



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 клиент-серверных приложений
по профилю: Разработка программных продуктов и проектирование информационных систем
направления профессиональной подготовки: Программная инженерия (09.03.04)
Студент: Никитина Валерия Александровна
Группа: ИКБО-32-21
Срок представления к защите: 20.05.2024
Руководитель: ст.п., Коваленко Михаил Андреевич

Тема: «Разработка клиент-серверного фуллстек-приложения для сбора отзывов пользователей»

Исходные данные: React, Vite, Redux, Express, NodeJS, MongoDB, Git.

Перечень вопросов, подлежащих разработке, и обязательного графического материала: 1. Провести анализ предметной области для выбранной темы, обосновать выбор клиент-серверной архитектуры и дать её детальное описание с помощью UML для разрабатываемого приложения; 2. Выбрать программный стек для реализации фуллстек разработки, ориентируясь на мировой опыт и стандарты в данной области; 3. Реализовать фронтенд и бекенд части клиент-серверного приложения, обеспечивать авторизацию и аутентификацию пользователя; 4. Разместить готовое клиент-серверное приложение в репозитории GitHub с указанием ссылки в тексте отчёта; 5. Развернуть клиент-серверное приложение в облаке. 6. Провести пользовательское тестирование функционирования минимально жизнеспособного продукта. 7. Разработать презентацию с графическими материалами.

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

Зав. кафедрой ИиППО: [подпись] / Болбаков Р. Г. /, « 26 » 02 2024 г.

Задание на КР выдал: [подпись] / Коваленко М.А. /, « 26 » 02 2024 г.

Задание на КР получил: [подпись] / Никитина В.А. /, « 26 » 02 2024 г.

Заведующему кафедрой
инструментального и прикладного
программного обеспечения (ИиППО)
Института информационных технологий (ИТ)
Болбакову Роману Геннадьевичу

От студента Никитиной Валерии Александровны
ФИО
ИКБО-32-21
группа
3 курс
курс

Контакт: +7 925 570 33 90

Заявление

Прошу утвердить мне тему курсовой работы по дисциплине «Разработка клиент-серверных приложений» образовательной программы бакалавриата 09.03.04 (Программная инженерия).

Тема: «Разработка клиент-серверного фуллстек-приложения для сбора отзывов пользователей».

Подпись студента



/Никитина В.А.


подпись

ФИО

Дата

26.02.2024

Подпись руководителя




/ст.п. Коваленко М.А.

подпись

Должность, ФИО

Дата

26.02.24

 / Коваленко М.А.
26.02.24

АННОТАЦИЯ

Отчёт 4 с., 28 рис., 1 табл., 20 источн.

MERN, MongoDB, Express, React, Redux, Node.js, REST, Fullstack-приложение

Объектом разработки является сервис для сбора отзывов пользователей.

Цель работы – разработка полноценного клиент-серверного приложения для сбора отзывов пользователей.

В ходе работы был проведён анализ предметной области и обзор сервисов с аналогичной тематикой. Методом сравнительного анализа определены необходимые сущности и функциональные возможности приложения, требуемые для его реализации. Спроектирована информационная система, разработаны серверная и клиентская часть приложения, настроены СУБД и подключение к ней, настроен облачный сервер.

Результатом работы является клиент-серверное приложение для сбора отзывов пользователей на стеке MERN.

Report 44 p., 28 fig., 1 table, 20 sources.

MERN, MongoDB, Express, React, Redux, Node.js, REST, Fullstack application

The object of the development is a service for collecting user feedback.

The aim of the work is to develop a full-fledged client-server application for collecting user feedback.

During the work, an analysis of the subject area and a review of services with similar topics were carried out. The comparative analysis method was chosen to determine the necessary entities and functional capabilities of the application required for its implementation. The information system was designed, the server and client parts of the application were developed, the DBMS and the connection to it were configured, and the cloud server was configured.

The result of the work is a client-server application for collecting user feedback on the MERN stack.

РТУ МИРЭА: 119454, Москва, пр-т Вернадского, д. 78

кафедра инструментального и прикладного программного обеспечения (ИиППО)

Тираж: 1 экз. (на правах рукописи)

Файл: «НикитинаВА_ИКБО-32-21.docx», исполнитель Никитина В.А.

© Никитина В.А.

ГЛОССАРИЙ

1. **Архитектура приложения** – структурный план приложения, описывающий организацию его компонентов, их назначение и взаимодействие друг с другом.

2. **Фуллстек-приложение** - программное приложение, охватывающее как серверный, так и клиентский уровни. Фуллстек разработчик способен работать как с фронтендом, так и с бэкендом приложения.

3. **Серверная часть** – часть приложения, выполняемая на сервере. Отвечает за обработку запросов клиента, взаимодействие с базой данных, реализацию бизнес-логики и другие серверные функции.

4. **Клиентская часть** - пользовательский интерфейс приложения. Обрабатывает действия пользователя, передает запросы на сервер и отображает полученную информацию.

5. **База данных** - хранилище данных, обеспечивающее долгосрочное сохранение и доступ к информации. Данные могут организовываться в различных форматах и структурах в зависимости от типа базы данных.

6. **СУБД** - система управления базами данных. Позволяет создавать, администрировать и получать доступ к базам данных. Обеспечивает обработку запросов, безопасность и целостность данных.

СОДЕРЖАНИЕ

АННОТАЦИЯ	3
ГЛОССАРИЙ	4
СОДЕРЖАНИЕ	5
ВВЕДЕНИЕ	7
1 ОБЩИЕ СВЕДЕНИЯ	9
1.1 Обозначение и наименование интернет-ресурса.....	9
1.2 Функциональное назначение	9
2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	10
2.1 Анализ функционала.....	12
Вывод к главе 2	13
3 ВЫБРАННЫЕ ТЕХНОЛОГИИ ДЛЯ РАЗРАБОТКИ ПРИЛОЖЕНИЯ	14
3.1 Выбор архитектуры.....	14
3.2 Выбор основных технологий.....	16
Вывод к главе 3	17
4 АРХИТЕКТУРА ВЕБ-ПРИЛОЖЕНИЯ	19
Вывод к главе 4	22
5 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ	23
5.1 Определение схемы базы данных	23
5.2 Аутентификация и авторизация	25
5.3 Обработка запросов	26
Вывод к главе 5	27
6 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ.....	28
6.1 Взаимодействие с сервером.....	28
6.2 Пример создания компонента web-приложения	29
6.3 Интерфейс веб-приложения	29
Вывод к главе 6	35
7 ТЕСТИРОВАНИЕ	36
Вывод к главе 7	38
8 РАЗВЕРТЫВАНИЕ ПРИЛОЖЕНИЯ	39

8.1 Выбор платформы для развертывания.....	39
Вывод к главе 8	41
ЗАКЛЮЧЕНИЕ	42
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	43

ВВЕДЕНИЕ

В современном бизнесе всё большую важность приобретает сбор отзывов от пользователей. Однако в настоящее время нет программных решений, которые бы полностью охватывали все аспекты этой области: управление отзывами, анализ обратной связи, управление обзорами и т.д. В связи с этим предлагается создать приложение, которое будет обеспечивать сбор и анализ пользовательских отзывов. Приложение будет включать серверную часть, разработанную с использованием методологии REST на платформе Express, с базой данных MongoDB, клиентскую часть, и будет развернуто на облачном сервисе Render.com.

Работа содержит следующие основные разделы:

1. ОБЩИЕ СВЕДЕНИЯ: выбор обозначение и наименования интернет-ресурса, а также определение функционального назначения.

2. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ: включает анализ существующих инструментов и постановку функциональных требований к приложению.

3. ВЫБРАННЫЕ ТЕХНОЛОГИИ ДЛЯ РАЗРАБОТКИ ПРИЛОЖЕНИЯ: описание выбранных технологий и обоснование их использования.

4. АРХИТЕКТУРА ВЕБ-ПРИЛОЖЕНИЯ: описание архитектуры серверной и клиентской частей приложения, а также архитектуры развертывания приложения.

5. РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ: описание структуры базы данных и разработка серверной части приложения.

6. РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ: краткое описание разработки клиентской части и представление интерфейса приложения.

7. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ: было проведено ручное тестирование для проверки функциональности веб-приложения.

8. РАЗВЕРТЫВАНИЕ ПРИЛОЖЕНИЯ: описание процесса развертывания приложения на облачных серверах.

Подводя итоги, создаваемое приложение предназначено для сбора и анализа пользовательских отзывов, предоставляя удобный инструмент для управления обратной связью.

1 ОБЩИЕ СВЕДЕНИЯ

1.1 Обозначение и наименование интернет-ресурса

Тема разработанного веб-приложения – «Сервис для сбора отзывов пользователя». Приложение получило название «Revvo»

1.2 Функциональное назначение

Приложение предназначено для написания и поиска отзывов, а также позволяет пользователям управлять ими. После входа в систему пользователи могут создавать и оценивать отзывы, оставлять комментарии, отправлять заявки и жалобы, управлять своими данными.

Администратор имеет возможность контролировать активность пользователей и принимать необходимые меры в случае нарушений.

2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Для установления необходимого функционала создаваемого приложения были проанализированы 3 веб-приложения с тематикой «сайт с отзывами»:

- <https://www.irecommend.ru>,
- <https://otzovik.com> ,
- <https://otzyv.ru>.

На следующих Рисунках 2.1 - 2.3 показаны главные страницы анализируемых сайтов.

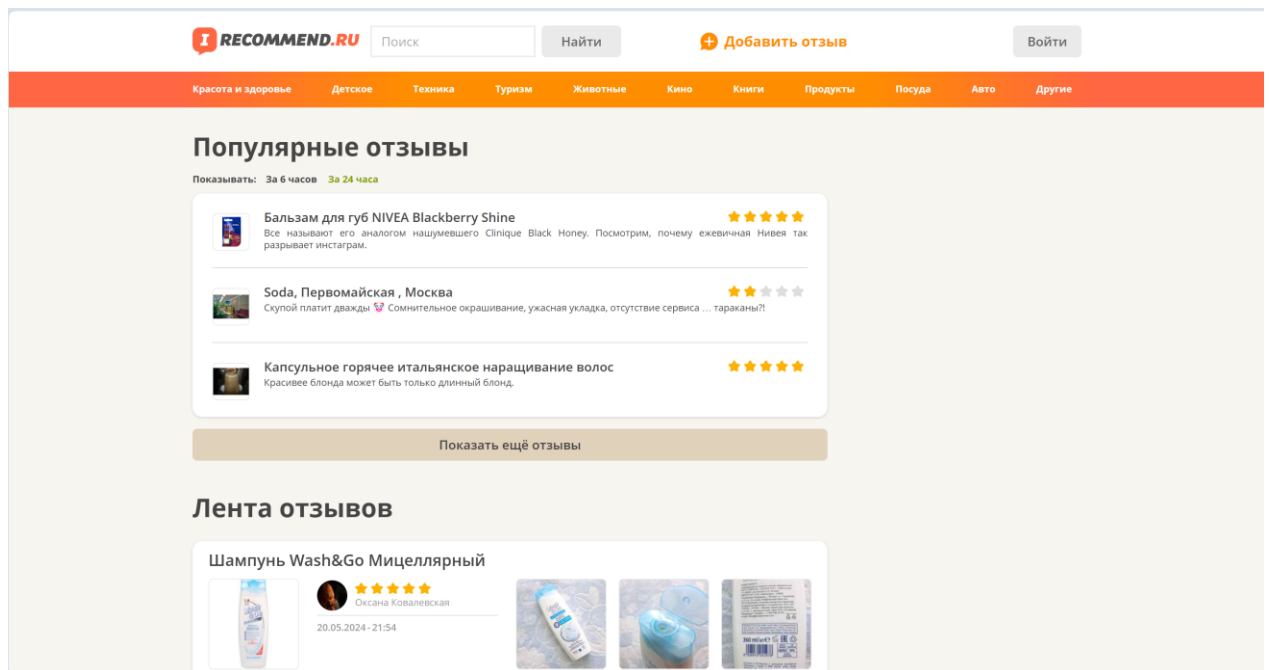


Рисунок 2.1 – Сайт сервиса «Recommend.ru»

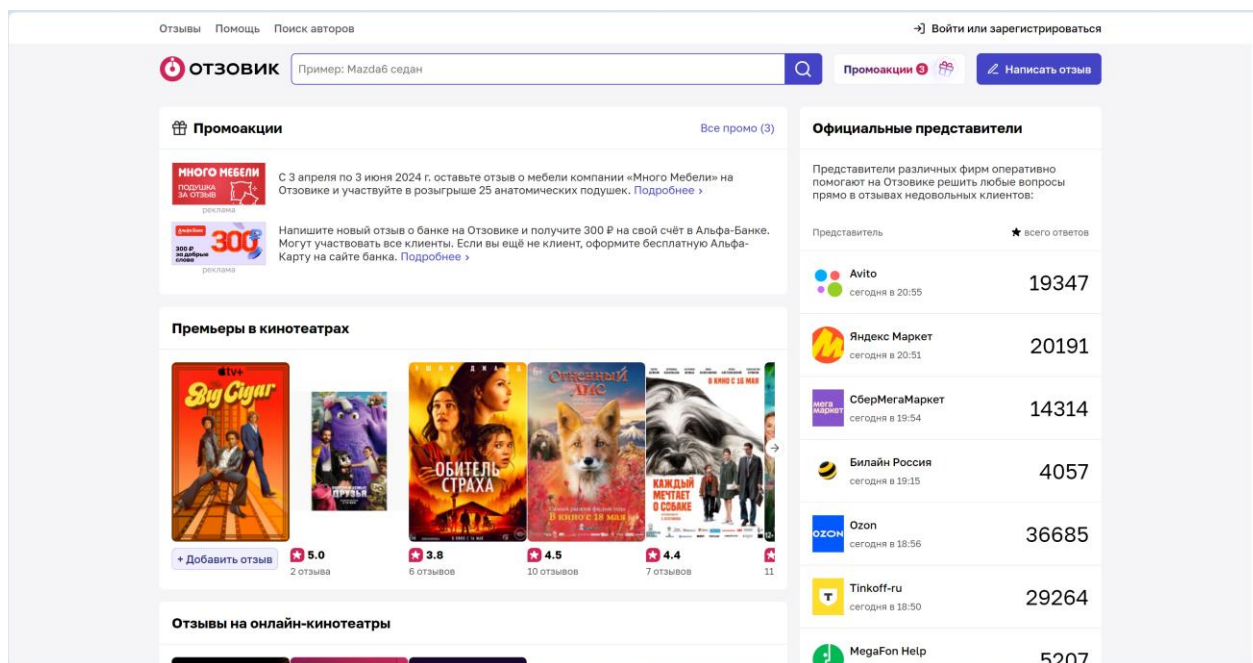


Рисунок 2.2 – Сайт сервиса «Отзовик»

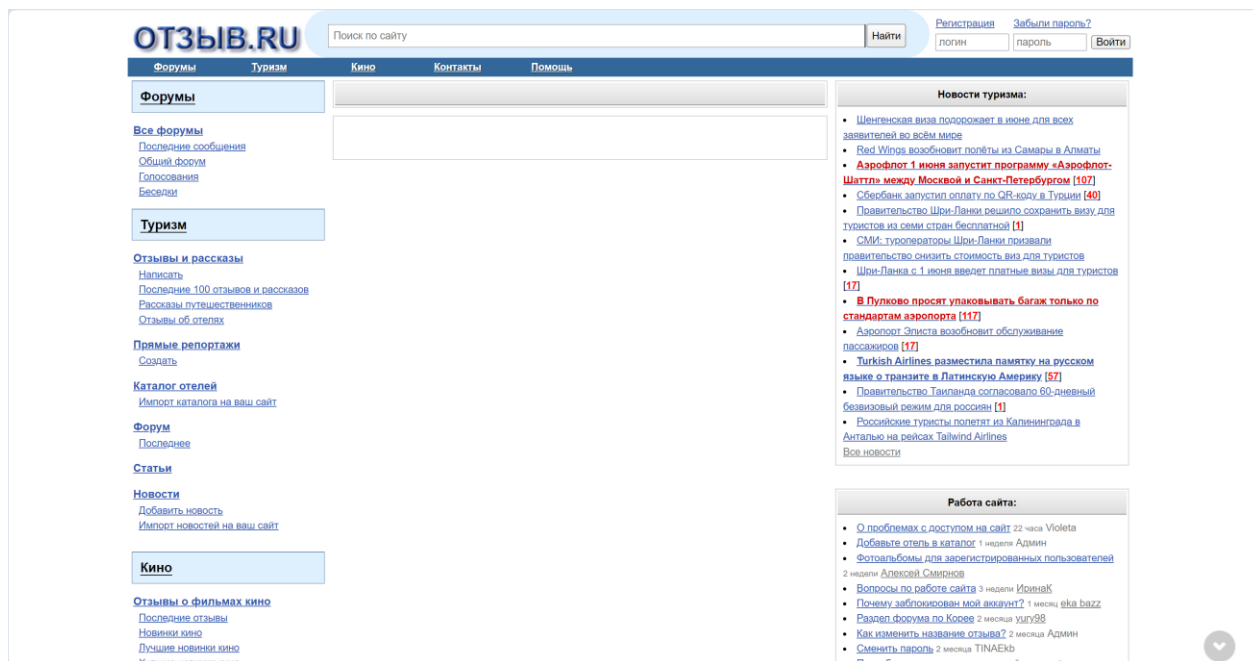


Рисунок 2.3 – Сайт сервиса «Отзыв.ру»

Можно заметить, что все представленные страницы довольно сильно перегружены лишними элементами, а на главных страницах отображаются не релевантные отзывы. Пользователь имеет возможность оставить отзыв на всех этих платформах, но не имеет возможности оставить заявку на добавление категории. Везде присутствует возможность фильтрации и комментирования существующих отзывов.

2.1 Анализ функционала

Составим сравнительную таблицу функционала аналогов разрабатываемого веб-сервиса, чтобы понять, что следует добавить в функционал нашего интернет-ресурса.

Таблица 2.1 - Сравнительный анализ аналогов разрабатываемого веб-сервиса

Название веб-сервиса	Recommend.ru	Отзовик	Отзыв.ru
Интуитивно понятный неперегруженный интерфейс	-	-	-
Возможность оставить отзыв	+	+	+
Возможность пользователей отправлять заявки на добавление новых категорий	-	-	-
Поддержка фильтров для поиска отзывов	+	+	+
Возможность комментирования существующих отзывов	+	+	+
Возможность пользователей оценивать релевантность отзывов	+	-	-
Возможность администрирования активности пользователей	+	+	+
Возможность взаимодействовать с личными данными и отзывами, комментариями, жалобами, запросами пользователей	+	+	+

Таким образом, создаваемый сервис должен иметь простой и удобный интерфейс, включать основные функции альтернативных приложений, использованных для анализа, а также предоставлять пользователям возможность оценивать качество, релевантность и другие характеристики отзывов.

При разработке приложения были учтены преимущества и недостатки существующих решений.

Вывод к главе 2

В процессе анализа предметной области были определены преимущества и недостатки различных веб-сервисов для сбора отзывов пользователей. На основании этого были сформулированы требования к приложению:

- действия с аккаунтами пользователей,
- действия с категориями отзывов,
- действия с отзывами
- действия с оценками
- действия с комментариями
- действия с заявками
- действия с жалобами

Для реализации нам понадобится:

- создание базы данных с таблицами: пользователи, сессии, категории, отзывы, комментарии, оценки, запросы, жалобы

3 ВЫБРАННЫЕ ТЕХНОЛОГИИ ДЛЯ РАЗРАБОТКИ ПРИЛОЖЕНИЯ

3.1 Выбор архитектуры

Для реализации веб-сервиса требуется использование клиент-серверной архитектуры. В ней можно выделить три класса архитектур:

1. Одноуровневая архитектура объединяет все компоненты приложения (интерфейс, бизнес-логику и данные) в одном месте, что делает ее простой в разработке и обслуживании.

2. Двухуровневая архитектура (клиент-сервер) разделяет приложение на клиент и сервер, что повышает масштабируемость и надежность. Клиент отвечает за пользовательский интерфейс, а сервер обрабатывает запросы и управляет данными.

3. Многоуровневая архитектура выходит за рамки двухуровневой, разделяя приложение на несколько уровней, что обеспечивает еще большую масштабируемость, гибкость и возможность повторного использования компонентов.

С целью улучшения масштабируемости, гибкости и независимости клиентской и серверной части и сервера базы данных была выбрана многоуровневая, а именно трехуровневая, архитектура.

Далее были проанализированы следующие популярные архитектурные паттерны:

1. Модель-представление-контроллер (MVC)
 - a. Модель: Представляет бизнес-логику и данные приложения.
 - b. Представление: Отображает данные для пользователя и собирает ввод пользователя.
 - c. Контроллер: Обрабатывает пользовательский ввод и направляет запросы к модели.
2. Микросервисная архитектура - вариант архитектуры на основе сервисов, где сервисы очень мелкие и независимые. Микросервисы

взаимодействуют друг с другом с использованием легких протоколов, таких как REST или gRPC.

3. Архитектура, ориентированная на сообщения
 - a. Приложение использует брокер сообщений для передачи сообщений между компонентами.
 - b. Сообщения содержат данные или команды, которые направляются получателям, подписанным на соответствующие темы.
4. Архитектура с обратным прокси
 - a. Прокси-сервер, работающий перед сервером приложений, направляет запросы клиентов на соответствующие серверы.
 - b. Прокси-сервер может выполнять балансировку нагрузки, кеширование и обработку безопасности.
5. Архитектура на основе контейнеров
 - a. Приложение упаковывается в контейнеры, которые независимо развертываются и управляются.
 - b. Контейнеры изолируют приложение от базовой инфраструктуры и упрощают развертывание и масштабирование.
6. Serverless-архитектура
 - a. Приложение размещается в облачной среде без необходимости управления серверами или инфраструктурой.
 - b. Провайдер облачных услуг автоматически выделяет ресурсы и масштабирует приложение в соответствии с требованиями.
7. Гибридная архитектура
 - a. Комбинация нескольких архитектурных паттернов для удовлетворения конкретных требований приложения.
 - b. Например, приложение может использовать MVC для пользовательского интерфейса, микросервисную архитектуру для бизнес-логики и архитектуру на основе контейнеров для развертывания.

Среди данных архитектурных паттернов был выбран MVC. Так как приложение не является нагруженным, использовать гибридный подход для объединения микросервисов, брокеров сообщений, разбиения на контейнеры, обратного проксирования было сочтено нецелесообразным в рамках данной курсовой работы, однако в перспективе развития веб-сервиса данный подход будет использован для совмещения всех положительных сторон, предоставляемых вышеперечисленными паттернами.

Исходя из того, что в приложении будут выполняться только CRUD операции, в качестве модели взаимодействия клиента с сервером был выбран REST.

3.2 Выбор основных технологий

1. Язык программирования JavaScript - JavaScript широко используется для разработки веб-приложений благодаря своей универсальности, широкому сообществу разработчиков и возможности запуска на стороне клиента и сервера.

2. Фреймворк Express - это минималистичный и гибкий фреймворк для Node.js, который позволяет быстро создавать веб-приложения и API. Он обладает простым API и хорошей производительностью.

3. JSON Web Tokens - стандарт для создания аутентификационных токенов, которые могут быть переданы между клиентом и сервером для безопасной аутентификации пользователя.

4. Cookie-parser - middleware для Express, который упрощает работу с куки в HTTP-запросах, обеспечивая удобный доступ к данным, хранящимся в куки.

5. Npm - менеджер пакетов для JavaScript, который позволяет быстро устанавливать, обновлять и управлять зависимостями проекта.

6. Mongoose - библиотека для работы с MongoDB в Node.js, предоставляющая удобный интерфейс для создания моделей данных и выполнения запросов.

7. MongoDB - NoSQL база данных, хранящая данные в формате JSON-подобных документов, что делает её гибкой и масштабируемой для хранения различных типов данных.

8. Git - распределённая система управления версиями, обеспечивающая контроль изменений в коде, совместную работу над проектом и отслеживание истории изменений.

9. GitHub - платформа для хостинга и совместной разработки проектов на основе Git, предоставляющая возможности для управления кодом, совместной работы и отслеживания изменений.

10. React - JavaScript библиотека для создания пользовательских интерфейсов, обеспечивающая эффективное обновление DOM и компонентный подход к разработке интерфейсов.

11. Redux-toolkit - официальный пакет для Redux, который упрощает работу с Redux, предоставляя удобные инструменты для управления состоянием приложения.

12. Axios - библиотека для выполнения HTTP-запросов в браузере и Node.js, обеспечивающая удобный интерфейс для обработки запросов и ответов.

13. SASS - препроцессор CSS, который добавляет дополнительные возможности к CSS, такие как переменные, вложенности, миксины, что облегчает работу с стилями.

14. Render.com - платформа для деплоя и хостинга веб-приложений, которая предлагает простую настройку, автоматическое масштабирование и управление сертификатами безопасности

15. Visual Studio Code - это бесплатный и открытый редактор кода, разработанный Microsoft. Он легкий, кроссплатформенный и расширяемый.

Вывод к главе 3

В ходе анализа архитектур, подходов и моделей взаимодействия клиент-серверных приложений были выбраны трехслойная архитектура, MVC и REST

на основе их положительных сторон и сформированных ранее требований к приложению.

Выбранные технологии разработки веб-приложений пользуются большой популярностью благодаря удобству, эффективности и поддержке со стороны сообщества разработчиков. Каждая технология имеет свои сильные стороны и может быть эффективно использована на разных этапах разработки веб-приложений, обеспечивая надежное и эффективное функционирование.

4 АРХИТЕКТУРА ВЕБ-ПРИЛОЖЕНИЯ

В качестве архитектуры разработки веб-приложения [6] была выбрана трехслойная архитектура, также были выбраны архитектурный паттерн MVC и модель взаимодействия клиента с сервером - REST.

Согласно выбранной архитектуре, приложение было разделено на три части, так что и клиентская часть, и серверная часть, и сервер базы данных работают на разных хостах, что обеспечивает определенный уровень надежности, масштабируемости, гибкости и независимости каждой из частей. Таким образом, при определенных действиях пользователя отправляются запросы на веб-сервер, который формирует соответствующие запросы на выполнение операций на сервере, сервер обрабатывает их, взаимодействует с сервером базы данных. В результате необходимые данные в обратном порядке возвращаются на клиентскую часть, где браузер производит обновление элементов страницы в соответствии с ними.

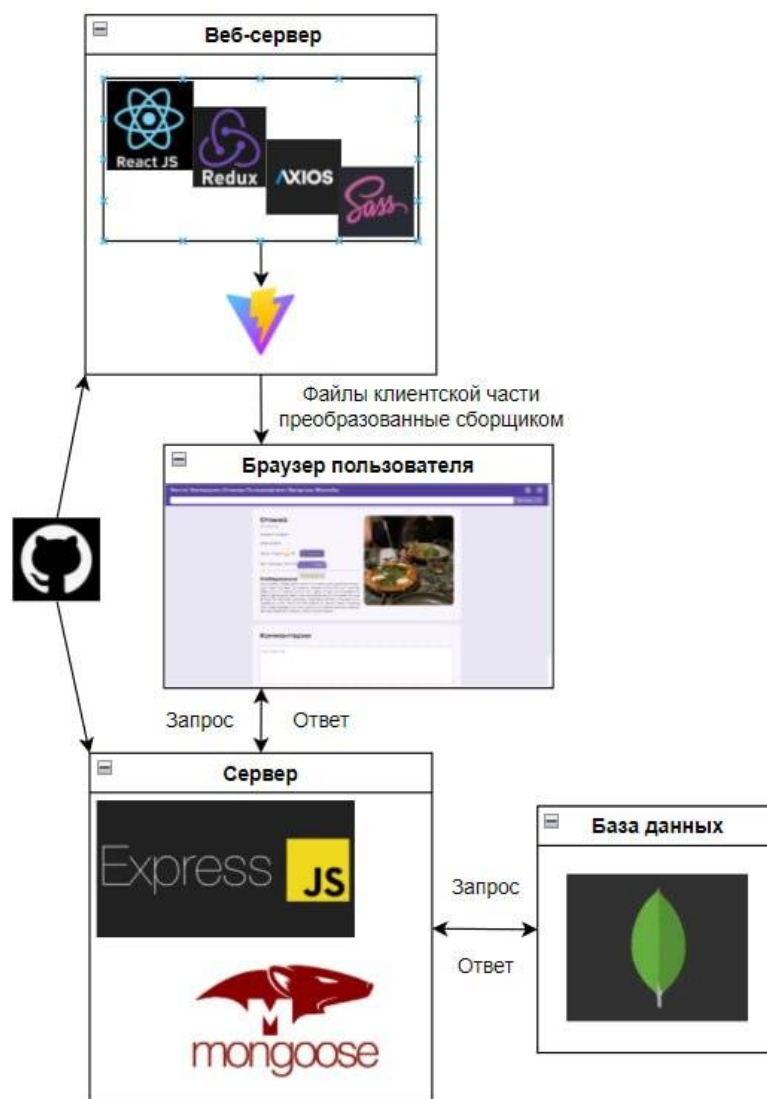


Рисунок 4.1 – Структура архитектуры веб-приложения

Архитектура MVC (Model-View-Controller) представляет собой парадигму разделения ответственностей в веб-приложении. С её помощью приложение делится на три основных компонента:

1. Модель (Model): Отвечает за представление данных и бизнес-логики приложения. При использовании MongoDB и Mongoose модель определяется схемой данных и методами для работы с базой данных.
2. Представление (View): Отвечает за интерфейс, визуализацию данных и взаимодействие с пользователем. В React представления создаются с использованием компонентов, которые отображают данные и обрабатывают события.
3. Контроллер (Controller): Управляет запросами от пользователя,

вызывает соответствующие методы модели для получения данных и передаёт их в представление. В Express контроллеры обрабатывают маршруты и вызывают методы модели для работы с данными.

Используя технологии Express, MongoDB, Mongoose и React вместе, процесс выглядит следующим образом:

1. Клиентское приложение на React отправляет HTTP-запросы к серверу Express через API-маршруты.
2. Сервер Express обрабатывает эти запросы, вызывая соответствующие методы контроллеров.
3. Контроллеры обращаются к сервисам, а те, в свою очередь, взаимодействуют с моделями, представляющими данные в базе MongoDB через Mongoose, чтобы получить или обновить информацию.
4. Данные, полученные из модели, передаются обратно в контроллер, который формирует ответ для клиентского приложения на React.
5. React обновляет интерфейс на основе полученных данных, взаимодействует с пользователем и при необходимости отправляет новые запросы на сервер.

Следовательно, применение архитектуры MVC с использованием Express, MongoDB, Mongoose и React обеспечивает ясное распределение задач между компонентами приложения, что гарантирует гибкость, масштабируемость и эффективное управление данными и интерфейсом.

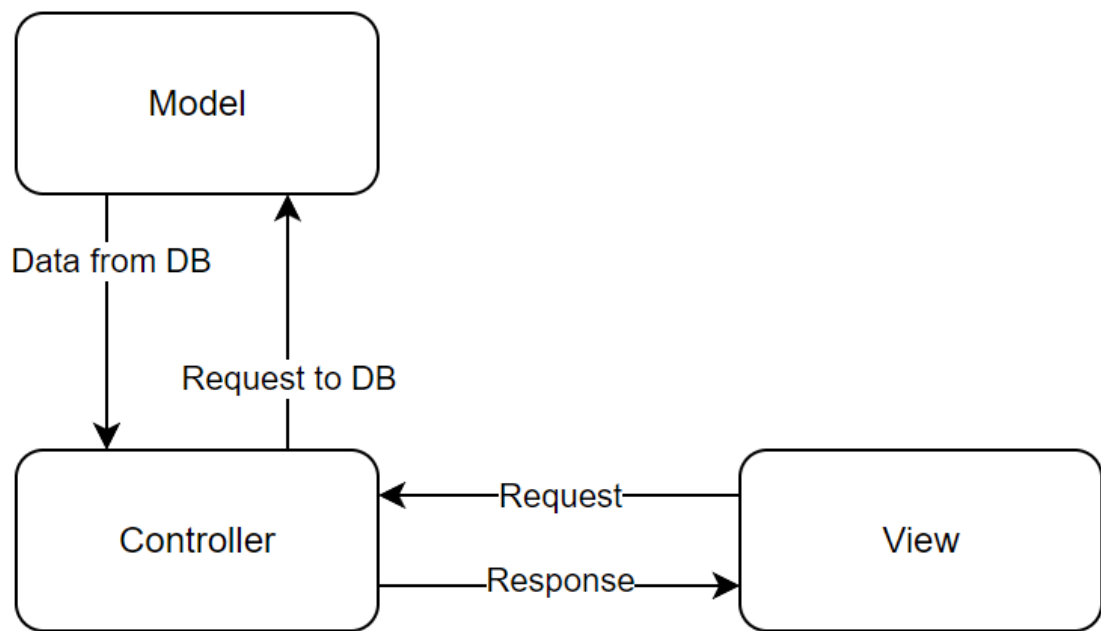


Рисунок 4.2 – Архитектура MVC

Вывод к главе 4

Таким образом, трехслойная архитектура, архитектурный шаблон MVC, реализованные с использованием Express, MongoDB, Mongoose и React, позволяют эффективно разделять роли компонентов приложения. Это разделение обеспечивает гибкость, масштабируемость и эффективное управление как данными, так и пользовательским интерфейсом, что приводит к надежной и удобной веб-системе.

5 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ

5.1 Определение схемы базы данных

В качестве базы данных выбрана NoSQL-система MongoDB [4][11], а для более удобного взаимодействия с базой данных с серверной стороны используется ORM Mongoose.

Для работы приложения были выбраны и реализованы следующие сущности:

1. Пользователи,
2. Сессии,
3. Категории,
4. Отзывы,
5. Оценки отзывов,
6. Комментарии отзывов.
7. Запросы.
8. Жалобы.

Структуру базы данных можно просмотреть на Рисунке 5.1, где показано её создание с помощью Draw.io.

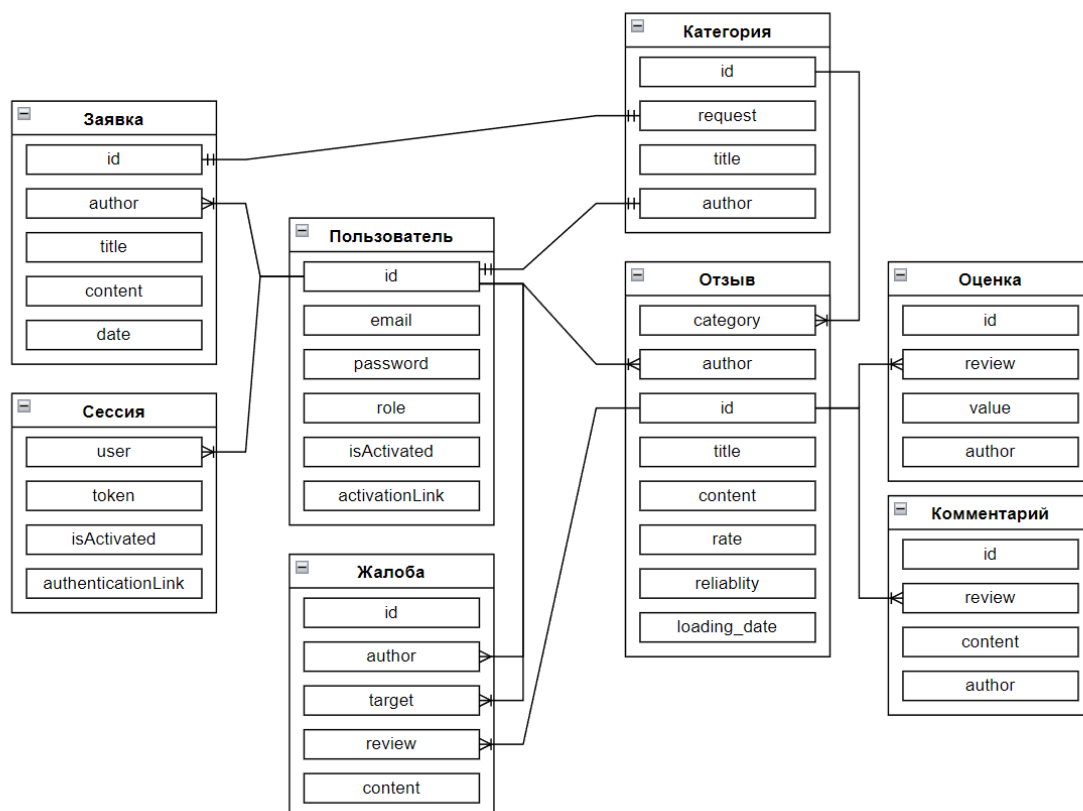


Рисунок 5.1 – Представление схемы базы данных, составленное с помощью Draw.io

5.2 Аутентификация и авторизация

Код функций сервиса для работы с аутентификацией и авторизацией [12] пользователей представлен на листинге 5.1 [10].

Листинг 5.1 – Часть кода для работы аутентификации и авторизации

```
class AuthService {
  async register(email, password) {
    let user = await userModel.findOne({ email })
    if (user) {
      throw ApiError.badRequestError(`Пользователь с адресом ${email} уже
существует!`)
    }

    const hashedPassword = await bcrypt.hash(password, 3)
    const activationLink = v4()
    const nickname = v4()
    user = await userModel.create({ email, password: hashedPassword,
activationLink, nickname })
    await mailService
      .sendActivationMail(email, `${process.env.MODE === "production"
? process.env.API_DEPLOYED_URL
: process.env.API_LOCAL_URL
}/api/auth/activate_account/${activationLink}`)
    const userDTO = new UserDTO(user)
    const tokens = jwtService.generateTokens({ ...userDTO })
    await jwtService.saveToken(userDTO.id, tokens.refreshToken, activationLink)

    return { ...tokens, user: userDTO }
  }

  async login(email, password) {
    let user = await userModel.findOne({ email })

    if (!user) {
```

Продолжение листинга 5.1

```
        throw ApiError.badRequestError(`Пользователь с адресом ${email} не  
существует!`)  
    }  
  
    const isPassEqual = await bcrypt.compare(password, user.password)  
    if (!isPassEqual) {  
        throw ApiError.badRequestError(`Неверный пароль`)  
    }  
}
```

5.3 Обработка запросов

В целях сокращения отчета, обработка запросов демонстрируется на примере контроллера для работы с пользователями (Листинг 5.2) [10][15]. На листинге представлены методы контроллера для обработки запросов на получение всех пользователей, получение данных одного пользователя, блокировку пользователя в системе [3][9].

Листинг 5.2 – Контроллер для обработки запросов взаимодействия с данными пользователей

```
class RequestService {  
    async create(title, content, author) {  
        const date = new Date()  
        const request = await requestModel.create({ title, content, author, date })  
        const relatedAuthor = await userModel.findById(request.author)  
        const requestData = new RequestDTO(request, new UserDTO(relatedAuthor))  
        return requestData  
    }  
  
    async getAll() {  
        const requests = Promise.all((await requestModel.find()).map(async model => {  
            const author = await userModel.findById(model.author)  
            return new RequestDTO(model, new UserDTO(author))  
        })))  
        return requests  
    }  
}
```

Продолжение листинга 5.2

```
    }

    async getUserRequests(userId) {
      const requests = Promise.all((await requestModel.find({ author: userId })))
        .map(async (model) => {
          const author = await userModel.findOne(model.author)
          return new RequestDTO(model, new UserDTO(author))
        })
    }
    return requests
  }
}
```

Вывод к главе 5

В этой главе была успешно разработана серверная логика приложения. Она включает в себя реализацию авторизации пользователей с использованием токенов JWT, обработку запросов от клиентской части и настройку взаимодействия с базой данных. Эти компоненты работают вместе, обеспечивая надежное и эффективное функционирование серверной части.

6 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ

6.1 Взаимодействие с сервером

Для реализации взаимодействия с сервером используются Axios и Redux actions [2,8]. Для удобства настройки взаимодействия был создан экземпляр Axios, в котором описана основная конфигурация, включая адрес сервера и перехватчики запросов и ответов от сервера, благодаря чему, при запросе к серверу в него добавляются заголовки запроса, а при ответе от сервера, в случае ошибки авторизации отправляется запрос на обновление токенов и далее повторяется исходный запрос (Листинг 6.1).

Листинг 6.1 – Создание экземпляров axios

```
const API_DEV_URL = 'http://localhost:5000'
const API_PROD_URL = 'https://rcsp-back.onrender.com'
export const API_URL = `${import.meta.env.MODE === "production"
  ? API_PROD_URL
  : API_DEV_URL}/api`
export const STATIC_URL = `${import.meta.env.MODE === "production"
  ? API_PROD_URL
  : API_DEV_URL}`

const cookies = new Cookies()

const api = axios.create({
  withCredentials: true,
  baseURL: API_URL
})

api.interceptors.request.use(config => {
  config.headers.Authorization = `Bearer ${cookies.get('token')}`
  return config
})
```

6.2 Пример создания компонента web-приложения

Для разработки клиентской части приложения использовалась файловая архитектура FSD (Feature-sliced-design). Все компоненты были организованы по семантическим уровням: приложение (app), сущности (entities), функциональные модули (features), страницы (pages), общие компоненты (shared), виджеты (widgets). На рисунке 6.1 представлен одна из страниц, реализованных в приложении [1,7].

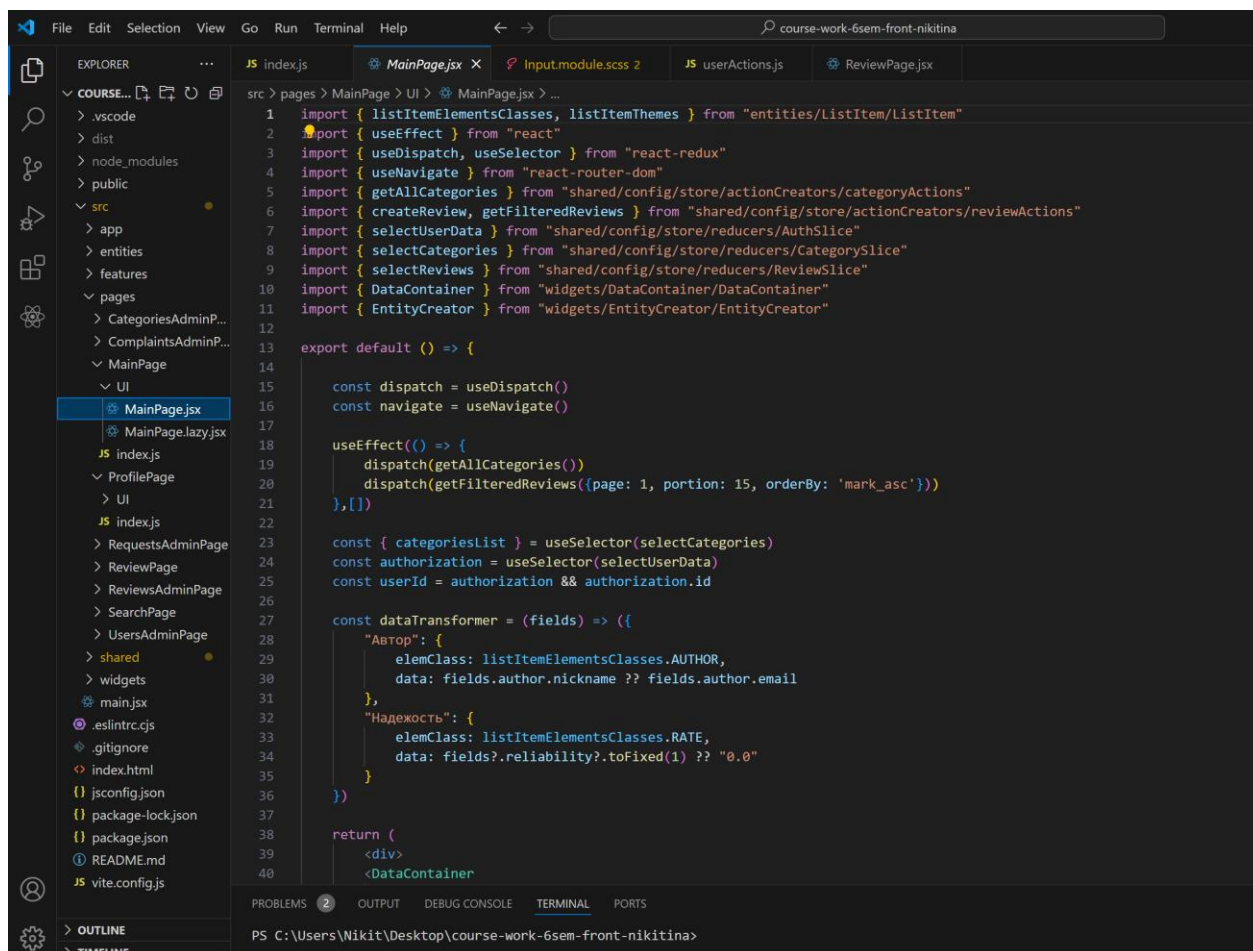


Рисунок 6.1 – Реализация главной страницы приложения

6.3 Интерфейс веб-приложения

Для оформления интерфейса приложения применяется препроцессор Scss. Внутри директории приложения "app" расположена папка "styles", которая содержит настройки базовых стилей приложения.

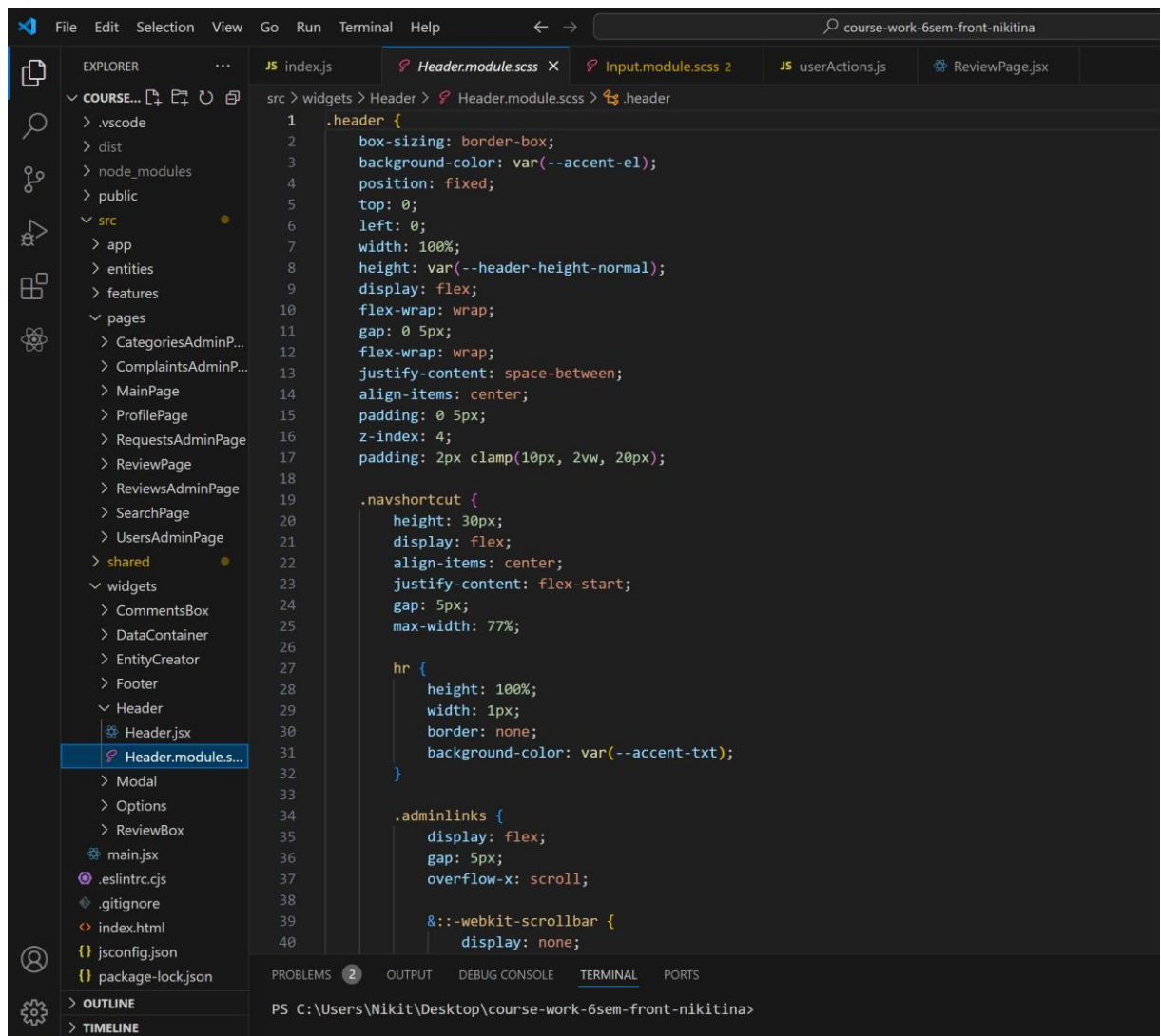
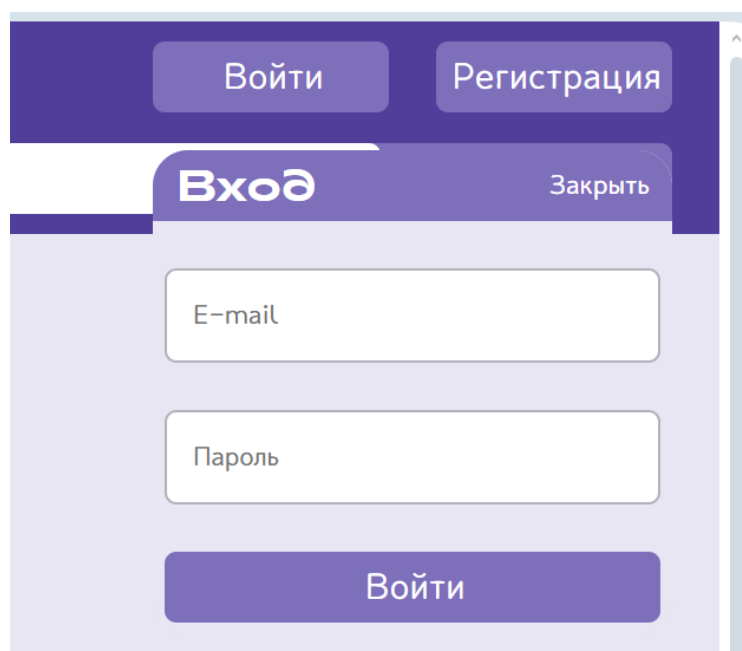


Рисунок 6.2 – Файл задания настроек стилей header

В результате разработки клиентской части приложения был реализован интерфейс, представленный на рисунках 6.3-6.11.

Окно входа/регистрации представлено на рисунке 6.3.



The image shows a login/register window with a purple header. At the top, there are two buttons: "Войти" (Login) and "Регистрация" (Registration). Below them is a tab labeled "Вход" (Login) with a "Закрыть" (Close) button. The main area contains two input fields: "E-mail" and "Пароль" (Password). At the bottom is a large purple button labeled "Войти" (Login).

Рисунок 6.3 – Страница авторизации

На Рисунке 6.4 показана главная страница.

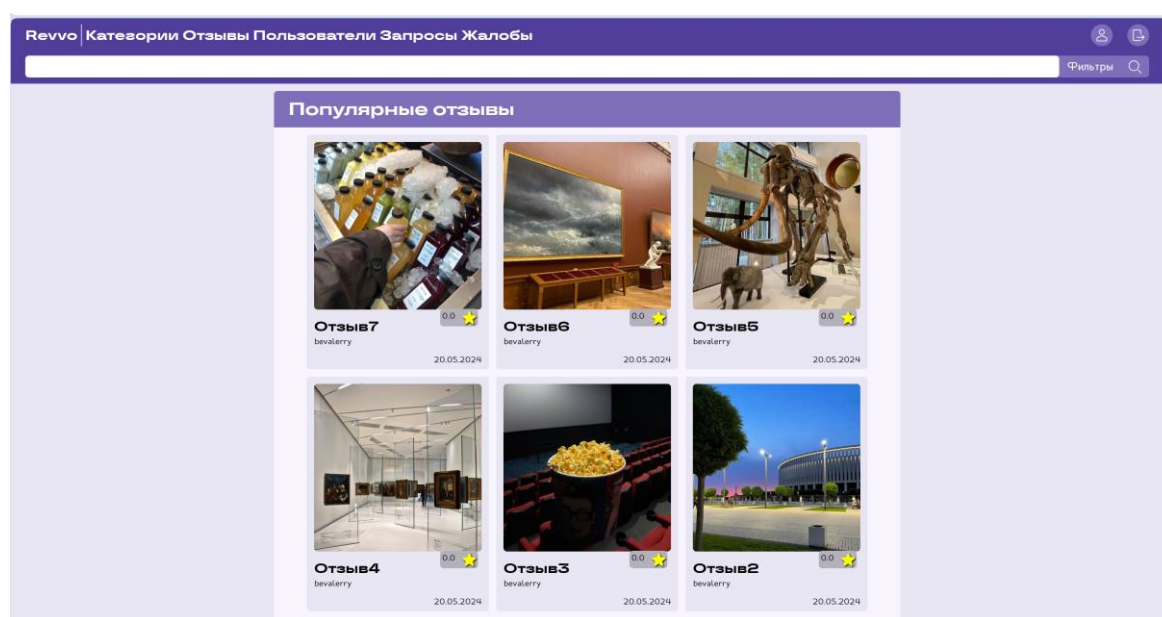


Рисунок 6.4 – Главная страница

На рисунке 6.5 представлена страница личного кабинета пользователя

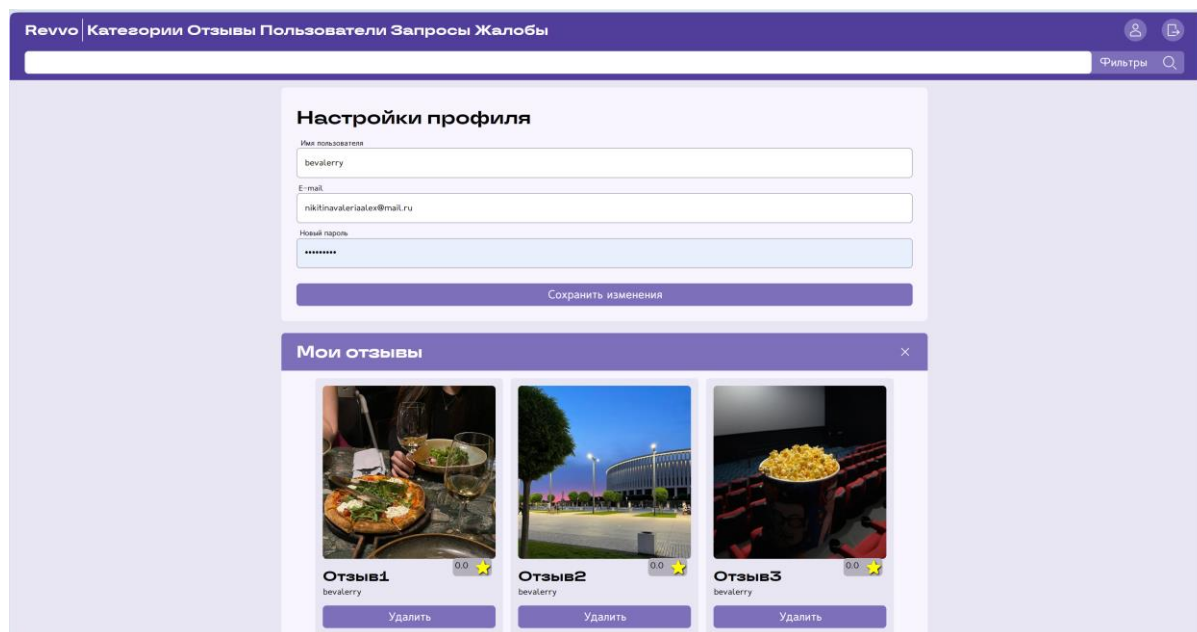


Рисунок 6.5 – Страница настроек профиля

На Рисунке 6.6 показана страница конкретного отзыва.

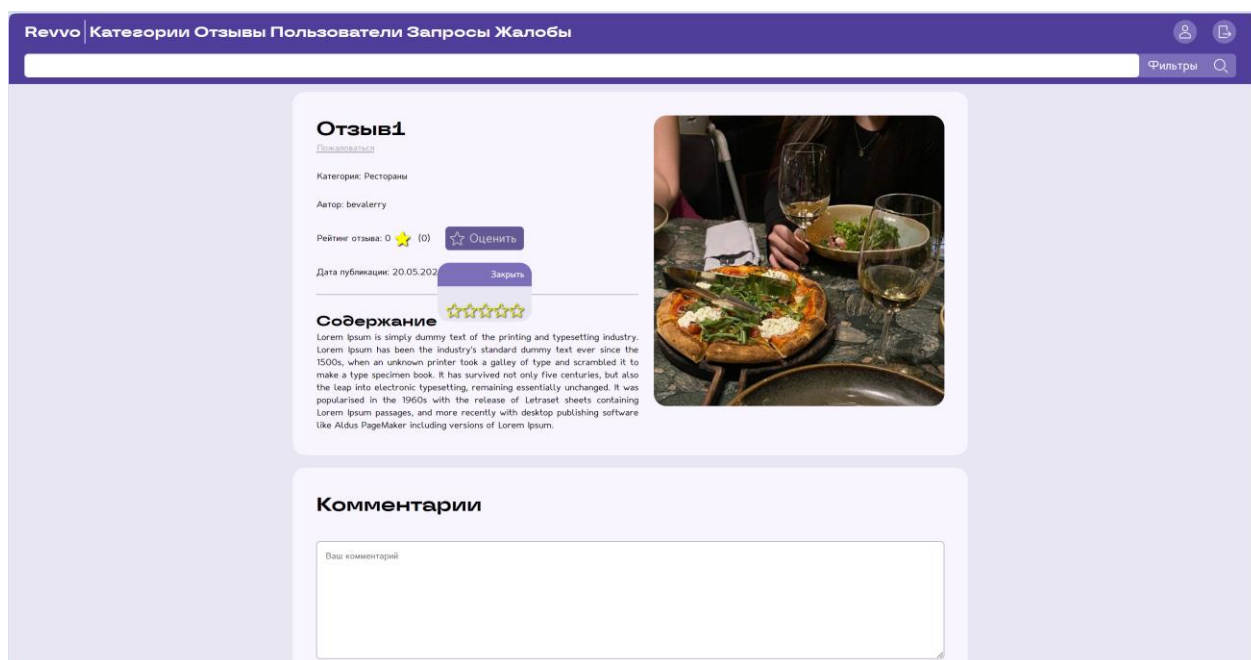


Рисунок 6.6 – Страница отзыва

На Рисунке 6.7 показана страница со всеми категориями отзывов.

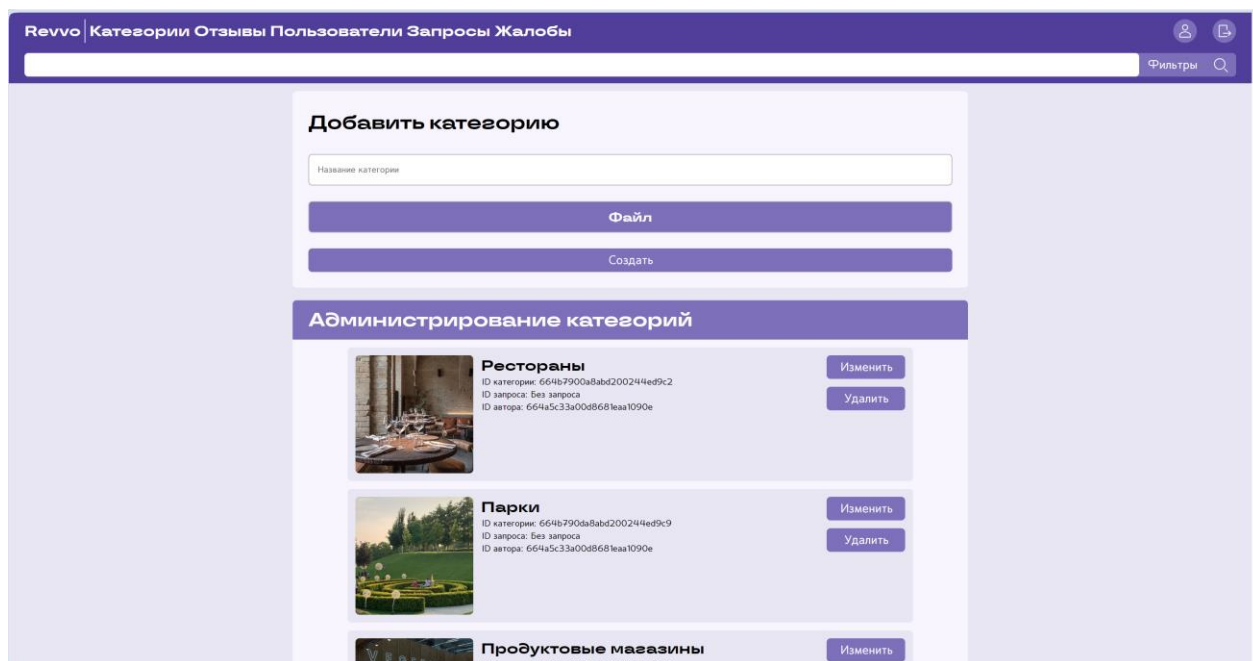


Рисунок 6.7 – Страница с категориями

На Рисунке 6.8 показана страница с администрированием отзывов (у администратора).

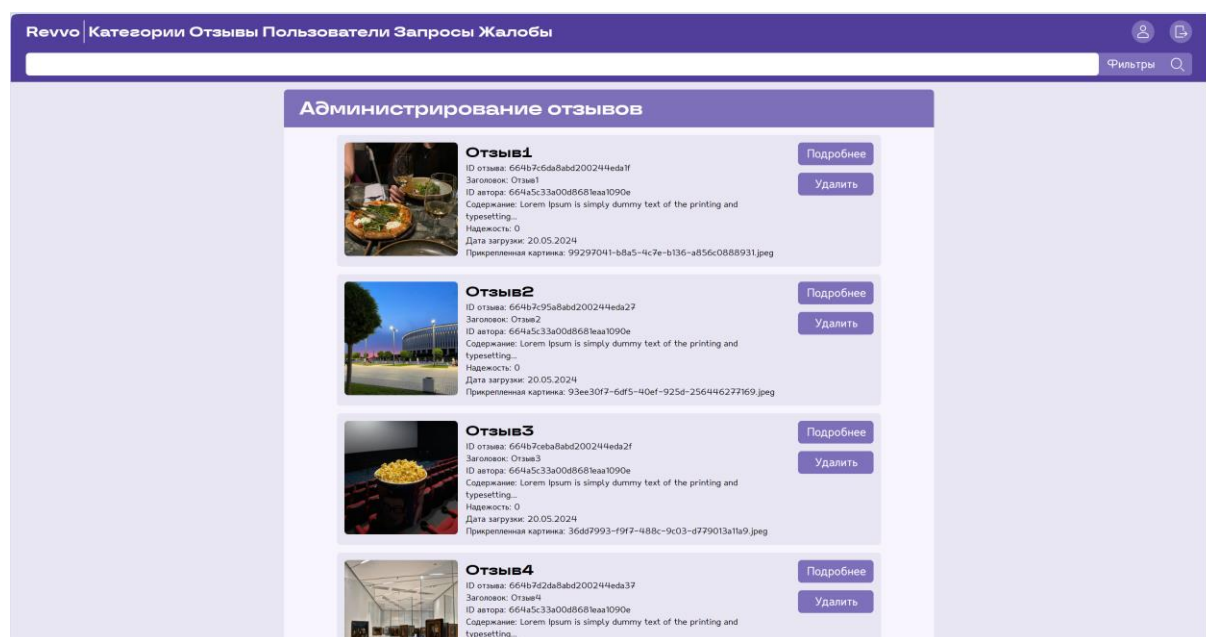


Рисунок 6.8 – Страница для администрирования отзывов

Специально для администратора создана страница для просмотра и удаления пользователей (Рисунок 6.9).

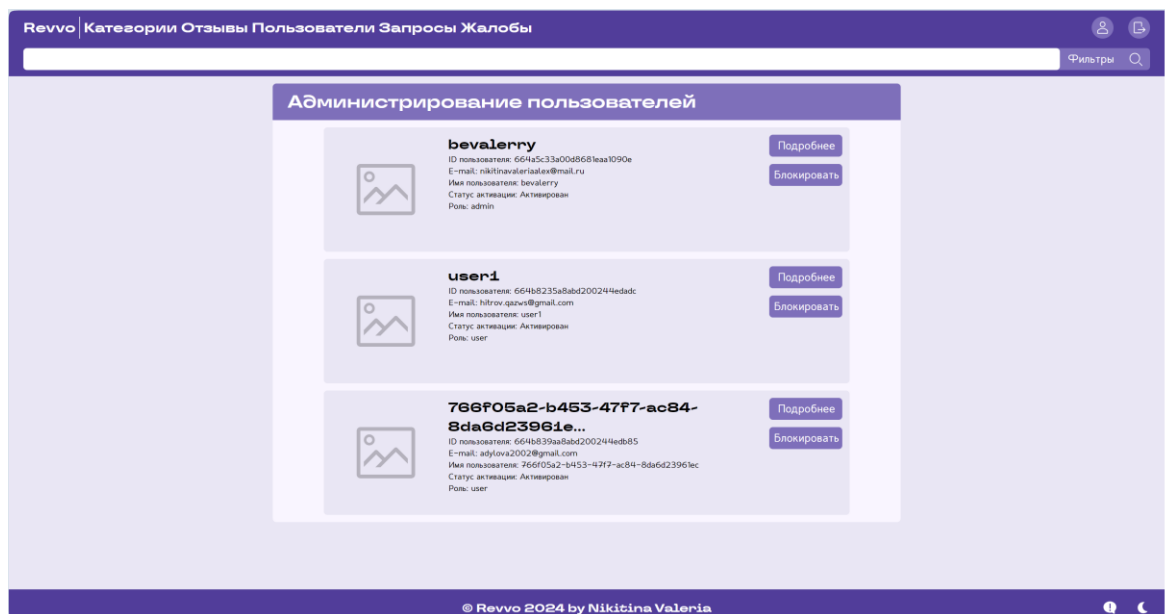


Рисунок 6.9 – Страница для администратора со списком пользователей
На Рисунке 6.10 показано окошко для написания жалобы на отзыв.

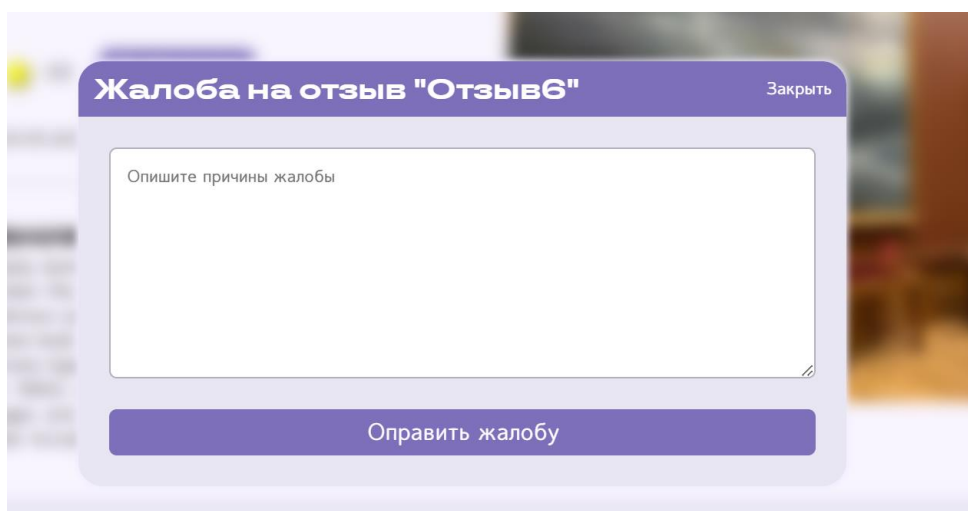


Рисунок 6.10 – Окошко для написания жалобы

На Рисунке 6.11 показан страница обработки заявки администратором.

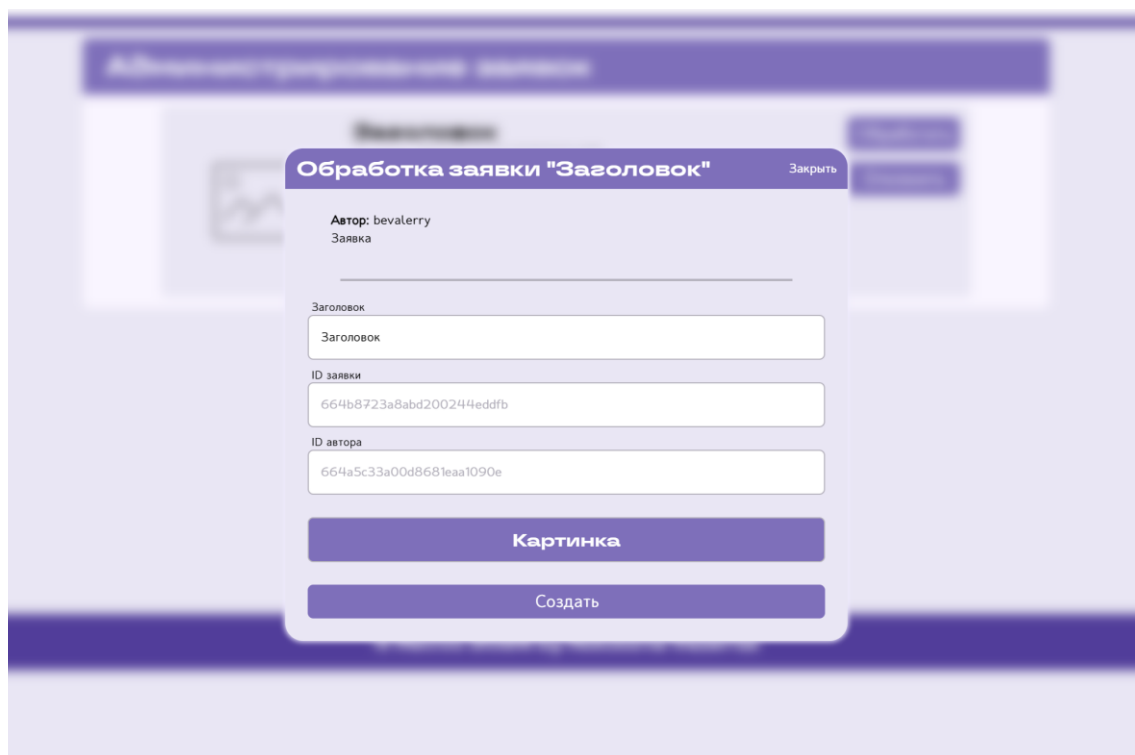


Рисунок 6.11 – Страница обработки заявки для администратора

Вывод к главе 6

В этой главе был успешно создан интуитивно понятный и функциональный пользовательский интерфейс. Для этого применялись инструменты и библиотеки, которые обеспечили современный и удобный дизайн, а также плавную и отзывчивую работу интерфейса.

7 ТЕСТИРОВАНИЕ

Были проведены тесты по работе приложения, продемонстрированные на Рисунках 7.1 – 7.7.

Для регистрации и аутентификации создана проверка подлинности, приходящая на электронную почту (Рисунок 7.1-7.2).

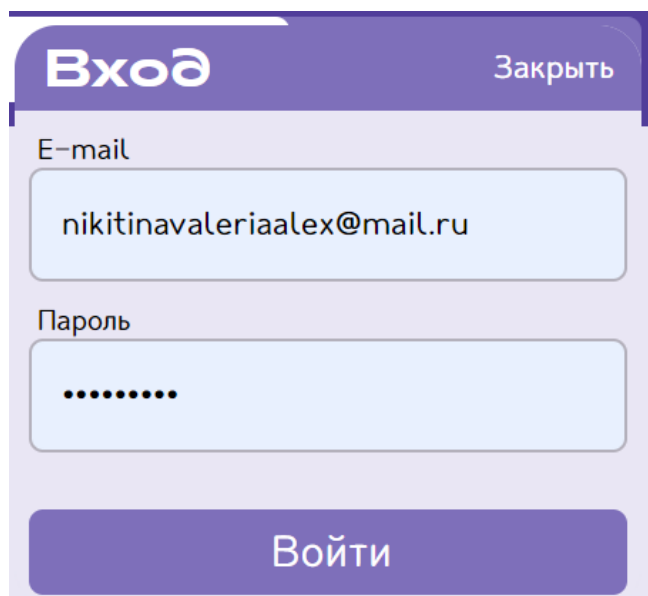



Рисунок 7.1 – Ввод данных для входа

Пройдите аутентификацию)

-  bevalerry@gmail.com Сегодня, 13:21
Кому: вам

Для аутентификации перейдите по ссылке

обновите страницу если аутентифицируетесь с другого устройства

[Активировать](#)

Рисунок 7.2 – Письмо, пришедшее на почту для активирования сеанса

На следующих рисунках проверим удаление отзывов. Результат удаления представлен на Рисунке 7.4.

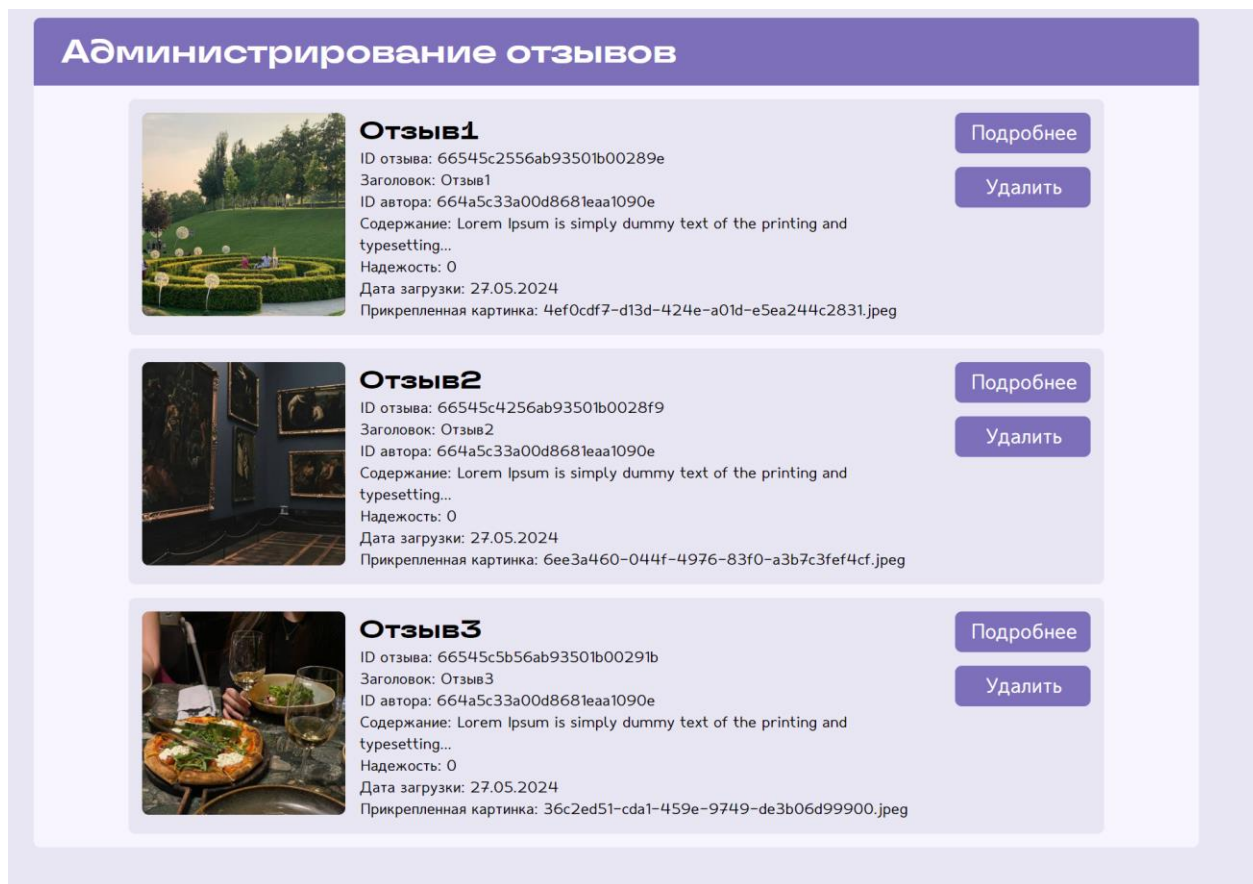


Рисунок 7.3 – Список отзывов до удаления

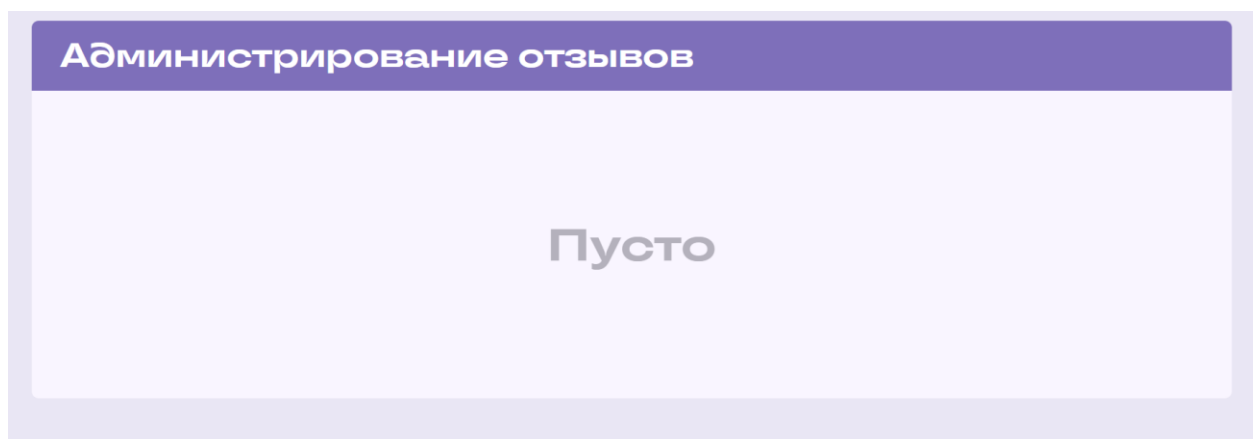


Рисунок 7.4 – Список после удаления

Далее проверим возможность изменения имени пользователя.

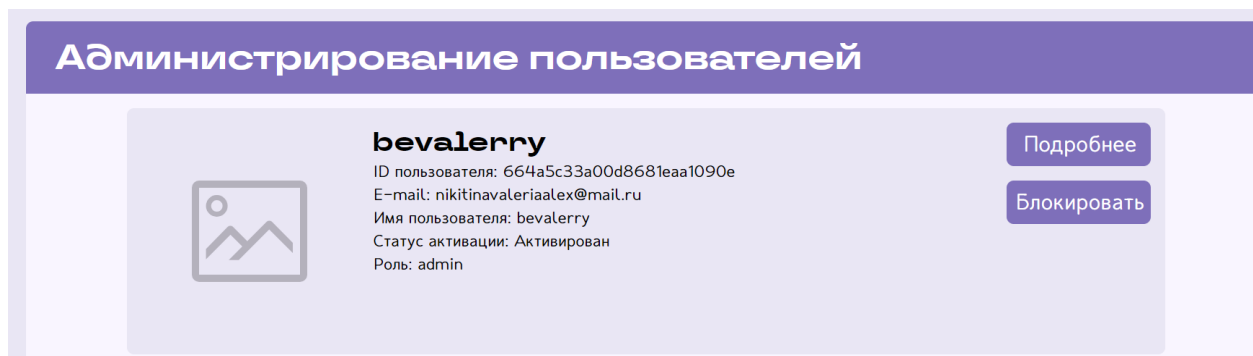


Рисунок 7.5 – Начальные данные пользователя

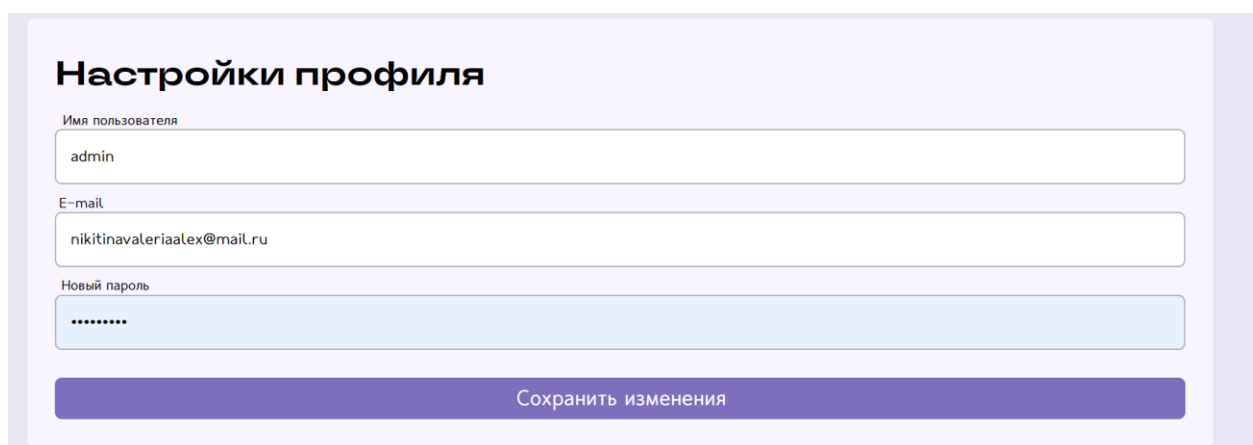


Рисунок 7.6 – Работа с настройками профиля

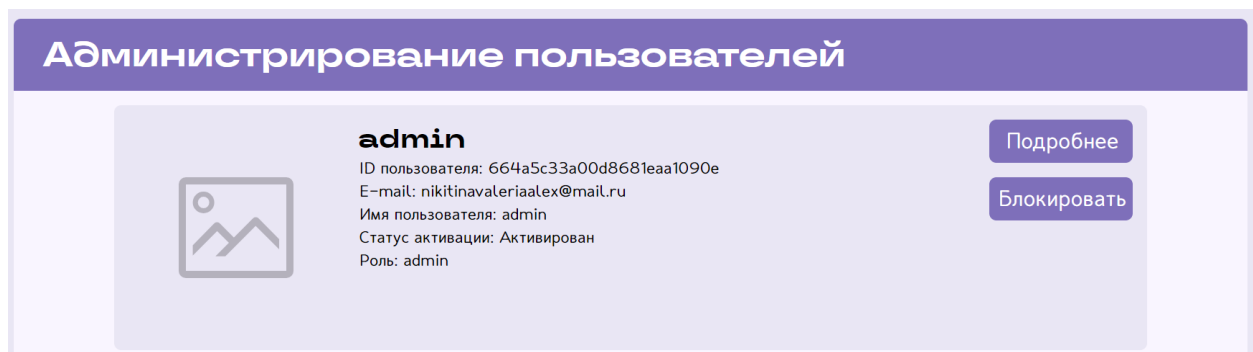


Рисунок 7.7 – Данные пользователя после сохраненных изменений

Вывод к главе 7

Проведя ручное тестирование, мы можем быть уверены, что основные функции веб-сайта работают должным образом и соответствуют требованиям. Результаты тестирования подтвердили надежность и функциональность сайта.

8 РАЗВЕРТЫВАНИЕ ПРИЛОЖЕНИЯ

8.1 Выбор платформы для развертывания

Развертывание приложения — это процесс подготовки и запуска приложения в реальной рабочей среде.

Для успешного развертывания интернет-ресурса была выбрана платформа Render [20], предлагающая удобные инструменты для развертывания веб-приложений. Этот выбор обусловлен простотой использования, интеграцией с GitHub [5][14][19] для автоматического развертывания из репозитория и масштабированием под нагрузкой. Render является универсальным решением для развертывания приложений различных типов.

8.2 Развертывание серверной и клиентской частей

Для развертывания сервисной и клиентской частей был указан репозиторий с исходным кодом и написаны необходимые команды для сборки и запуска (Рисунок 8.1 – 8.2).

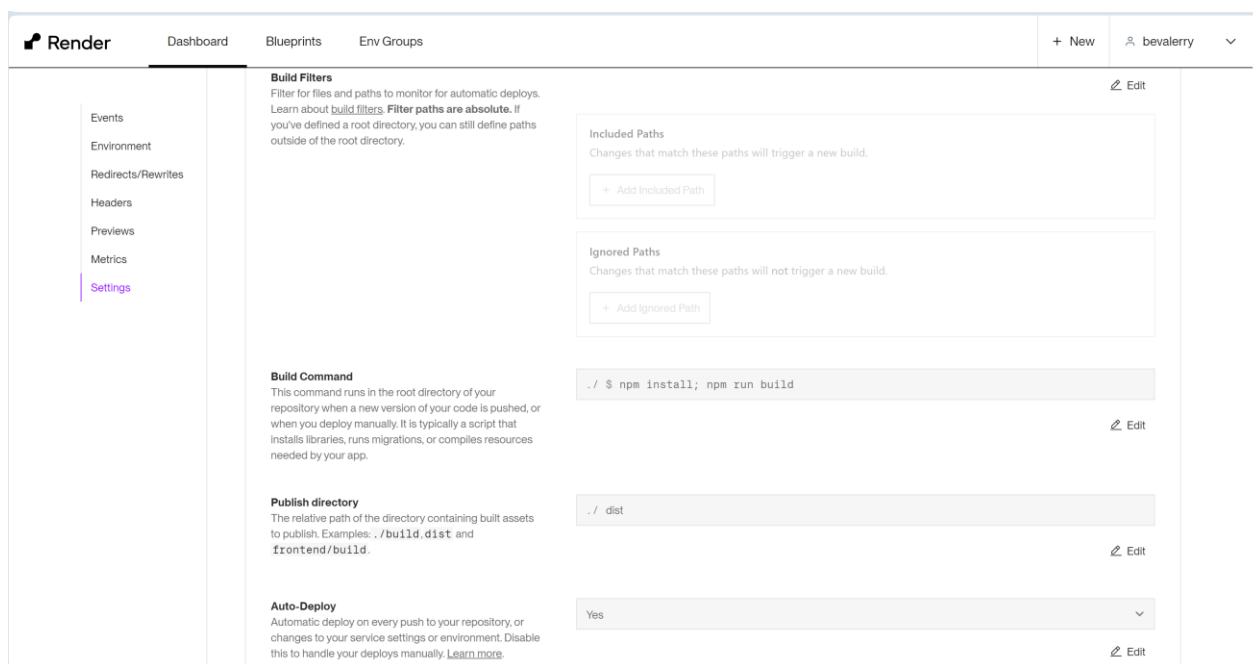


Рисунок 8.1 – Настройки развертывания клиентской части

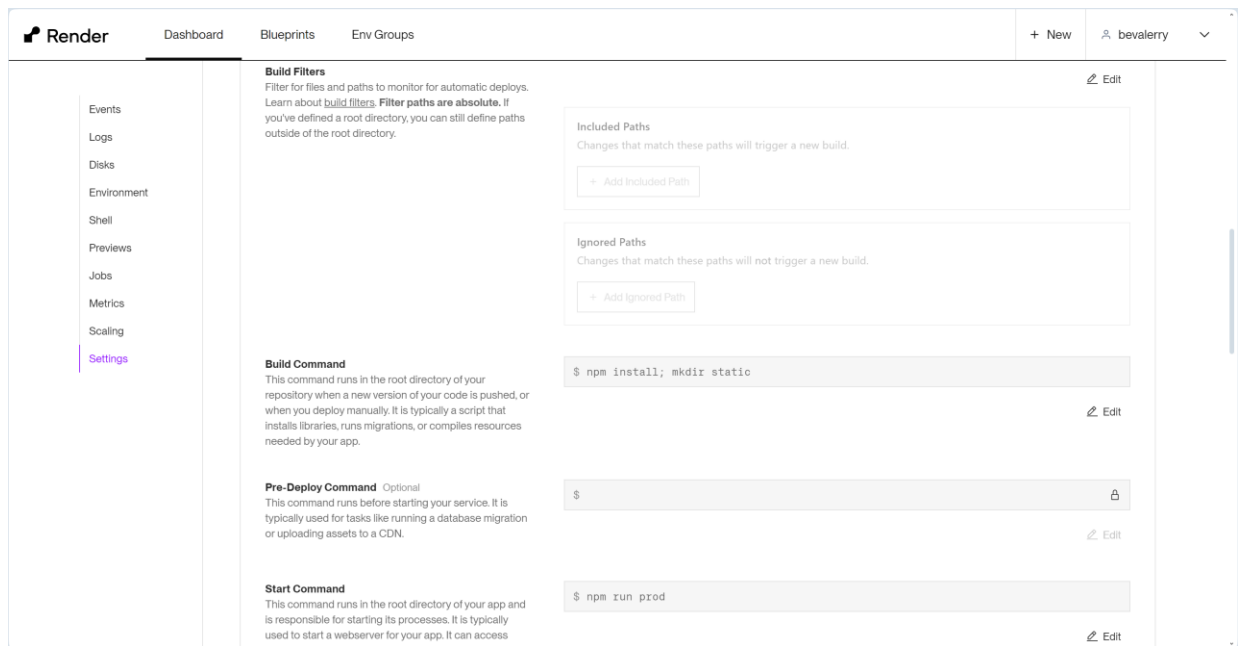


Рисунок 8.2 – Настройки развертывания серверной части

На рисунках 8.3 – 8.4 изображена успешность развертывания.

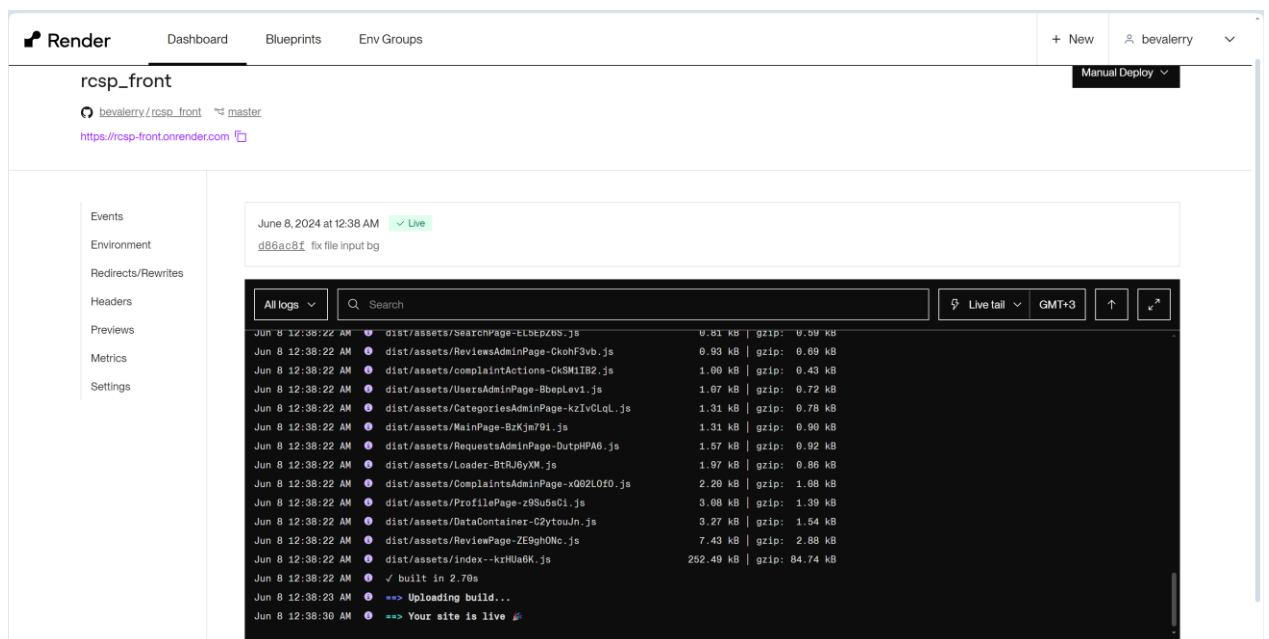


Рисунок 8.3 – Логи развертывания клиентской части

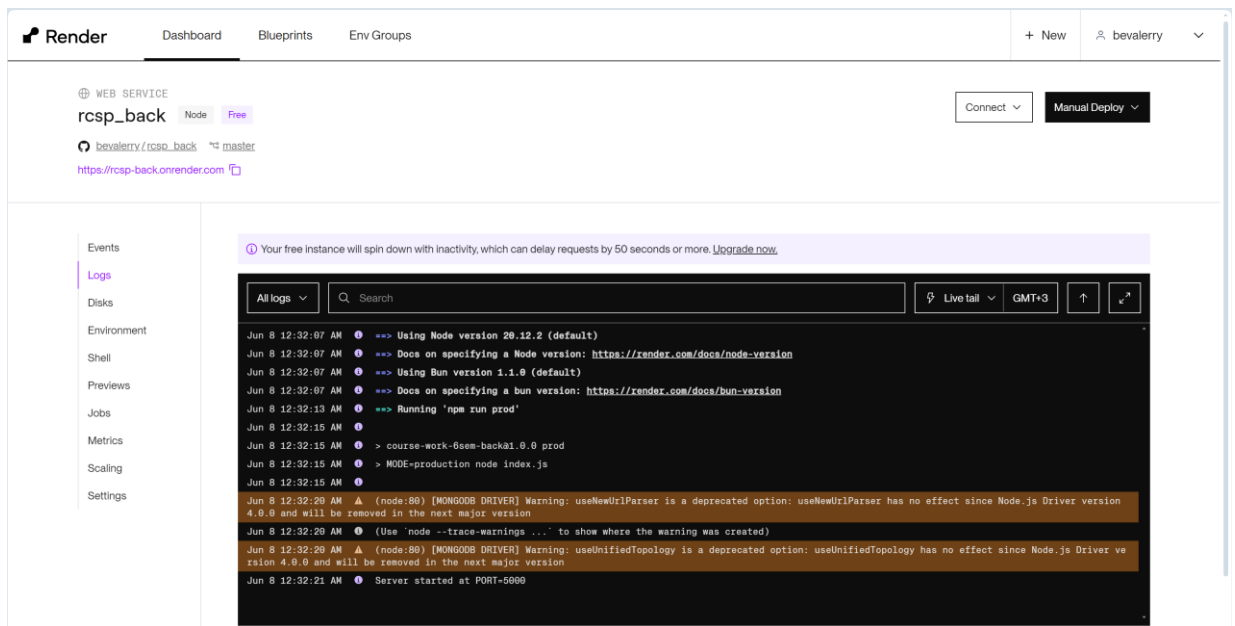


Рисунок 8.4 – Логи развертывания серверной части

Вывод к главе 8

Развертывание на платформе Render прошло успешно и без каких-либо ошибок. Это свидетельствует о надежном и эффективном процессе развертывания, который гарантирует бесперебойную доставку веб-сайта пользователям. Благодаря успешному развертыванию сайт готов к использованию и может обеспечить пользователям беспрепятственный и надежный доступ к функциям и контенту.

ЗАКЛЮЧЕНИЕ

Для успешного завершения проекта были выполнены следующие шаги:

1. Глубокое изучение предметной области разрабатываемого веб-приложения.
2. Обоснованный выбор технологий, необходимых для создания веб-приложения.
3. Разработка архитектуры веб-приложения на основе выбранного паттерна проектирования.
4. Реализация серверной логики веб-приложения с использованием соответствующей технологии.
5. Создание слоя логики базы данных для управления данными приложения.
6. Разработка клиентского представления веб-приложения, обеспечивающего взаимодействие с пользователем.

В результате выполнения проекта достигнуты все поставленные цели. Разработан веб-сервис для сбора отзывов пользователей с использованием трехслойной архитектуры, паттерна MVC, и модели клиент-серверного взаимодействия REST. Также проведено ручное тестирование приложения с помощью пользовательского веб-интерфейса, написанного на ReactJS.

Отчет подготовлен в соответствии с инструкциями по курсовому проектированию и ГОСТ 7.32. Дополнительно создан демонстрационный материал в виде презентации. Для проверки оригинальности работы использован сервис антиплагиата.

Исходный код серверной части приложения доступен по ссылке:
https://github.com/bevalerry/rcsp_back

Исходный код клиентской части приложения доступен по ссылке:
https://github.com/bevalerry/rcsp_front

Приложение развернуто на сервисе render.com и доступно по ссылке:
<https://rcsp-front.onrender.com>

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. React: Быстрый старт. Практическое руководство по созданию веб-приложений на React.js (Дэвид Томас, Адам Крокфорд, 2020)
2. Redux: Руководство для разработчиков (Дэн Абрамов, 2020)
3. Node.js в действии (Илья Кантор, 2019)
4. MongoDB: Руководство разработчика (Кристина Чодоров, Майкл Монгалло, 2018)
5. Git для программистов (Скотт Чакон, Бен Страуб, 2019)
6. 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: 01.04.2024).
7. React [Электронный ресурс] – URL: <https://reactjs.org/> (дата обращения 12.05.2024).
8. Redux [Электронный ресурс] – URL: <https://redux.js.org/> (дата обращения 27.04.2024).
9. Node.js [Электронный ресурс] – URL: <https://nodejs.org/en/> (дата обращения 08.05.2024).
10. Express.js [Электронный ресурс] – URL: <https://expressjs.com/> (дата обращения 07.05.2024).
11. MongoDB [Электронный ресурс] – URL: <https://www.mongodb.com/> (дата обращения 10.04.2024).
12. Хоффман Эндрю X85 Безопасность веб-приложений. — СПб.: Питер, 2021. — 336 с.: ил. — (Серия «Бестселлеры O'Reilly»).
13. Vite [Электронный ресурс] – URL: <https://vitejs.dev/> (дата обращения 15.05.2024).
14. GitHub [Электронный ресурс] – URL: <https://docs.github.com/en> (дата обращения 15.05.2024).
15. Node.js для начинающих [Электронный ресурс] – URL:

<https://nodejs.org/ru/docs/guides/getting-started-guide/> (дата обращения 03.04.2024).

16. Express.js для начинающих [Электронный ресурс] – URL: <https://expressjs.com/ru/starter/hello-world.html> (дата обращения 12.04.2024).

17. MongoDB для начинающих [Электронный ресурс] – URL: <https://docs.mongodb.com/manual/tutorial/> (дата обращения 02.03.2024).

18. Vite для начинающих [Электронный ресурс] – URL: <https://vitejs.dev/guide/> (дата обращения 12.05.2024).

19. Git для начинающих [Электронный ресурс] – URL: <https://git-scm.com/book/ru/v2> (дата обращения 29.04.2024).

20. Render [Электронный ресурс] – URL: <https://render.com/> (дата обращения 02.05.2024).