевых элементов автоматизации – информационная система, основой которой является программное обеспечение.

Программное обеспечение обеспечивает выполнение основных функций информационной системы, включая:

- сбор, хранение и обработку информации.
- автоматизацию рутинных операций.
- поддержку принятия управленческих решений.
- взаимодействие с пользователями и внешними системами.

Для компаний, которые работают в сфере аренды и продажи специализированной техники, информационные системы позволяют повысить прозрачность и управляемость операций. Программное обеспечение может обеспечить такие функции как:

- ведение базы данных техники, с учетом ее характеристик, состояния и т.д;
- прием и обработка заявок на покупку или аренду;
- автоматическое формирование коммерческих предложений и договоров;
- интеграция с платежными системами и системами документооборота;
- различного рода аналитика.

На рисунке 1 представлена диаграмма Исикавы (причинно-следственная диаграмма), которая иллюстрирует причины низкой эффективности бизнес-процессов в компаниях без автоматизации.

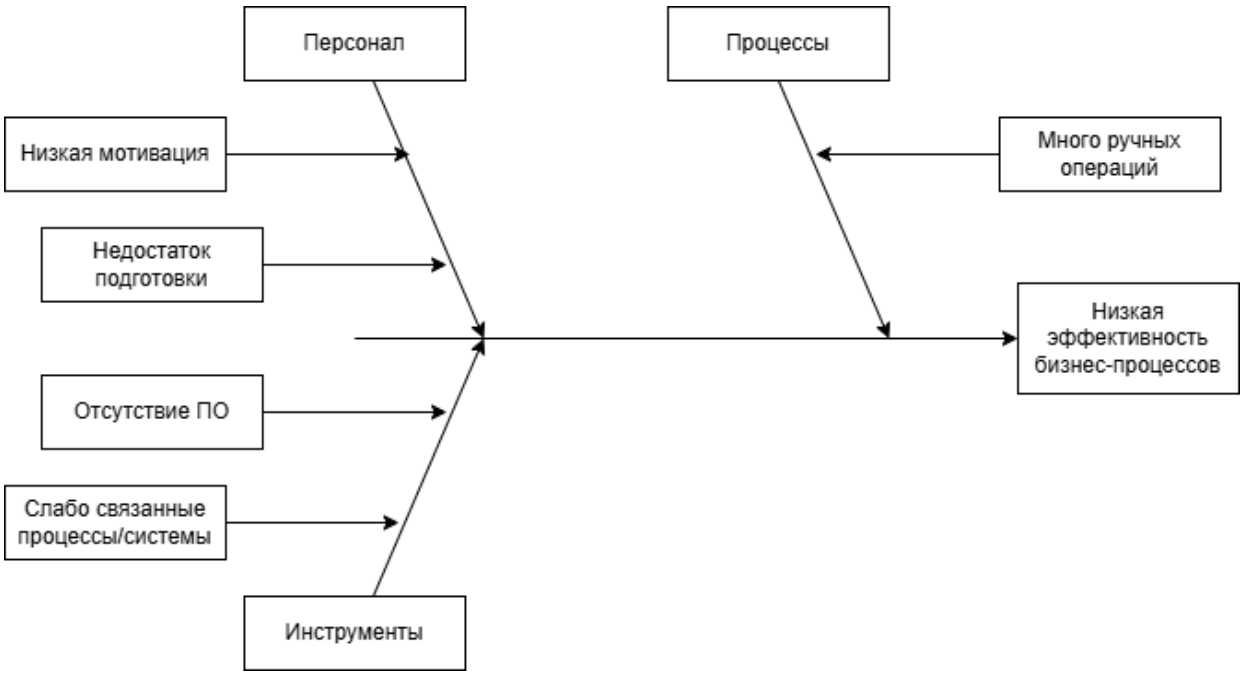


Рисунок 1 - Диаграмма “Причина-следствие”

Диаграмма показывает, что отсутствие программного обеспечения или его неэффективное использование может быть одной из ключевых причин снижения производительности. Программное обеспечение позволяет устранить ручные операции, улучшить координацию между отделами и снизить зависимость от человеческого фактора. Следовательно, разработка специализированного веб-приложения способствует устранению множества первопричин низкой эффективности. Как показывает практика, компании, использующие автоматизированные информационные системы, имеют следующие преимущества:

- сокращение времени обработки заявок на 50-70%;
- меньшее количество ошибок и дублирующих операций;
- повышение удовлетворенности клиентов за счет оперативной и точной обработки обращений;
- уменьшение издержек на персонал и бумажный документооборот.

Разработка собственного программного обеспечения позволяет учесть особенности данной предметной области, адаптировать логику системы под

бизнес-процессы компании, обеспечить конкурентное преимущество. Особенно это актуально для компаний, которые работают с юридическими лицами, для которых важны стандартизированное оформление документов, контроль выполнения договоров и отслеживание операций через личный кабинет.

Таким образом, программное обеспечение является неотъемлемой частью эффективной информационной системы предприятия и играет ключевую роль в цифровой трансформации бизнеса.

1.2 Основные функции программного обеспечения для информационной системы компании

Программное обеспечение информационной системы компании представляет собой совокупность программных модулей, обеспечивающих автоматизацию бизнес-процессов, поддержку принятия управленческих решений и взаимодействие с внешними системами. Основной задачей такого ПО является повышение эффективности деятельности предприятия за счёт цифровизации процессов.

Функции программного обеспечения можно классифицировать по следующим направлениям. Сбор и обработка данных - обеспечивается сбор, хранение и валидация информации, которая поступает из различных источников:

– регистрация заявок от клиентов;
– ввод данных о наличии и характеристиках спецтехники;
– импорт информации из бухгалтерских, складских и CRM-систем;
– сбор пользовательской активности для аналитики.

На рисунке 2 представлена DFD модель уровня 0, описывающая ключевые потоки данных между сущностями, хранилищами и процессами:

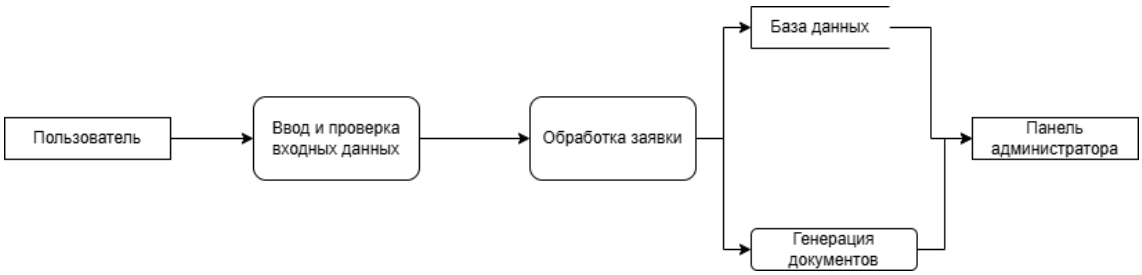


Рисунок 2 - Схема потока данных.

DFD-модель описывает, как данные от внешнего пользователя поступают в систему, проходят обработку на разных логических уровнях (валидация, логика обработки заявок, генерация документов), сохраняются в хранилище, используются для построения отчётов и передаются в административный интерфейс. Такая модель подчёркивает поток информации и взаимодействие между ключевыми компонентами ИС и открытые API предоставляют доступ к данным и управлению системой для внутренних пользователей и внешних интеграций.

Функции управления бизнес-процессами. Программное обеспечение позволяет реализовать:

- контроль жизненного цикла заявки;
- управление договорами и расчетами;
- контроль наличия и бронирования техники;
- маршрутизацию процессов между отделами.

Система может автоматически уведомлять ответственных сотрудников и клиентов о статусе процессов, сокращая время на согласования.

В таблице 1 можно ознакомиться с уже имеющимися на рынке решениями, их кратким описанием и преимуществами.

Таблица 1 – сравнение систем аренды и покупки спецтехники

Система	**Описание**	**Преимущества**
PlantHire	Гибко настраиваемое ПО для управления арендой оборудования	Интеграция с учетными системами Гибкое выставление счетов Мобильная доставка Полный набор отчетов
RentalWorks	Комплексное решение для управления арендой	Доступность и статусы в реальном времени Интегрированный модуль бухгалтерии и закупок Сканирование штрих-кодов и RFID-меток Настраиваемые пакеты и тарифы услуг
MachineryTrader	Онлайн площадка для купли-продажи б/у строительной и спецтехники	Широкая аудитория Маркетинговые инструменты Мобильность Рыночная аналитика

Функции взаимодействия с пользователем. Программное обеспечение реализует пользовательские интерфейсы, через которые осуществляется:

– размещение и редактирование объявлений;

– фильтрация и поиск спецтехники по параметрам;

– просмотр истории заявок и документов;

– получение уведомления о смене статуса или необходимости действия.

Клиентоориентированный интерфейс повышает лояльность пользователей и снижает нагрузку на операторов.

Функции аналитики и отчетности. Современные информационные системы включают модули формирования отчетов и визуализации данных:

– динамика спроса и предложения;

– загрузка техники по периодам;

– показатели по доходности и эффективности сделок;

– прогнозирование потребностей.

На основе собранных данных система может формировать рекомендации для менеджмента.

Интеграционные функции. Программное обеспечение поддерживает интеграцию с различными внешними сервисами и внутренними системами:

– платежные шлюзы и банковские API;

– бухгалтерские и складские системы;

– электронный документооборот и цифровая подпись;

– сторонние сервисы аренды и логистики.

Такая интеграция позволяет построить единую цифровую экосистему предприятия.

Основные функции программного обеспечения охватывают все ключевые аспекты работы информационной системы предприятия — от приёма информации и управления заявками до аналитики и взаимодействия с пользователями. Именно такое программное обеспечение становится основой эффективной, прозрачной и масштабируемой цифровой среды, способной адаптироваться под требования бизнеса.

1.3 Обзор существующих решений и их анализ

Далее представлен анализ популярных решений на рынке аренды и продаже спецтехники – RentalWorks, MachineryTrader и HirePOS (PlantHire) – с анализом их ключевых возможностей, сильных и слабых сторон.

RentalWorks – ПО для управления арендной деятельностью, сочетающее функции учета инвентаря и финансового учета в единой системе. Оно поддерживает работу нескольких офисов и складов, оплату в нескольких валютах, а также обеспечивает настраиваемые формы, отчеты и оповещения в реальном времени. Благодаря обширным API RentalWorks легко интегрируется с внешними бухгалтерскими системами и CRM, что позволяет обеспечить прозрачность бизнес-процессов.

Преимущества:

– полный цикл учета и управления договорами;

– высокая степень настройки интерфейсов и отчетов;

– надежная интеграция с учетными системами.

Недостатки:

– отсутствие бесплатного пробного периода;

– сложность внедрения для очень малых компаний.

MachineryTrader – облачная платформа-рынок для размещения объявлений о продаже и аренде новой и б/у строительной техники. Продавцы могут создавать детализированные лоты с техническими характеристиками, фото, видео, а также настраивать собственный “e-commerce shop” для прямых продаж через сайт. Платформа представляет из себя маркетплейс.

Преимущества:

– масштабируемость и доступ международной аудитории;

– простота создания и продвижения объявлений;

– инструменты аналитики трафика и заинтересованности покупателей.

Недостатки:

– отсутствие встроенных инструментов учета инвентаря и документооборота;

– комиссия за размещение или платные пакеты для премиум-листингов.

HirePOS – специализированное ПО для plant hire-бизнеса, ориентированное на управление парком оборудования, техническое обслуживание и документооборот. Система позволяет отслеживать состояние техники через плановые инспекции, автоматизировать сервисные заказы на основе наработки часов и вести историю ремонтов и ТО каждого агрегата. Поддерживает несколько площадок хранения, настраиваемые шаблоны документов и гибкую систему оплаты, что ценно для малых и средних компаний.

Преимущества:

– специализация на plant hire с учетом нормативов;

– локальная поддержка и возможность адаптации под требования заказчика;
– простой интерфейс и быстрый запуск.

Недостатки:

– узкая географическая направленность (Австралия, Новая Зеландия);
– меньше возможностей для международных интеграций по сравнению с глобальными платформами.

Таблица 2 представляет собой таблицу сравнительного анализа данных решений по следующим критериям: функциональность, стоимость, удобство использования, поддержка и безопасность.

Таблица 2. Сравнительный анализ существующих решений

Решение	**Функциональность**	**Стоимость**	**Удобство использования**	**Поддержка**	**Безопасность**
RentalWorks	5	3	4	5	5
MachineryTrader	3	4	5	3	4
HirePOS	4	4	4	4	4

Сравнив эти решения, становится понятно, что RentalWorks демонстрирует максимальную сбалансированность по большинству критериев, MachineryTrader выгоден по стоимости и простоте, а HirePOS представляет собой универсальное решение с умеренными показателями по всем направлениям.

С визуализацией анализа в виде столбчатой диаграммы можно ознакомиться на рисунке 3.

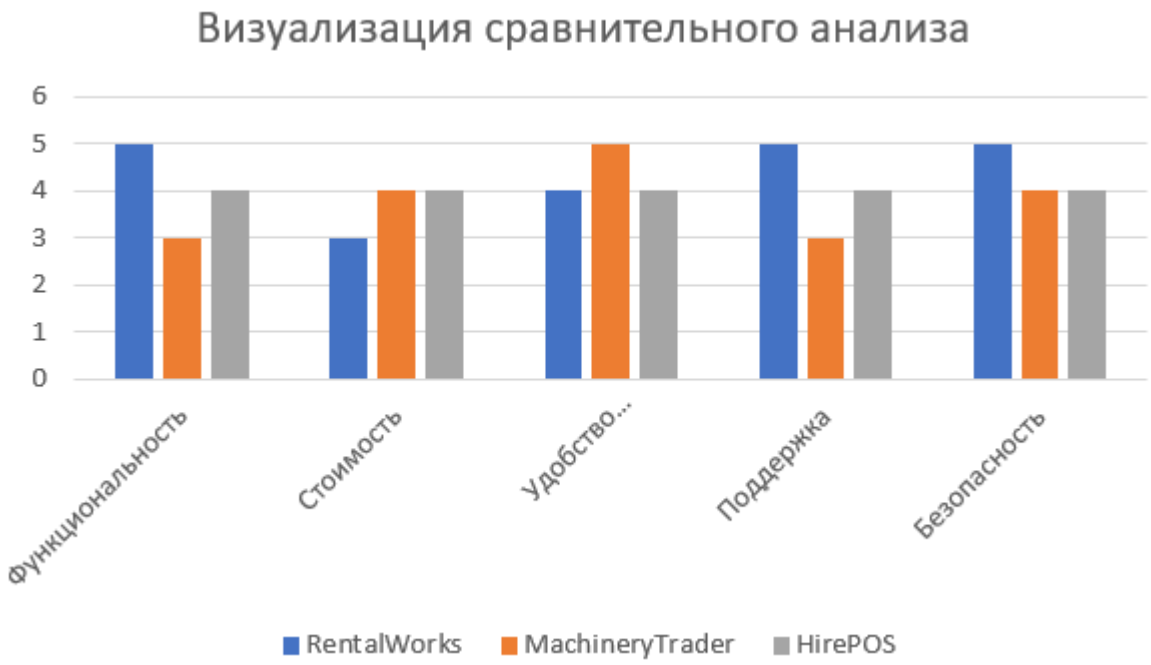


Рисунок 3 - Столбчатая диаграмма сравнительного анализа решений.

По результатам анализа можно сделать следующий вывод: RentalWorks идеально подходит крупным и средним прокатным компаниям, которым необходим полный контроль над техникой, документооборотом и отчётностью, а также интеграция с бухгалтерскими системами. MachineryTrader больше подойдёт малым и средним компаниям, желающим быстро выйти на рынок и привлечь клиентов через онлайн-платформу без необходимости внедрения сложного ПО. HirePOS наилучшим образом подойдёт небольшим региональным организациям, осуществляющим аренду техники с полным контролем за техническим состоянием и локальной спецификой бизнеса.

Таким образом, выбор системы зависит от целей бизнеса: от лёгкого маркетинга до глубокой автоматизации и масштабируемости внутренних процессов.

1.4 Требования к функционалу и интерфейсу приложения

Разработка веб-приложения для размещения объявлений о продаже и аренде спецтехники юридическими лицами требует учёта как бизнес-потребностей клиентов, так и современных стандартов проектирования информационных систем. Основные требования к функционалу и интерфейсу приложения сформулированы на основе анализа предметной области, изучения аналогичных решений и предполагаемых сценариев использования.

Функциональные требования. Ключевые функции веб-приложения можно разделить на следующие блоки:

Управление объявлениями:

- добавление, редактирование объявлений;
- загрузка фотографий;
- указание стоимости, доступности, технических характеристик;
- возможность временной деактивации объявлений;
- просмотр объявлений.

Работа с пользователями:

- регистрация и авторизация юридических лиц через электронную почту;
- разграничение прав пользователя;
- просмотр профиля пользователя.

На рисунке 4 представлены блок-схемы регистрации и авторизации на стороне клиента, на рисунке 5 – на стороне сервера.

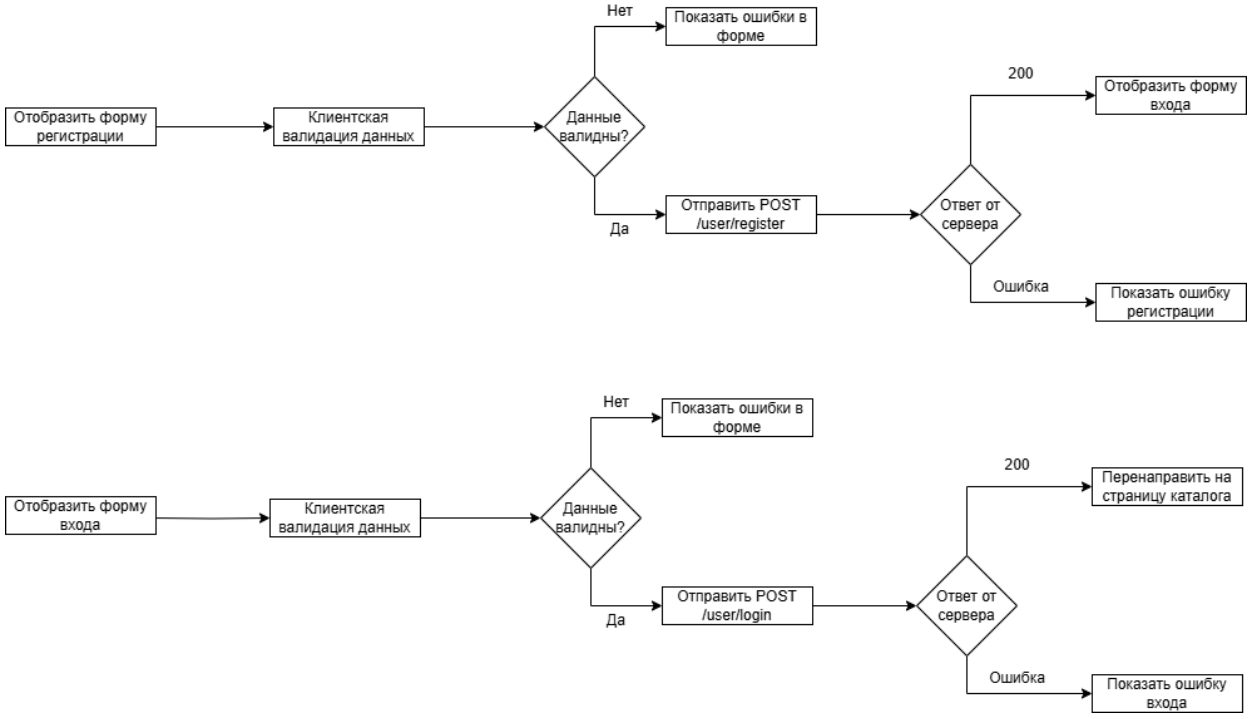


Рисунок 4 - Блок-схема регистрации и авторизации на стороне клиента.

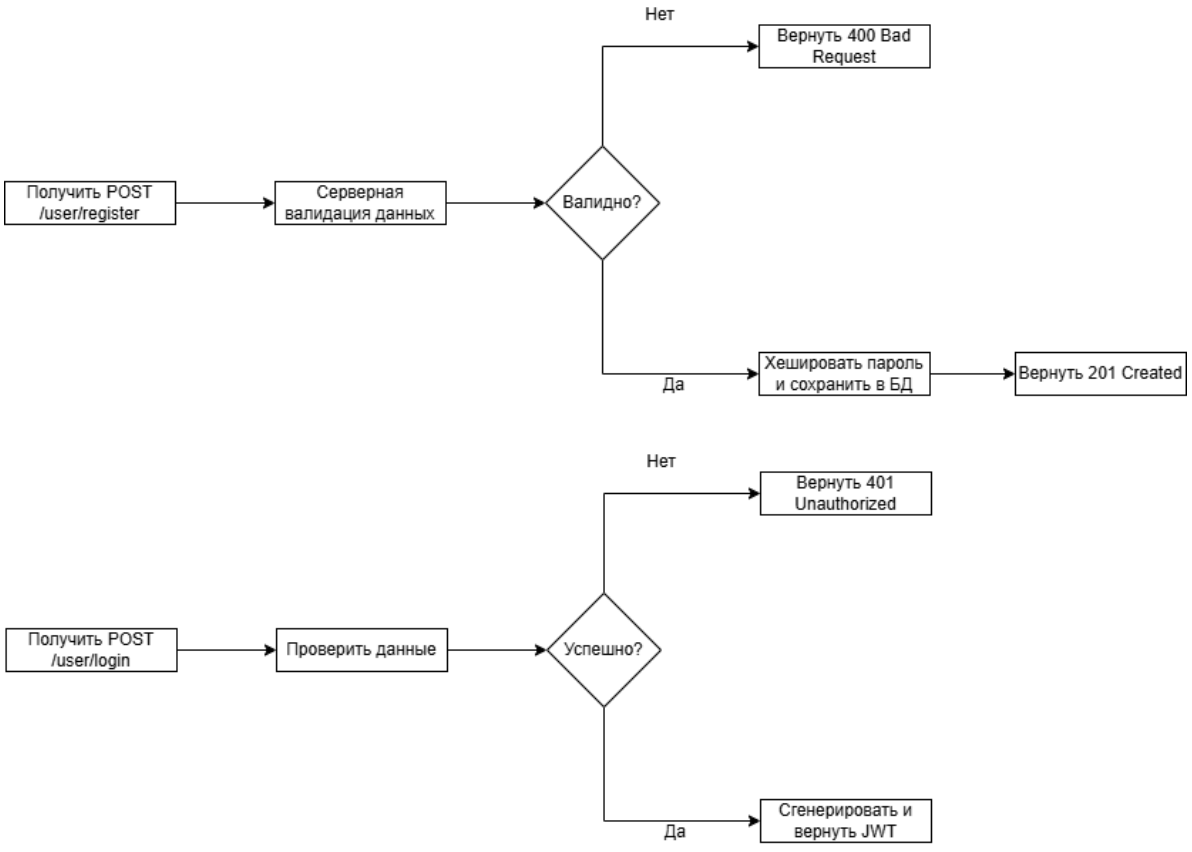


Рисунок 5 - Блок-схема регистрации и авторизации на стороне сервере.

В обоих случаях клиент отправляет POST запрос на нужные маршруты, после чего они обрабатываются сервером и возвращается ответ на клиент.

Нефункциональные требования:

– адаптивность – поддержка мобильных устройств и планшетов;

– надежность – устойчивость к сбоям;

– безопасность – защита от CSRF или и XSS.

Требования к пользовательскому интерфейсу:

Интерфейс должен быть интуитивно понятным и соответствовать современным принципам UX-дизайна:

– простая навигация по разделам;

– единый стиль оформления;

– отображение ключевой информации в виде карточек техники;

– валидация при заполнении форм.

Для реализации интерфейса выбран React с библиотекой Material UI, что обеспечивает гибкость и соответствие требованиям доступности. Серверная часть предусматривает использование NestJS и NoSQL базы данных, обеспечивающих модульность, масштабируемость и высокую производительность.

Вывод по главе 1

Результаты главы 1 создают прочную методологическую и исследовательскую основу для последующей проектной и реализационной части работы.

Глава 2 Процесс разработки программного обеспечения

2.1 Проектирование программного обеспечения

Проектирование программного обеспечения является ключевым этапом разработки информационной системы. От качества архитектурных и структурных решений на этом этапе зависит устойчивость, масштабируемость и удобство сопровождения всей системы. На данном этапе определяются общая архитектура приложения, структура хранения данных, принципы взаимодействия между модулями, а также технология реализации.

Архитектурный подход. Для веб-приложения, которое ориентировано на размещение объявлений о аренде и продаже спецтехники юридическими лицами, была выбрана клиент-серверная архитектура, включающая два основных компонента:

- клиентский (Frontend) – реализуется на основе React с использованием Material UI. Отвечает за отображение пользовательского интерфейса, валидацию на клиенте и взаимодействие с сервером через HTTP-запросы [1];
- серверный (Backend) – построена на NestJS (фреймворк для Node.js). Принимает запросы от клиентов, обрабатывает бизнес-логику, обеспечивает аутентификацию и авторизацию, а также взаимодействует с системой хранения данных.

Компонент хранения данных представлен нереляционной, документно-ориентированной базой данных, обеспечивающей гибкое и масштабируемое хранение информации.

Отделение клиентской и серверной частей позволяет независимо развивать и масштабировать интерфейс и логику приложения, а также упрощает развертывание и сопровождение системы.

Выбор архитектурного шаблона. Приложение реализует классический клиент-серверный шаблон, где:

– клиент отвечает за отображение и сбор пользовательских данных, формирует HTTP запросы к серверу обрабатывает его ответы;
– сервер выполняет бизнес-логику, маршрутизирует запросы, выполняет валидацию данных, работает с базой данных и возвращает результаты на клиент.

Взаимодействие компонентов происходит по стандартам REST API, что упрощает интеграцию, масштабирование и тестирование.

Информационные потоки. Приложение предусматривает следующий типовой цикл обработки пользовательских запросов:

– пользователь отправляет запрос на просмотр или размещение объявления;
– клиентская часть формирует запрос и отправляет его через REST API на сервер;
– серверный контроллер валидирует данные и передает их в сервисный слой [4], [6];
– бизнес-логика обрабатывает запрос, взаимодействует с базой данных;
– формируется ответ и отправляется обратно на клиентскую часть;
– интерфейс отображает результат пользователю.

Обоснование выбора технологий. Выбор технологий обусловлен следующими факторами:

– React и Material UI – высокая скорость разработки и адаптивный дизайн;
– NestJS – строгая структура, поддержка TypeScript, масштабируемость;
– MongoDB – гибкое хранение структурированных и слабо структурированных данных;
– REST API – стандартизированный способ обмена данных между компонентами.

На рисунке 6 представлена диаграмма компонентов, которая показывает, как компоненты приложения взаимодействуют между собой.

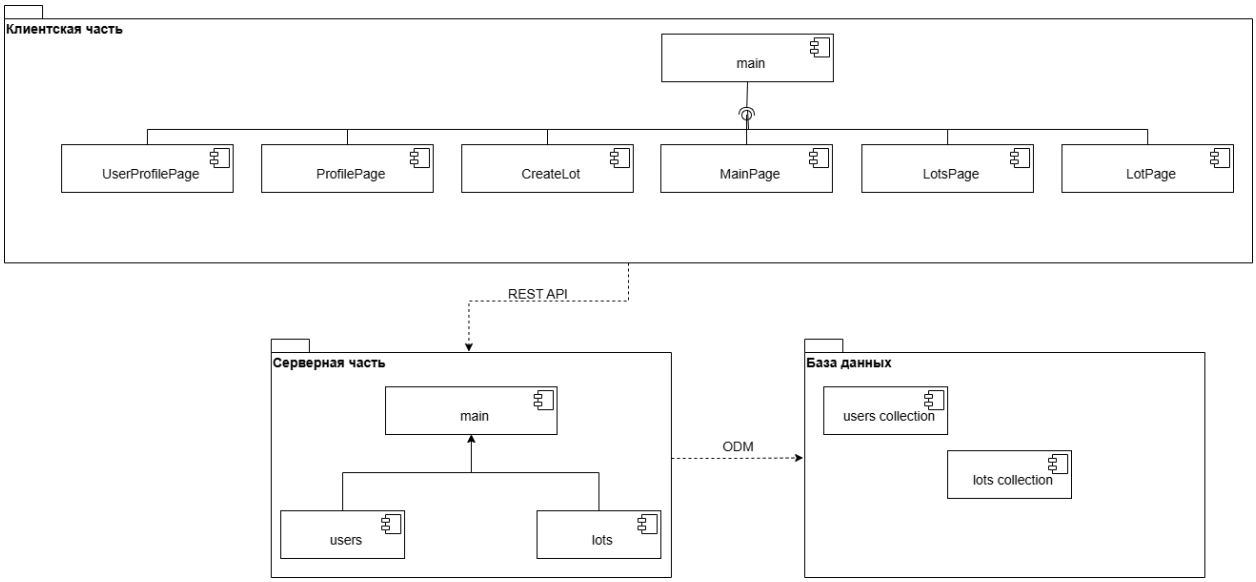


Рисунок 6 - Диаграмма компонентов приложения.

Клиентская часть взаимодействует с серверной частью посредством REST API, а с базой данных взаимодействует через ODM.

Проектирование интерфейса пользователя. При проектировании пользовательского интерфейса (UI) были учтены принципы современной веб-разработки и требования к UX:

- основные страницы – Главная, Каталог, Профиль;
- меню навигации закреплено в шапке сайта и адаптируется под разные экраны;
- использование Material UI;
- карточки техники содержат фото, основные характеристики, кнопку быстрого действия;
- валидация полей форм.

Таким образом, проектирование UI предусматривает удобство взаимодействия пользователя с системой, согласованность визуальных элементов и гибкость адаптации под различные устройства.

Проектирование базы данных включает определение структур хранения информации и взаимосвязей между ключевыми сущностями. В качестве хранилища используется нереляционная база данных, где данные организованы в коллекции. Свойства документов коллекции users можно увидеть в таблице 3, коллекции lots – в таблице 4.

Таблица 3. Свойства документов коллекции users

Свойство	**Тип данных**
id	string
email	string
name	string
businessType	string
login	string
password	string
favoriteLots	Array<string>

Таблица 4. Свойства документов коллекции lots

Свойство	**Тип данных**
id	string
title	string
owner	string
images	Array<string>
description	string
price	number
createdTimestamp	number
isArchived	boolean
tags	Array<string>

Продолжение таблицы 4.

Свойство	**Тип данных**
views	Array<{ id: string }>
vehicleType	string
productionYear	number
vinNumber	string
brand	string
model	string
hp	number
engineVolume	number
availability	string
condition	string
milage	number
pts	string
color	string

С данным набором свойств визуальное наполнение клиентской части будет максимально полным, полезным и удобным.

Чтобы лучше понять, как эти модели могут быть связаны между собой, обратимся к рисунку 7, на котором показана схема представления данных в формате NoSQL. На нем видно, что документы из коллекции lots можно связать с документами из коллекции users по полям “owner” и “id” соответственно. Это будет полезно как при просмотре собственного профиля, так и профилей других пользователей, для упрощения поиска нужных объявлений.

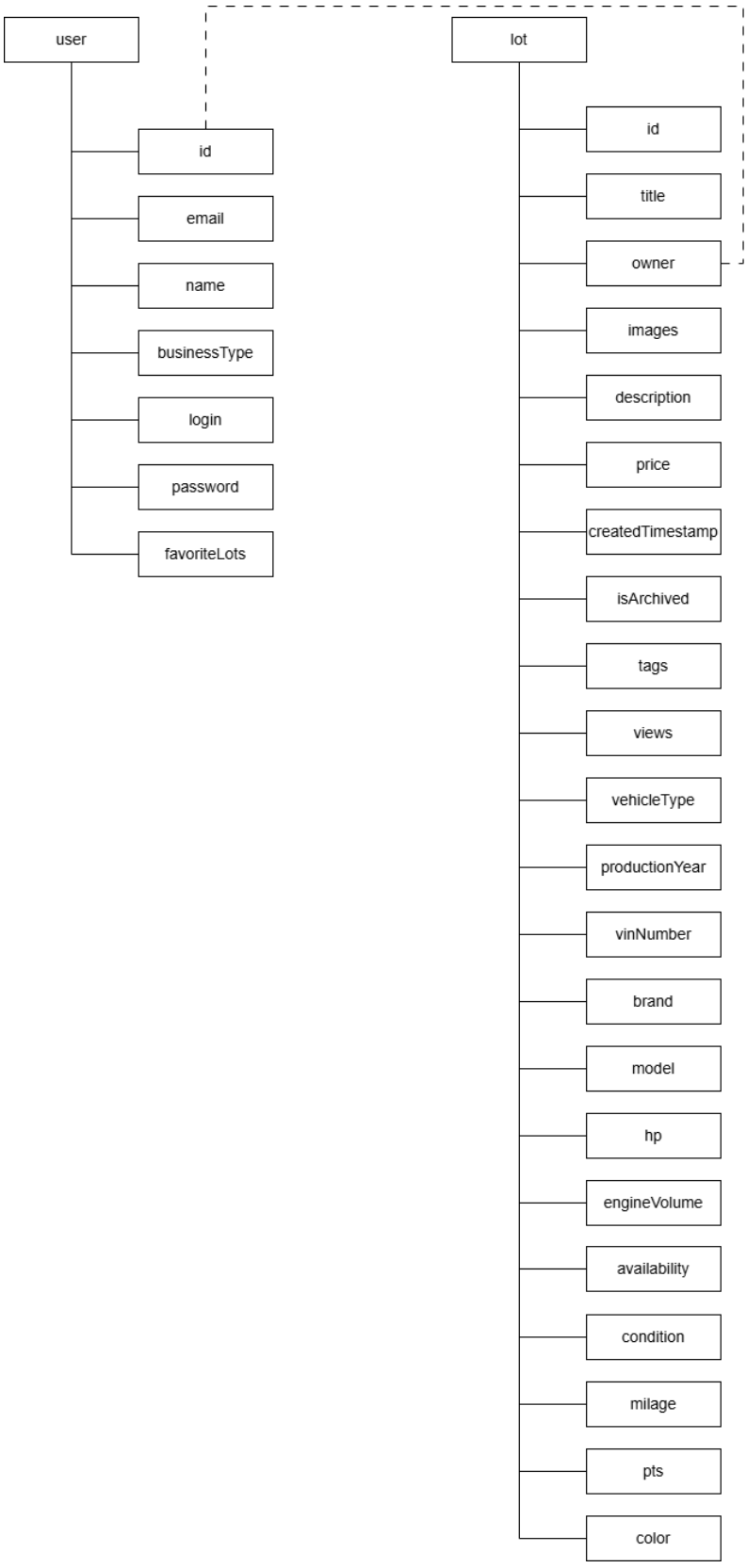


Рисунок 7 - Схема представления данных в формате NoSQL.

Такое проектирование обеспечивает быстродействие операций чтения/записи и гибкость в хранении разнородных данных, характерных для объявлений спецтехники.

2.2 Выбор технологий и инструментов для разработки

Выбор технологий и инструментов разработки играет ключевую роль в обеспечении эффективности, надёжности и масштабируемости создаваемого веб-приложения. При выборе учитывались особенности предметной области, предполагаемая нагрузка, удобство поддержки, а также опыт команды разработки.

Критерии выбора технологий. Основными критериями выбора стали:

- сообщество и документация;
- совместимость с REST архитектурой;
- поддержка модульности и масштабирования.

Выбранные технологии. В таблице 5 приведены выбранные для каждого уровня технологий и их краткое описание.

Таблица 5. Выбранные технологии

Уровень системы	**Технология/инструмент**	**Назначение**
Клиент (Frontend)	React	Построение пользовательского интерфейса [9], [15], [16]
	Material UI (MUI)	Готовые адаптивные компоненты интерфейса
	React Router	Маршрутизация между страницами [7]
	Axios	Отправка HTTP запросов к серверу

Продолжение таблицы 5.

Уровень системы	**Технология/инструмент**	**Назначение**
	Yup [19]	Валидация данных на клиенте
Сервер (Backend)	Node.js + NestJS [10], [18]	Реализация серверной логики и REST API
	JWT [3]	Аутентификация и авторизация пользователей
	class-validator [20]	Валидация данных на сервере
База данных	MongoDB	Документно-ориентированное хранилище данных
	Mongoose [12]	ODM-библиотека для взаимодействия с MongoDB
Прочие инструменты	Git и GitHub	Контроль версий
	Bruno	Тестирование API запросов
	Visual Studio Code	Интегрированная среда разработки с поддержкой расширений для JS/TS и отладки

Обоснование выбора для клиента (Frontend) - React является современным фреймворком с виртуальным DOM, высокой производительностью и поддержкой компонентного подхода. Он отлично подходит для построения динамичных и отзывчивых интерфейсов, а Material UI позволяет ускорить верстку за счёт использования готовых компонентов. Дополнительно в клиентской части для работы с данными и состоянием используется TanStack Query. Эта библиотека упрощает загрузку, кэширование и синхронизацию данных с сервером, обеспечивая автоматическое обновление данных, управление состоянием загрузки и обработку ошибок.

Сервер (Backend) - NestJS обеспечивает строгую архитектуру, поддержку модульности и TypeScript [6], [10], [18]. Это даёт возможность разделить логику приложения на изолированные модули, упрощая сопровождение и расширение.

База данных - MongoDB предоставляет гибкое хранение документов в формате JSON и хорошо масштабируется. Mongoose обеспечивает типизацию и удобную работу с коллекциями данных [12].

Инструменты разработки - Git и GitHub позволяют эффективно отслеживать изменения в проекте и работать в команде. Bruno применяется для проверки корректности запросов на всех этапах разработки [5].

Основной IDE для разработки является Visual Studio Code — лёгкая, расширяемая среда с поддержкой TypeScript, ESLint и Prettier, что обеспечивает удобное написание кода и интеграцию с системами контроля версий.

2.3 Реализация функциональных требований

Реализация ключевых функциональных требований основана на выбранном стеке технологий (React, Material UI, TanStack Query для клиента; NestJS, Mongoose и MongoDB для сервера) [12]-[17]. Далее описаны подходы к реализации каждого требования.

Управление объявлениями

Для создания объявлений на сервере написаны следующие обработчики запросов “/lot/beforeCreate” и “/lot/create”, как видно на рисунке 8. Именно они отвечают за выполнение бизнес-логики создания объявления, обрабатывая соответствующие запросы с клиентской части веб-приложения.

```
@Post("/beforeCreate")
    @RequiresCookie()
    @UseInterceptors(FilesInterceptor('photos', 15, { dest: './files' }))
    create(@UploadedFiles() files: Express.Multer.File[]): Promise<string[]> {
        return this.lotService.beforeCreate(files);
    };

    @Post("/create")
    @RequiresCookie()
    createLot(@Body() createLot: CreateLotDto, @Req() req: Request): Promise<Partial<ILot>> {
        return this.lotService.create(createLot, req);
    };
```

Рисунок 8 - Обработчики запросов “/beforeCreate” и “create” в контроллере.

Обработчик POST запроса “/beforeCreate” отвечает за загрузку изображений на сервер и возвращает их в виде массива ссылок, для использования в запросе “/create”. Сам метод beforeCreate класса lotsService можно просмотреть на рисунке 9.

```
async beforeCreate(files: Express.Multer.File[]): Promise<Array<string>> {

        if (files.length === 0) return [];

        cloudinary.config({
            cloud_name: process.env.API_CLOUD,
            api_key: process.env.API_KEY,
            api_secret: process.env.API_SECRET
        });

        const uploadPromises = Array.from(files).map((photo) => {
            return cloudinary.uploader.upload(photo.path);
        });

        const uploadedImages = await Promise.all(uploadPromises);

        const folderFiles = await readdir("./files");
        folderFiles.forEach((file) => {
            unlink("./files/" + file);
        });

        return uploadedImages.map((image) => image.secure_url);
    };
```

Рисунок 9 - Реализация метода beforeCreate.

На вход метод принимает массив файлов, с которым работает далее. Создается массив файлов на загрузку и загружается, используя Promise.all. После загрузки изображений сами файлы удаляются, а в ответ возвращается массив с ссылками на изображения.

Метод create можно просмотреть на рисунке 10.

Рисунок 10 - Реализация метода create.

Данный метод принимает на вход объект с данными о новом объявлении, включая изображения из метода beforeCreate. Вторым параметром является запрос, отправленный клиентом. Это необходимо для извлечения JWT пользователя [3], чтобы узнать id человека, который создает объявление.

В объект нового объявления передаются id создателя, время создания в формате UNIX Timestamp (количество секунд с 1 января 1970 года) и данные о объявлении, после чего на клиент возвращается объект нового объявления в формате JSON.

В базе данных, в коллекции lots, при этом, создается документ, со свойствами указанными в таблице 4.

Внешний вид формы можно увидеть на рисунке 11.

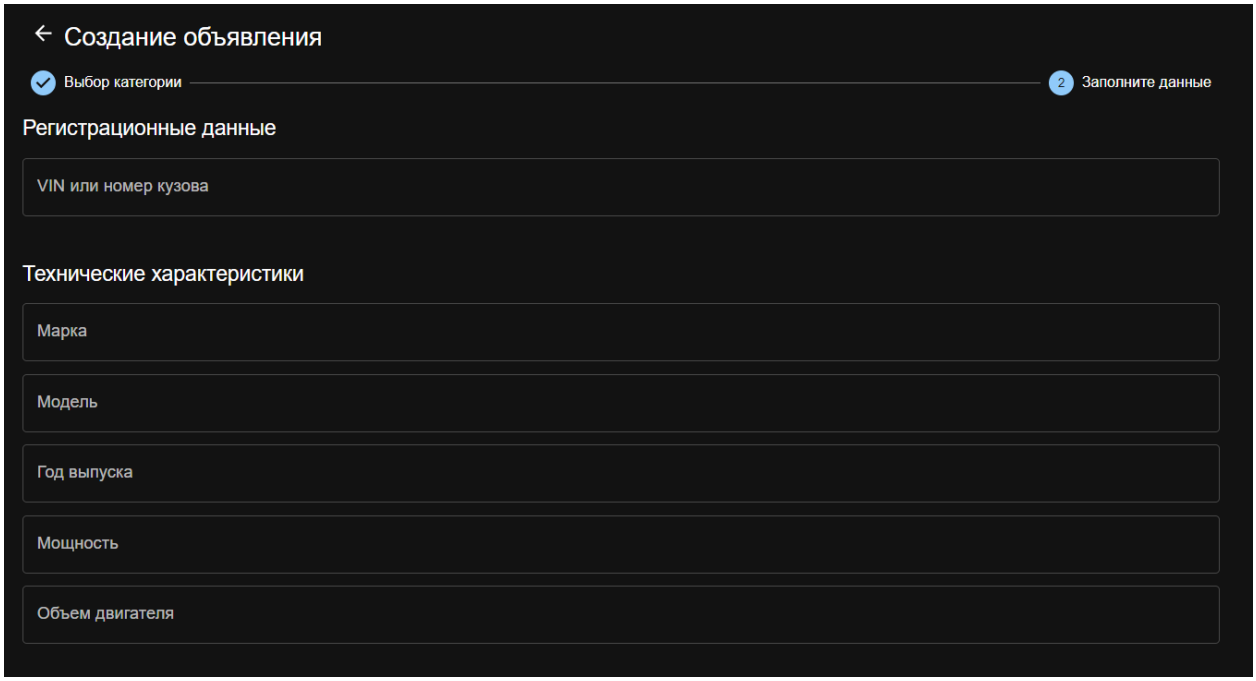
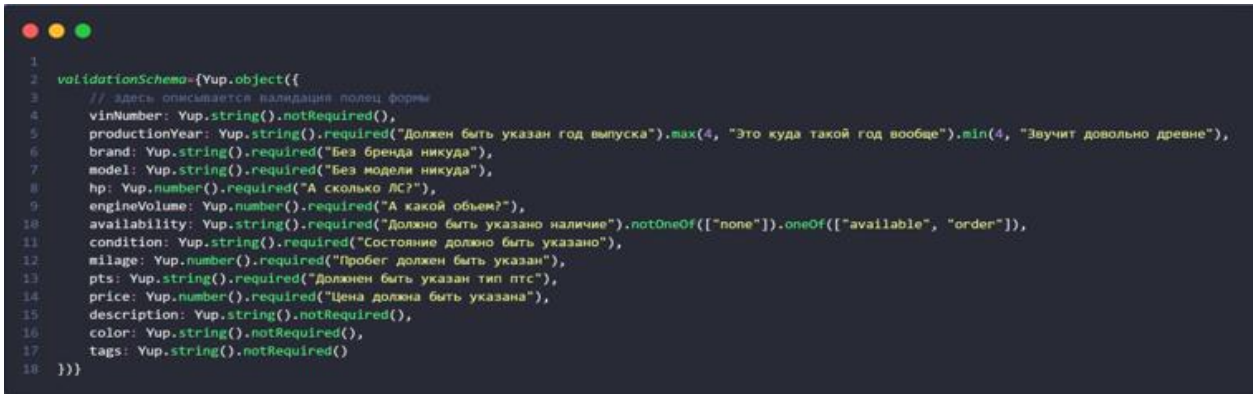


Рисунок 11 - Форма создания объявления.

Отправить ее пустой у пользователя не получится, так как присутствует валидация как на стороне клиента, так и на стороне сервера. На стороне клиента валидация данных представляет собой объект, ключами которого являются названия полей формы, а значениями являются валидаторы разного рода, как показано на рисунке 12.



```
validationSchema={Yup.object({
    // здесь описывается валидация полец формы
    vinNumber: Yup.string().notRequired(),
    productionYear: Yup.string().required("Должен быть указан год выпуска").max(4, "Это куда такой год вообще").min(4, "Звучит довольно древне"),
    brand: Yup.string().required("Без бренда никуда"),
    model: Yup.string().required("Без модели никуда"),
    hp: Yup.number().required("А сколько ЛС?"),
    engineVolume: Yup.number().required("А какой объем?"),
    availability: Yup.string().required("Должно быть указано наличие").notOneOf(["none"]).oneOf(["available", "order"]),
    condition: Yup.string().required("Состояние должно быть указано"),
    milage: Yup.number().required("Пробег должен быть указан"),
    pts: Yup.string().required("Должнен быть указан тип птс"),
    price: Yup.number().required("Цена должна быть указана"),
    description: Yup.string().notRequired(),
    color: Yup.string().notRequired(),
    tags: Yup.string().notRequired()
})}
```

Рисунок 12 - Валидация формы создания объявления.

Указывается тип значения поля, обязательно ли оно к заполнению и так далее. При ошибке валидации поля пользователю будет показано сообщение об ошибке и поле подсветится красным цветом, как на рисунке 13.

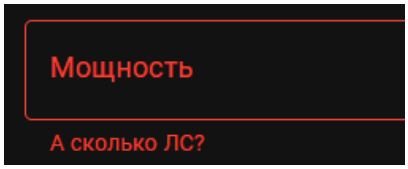


Рисунок 13 - Ошибка валидации поля.

Если валидация полей была успешно выполнена – данные отправляются на сервер.

```
// обработка события отправки формы
console.log(JSON.stringify(values, null, 2));
const photosForm = new FormData();
Array.from(values.photos).forEach((photo) => photosForm.append('photos', photo));
formRef.current?.setSubmitting(true);
// первым делом загружаются изображения
request.postForm("/lot/beforeCreate", photosForm, { withCredentials: true }).then((res) => {
    const images = res.data;
    // после этого отправляется запрос на создание объявления
    request.post("/lot/create", { ...values, vehicleType: selectedCategory, images, tags: values.tags.split(" ") }, { withCredentials: true }).then((res) => {
        console.log(res.data);
        formRef.current?.setSubmitting(false);
        navigate(`/lots/${res.data.id}`);
    });
}).catch((e) => console.log(e));
```

Рисунок 14 - Отправка данных нового объявления на сервер.

Как видно на рисунке 14, сначала на сервер отправляется запрос на загрузку изображений (/lot/beforeCreate), после чего отправляется запрос на создание объявления (/lot/create). В случае если объявление создано успешно – пользователя перенаправит на страницу этого объявления.

Для редактирования объявлений написан обработчик запроса PATCH “/edit/:lotId”, где lotId – id объявления, код представлен на рисунке 15.

```
@Patch("/edit/:lotId")
    @RequiresCookie()
    editLot(@Body() newData: EditLotDto, @Param('lotId') LotId:
string, @Req() req: Request): Promise<{ lotUpdated: boolean }>
{
        return this.lotService.editLot(newData, LotId, req);
    };
```

Рисунок 15 - Обработчик запроса “/edit/:lotId”.

Сам метод editLot принимает как параметры измененные данные объявления, id объявления, которое нужно отредактировать и запрос.

```
async editLot(newData: EditLotDto, LotId: string, req: Request): Promise<{ lotUpdated:
boolean }> {
        const tokenCookie = req.signedCookies["VehToken"];
        const payload = this.jwtService.verify(tokenCookie, { secret: process.env.JWT_
SECRET || "local_development" });

        const updatedLot = await Lot.findOneAndUpdate({ id: LotId, owner: payload.id
}, { price: newData.price, description: newData.description, tags: newData.tags }, { n
ew: true });

        if (updatedLot) {
            this.logger.log(`Successfully updated lot id:${updatedLot.id}`);
            return { lotUpdated: true };
        } else {
            throw new HttpException(`Couldn't find lot with id ${LotId}`, 404);
        };
    };
```

Рисунок 16 - Реализация метода editLot.

На рисунке 16 видно, что реализация довольно проста – в базе данных находится объявление с указанным id и владельцем и обновляется новыми данными из запроса. Если объявление было изменено – пользователь получает ответ “{ lotUpdated: true }”, а если такое объявление найдено не было – ошибку со статусом 404 и соответствующим сообщением.

Внешний вид диалога на клиенте показан на рисунке 17.

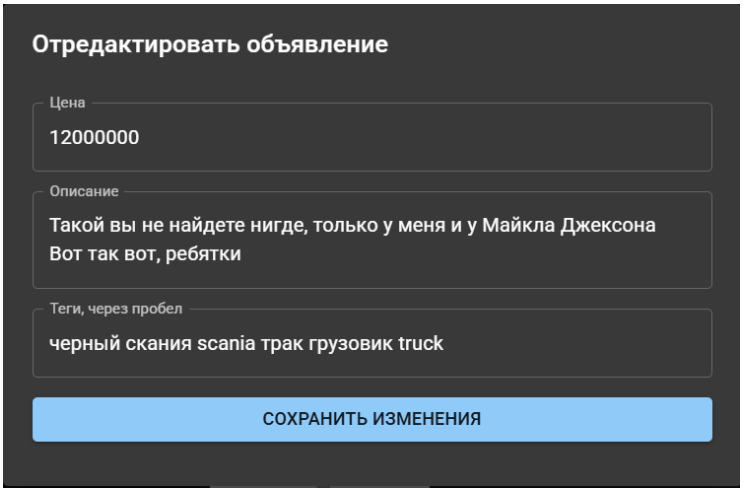


Рисунок 17 - Диалог редактирования объявления.

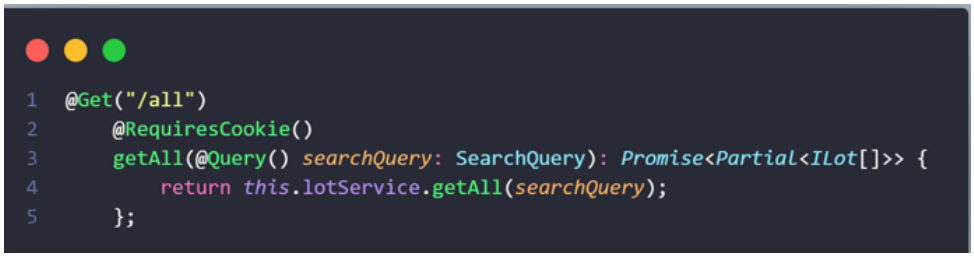
При нажатии на “Сохранить изменения” выполняется код с рисунка 18.

```
request.patch(`/lot/edit/${lotId}`, {
    ...values,
    tags: values.tags.split(" ")
}, { withCredentials: true }).then((res) => {
    if (res.data.lotUpdated) {
        queryClient.refetchQueries({
            queryKey: ["userLots"]
        });
        set((prev) => !prev);
    };
});
```

Рисунок 18 - Обработка нажатия на кнопку сохранения.

На сервер отправляется PATCH запрос “/lot/edit/id” с измененными значениями. Если объявление было изменено – обновляется список пользовательских объявлений и закрывается диалог изменения.

За просмотр объявлений на сервере отвечает обработчик GET запроса “/lot/all”, код представлен на рисунке 19.



```
@Get("/all")
    @RequiresCookie()
    getAll(@Query() searchQuery: SearchQuery): Promise<Partial<ILot[]>> {
        return this.lotService.getAll(searchQuery);
    };
```

Рисунок 19 - Обработчик запроса “/lot/all”.

За фильтрацию, пагинацию и другое отвечает метод getAll класса lotService. В нем выполняется агрегация, в которой применяются фильтры по цене, тексту с поля поиска, выбираются только нужные для отображения на клиенте поля. После выполнения агрегации на клиент отправляется массив из 10 объявлений, чтобы избежать как большой нагрузки на сервер, так и использовать меньше трафика для передачи данных.

Получение данных с помощью библиотеки Tanstack Query на клиенте показано на рисунке 20.

```

const infiniteLotsQuery = useInfiniteQuery({
    queryKey: ["iLots", filters],
    initialPageParam: 0,
    queryFn: async ({ pageParam }: { pageParam: number }) => {
        const { data } = await request.get(`/lot/all${getFiltersQuery(filters)}&page=${pageParam}`, { withCredentials: true });
        console.log(data);
        return data;
    },
    getNextPageParam: (lastPage, pages) => lastPage.length < 10 ? undefined : pages.length
});
```

Рисунок 20 - Получение данных на клиенте.

Отправляется запрос на сервер со всеми фильтрами и указанием страницы для пагинации, после чего объявления можно показывать пользователю, код представлен на рисунке 21.

```
infiniteLotsQuery.data?.pages.flat().map((lot: Lot, idx: number) => (
    lot ? <Lot key={lot.id} lot={lot} maxIdx={infiniteLotsQuery.data?.pages.flat().length - 1} viewRef={ref} idx={idx} /> : null
))
```

Рисунок 21 - Отображение объявлений пользователю.

Выполняется цикл по списку объявлений с сервера, в котором пользователю отображается компонент Lot, в который как параметры передаются объект самого объявления, его индекс в списке и некоторые другие параметры.

Внешний вид страницы с объявлениями представлен на рисунке 22, на рисунке 23 представлен внешний вид страницы отдельного объявления.

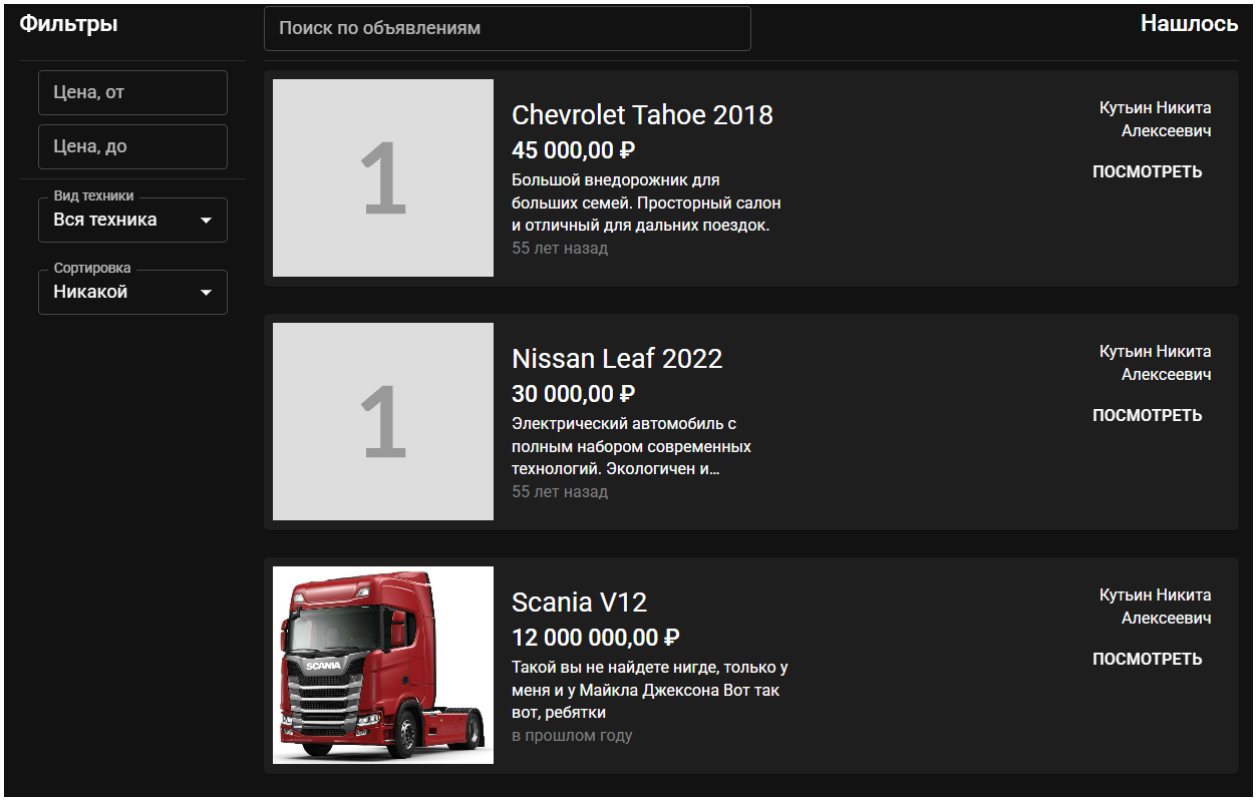


Рисунок 22 - Внешний вид страницы объявлений.

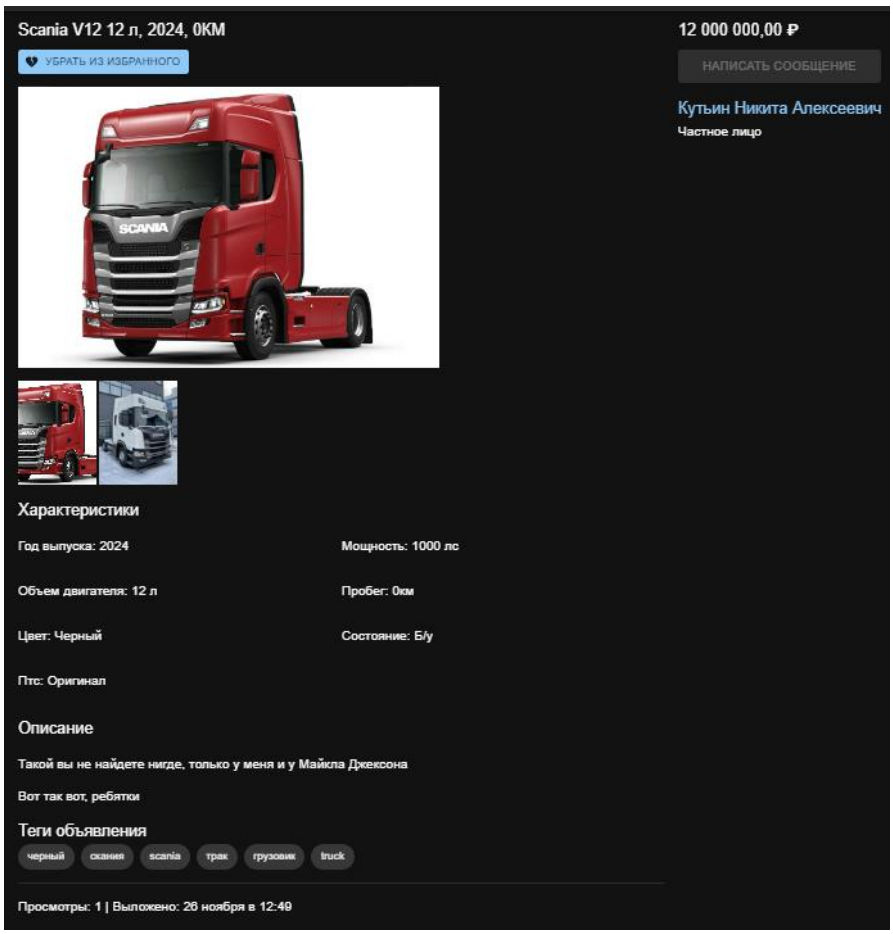


Рисунок 23 - Страница отдельного объявления.

Работа с пользователями

Для регистрации пользователей был написан обработчик запросов POST “/user/register”, код представлен на рисунке 24.

```
1
2  @Post("/register")
3  register(@Body() createUser: CreateUserDto): Promise<Partial<IUser>> {
4      return this.userService.register(createUser);
5  };
```

Рисунок 24 - Обработчик запроса на регистрацию пользователя.

Также был написан метод register для класса userService, код представлен на рисунке 25.

```
async register(createUser: CreateUserDto): Promise<Partial<IUser>> {
    const { login } = createUser;

    const userExistsCheck = await User.findOne({ login });

    if (!userExistsCheck) {
        const user = new User({
            ...createUser
        });

        return user.save().then((saved) => {
            this.logger.log(`Saved user ${saved.id} in the database`);
            return { email: saved.email, login: saved.login, id: saved.id, name: saved.name };
        });
    } else {
        throw new HttpException("Something went wrong creating user", 400);
    };
};
```

Рисунок 25 - Метод register.

В этом методе проверяется существует ли уже такой пользователь или нет. Если нет – пользователь сохраняется в базе данных. Если такой пользователь уже есть - в ответ на клиент отправляется ошибка со статусом 400 и соответствующим сообщением.

Форма регистрации на клиентской части состоит из следующих полей – выбор типа бизнеса (ИП или ООО), ФИО пользователя, электронная почта, логин, пароль и кнопки регистрации, после нажатия на которую отправляется запрос на сервер. С ее внешним видом можно ознакомиться на рисунке 26.

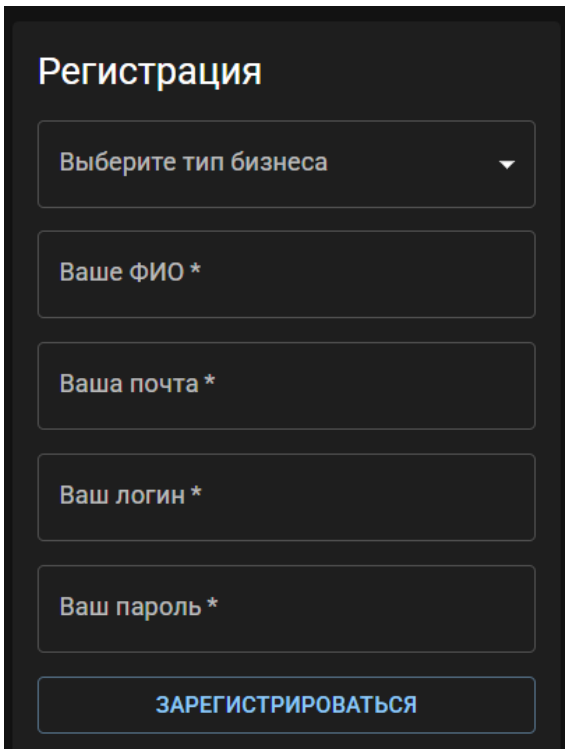


Рисунок 26 - Внешний вид формы регистрации.

При авторизации пользователь вводит в форму свой логин и пароль. При нажатии на кнопку авторизации на сервере проверяется есть ли такой пользователь, проверяются пароли на соответствие введенного с тем, что записан в базе данных и в ответ сервером отправляется Cookie, содержащее в себе JWT и сообщение о том, что пользователь успешно авторизован. При успешной авторизации пользователя перенаправит на страницу с объявлениями.

Для просмотра профиля нужно:

– получить данные пользователя для отображения;

– получить объявления для отображения.

Чтобы получить данные пользователя используется маршрут GET “/user/check”, его код представлен на рисунке 27.

```
async check(req: Request): Promise<IUser> {
    const tokenCookie = req.signedCookies["VehToken"];
    const payload = this.jwtService.verify(tokenCookie, { secret: process.env.JWT_SECRET || "local_development" });

    return User.findOne({ id: payload.id }, { _id: 0, password: 0 });
};
```

Рисунок 27 - Метод check класса userService.

В этом методе ищется пользователь с id, которое было в JWT запроса, после на клиент возвращаются все поля, кроме _id и password.

Для получения объявлений пользователя используется маршрут “/lot/user/:userId”, код представлен на рисунке 28.

```
async getForUser(userId: string): Promise<Partial<ILot[]>> {
    const foundLots = await Lot.aggregate([
        {
            $match: {
                owner: userId
            }
        },
        {
            $project: {
                id: 1,
                title: 1,
                images: 1,
                description: 1,
                price: 1,
                createdTimestamp: 1,
                isArchived: 1,
                tags: 1
            }
        },
        {
            $sort: {
                createdTimestamp: -1
            }
        }
    ]);

    this.logger.log(`Found ${foundLots.length} lots with owner:${userId}`);
    return foundLots;
};
```

Рисунок 28 - Метод getForUser класса lotService.

В этом методе находятся все объявления пользователя, профиль которого просматривается, выбираются только нужные для отображения поля,

происходит сортировка по времени, когда выложили объявление и после этого данные отправляются на клиент для их дальнейшего отображения.

Код отображения этих данных на клиенте представлен на рисунке 29.

```
{userLotsQuery.data &&
    <TabContext value={activeTab}>
        <Box sx={{ borderBottom: 1, borderColor: 'divider' }}>
            <TabList onChange={(_, val) => { setTab(val); }}>
                <Tab label={<p>Активные<sup style={{ margin: '5px' }}>{userLotsQuery.data.filter((lot: Lot) => !lot.isArchived).length}</sup></p>} value="active" />
                <Tab label={<p>В архиве<sup style={{ margin: '5px' }}>{userLotsQuery.data.filter((lot: Lot) => lot.isArchived).length}</sup></p>} value="archived" />
            </TabList>
        </Box>
        <TabPanel value="active">
            {
                userLotsQuery.data.filter((lot: Lot) => !lot.isArchived).map((lot: Lot) => (
                    <UserProfileLot key={lot.id} lot={lot} />
                ))
            }
        </TabPanel>
        <TabPanel value="archived">
            {
                userLotsQuery.data.filter((lot: Lot) => lot.isArchived).map((lot: Lot) => (
                    <UserProfileLot key={lot.id} lot={lot} />
                ))
            }
        </TabPanel>
    </TabContext>}
```

Рисунок 29 - Отображение списка объявлений на странице профиля на клиенте.

В двух циклах отображаются объявления с проверкой на то, находятся ли они в архиве или нет.

Выводы по главе 2

Во второй главе были спроектированы и реализованы основные компоненты веб-приложения, предназначенного для размещения объявлений о продаже и аренде спецтехники юридическими лицами.

Глава 3 Оценка эффективности разработанного программного обеспечения

3.1 Тестирование и отладка приложения

После завершения этапов проектирования и реализации важной задачей является проведение тестирования и отладки веб-приложения. Это позволяет выявить и устранить ошибки, проверить соответствие системы функциональным требованиям и убедиться в стабильной работе всех компонентов.

Цели тестирования

Целями тестирования являются:

- проверка корректности реализации бизнес-логики;
- проверка взаимодействия клиентской и серверной частей;
- выявление и устранение критических и логических ошибок;
- оценка стабильности системы при различных сценариях использования;

В таблице 6 предоставлен плана тестирования.

Таблица 6. Тест-план

Цель тестирования	**Метод тестирования**	**Ответственный**
Проверка корректности бизнес-логики	Unit-тестирование	QA Engineer
Проверка взаимодействия клиент/сервер	Интеграционное тестирование	Integration tester
Оценка производительности и времени отклика	Нагрузочное тестирование	Performance Engineer
Проверка обработки пользовательских данных	Функциональное тестирование	Functional tester

Далее перейдем к проведению тестов.

Unit-тестирование

Unit-тестирование проводилось для серверных контроллеров и отдельных React компонентов [2], [8].

При тестировании клиента были написаны 7 тестов в общей сложности, все они успешно выполняются, как видно на рисунке 30.

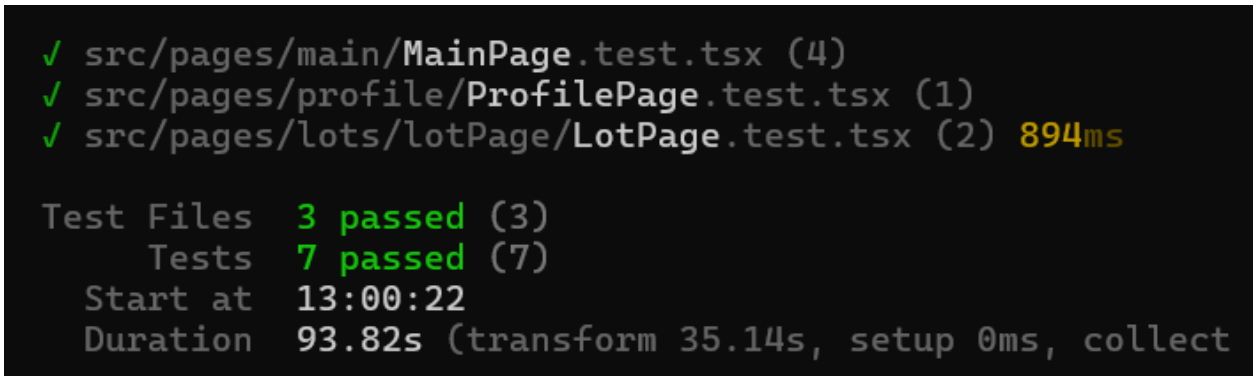


Рисунок 30 - Выполненные тесты на клиенте.

При тестировании сервера было написано 14 тестов в общей сложности, все они успешно выполняются, как видно на рисунке 31.

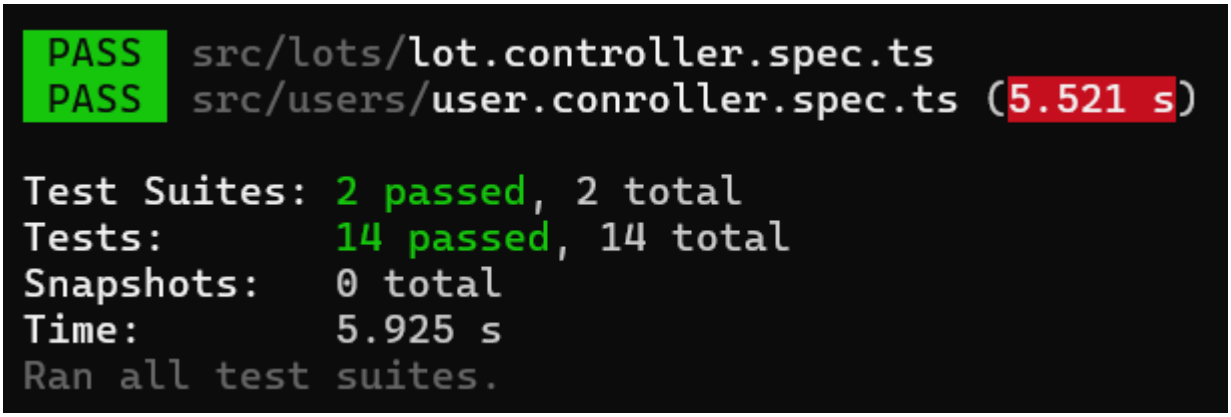


Рисунок 31 - Выполненные тесты на сервере.

Так как проблем на этапе unit-тестирования выявлено не было – перейдем к функциональному тестированию.

Функциональное тестирование

При функциональном тестировании вручную проверялись сценарии из таблицы 7.

Таблица 7. Сценарии функционального тестирования.

ID	**Тест-кейс**	**Ожидаемый результат**	**Статус**
1	Добавление нового объявления	Объявление создано, отображается в списке	Пройден
2	Редактирование существующего объявления	Изменения сохранены, обновленная информация видна	Пройден
3	Загрузка фотографий к объявлению	Фотографии загружены и отображаются в объявлении	Пройден
4	Указание стоимости и т.д при создании объявления	Указанные параметры корректно сохраняются	Пройден
5	Просмотр списка объявлений	Список объявлений загружается и отображается	Пройден
6	Регистрация через электронную почту	Новый пользователь создается	Пройден
7	Авторизация пользователя	Успешный вход по корректным учетным данным	Пройден
8	Просмотр профиля пользователя	Информация о пользователе отображается корректно	Пройден

Каждый из этих сценариев проверялся на наличие ошибок и соответствие требованиям.

Интеграционное тестирование

Были протестированы взаимодействия между модулями — например, между модулем пользователей и модулем объявлений, проверка корректного сохранения и отображения информации через API [5]. Результаты отображены в таблице 8. По результатам интеграционного тестирования никаких проблем также замечено не было.

Таблица 8. Результаты интеграционного тестирования.

ID	Тест-кейс	Компоненты	Ожидаемый результат	Статус
1	Создание объявления через клиент и запись в БД	React UI Lots API MongoDB	Новое объявление сохраняется в БД и клиент перенаправляется на его страницу	Пройден
2	Редактирование объявления и запись новых данных в БД	React UI Lots API MongoDB	Обновленные данные сохраняются и загружаются при новом запросе	Пройден
3	Загрузка фотографий и их доступность в карточке объявления	React UI Images API	Фотографии сохраняются, ссылки сохраняются в БД и отображаются в карточке	Пройден
4	Просмотр списка объявлений	React UI Lots API	Возвращаются только активные объявления	Пройден
5	Регистрация нового пользователя	React UI User API MongoDB	Новый пользователь создается в БД	Пройден
6	Аутентификация и получение токена	React UI User API	При корректных данных выдается JWT	Пройден
7	Попытка доступа к защищенным маршрутам без токена	React UI User API	Возврат статуса 401 Unauthorized	Пройден
8	Получение и отображение данных в профиле пользователя	React UI User API MongoDB	Данные профиля получены и отображены в интерфейсе	Пройден

Нагрузочное тестирование

Нагрузочное тестирование проводится с целью оценки устойчивости и производительности веб-приложения при одновременном обращении большого количества пользователей. Основная задача — определить, выдерживает ли система пиковые нагрузки без значительной деградации времени отклика и сбоев в работе.

Для проведения тестирования использовался инструмент loadtest [11], результаты представлены на рисунке 32.

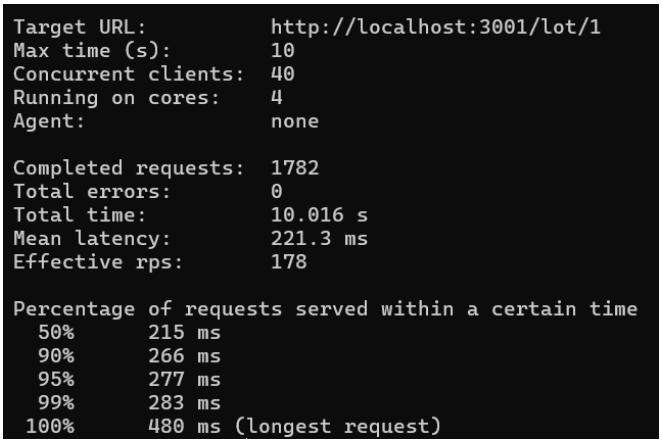


Рисунок 32 - Результаты нагрузочного тестирования.

Тестирование проводилось на запросе получения объявления. Как видно на рисунке 32 – максимальная задержка ответа не превысила 500мс, что можно считать приемлемым результатом.

Вывод по главе 3

В третьей главе было проведено тестирование разработанного веб-приложения. Проведены модульные, интеграционные, функциональные и нагрузочные тесты, что позволило убедиться в корректности реализации ключевых функций, устойчивости системы к ошибкам и высокой стабильности при типовых сценариях использования.

Результаты тестирования подтвердили, что приложение соответствует заявленным функциональным требованиям.

Заключение

Выпускная квалификационная работа была посвящена разработке веб-приложения для размещения объявлений о продаже и аренде спецтехники, ориентированного на юридические лица. В ходе выполнения проекта была достигнута поставленная цель — создание функциональной, масштабируемой и надёжной информационной системы, способной обеспечивать автоматизацию ключевых процессов в сфере аренды и реализации специализированной техники. Веб-приложение реализует все заявленные функциональные и нефункциональные требования.

В первой главе был проведён обзор предметной области и анализ существующих решений. Рассмотрены современные программные платформы (RentalWorks, MachineryTrader, HirePOS), определены их преимущества и ограничения, что позволило обосновать необходимость собственной разработки ПО. На основе анализа сформулированы функциональные и нефункциональные требования к разрабатываемому программному обеспечению, охватывающие управление объявлениями, работу с пользователями и интеграционные возможности.

Во второй главе выполнено проектирование программного обеспечения. Система реализована по классической клиент-серверной архитектуре с использованием актуального технологического стека: React и Material UI на клиентской стороне позволили обеспечить высокую отзывчивость и современный внешний вид, NestJS и MongoDB на серверной, в свою очередь, позволили обеспечить модульность бизнес-логики и надежное хранение данных. Разработан интерфейс пользователя, структура базы данных и реализована ключевая функциональность, включая регистрацию, авторизацию, размещение и редактирование объявлений. Работа велась с учетом современных подходов к архитектуре.

В третьей главе проведено комплексное тестирование приложения, включающее модульные, интеграционные и нагрузочные испытания.

Используемые при тестировании инструменты позволили подтвердить корректность бизнес-логики, устойчивость системы при работе с большим числом пользователей, а также соответствие функционала заявленным требованиям и общую надежность веб-приложения.

Разработанное приложение обладает высокой степенью адаптивности, простым и понятным интерфейсом, а также предоставляет широкие возможности для масштабирования и интеграции с внешними сервисами. Оно может быть использовано в качестве основы для внедрения корпоративных решений в сфере аренды и продажи техники, а также расширено дополнительными модулями, такими как документооборот, платёжные шлюзы, системы аналитики.

Таким образом, поставленные цели и задачи выпускной квалификационной работы были достигнуты, была подтверждена практическая значимость создания специализированных информационных систем на основе современных веб-технологий для цифровой трансформации бизнес-процессов в сфере аренды и продажи спецтехники. Полученный опыт и разработанное веб-приложение могут быть использованы для внедрения в деятельность реальных предприятий.

Список используемой литературы и используемых источников

1. Введение | Axios [Электронный ресурс] URL: https://axios-http.com/ru/docs/intro (дата обращения: 07.05.2025)
2. Начало работы – Jest [Электронный ресурс] URL: https://jestjs.io/ru/docs/getting-started (дата обращения: 07.05.2025)
3. Подробно про JWT [Электронный ресурс] URL: https://habr.com/ru/articles/842056/ (дата обращения: 07.05.2025)
4. Best NestJS Practicies and Techniques [Электронный ресурс] URL: https://dev.to/drbenzene/best-nestjs-practices-and-advanced-techniques-9m0 (дата обращения: 07.05.2025)
5. Bruno Docs – API Client [Электронный ресурс] URL: https://docs.usebruno.com/introduction/what-is-bruno (дата обращения: 07.05.2025)
6. Documentation | NestJS [Электронный ресурс] URL: https://vite.dev/guide/ (дата обращения: 07.05.2025)
7. Docs v6.30.1 | ReactRouter [Электронный ресурс] URL: https://reactrouter.com/6.30.1 (дата обращения: 07.05.2025)
8. Getting Started | Guide | Vitest [Электронный ресурс] URL: https://vitest.dev/guide/ (дата обращения: 07.05.2025)
9. Getting Started | Vite [Электронный ресурс] URL: https://vite.dev/guide/ (дата обращения: 07.05.2025)
10. JavaScript Guide – JavaScript | MDN [Электронный ресурс] URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide (дата обращения: 07.05.2025)
11. loadtest – npm [Электронный ресурс] URL: https://www.npmjs.com/package/loadtest (дата обращения: 07.05.2025)
12. Mongoose v8.15.0: Schemas [Электронный ресурс] URL: https://mongoosejs.com/docs/guide.html (дата обращения: 07.05.2025)

13. Overview – Material UI [Электронный ресурс] URL: https://mui.com/material-ui/getting-started/ (дата обращения: 07.05.2025)

14. Overview | Tanstack Query React Docs [Электронный ресурс] URL: https://tanstack.com/query/latest/docs/framework/react/overview (дата обращения: 07.05.2025)

15. React Architecture Patterns and Best Practices in 2025 [Электронный ресурс] URL: https://www.geeksforgeeks.org/react-architecture-pattern-and-best-practices/ (дата обращения: 07.05.2025)

16. React Reference Overview – React [Электронный ресурс] URL: https://react.dev/reference/react (дата обращения: 07.05.2025)

17. react-intersection-observer – npm [Электронный ресурс] URL: https://www.npmjs.com/package/react-intersection-observer (дата обращения: 07.05.2025)

18. TypeScript: Documentation [Электронный ресурс] URL: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html (дата обращения: 07.05.2025)

19. typestack/class-validator [Электронный ресурс] URL: https://github.com/typestack/class-validator (дата обращения: 07.05.2025)

20. yup – npm [Электронный ресурс] URL: https://www.npmjs.com/package/yup (дата обращения: 07.05.2025)