Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ

УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

Специализация 1-40 05 01 03«Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОЙ РАБОТЫ**

по дисциплине «Программирование в Internet».

Тема: Web-приложение «Музыкальный стриминговый сервис»

Исполнитель

студент 4 курса 1 группы

(подпись) (инициалы, фамилия)

Руководитель

ассистент

(подпись) (инициалы, фамилия)

Курсовая работа защищена с оценкой

Руководитель

ассистент

(подпись) (инициалы, фамилия)

Минск 2023

**Содержание**

[Введение 5](#_Toc154148779)

[1 Обзор аналогичных решений и постановка задач. 6](#_Toc154148780)

[1.1 Обзор аналогичных решений 6](#_Toc154148781)

[1.1.1 Spotify 6](#_Toc154148782)

[1.1.2 Яндекс Музыка 8](#_Toc154148783)

[1.1.3 YouTube Music 9](#_Toc154148784)

[1.2 Постановка задач 10](#_Toc154148785)

[1.3 Актуальность задачи 11](#_Toc154148786)

[1.4 Вывод по разделу 11](#_Toc154148787)

[2 Проектирование программного средства 12](#_Toc154148788)

[2.1 Выбор средств реализации программного средства 12](#_Toc154148789)

[2.1.1 Выбор средств реализации для клиентской части 12](#_Toc154148790)

[2.1.2 Выбор средств реализации для серверной части 13](#_Toc154148791)

[2.1.3 Выбор средств реализации для серверной части 14](#_Toc154148792)

[2.2 Архитектура приложения 14](#_Toc154148793)

[2.3 Диаграмма вариантов использования 15](#_Toc154148794)

[2.4 Проектирование базы данных 17](#_Toc154148795)

[2.5 Проектирование основных алгоритмов 18](#_Toc154148796)

[2.6 Вывод по разделу 19](#_Toc154148797)

[3 Разработка программного средства 20](#_Toc154148798)

[3.1 Разработка серверной части 20](#_Toc154148799)

[3.2 Разработка клиентской части 23](#_Toc154148800)

[3.3 Контейнеризация 25](#_Toc154148801)

[3.4 Вывод по разделу 27](#_Toc154148802)

[4 Тестирование веб-приложения 28](#_Toc154148803)

[4.1 Вывод по разделу 31](#_Toc154148804)

[5 Руководство пользователя 32](#_Toc154148805)

[5.1 Методика использования приложения пользователем 32](#_Toc154148806)

[5.2 Методика использования приложения администратором 35](#_Toc154148807)

[5.3 Вывод по разделу 38](#_Toc154148808)

[Заключение 39](#_Toc154148809)

[Список использованных источников 40](#_Toc154148810)

[Приложение А 41](#_Toc154148811)

[Приложение Б 42](#_Toc154148812)

Введение

В современном мире музыкальная индустрия пережила глубокие и радикальные изменения, в значительной степени благодаря стремительному развитию информационных технологий и распространению интернета. Эти изменения касаются как потребителей музыки, так и самих музыкантов, и они изменили способ, которым мы воспринимаем, создаем и распространяем музыку.

Доступ к музыке стал более универсальным и мгновенным, чем когда-либо. С появлением веб-приложений и музыкальных стриминговых платформ, люди теперь имеют неограниченный доступ к миллионам композиций из разнообразных музыкальных жанров и исполнителей. Слушатели могут просто открыть мобильное устройство или компьютер, подключиться к интернету и начать прослушивать музыку по своему выбору. Это предоставляет им свободу наслаждаться музыкой в любое время и в любом месте, будь то в дороге, дома, в спортзале или на работе.

С другой стороны, для музыкантов современные технологии создали новые возможности. Они могут легко загружать свои композиции на цифровые платформы, добираясь до аудитории по всему миру без необходимости заключения крупных музыкальных контрактов. Это означает, что начинающие и независимые артисты могут самостоятельно продвигать свои произведения и получать обратную связь от слушателей. Благодаря социальным сетям и стриминговым сервисам, они также могут легко взаимодействовать с публикой, строить свою базу поклонников и даже финансироваться через краудфандинговые платформы.

Целью данного курсового проекта является создание полноценного веб-приложения для музыкальной платформы, отражающей современные тенденции в музыкальной индустрии. Проект направлен на разработку серверной и клиентской частей приложения, включая базу данных, обеспечивающую хранение информации о пользователях, артистах, альбомах, треках и плейлистах. Основной задачей является обеспечение удобного взаимодействия пользователей с сервисом, предоставление им мгновенного доступа к множеству музыкальных композиций.

Конечным результатом проекта должно быть полностью функциональное веб-приложение, готовое к использованию, с интуитивно понятным интерфейсом и высокой степенью взаимодействия между пользователями и музыкальным контентом.

# Обзор аналогичных решений и постановка задач.

Одним из необходимых этапов разработки программного обеспечения является обзор и анализ аналогов. В ходе выполнения данного курсового проекта были рассмотрены музыкальные стриминговые сервисы Spotify, Яндекс Музыка и YouTube Music.

# Обзор аналогичных решений

# Spotify

Spotify – это онлайн-платформа для потоковой передачи музыки, которая предоставляет доступ к миллионам песен различных жанров и исполнителей со всего мира [1]. Она была основана в 2006 году в Швеции и стала одной из самых популярных музыкальных платформ в мире.

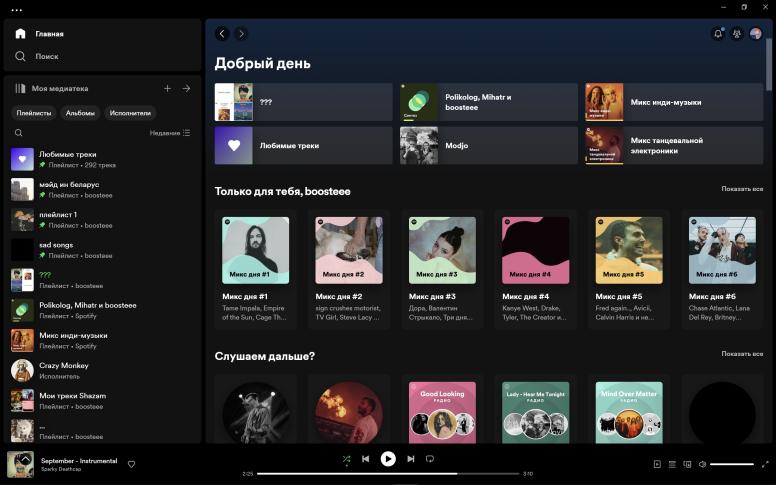


Рисунок 1.1 – Главная страница Spotify

Рассмотрим главную страницу приложения. Слева располагается меню навигации и включает в себя разделы «Главная», «Поиск», «Моя медиатека» и «Любимые треки». Там же расположены пользовательские плейлисты и любимые исполнители пользователя. Главная страница предлагает пользователю персонализированные рекомендации на основе его музыкальных предпочтений. Здесь могут быть представлены плейлисты, альбомы, и треки, которые могут заинтересовать пользователя.

На странице «Поиск» пользователь может найти музыку и артистов, а также других пользователей и плейлисты. Здесь можно искать по ключевым словам, названиям треков, артистам и даже по жанрам.

В нижней части интерфейса приложения находятся элементы управления воспроизведением, такие как информация о текущем треке, кнопки воспроизведения, переключения на следующий и предыдущий трек, регулировка громкости и добавление треков в свои плейлисты.

По желанию пользователь может перейти на страницу исполнителя. Там он может посмотреть информацию о музыканте, его альбомы и популярные треки.

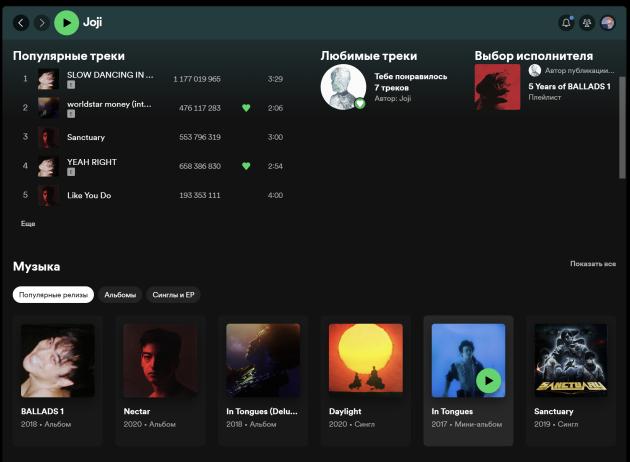


Рисунок 1.2 – Страница исполнителя

Также рассмотрим возможности пользователя в контексте управления треками. Нажав правой кнопкой мыши по треку, пользователь откроет контекстное меню, при помощи которого сможет перейти к исполнителю трека, альбому; добавить/удалить трек в «Любимые» или свой плейлист.

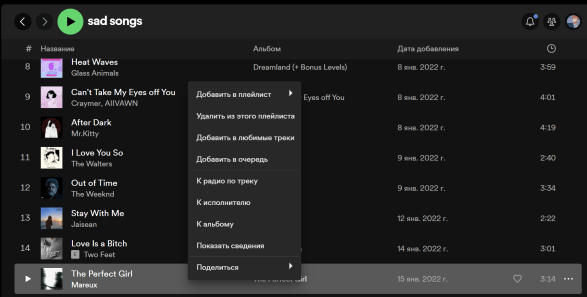


Рисунок 1.3 – Контекстное меню трека

По итогу можно сказать, что Spotify разработан с акцентом на легкость использования и обеспечивает широкие возможности для нахождения и прослушивания музыки. Его интерфейс обеспечивает комфортное взаимодействие пользователей с сервисом.

# Яндекс Музыка

Яндекс Музыка – это российский музыкальный стриминговый сервис, предоставляющий доступ к широкому каталогу музыки, плейлистам и подкастам [2].

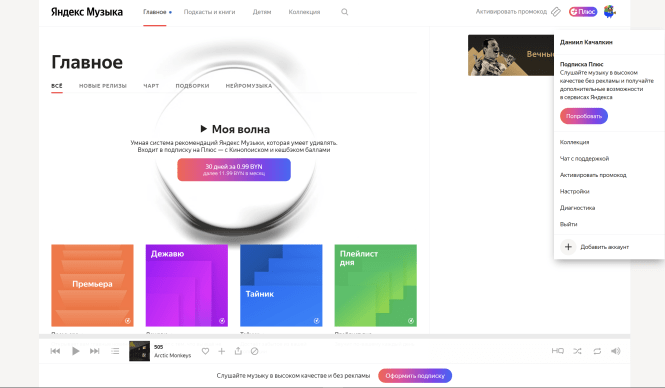


Рисунок 1.4 – Главная страница сервиса Яндекс Музыка

Попадая на главную страницу приложения, пользователь наблюдает следующую картину: сверху находится поисковая строка и меню навигации с разделами «Главное», «Подкасты и книги», «Детям», «Коллекция». Основное содержимое страницы включает в себя плейлисты с персональными рекомендациями, недавно прослушанные треки, новинки, популярные плейлисты и чарты вашей страны.

Внизу, аналогично Spotify, расположен функциональный плеер. С помощью него можно переключать треки, увеличивать/уменьшать громкость трека, добавить трек в плейлист или избранное, выбирать качество воспроизведения.

Теперь рассмотрим раздел с настройками. Здесь можно изменить вид плеера, поменять тему приложения, предоставить публичный доступ к своей фонотеке. Возможности редактирования профиля у пользователей нет, так как Яндекс Музыка тесно интегрирована с другими сервисами Яндекса (редактировать можно общий аккаунт).

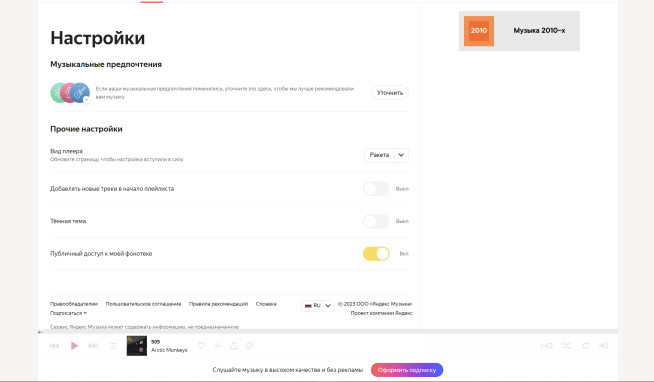


Рисунок 1.5 – Настройки Яндекс Музыки

По итогу можно сказать, что Яндекс Музыка имеет интуитивно понятный интерфейс, а в плане функционала этот сервис не сильно отличается от Spotify, рассмотренного ранее.

# YouTube Music

YouTube Music – музыкальной стриминговый сервис, разработанный компанией Google [3]. Он пришел на смену Google Play Music. В приложении YouTube Music можно следить за творчеством любимых музыкантов, слушать последние хиты и подкасты, а также открывать новых исполнителей. Сервис доступен на разных устройствах.

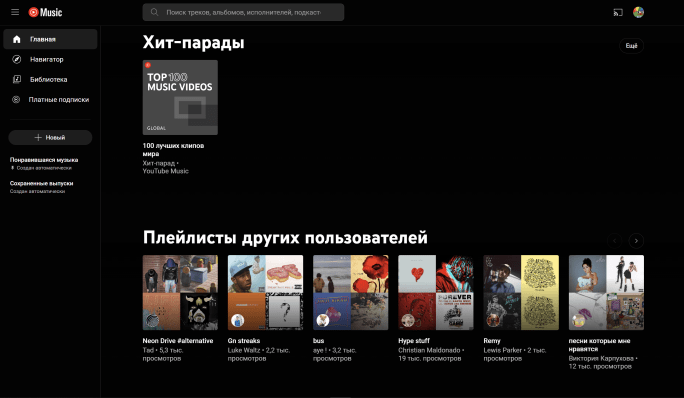


Рисунок 1.6 – Главная страница сервиса YouTube Music

В приложении есть три вкладки. Найти трек, исполнителя, альбом или подкаст можно через поиск или на этих трех вкладках:

Главная. Здесь вы увидите персональные рекомендации и радиостанции с учетом истории прослушивания, а также подборки под любое настроение и для разных занятий. Если вы войдете в аккаунт, то также будут учитываться треки и подкасты, которые вы слушаете на YouTube.

Семплы. Это лента с фрагментами музыкальных видео, в которой вы наверняка найдете для себя что-то новое.

Навигатор. На этой вкладке собраны новые треки в сервисе YouTube Music, а также плейлисты под настроение.

Библиотека. Здесь можно сохранять треки, альбомы, плейлисты и подкасты. У подписчиков YouTube Music Premium в этом разделе также представлен скачанный контент и недавно воспроизведенные песни и плейлисты.

В верхней части приложения расположен поиск, с помощью которого можно найти исполнителей, треки, альбомы, плейлисты и подкасты. В нижней части приложения расположен плеер для управления воспроизведением музыкальных композиций.



Рисунок 1.7 – Музыкальный плеер YouTube Music

Делая вывод, можно сказать, что функционал как YouTube Music, Spotify и Яндекс Музыки, так и других стриминговых музыкальных сервисов в целом схож. Основные отличия проявляются в системе рекомендаций и пользовательском интерфейсе.

# Постановка задач

Основная задача данного курсового проекта – разработка современного музыкального стримингового сервиса, который предоставит пользователям доступ к обширной музыкальной библиотеке, возможность создания персонализированных плейлистов, а также удобные инструменты для прослушивания музыки.

Функционально должны быть выполнены следующие задачи:

* обеспечивать возможность регистрации и авторизации;
* обеспечивать реализацию двух ролей: администратор и пользователь;
* предоставлять возможность пользователю прослушивать музыкальные композиции;
* позволять пользователю редактировать свой профиль;
* предоставлять возможность пользователю создавать, редактировать и удалять плейлисты, добавлять и удалять из них треки;
* предоставлять пользователю возможность осуществлять поиск музыкальных композиций;
* позволять администратору добавлять и удалять исполнителей, альбомы исполнителей, треки;
* обеспечивать возможность администратору ограничивать доступ пользователю к сервису.

# Актуальность задачи

Актуальность задачи разработки современного музыкального стримингового сервиса подчеркивается рядом факторов.

Во-первых, изменения в потребительском поведении в сфере музыки. С появлением универсального доступа к интернету пользователи стремятся получить мгновенный и неограниченный доступ к огромной библиотеке музыкальных композиций.

Во-вторых, важность персонализации и удобства использования. Пользователи ожидают не только доступа к обширной библиотеке, но и возможности создавать персонализированные плейлисты, редактировать свои профили и настраивать воспроизведение музыки под свои предпочтения. Эти функциональные возможности обеспечивают не только удовлетворение потребительских запросов, но и создание уникального пользовательского опыта.

Таким образом, разработка современного музыкального стримингового сервиса не только соответствует текущим трендам в потребительском поведении, но и отвечает потребности как слушателей, так и музыкантов, создавая уникальное пространство для взаимодействия и распространения музыкального контента.

# Вывод по разделу

В данном разделе были рассмотрены три популярных музыкальных стриминговых сервиса: Spotify, Яндекс Музыка и YouTube Music. Все три сервиса обеспечивают базовые функции поиска, воспроизведения и управления музыкой, но имеют свои уникальные особенности. Подводя итог, нужно отметить, что выбор музыкального стримингового сервиса зависит от предпочтений пользователя в интерфейсе, системе рекомендаций и интеграции с другими сервисами. Каждый из рассмотренных сервисов имеет свои преимущества, и конечный выбор будет зависеть от индивидуальных потребностей и предпочтений пользователей.

# Проектирование программного средства

## **Выбор средств реализации программного средства**

### Выбор средств реализации для клиентской части

React JS – это библиотека JavaScript для создания пользовательских интерфейсов, разработанная Facebook. React позволяет разработчикам создавать большие веб-приложения, которые могут изменять данные без перезагрузки страницы. Она использует синтаксис HTML-in-JavaScript, называемый JSX (JavaScript и XML). Это позволяет разработчикам писать HTML-подобный код прямо в JavaScript. Однако, стоит отметить, что использование JSX не является обязательным, и можно использовать React без JSX, но это усложнит код и сделает его менее читаемым.

React основывается на компонентах. Компонент – это самодостаточная часть пользовательского интерфейса, которая может быть повторно использована. Они могут быть функциональными или классовыми. Функциональные компоненты – это простые JavaScript-функции, которые принимают свойства (props) и возвращают React-элементы. Классовые компоненты – это более сложные и могут содержать состояние (state), методы жизненного цикла и другие функции.

Библиотека также использует концепцию "однонаправленного потока данных". Это означает, что данные в приложении должны постоянно потоковаться в одном направлении, от родительских компонентов к дочерним. Это упрощает отслеживание изменений в данных и помогает избегать ошибок.

React также предоставляет инструменты для управления состоянием приложения, такие как хуки (hooks) и контекст (context). Хуки позволяют использовать состояние и другие функции React без написания классовых компонентов. Контекст позволяет передавать данные через дерево компонентов без необходимости передавать пропсы вручную на каждом уровне.

Важно отметить, что React – это только библиотека для создания пользовательских интерфейсов, и она не предоставляет решений для других аспектов веб-разработки, таких как маршрутизация или управление состоянием приложения на уровне сервера. Для этих задач часто используются дополнительные библиотеки, такие как React Router для маршрутизации и Redux или Context API для управления состоянием приложения.

Tailwind CSS – это современный, низкоуровневый CSS-фреймворк, созданный для быстрого и удобного создания пользовательских интерфейсов.

Основная идея Tailwind CSS заключается в предоставлении разработчикам набора утилитарных классов, которые можно использовать для быстрой и простой стилизации элементов на странице. Вместо того чтобы писать традиционный CSS, разработчики могут просто добавлять классы к HTML-элементам для применения стилей.

Tailwind предоставляет инструменты для создания адаптивных дизайнов. Вы можете использовать классы, которые применяются только на определенных размерах экрана, чтобы легко создавать отзывчивые дизайны прямо в HTML.

Фреймворк автоматически удаляет все неиспользуемые CSS при сборке для продакшена, что означает, что ваш окончательный пакет CSS будет как можно меньше. Фактически, большинство проектов Tailwind отправляют менее 10kB CSS на клиент.

Tailwind также поддерживает все последние и лучшие CSS-функции, чтобы сделать опыт разработчика максимально приятным. У него есть встроенная поддержка CSS Grid, композируемые преобразования и градиенты, поддержка современных селекторов состояния и многое другое.

### Выбор средств реализации для серверной части

NestJS – это серверный фреймворк Node.js, который используется для создания эффективных, надежных и масштабируемых приложений. Он был вдохновлен Angular и разработан с использованием TypeScript, что делает его совместимым с большинством middleware Express.

Основные строительные блоки NestJS включают модули, контроллеры и сервисы.

Модули: Модули в NestJS позволяют группировать связанные файлы. Они представляют собой файлы, декорированные декоратором @Module. Этот декоратор предоставляет метаданные, которые Nest использует для организации структуры приложения.

Контроллеры: Контроллеры в NestJS отвечают за обработку входящих запросов и возвращение ответов на клиентскую сторону приложения. Для определения контроллера в NestJS создается файл и включается декоратор @Controller().

Сервисы: Сервисы в NestJS представляют собой классы с специальным декоратором @Injectable() в верхней части. Они предназначены для абстракции любой формы сложности и логики. Сервис-провайдер в NestJS может быть использован для создания и возвращения всех пользователей.

NestJS также предоставляет инструменты для создания RESTful API, GraphQL приложений, MVC приложений, WebSockets и многого другого. Он поддерживает и предоставляет большое количество модулей, поддерживаемых сообществом, которые можно использовать для создания любой специфической функции вашего выбора, от концепций, таких как TypeORM, Mongoose и GraphQL, валидации, кеширования, WebSockets и многого другого.

NestJS также поддерживает TypeScript, который обеспечивает проверку типов и упрощает написание поддерживаемых приложений, предоставляя ошибки и предупреждения компиляции. Это делает его хорошо интегрированным в VSCode для доступной среды разработки.

Важно отметить, что NestJS – это не просто фреймворк для создания веб-API, он предоставляет полноценную архитектуру приложения, которая позволяет разработчикам и командам создавать высокопроизводительные, масштабируемые, слабосвязанные и легко поддерживаемые приложения.

### Выбор средств реализации для серверной части

MongoDB – это обобщенная, многофункциональная и мощная документоориентированная NoSQL база данных, которая предоставляет эффективное и гибкое хранилище для различных типов наборов данных.

В MongoDB, вместо таблиц, используемых для определения структуры набора данных в традиционных реляционных базах данных SQL (например, MySQL, PostgreSQL), мы имеем коллекции и текстовые документы, содержащие пары ключ-значение. Коллекция содержит номер документов, а документ содержит поля данных. Поля данных могут быть произвольного числа и типа, даже в рамках одной и той же коллекции. По сути, каждый документ MongoDB следует формату Javascript Object Notation (JSON).

MongoDB может обрабатывать динамические модели данных и схемы, которые могут легко хранить неструктурированные наборы данных. В случае изменчивых нагрузок данных, она может масштабироваться горизонтально, что является крайне экономически эффективным по сравнению с традиционными SQL базами данных, которые обычно могут масштабироваться только вертикально. Также эта БД поддерживает динамические запросы документов, используя язык запросов на их основе, практически не уступающий в эффективности SQL. MongoDB не требует преобразования объектов приложения в объекты базы данных. СУБД использует внутреннюю память для хранения рабочего набора, обеспечивая ускоренный доступ к данным.

## **Архитектура приложения**

Чтобы приступить к реализации проекта, необходимо определится с его архитектурой. Для данного проекта был выбран архитектурный подход Single Page Application (SPA). SPA представляет собой тип веб-приложения, в котором весь контент и код загружаются однократно в начале сеанса, а затем динамически обновляются по мере взаимодействия пользователя с приложением. Это позволяет избежать полной перезагрузки страницы при каждом запросе. Основные характеристики SPA:

* Одностраничность: Весь интерфейс приложения загружается однократно.
* Асинхронная загрузка данных: Запросы данных выполняются асинхронно, обеспечивая более быстрый отклик приложения.
* Маршрутизация на клиенте: Управление навигацией осуществляется на стороне клиента без перезагрузки страницы.

Преимущества SPA:

* Быстродействие: SPA обеспечивает более быстрый пользовательский опыт благодаря асинхронной загрузке данных и отсутствию полной перезагрузки страницы.
* Оптимизированный пользовательский опыт: Пользователи могут взаимодействовать с приложением более естественным образом, поскольку не сталкиваются с перезагрузкой страницы при каждом действии
* Меньший объем передаваемых данных: SPA обычно передает только данные, необходимые для обновления контента, что снижает объем трафика и ускоряет загрузку.

Для взаимодействия внутри приложения была выбрана монолитная клиент-серверная архитектура. Монолитная клиент-серверная архитектура веб-приложений представляет собой структуру, в которой все компоненты приложения (клиентский и серверный код) находятся внутри одного проекта. Это означает, что клиентский и серверный код тесно связаны между собой.

В контексте веб-приложений, клиентская часть обычно отвечает за большую часть того, что видит пользователь. Это включает в себя реализацию механизма пользовательского взаимодействия (нажатие кнопок, ввод текста и т.д.), структуру веб-страницы (определяется с помощью HTML), внешний вид веб-страницы (определяется с помощью CSS), и взаимодействие с пользователем (определяется с помощью JavaScript).

Серверная часть обычно обрабатывает запросы от клиентской части, включая обработку данных, взаимодействие с базой данных и отправку ответов обратно на клиент.

В монолитной архитектуре, все приложение развертывается как единое целое. Любые изменения или обновления требуют повторного развертывания всего приложения, что ведет к увеличению времени простоя и возможным сбоям.

Однако, монолитная архитектура имеет свои преимущества. Например, она требует меньше планирования на начальном этапе, но сложность понимания и поддержки постепенно растет. Также, весь путь кода можно отслеживать в одной среде, что упрощает процесс отладки.

## **Диаграмма вариантов использования**

Диаграмма вариантов использования показывает все возможные варианты использования приложения и взаимодействия пользователей с различными ролями. Данная диаграмма представлена на рисунке 2.1.

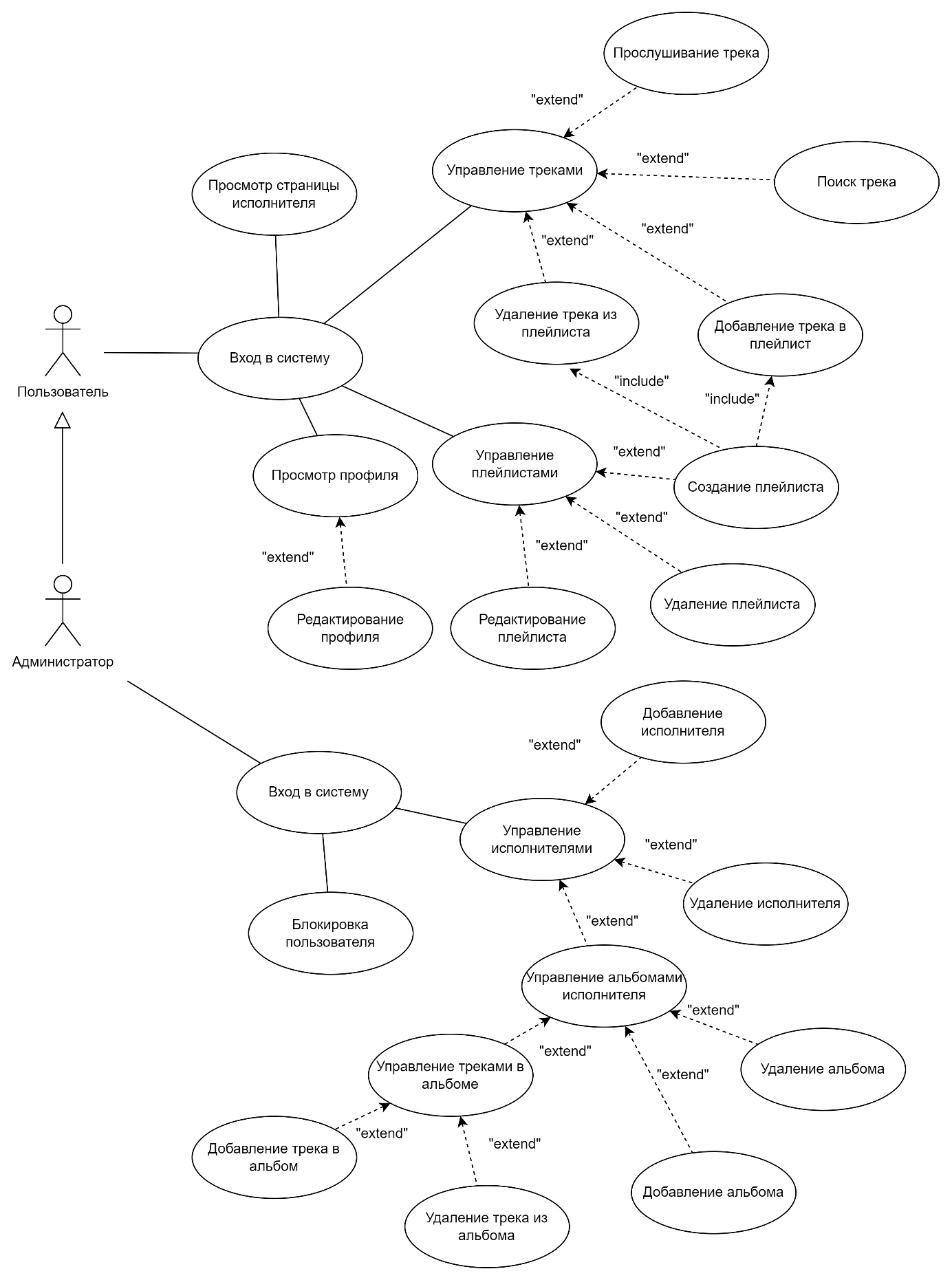


Рисунок 2.1 – Диаграмма вариантов использования

Как видно из рисунков, в проекте будут реализованы две роли: администратор и пользователь. Пользователь может искать и прослушивать музыкальные композиции, создавать и изменять плейлисты. Администратор, в свою очередь, может добавлять исполнителей, альбомы и треки, удалять их.

## **Проектирование базы данных**

Как отмечалось ранее, была выбрана нереляционная база данных. Вместо таблиц у таких БД коллекции. Для данного проекта были спроектированы шесть коллекций: пользователи, жанры, альбомы, треки, плейлисты и исполнители. Схема коллекций и связи между ними представлены на рисунке 2.2.

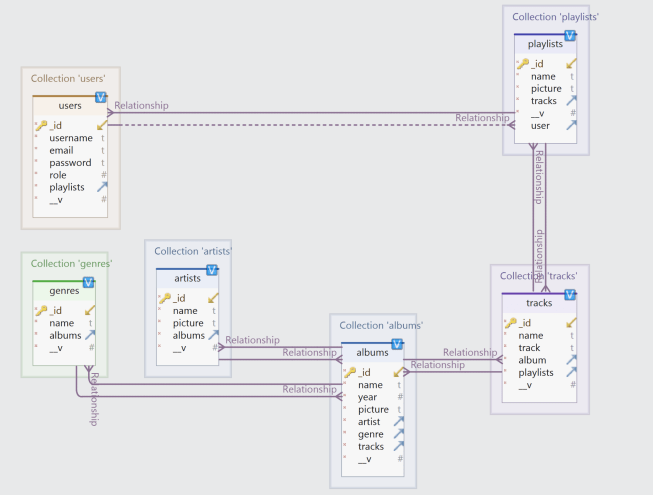


Рисунок 2.2 – Схема спроектированной базы данных

Коллекция «Users» хранит данные пользователя: имя, электронную почту, захешированный пароль, роль и массив плейлистов.

Коллекция «Playlists» содержит название плейлиста, обложку, массив id треков и id пользователя – владельца плейлиста.

Коллекция «Треки» содержит название трека, сам трек, id альбома и массив id плейлистов, в которых этот трек находится.

Коллекция «Альбомы» содержит название альбома, год его выпуска, обложку, id жанра и исполнителя, а также массив id треков, входящих в альбом.

Коллекция «Исполнители» содержит имя исполнителя, его аватар (изображение) и массив id альбомов этого исполнителя.

Коллекция «Жанры» содержит название жанра и массив id альбомов этого жанра.

## **Проектирование основных алгоритмов**

Для описания были выбраны алгоритм добавления исполнителя администратором и алгоритм прослушивания трека пользователем.

Добавление исполнителя состоит из 7 этапов. Для начала нужно войти в приложение с ролью администратора, после этого выбрать соответствующую опцию. Система проверит есть ли такой исполнитель в базе данных, и в зависимости от ответа завершит или отменит действия. Блок-схема алгоритма представлена ниже.

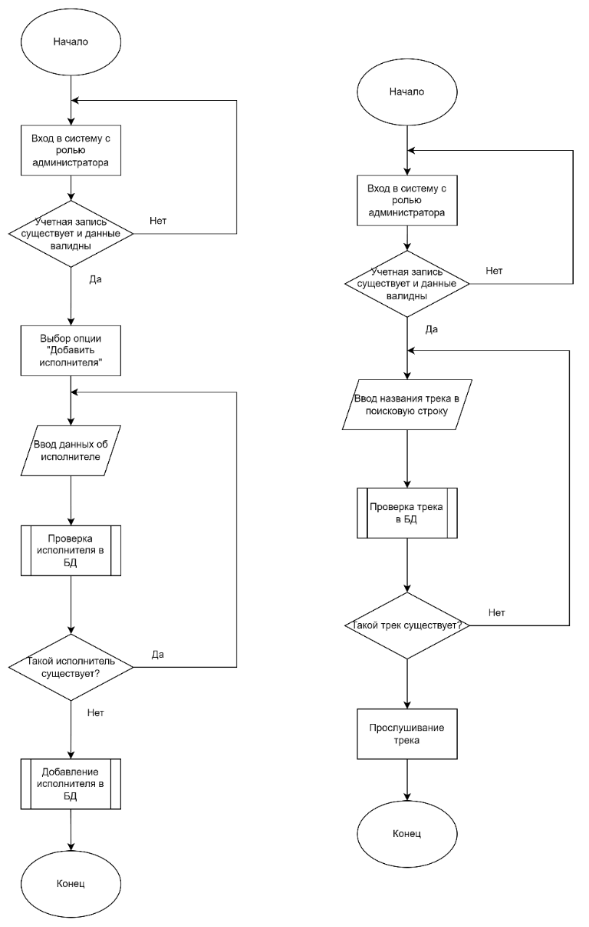


Рисунок 2.3 – Блок-схемы алгоритмов

Чтобы прослушать трек, нужно войти в систему с ролью пользователя. Далее необходимо ввести название композиции в поисковую строку и проиграть песню. Структура алгоритма представлена на рисунке 2.3.

## **Вывод по разделу**

Таким образом была рассмотрены инструменты разработки, спроектирована база данных, описаны UML-диаграммы вариантов использования для разных типов пользователей (администратора и пользователя). Также были описаны алгоритмы добавления исполнителя и прослушивания трека, а также определен стек технологий, используемый в данном проекте.

# Разработка программного средства

Как отмечалось ранее, приложение будет разрабатываться при помощи фреймворка NestJS для серверной части, и библиотеки ReactJS – для клиентской.

## **Разработка серверной части**

Архитектура приложения – это структурный дизайн программного обеспечения, который определяет организацию компонентов приложения, их взаимодействие и внешний интерфейс. Это набор принципов и паттернов, которые определяют, как приложение будет построено, чтобы обеспечить его эффективность, масштабируемость, удобство сопровождения и другие характеристики.

Структура серверной части представлена ниже:

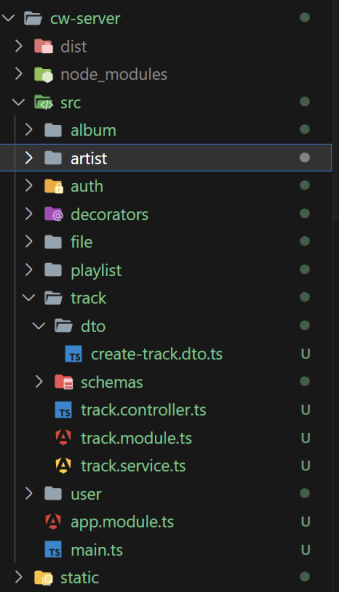


Рисунок 3.1 – Структура серверной части веб-приложения

В папке dist расположены скомпилированные файлы сервера. Node\_modules содержит зависимости проекта, а static – статические файлы. Основной код приложения расположен в папке src. Здесь легко заметить директории, соответствующие названиям сущностей из бд. В каждой такой папке есть модули, контроллеры, сервисы, dto, схемы. Рассмотрим для чего нужна каждая из сущностей.

Модули содержат связанные контроллеры, сервисы, провайдеры и другие компоненты. Это помогает организовать код, делает его более модульным и улучшает его переиспользование. Ниже представлен листинг кода для модуля плейлиста.

|  |
| --- |
| @Module({  imports: [  UserModule,  TrackModule,  MongooseModule.forFeature([  { name: Playlist.name, schema: PlaylistSchema },  ]),  MongooseModule.forFeature([{ name: User.name, schema: UserSchema }]),  MongooseModule.forFeature([{ name: Track.name, schema: UserSchema }]),  ],  controllers: [PlaylistController],  providers: [PlaylistService, FileService],  })  export class PlaylistModule {} |

Листинг 3.1 – Playlist.module.ts

Контроллеры принимают входящие запросы, вызывают соответствующие методы сервисов и возвращают HTTP-ответы. Они являются точкой входа для внешних запросов. Ниже представлена часть кода контроллера исполнителя.

|  |
| --- |
| @Controller('/artists')  export class ArtistController {  constructor(private artistService: ArtistService) {}  @Post()  @UseInterceptors(FileInterceptor('picture'))  create(@UploadedFile() file, @Body() dto: CreateArtistDto) {  return this.artistService.create(dto, file);  }  @Get()  getAll() {return this.artistService.getAll();  }  @Get('/search')  search(@Query('query') query: string) {  return this.artistService.search(query }  } |

Листинг 3.2 – Artist.controller.ts

Сервисы содержат методы, которые реализуют бизнес-логику приложения. Они обрабатывают данные, взаимодействуют с базой данных, выполняют операции и предоставляют результаты контроллерам. Часть кода сервиса для треков представлен ниже.

|  |
| --- |
| @Injectable()  export class TrackService {  constructor(  @InjectModel(Track.name) private trackModel: Model<Track>,  @InjectModel(Album.name) private albumModel: Model<Album>,  @InjectModel(Playlist.name) private playlistModel: Model<Playlist>,  private readonly fileService: FileService,  ) {}  async addTrack(dto: CreateTrackDto, audio: any): Promise<Track> {  const audioPath = this.fileService.createFile(FileType.AUDIO, audio);  const album = await this.albumModel.findById(dto.albumId);  const track = await this.trackModel.create({...dto,  album: dto.albumId, track: audioPath,});  album.tracks.push(track.id);  await Promise.all([track.save(), album.save()]);  return track;  }  async search(query: string): Promise<Track[]> {  const tracks = await this.trackModel.find({  name: { $regex: new RegExp(query, 'i')  },});  return tracks;  }  async getById(trackId: ObjectId): Promise<Track> {  const track = await this.trackModel.findById(trackId);  return track;  } |

Листинг 3.3 –Track.service.ts

Здесь представлены методы для добавления, поиска трека и его получения по id.

DTO представляют объекты данных, которые используются для передачи данных между различными частями приложения. Они помогают определить формат и структуру данных, которые передаются через API.

Схемы используются, например, при валидации данных, полученных из запросов. Они определяют, какие данные ожидаются и как они должны быть представлены. Схема для исполнителей представлена ниже.

|  |
| --- |
| import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';  import { HydratedDocument } from 'mongoose';  import { Album } from 'src/album/schemas/album.schema';  import \* as mongoose from 'mongoose';  export type ArtistDocument = HydratedDocument<Artist>;  @Schema()  export class Artist {  @Prop()  name: string;  @Prop()  picture: string;  @Prop({ type: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Album' }] })  albums: Album[];  }  export const ArtistSchema=SchemaFactory.createForClass(Artist); |

Листинг 3.4 – Artist.schema.ts

Все остальные контроллеры, сервисы, модули, схемы и dto в проекте имеют схожую структуру.

## **Разработка клиентской части**

Структура разработанного React-приложения представлена на рисунке 4.2. Оно содержит следующие директории и файлы:

* public: этот каталог содержит статические ресурсы, такие как файлы HTML, изображения и т. д.;
* index.html обычно используется как точка входа для приложения;
* src: этот каталог содержит исходный код приложения;
* assets: здесь могут находиться различные ресурсы, такие как изображения и стили;
* components: каталог, содержащий компоненты React. Каждый компонент может иметь свой собственный каталог, включая файлы TypeScriptXML и CSS;
* pages: подкаталог components, содержащий компоненты страниц. Каждая страница может иметь свой собственный каталог;
* services: как для клиентских запросов на серверную часть приложения;
* App.tsx: основной компонент приложения, который обычно содержит маршрутизацию и определяет структуру приложения;
* main.tsx: точка входа для приложения, где происходит рендеринг корневого компонента в DOM;
* package.json: файл с метаданными о проекте, зависимостями и скриптами;
* index.css: главный файл со стилями.

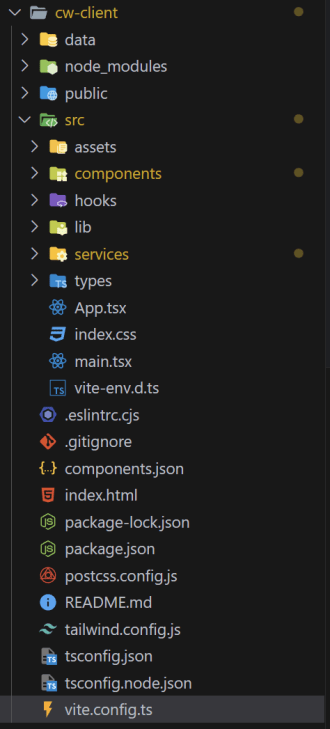


Рисунок 3.2 – Структура клиентской части приложения

Теперь рассмотрим работу конкретного компонента на примере AlbumDemoPage. Он представляет собой страницу, отображающую список альбомов и предоставляющую возможность поиска альбомов. В нем реализована логика для загрузки альбомов с использованием хука useEffect и сервиса AlbumService (код представлен на листинге ниже) при изменении поискового запроса (searchQuery). Компонент включает также обработчик клика на альбоме (handleAlbumClick), который осуществляет навигацию к детальной странице альбома.

Импорты включают интерфейс IAlbum, компоненты для работы с областями прокрутки, хуки React, сервис для взаимодействия с данными альбомов, компонент отображения информации об альбоме, компонент для текстового поля ввода и хук для навигации.

|  |
| --- |
| export const AlbumService = {  async getAllAlbums(): Promise<IAlbum[]> {  const response = await axios.get<IAlbum[]>(`http://localhost:5000/albums`);  return response.data;  },  async getAlbumById(albumId: string | undefined): Promise<IAlbum> {  const response = await axios.get<IAlbum>(  `http://localhost:5000/albums/get/${albumId}`  );  return response.data;  },  async searchAlbums(albumName?: string): Promise<IAlbum[]> {  const response = await axios.get('http://localhost:5000/albums/search', {  params: {  query: albumName,  },  });  return response.data;  },  }; |

Листинг 3.5 – Сервис для отправки запросов на сервер для альбома

Структура компонента включает состояния для хранения списка альбомов (albums) и поискового запроса (searchQuery). На странице присутствуют две области прокрутки: верхняя с заголовком "Альбомы" и полем ввода для поиска, а нижняя с отображением альбомов в виде компонентов AlbumCard. Горизонтальная полоса прокрутки добавлена к нижней области. Этот компонент обеспечивает пользовательский интерфейс для просмотра и поиска альбомов, а также предоставляет навигацию к деталям выбранного альбома.

## **Контейнеризация**

Docker – это открытая платформа для разработки, доставки и запуска приложений в контейнерах. Она обеспечивает стандартизированный способ упаковки приложений и всех их зависимостей в легкий и переносимый контейнер. Этот контейнер может быть запущен на любой системе, поддерживающей Docker, обеспечивая при этом изолированное и надежное выполнение приложения.

Чтобы создать контейнер приложения, используются специальные Docker файлы (Dockerfile). Пример Dockerfile для серверной части приложения представлен на рисунке ниже.

|  |
| --- |
| FROM node:18-alpine  RUN mkdir -p /usr/src/app  WORKDIR /usr/src/app  COPY package\*.json ./  RUN npm install  COPY . .  EXPOSE 5000  CMD ["npm", "run", "start:prod"] |

Листинг 3.6 – Dockerfile для серверной части приложения

Теперь займемся его подробным описанием. Этот Dockerfile используется для создания Docker-образа на основе образа Node.js версии 18 на базе Alpine Linux. Docker-образ предназначен для запуска Node.js-приложения.

* FROM node:18-alpine: Эта инструкция указывает базовый образ для текущего Docker-образа. В данном случае, используется официальный образ Node.js версии 18, основанный на Alpine Linux. Alpine Linux — это легковесный дистрибутив Linux, что помогает уменьшить размер итогового Docker-образа.
* RUN mkdir -p /usr/src/app: Эта инструкция создает директорию /usr/src/app внутри контейнера. Обычно в ней будут размещаться файлы приложения.
* WORKDIR /usr/src/app: Эта инструкция устанавливает рабочий каталог для всех последующих инструкций в Dockerfile. В данном случае, это /usr/src/app.
* COPY package\*.json ./: Эта инструкция копирует файлы package.json и package-lock.json из контекста сборки (текущей директории) внутрь рабочего каталога контейнера.
* RUN npm install: Эта инструкция выполняет установку зависимостей Node.js, указанных в файле package.json;
* COPY .. : Эта инструкция копирует все файлы из контекста сборки в текущий рабочий каталог в контейнере. Это включает в себя код приложения.
* EXPOSE 5000: Эта инструкция объявляет порт 5000, который будет использоваться приложением. Однако, это не фактическое открытие порта; она служит для информирования о том, что контейнер ожидает прослушивания порта 5000.
* CMD ["npm", "run", "start:prod"]: Эта инструкция задает команду, которая будет выполнена при запуске контейнера. В данном случае, это команда для запуска приложения в режиме production с использованием npm.

Таким образом, этот Dockerfile определяет окружение для Node.js приложения, устанавливает зависимости, копирует код приложения и указывает команду для запуска приложения в контейнере.

После создания конфигурационного файла необходимо выполнить команду docker build «название» проекта. После успешного завершения сборки контейнер появится в списке доступных в приложении Docker Desktop. Запустить его можно при помощи команды docker run.

## **Вывод по разделу**

В данной главе курсовой работы рассмотрены ключевые аспекты разработки программного средства с использованием фреймворка NestJS для серверной части и библиотеки ReactJS для клиентской. Основное внимание уделено архитектуре серверной части, где подробно описаны структура папок, роли модулей, контроллеров, сервисов, DTO и схем. Данный раздел предоставляет комплексное понимание архитектуры и структуры разрабатываемого программного средства, а также демонстрирует примеры практической реализации на сервере и клиенте.­­­­­­ ­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­

# Тестирование веб-приложения

Для проверки работоспособности приложения было проведено ручное тестирование. Ниже будут предоставлены скриншоты с описанием проведенных тестов.

При попытке входа в приложение под несуществующим пользователем выводится модальное окно с сообщением, приведенным на рисунке ниже.

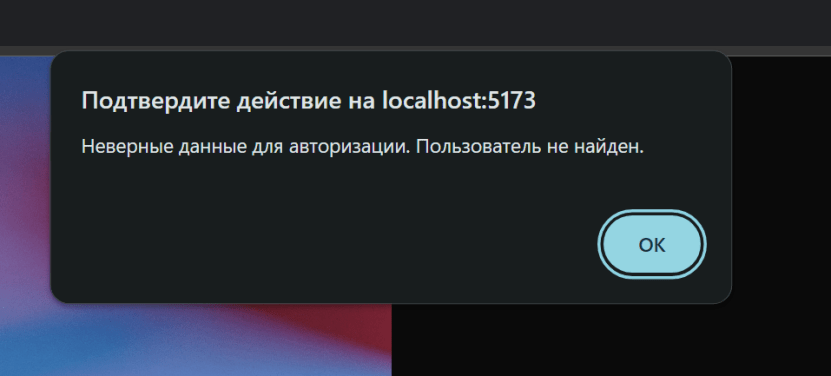


Рисунок 4.1 – Тестирование процесса авторизации под несуществующим пользователем

Далее выполним тестирование поисковой строки для альбомов. В состоянии, когда поисковая строка пустая, выводятся все альбомы.



Рисунок 4.2 – Состояние списка альбомов при пустой поисковой строке

Стоит пользователю ввести какие-либо символы в поисковую строку, тут же отправится запрос на сервер и в ответе придут только те альбомы, в названиях которых присутствуют символы, введенные пользователем.

Демонстрация данного состояния приведена на рисунке 4.3.



Рисунок 4.3 – Состояние списка альбомов при введеных символах в поисковую строку.

Следующим на очереди станет тест добавления плейлиста. При попытке добавления плейлиста без названия или обложки будет выведено соответствующее сообщение.

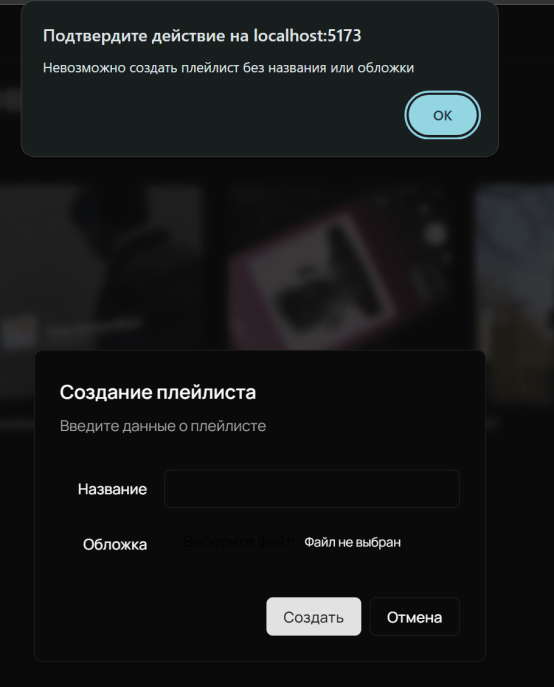


Рисунок 4.4 – Попытка создания плейлиста без обложки и названия

В противном же случае, когда имя введено, а обложка добавлена, плейлист появится в пользовательской библиотеке.

Демонстрация этого кейса представлена на рисунке 4.5.

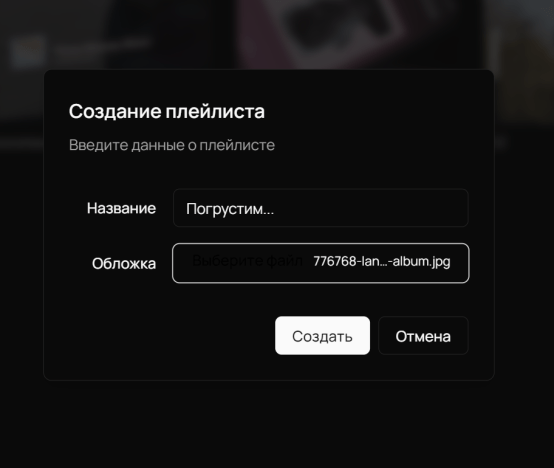


Рисунок 4.5 – Заполненная форма создания плейлиста

После нажатия кнопки «Создать» плейлист появляется в библиотеке пользователя.



Рисунок 4.6 – Успешно добавленный плейлист

Теперь рассмотрим ситуацию, когда пользователь пытается зарегистрироваться в приложении с некорректными данными. Например, при попытке регистрации с пустыми данными, кнопка «Зарегистрироваться» будет недоступна.

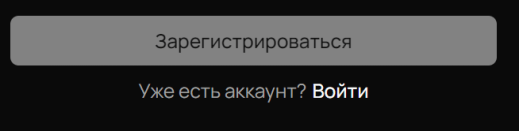


Рисунок 4.7 – Отключенная кнопка «Зарегистрироваться»

Также, если пользователь введет неверный формат электронной почты, система оповестит его об этом.

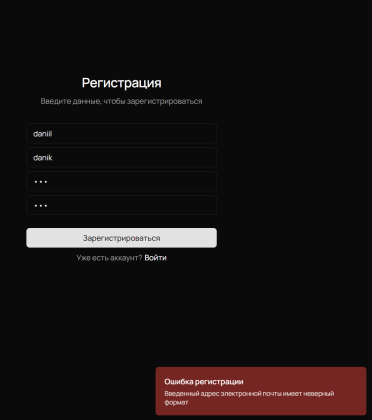


Рисунок 4.8 – Оповещение системы о неверном формате Email

Таким образом было предотвращены действия по добавлению плейлиста и регистрации с некорректными данными.

## **Вывод по разделу**

В данном разделе было проведено ручное тестирование клиентской части приложения. Тестирование включало в себя сценарии входа с несуществующим пользователем, использование поисковой строки для альбомов, а также добавление плейлиста с корректными и некорректными данными. С помощью модальных окон и соответствующих сообщений пользователю было предотвращено введение некорректных данных и обеспечена корректная обработка различных сценариев использования.

# Руководство пользователя

## **Методика использования приложения пользователем**

Первое, что встречает пользователь при запуске приложения – это форма авторизации, она приведена на рисунке 5.1.

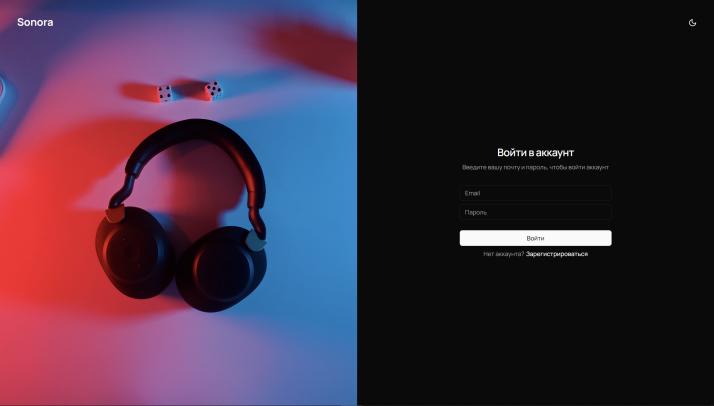


Рисунок 5.1 – Страница регистрации и авторизации

Для входа в систему пользователю необходимо зарегистрироваться, после чего ввести свою почту и пароль. После входа пользователь попадает на страницу со своими плейлистами.

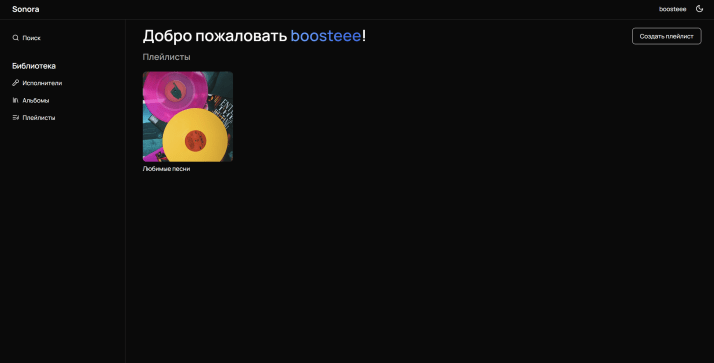


Рисунок 5.2 – Страница пользователя с плейлистами

На этой странице пользователь может создать свой собственный плейлист или перейти на страницу с отдельным плейлистом.

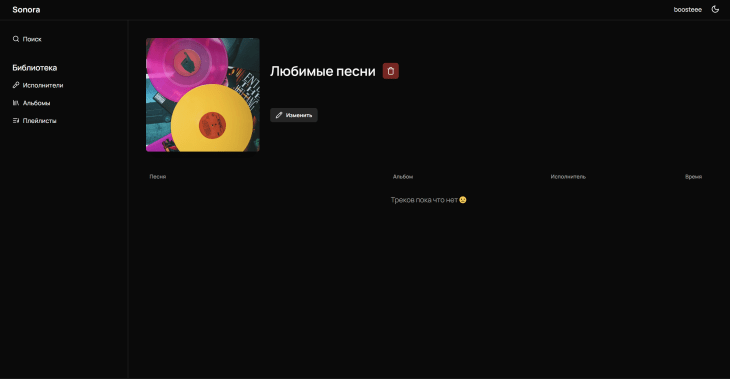


Рисунок 5.3 – Страница с отдельным плейлистом

На этой странице пользователь может изменить свой плейлист или удалить его. Следующей страницей, которую при желании может посетить пользователь станет страница «Исполнители».

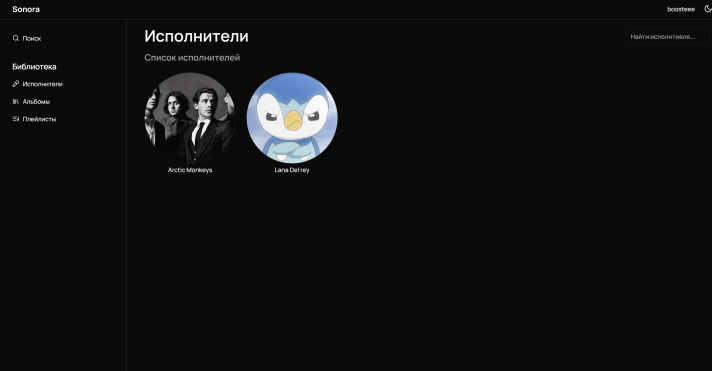


Рисунок 5.4 – Страница с исполнителями

На этой странице пользователь может найти интересующего исполнителя при помощи поисковой строки, а также перейти на страницу конкретного исполнителя, где можно посмотреть некоторые композиции музыканта, а также все его альбомы. Скриншот страницы с конкретным исполнителем представлен на рисунке 5.5.

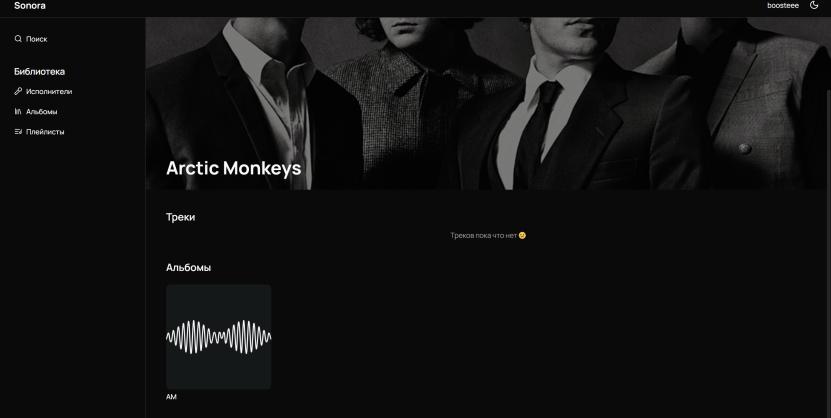


Рисунок 5.5 – Страница с отдельным исполнителем

С этой страницы пользователь может перейти на страницу конкретного альбома.

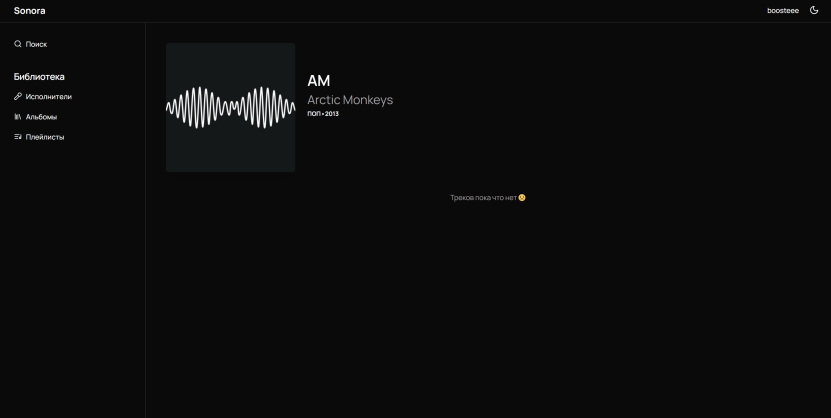


Рисунок 5.6 – Страница конкретного альбома

Страница «Альбомы» имеет схожую функциональность со страницей «Исполнители». На странице «Поиск» пользователь имеет возможность найти интересующий его трек и прослушать его. Также пользователь может изменить цветовое оформление приложения и личные данные.

Данный функционал продемонстрирован на рисунке 5.7.

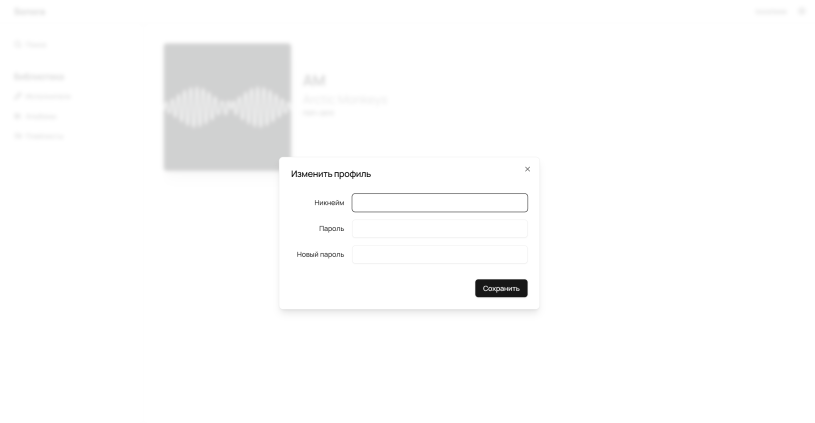


Рисунок 5.7 – Форма для изменения личной информации и светлая тема оформления приложения

## **Методика использования приложения администратором**

При запуске приложения, администратор попадает на страницу авторизации. Здесь ему нужно зарегистрироваться или войти, используя свой логин и пароль. После успешной авторизации, пользователь попадает на главную страницу приложения. Здесь расположены главные элементы приложения: музыкальный плеер, главное меню и сайдбар, при помощи которого можно переходить между основными экранами.

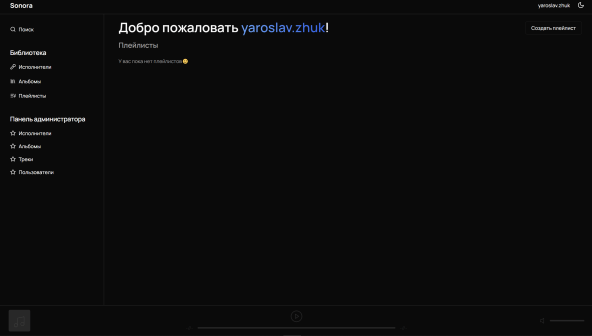


Рисунок 5.8 – Главный экран музыкального сервиса

Важно отметить, что у администратора есть особые разделы меню, перейдя по которым можно управлять содержимым приложения и пользователями. Пройдемся конкретно по каждому.

Первый раздел – исполнители. Здесь администратор может создавать и удалять музыкантов.

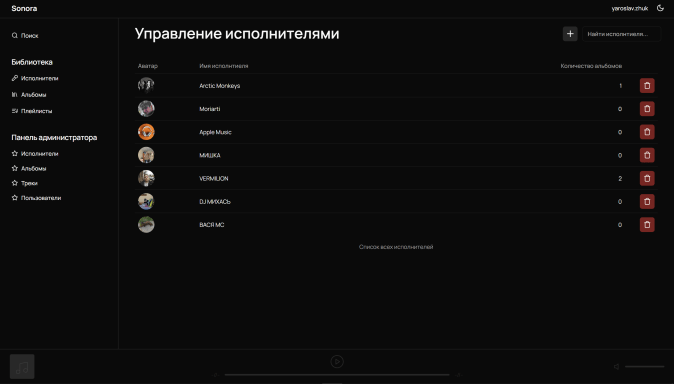


Рисунок 5.9 – Страница управления исполнителями

Нажав на кнопку «+», открывается форма добавления исполнителя. Скриншот формы представлен ниже:

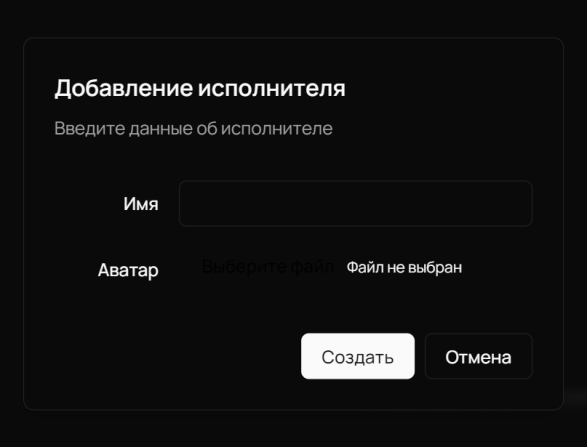


Рисунок 5.10 – Форма добавления исполнителя

Следующим рассмотренным разделом будет раздел «Альбомы». Здесь администратор может добавлять, удалять, искать альбомы и сортировать по жанру. Страница организована в виде таблицы, в каждой строке которой имеется подробная информация об альбоме: обложка, исполнитель, жанр и количество треков. Ниже представлен скриншот данной страницы.

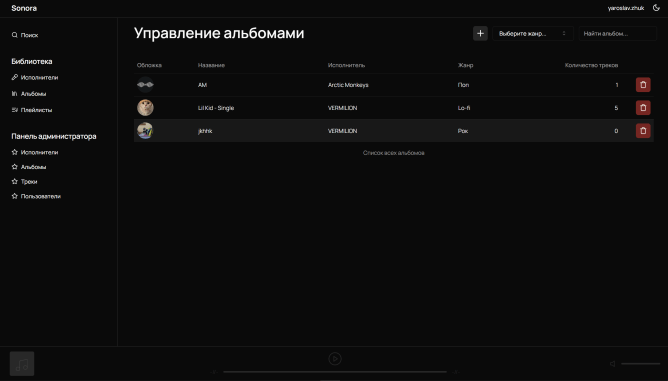


Рисунок 5.11 – Страница управления альбомами

Следующей на очереди станет страница с треками, на ней аналогично предыдущим, можно создать, найти или удалить трек. На рисунке 5.12 продемонстрирован внешний вид данной страницы.

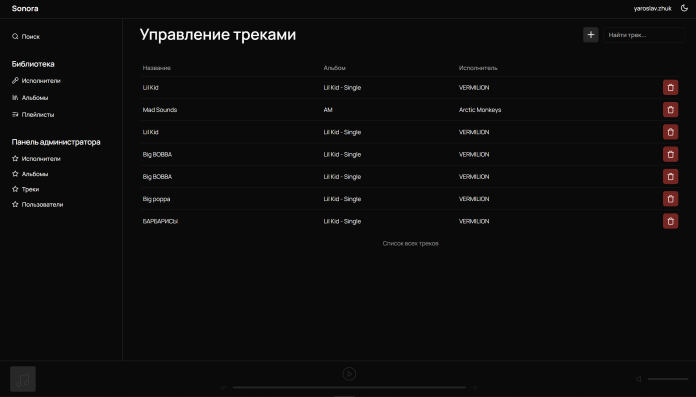


Рисунок 5.12 – Страница управления треками

Так же, как и на предыдущих страницах, нажав на кнопку «+» можно добавить сущность, в частности, трек. В ней нужно ввести название, выбрать альбом из существующих и прикрепить соответствующую аудиозапись, чтобы успешно завершить процедуру добавления.

Последней рассмотренной страницей будет страница «Пользователи». Ниже представлен её внешний вид.

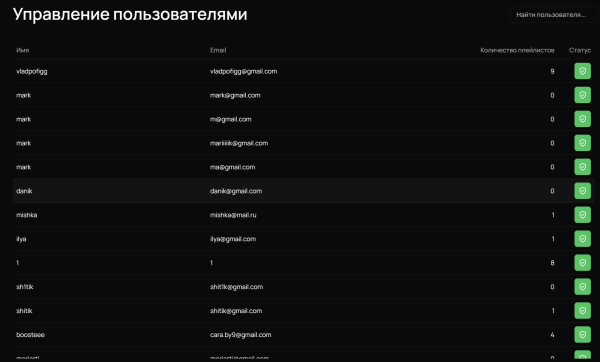


Рисунок 5.13 – Страница управления пользователями

Здесь администратор может найти определенного юзера и заблокировать ему доступ к сервису.

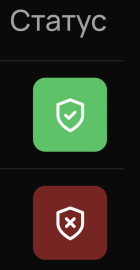


Рисунок 5.14 – Статус пользователя

## **Вывод по разделу**

В разделе представлены основные моменты работы с веб-приложением. Были приведены скриншоты самого приложения для большей наглядности использования с подробным описанием алгоритмов взаимодействия пользователя с веб-приложением. Описано то, что пользователь может наблюдать на различных страницах и функционал, который ему доступен.

Интерфейс приложения достаточно прост, поэтому у пользователя не должно возникнуть трудностей с его эксплуатацией.

Заключение

В рамках данного курсового проекта была выполнена основная задача – разработка современного музыкального стримингового сервиса. Это веб-приложение предоставляет широкие возможности для прослушивания музыки и управления контентом. Для достижения этой цели были решены следующие функциональные задачи.

Реализованы механизмы регистрации и авторизации, обеспечивающие безопасность и удобство пользователей. Пользователи могут взаимодействовать с сервисом в различных ролях, включая администратора и обычного пользователя. Разработаны средства прослушивания музыкальных композиций. Пользователи имеют возможность редактировать свой профиль, что способствует персонализации опыта использования. Реализован функционал создания, редактирования и удаления плейлистов, а также добавления и удаления треков из плейлистов. Обеспечен поиск музыкальных композиций, что позволяет пользователям легко находить интересующие их треки. Администратору предоставлены инструменты для добавления и удаления исполнителей, альбомов и треков, а также для ограничения доступа пользователей к сервису.

Разработанное приложение успешно прошло ручное тестирование, подтвердив свою функциональность и соответствие поставленным требованиям. Исходя из результатов работы программы, можно с уверенностью сказать, что созданное веб-приложение предоставляет пользователям широкий спектр возможностей для взаимодействия с музыкальным контентом и полностью соответствует поставленным задачам.

Список использованных источников

1. Spotify [Электронный ресурс]/ Spotify Web Player – Режим доступа: https://open.spotify.com/ – Дата доступа: 18.11.2023.
2. Яндекс Музыка [Электронный ресурс] /Yandex Music – Режим доступа: https://music.yandex.by/– Дата доступа: 20.11.2023.
3. YouTube Music [Электронный ресурс] /YouTube Music Service – Режим доступа: https://music.youtube.com/– Дата доступа: 22.11.2023.
4. Фаулер, М. UML. Основы. Краткое руководство по стандартному языку объектного моделирования. / М. Фаулер. – 3-е изд. – СПб: Символ-Плюс, 2004. – 192с.
5. Nest documentation [Электронный ресурс] / nestjs.com. – Режим доступа: https://nestjs.com. – Дата доступа: 25.10.2023.
6. Бэнкс, А. React и Redux. Функциональная веб-разработка. / А. Бэнкс, Е. Порселло. – СПб: Питер, 2020. – 336с.
7. TailwindCSS documentation [Электронный ресурс]/ tailwindcss.com. – Режим доступа: https://tailwindcss.com/. – Дата доступа: 12.11.2023.
8. MongoDB documentation [Электронный ресурс]/ mongodb.com/docs. – Режим доступа: https://www.mongodb.com/docs/. – Дата доступа: 8.12.2023.

Приложение А

**package.json**

|  |
| --- |
| {  "name": "cw-client",  "private": true,  "version": "0.0.0",  "type": "module",  "scripts": {  "dev": "vite",  "build": "tsc && vite build",  "lint": "eslint . --ext ts,tsx --report-unused-disable-directives --max-warnings 0",  "preview": "vite preview"  },  "dependencies": {  "@radix-ui/react-alert-dialog": "^1.0.5",  "@radix-ui/react-aspect-ratio": "^1.0.3",  "@radix-ui/react-avatar": "^1.0.4",  "@radix-ui/react-context-menu": "^2.1.5",  "@radix-ui/react-dialog": "^1.0.5",  "@radix-ui/react-dropdown-menu": "^2.0.6",  "@radix-ui/react-icons": "^1.3.0",  "@radix-ui/react-label": "^2.0.2",  "@radix-ui/react-menubar": "^1.0.4",  "@radix-ui/react-popover": "^1.0.7",  "@radix-ui/react-progress": "^1.0.3",  "@radix-ui/react-scroll-area": "^1.0.5",  "@radix-ui/react-separator": "^1.0.3",  "@radix-ui/react-slider": "^1.1.2",  "@radix-ui/react-slot": "^1.0.2",  "@radix-ui/react-tabs": "^1.0.4",  "@radix-ui/react-toast": "^1.1.5",  "@types/react-redux": "^7.1.33",  "axios": "^1.6.2",  "class-variance-authority": "^0.7.0",  "clsx": "^2.0.0",  "cmdk": "^0.2.0",  "lottie-react": "^2.4.0",  "lucide-react": "^0.294.0",  "music-metadata-browser": "^2.5.10",  "react": "^18.2.0",  "react-dom": "^18.2.0",  "react-redux": "^9.0.4",  "react-router-dom": "^6.20.1",  "redux": "^5.0.0",  "redux-thunk": "^3.1.0",  "tailwind-merge": "^2.0.0",  "tailwindcss-animate": "^1.0.7"  },  "devDependencies": {  "@types/node": "^20.10.0",  "@types/react": "^18.2.37",  "@types/react-dom": "^18.2.15",  "@typescript-eslint/eslint-plugin": "^6.10.0",  "@typescript-eslint/parser": "^6.10.0",  "@vitejs/plugin-react-swc": "^3.5.0",  "autoprefixer": "^10.4.16",  "eslint": "^8.53.0",  "eslint-plugin-react-hooks": "^4.6.0",  "eslint-plugin-react-refresh": "^0.4.4",  "postcss": "^8.4.31",  "tailwindcss": "^3.3.5",  "typescript": "^5.2.2",  "vite": "^5.0.0"  }  } |

Приложение Б

**AddGenreForm.tsx**

|  |
| --- |
| export interface AddGenreFormProps {  setCreated: React.Dispatch<React.SetStateAction<boolean>>;  }  const AddGenreForm: React.FC<AddGenreFormProps> = ({ setCreated }) => {  const genreName = useInput('');  const addGenreHandle = async () => {  const data = await AlbumService.addGenre(genreName.value);  setCreated(true);  return data;  };  return (  <AlertDialog>  <AlertDialogTrigger asChild>  <Button variant='outline'>Добавить жанр</Button>  </AlertDialogTrigger>  <AlertDialogContent>  <AlertDialogHeader>  <AlertDialogTitle>Добавление жанра</AlertDialogTitle>  <AlertDialogDescription>  Введите название жанра  </AlertDialogDescription>  </AlertDialogHeader>  <div className=' grid grid-cols-4 items-center gap-4'>  <Label htmlFor='name' className='text-right'>  Название  </Label>  <Input {...genreName} id='name' className='col-span-3' />  </div>  <AlertDialogFooter>  <AlertDialogCancel>Отмена</AlertDialogCancel>  <AlertDialogAction  onClick={() => {  addGenreHandle();  }}  >  Добавить  </AlertDialogAction>  </AlertDialogFooter>  </AlertDialogContent>  </AlertDialog>  );  };  export default AddGenreForm; |