

**Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Институт вычислительной математики и информационных технологий
Кафедра системного анализа и информационных технологий

Направление подготовки: 02.03.02 – Фундаментальная информатика
и информационные технологии
Профиль: Системный анализ и информационные технологии

КУРСОВАЯ РАБОТА

Музыкальный web-сервис «SoundScape»

Студент 3 курса

группы 09-132

«__» _____ 2024 г.

Чирков Л.С.

Научный руководитель

канд. физ.-мат. наук, доцент, доцент КСАИТ

«__» _____ 2024 г.

Шаймухаметов Р.Р.

Казань-2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
1. Актуальность выбранной темы.....	6
2. Анализ приложений для прослушивания музыки	7
3. Технологические аспекты и инструменты разработки web-приложений...	10
3.1. Понятие web-приложения	10
3.2. Основы разработки web-приложений	12
3.3. Обзор и описание используемых технологий и инструментов разработки.....	13
4. Разработка web-сервиса «SoundScape»	16
4.1. Интерфейс	16
4.2. Структура приложения	19
4.3. Компонент Sidebar	21
4.4. Основная страница приложения.....	22
4.5. База данных.....	23
4.6. Модальное окно.....	25
4.7. Страница Search	26
4.8. Страница News	27
4.9. Библиотека пользователя	28
4.10. Альбомы и плейлисты	29
4.11. Плеер	33
4.12. Страница аккаунта и интеграция Stripe	35
ЗАКЛЮЧЕНИЕ	41
СПИСОК ЛИТЕРАТУРЫ.....	43
Приложение 1. Код компонента Sidebar	45
Приложение 2. Код основной страницы приложения	47
Приложение 3. Код SupabaseProvider и UserProvider	49
Приложение 4. Код компонента Modal.....	50
Приложение 5. Экшен getSongsByTitle.....	53
Приложение 6. Код компонента SearchInput.....	54
Приложение 7. Код страницы Search	55

Приложение 8. Код страницы News	56
Приложение 9. Экшен getNews.....	57
Приложение 10. Код компонента NewsContent	58
Приложение 11. Экшены getSongsForPlaylist и getSongsForAlbum.....	59
Приложение 12. Код компонента LikedContent	62
Приложение 13. Код компонента MusicMenu.....	68
Приложение 14. Код компонента LikeButton.....	71
Приложение 15. Код компонента AlbumModal.....	74

ВВЕДЕНИЕ

С развитием интернет-технологий web-сервисы становятся нашими верными спутниками в мире развлечений. Они открывают двери разнообразным формам деятельности и различным способам проведения свободного времени.

В этом мире музыкальные сервисы занимают особенно важное место. Музыка сопровождает нас повсюду: мы идем по улице, слушая любимые песни через наушники, наслаждаемся ими дома в атмосфере уюта и спокойствия. Музыка становится нашим лучшим другом в дороге, верным спутником во время занятий спортом и незаменимым компаньоном во время работы или отдыха. Музыка вдохновляет, успокаивает, поднимает настроение и заставляет нас чувствовать себя живыми. В мире, где мы проводим все больше времени в сети, музыкальные сервисы предлагают нам возможность наслаждаться музыкой в любое удобное время, создавая собственный звуковой мир, соответствующий нашему настроению и желаниям.

Целью данной курсовой работы является разработка музыкального web-сервиса «SoundScape», предоставляющего пользователям легкий доступ к широкому спектру музыкальных произведений, а также удобный функционал для прослушивания музыки и взаимодействия с приложением.

Для достижения поставленной цели необходимо решить следующие задачи:

- 1) Проанализировать различные музыкальные сервисы для определения основных категорий и функционала, при этом выявляя наиболее нужные и необходимые функции для их последующей разработки и реализации в приложении;

- 2) Создать пользовательский интерфейс музыкального web-сервиса с учетом особенностей взаимодействия пользователя с контентом и функционалом сервиса;

- 3) Разработать музыкальный web-сервис «SoundScape», обеспечивающий удобный поиск и прослушивание музыки, а также

предоставляющий разнообразный функционал для взаимодействия с музыкой и экосистемой приложения.

1. Актуальность выбранной темы

Музыкальные сервисы становятся важным способом доступа к музыкальному контенту для миллионов пользователей по всему миру. Они предоставляют доступ к обширному набору различных музыкальных композиций, создают уникальные возможности для обмена музыкальными предпочтениями и взаимодействия с другими пользователями.

В этом контексте разработка музыкального web-сервиса, отвечающего потребностям современного пользователя, является чрезвычайно актуальной задачей. Хотя существующие музыкальные платформы обладают большой функциональностью, они часто страдают от избыточности и сложности в использовании.

Музыкальный web-сервис «SoundScape» направлен на решение этой проблемы. Он стремится предоставить пользователям удобный, интуитивно понятный и легко доступный функционал для поиска, прослушивания и обмена музыкой. Фокус на основных и наиболее востребованных функциях делает «SoundScape» привлекательным выбором для тех, кому нравится простота и удобство в использовании.

Кроме того, «SoundScape» открывает двери для начинающих музыкантов, предоставляя им возможность загружать свои собственные композиции на платформу и делиться ими с аудиторией. Это помогает раскрыть потенциал новых талантов в музыкальной индустрии.

Таким образом, музыкальный web-сервис «SoundScape» не только улучшает пользовательский опыт в области прослушивания музыки, но и способствует развитию музыкального сообщества.

2. Анализ приложений для прослушивания музыки

Анализ существующих приложений для прослушивания музыки включает в себя процесс изучения различных аспектов этих приложений с целью понимания их общей структуры, интерфейса, функциональности и других характеристик, что позволяет выявить их сильные и слабые стороны, а также определить наиболее востребованные функции и элементы, которые привлекают пользователей.

Понимание общей структуры приложения является ключевым шагом в анализе. Это включает в себя изучение основных разделов и функций, которые предоставляет приложение. При этом особое внимание уделяется интерфейсу – удобству его использования, навигации и визуальной привлекательности. Часто важно учитывать именно те элементы, которые делают приложение интуитивно понятным и привлекательным для пользователя.

С точки зрения функциональности, анализ приложений для прослушивания музыки включает в себя изучение доступных возможностей. Важно определить, какие функции наиболее важны для пользователей и как их можно реализовать наиболее эффективно.

Выбор только самого необходимого функционала – немаловажный аспект анализа. Он позволяет исключить визуальную перегрузку и упростить пользовательский опыт. Это означает, что необходимо выбирать только те функции и элементы, которые действительно нужны пользователям и которые помогут им достичь своих целей с минимальными усилиями.

Изучив существующие приложения и проанализировав их особенности, можно сформулировать концепцию своего будущего приложения. Это позволит создать музыкальный web-сервис, который сочетает в себе лучшие практики существующих приложений и собственноручно внедренные решения.

Существует множество различных приложений для прослушивания музыки, рассмотрим самые популярные отечественные и зарубежные приложения и выявим их достоинства и недостатки:

– Spotify – один из самых популярных музыкальных стриминговых сервисов в мире с огромным каталогом музыки, персонализированными рекомендациями и возможностью создания собственных плейлистов. Из минусов можно выделить частые прерывания музыкального прослушивания рекламой, особенно в бесплатной версии, что может раздражать пользователей, а также то, что некоторые треки или артисты могут быть недоступны в определенных регионах из-за ограничений лицензий, что может огорчать пользователей [1]);

– Яндекс.Музыка – популярный отечественный музыкальный стриминговый сервис с широким каталогом музыки и возможностью создания персональных плейлистов. Некоторые пользователи отмечают систематические проблемы с поиском и организацией плейлистов, что затрудняет пользовательский опыт. Также, ограничения в доступности определенных треков или артистов для прослушивания могут вызвать неудовлетворенность пользователей [2]);

– VK Музыка – музыкальный сервис, интегрированный в социальную сеть «ВКонтакте», предоставляющий доступ к большому каталогу музыки. Однако, нестабильная работа приложения и технические проблемы, такие как зависания и вылеты, могут вызывать неудобства для пользователей. Ограничения доступа к некоторым трекам из-за авторских прав также являются существенным недостатком [3]);

– Deezer – музыкальный стриминговый сервис с широким каталогом треков и высоким качеством звука. Недоступность в некоторых странах и сложности с навигацией и поиском музыкальных треков в приложении могут снижать удовлетворенность пользователей от использования сервиса [4]);

– SoundCloud – платформа для обмена и прослушивания музыки, позволяющая пользователям загружать свои треки и делиться ими с другими пользователями. Низкое качество аудиозаписей и ограничения на доступность

некоторых треков из-за авторских прав являются недостатками использования этой платформы [5]).

После анализа существующих приложений были выявлены следующие недостатки:

- частые прерывания музыкального прослушивания рекламой,
- проблемы с организацией и поиском музыкальных треков,
- технические неполадки и нестабильная работа приложений,
- ограничения доступности некоторых треков из-за авторских прав и региональных ограничений,
- низкое качество аудиозаписей и сложности с навигацией по приложению.

Таким образом, проанализировав различные музыкальные приложения, мы смогли выявить основные недостатки, что поможет учесть эти аспекты при разработке музыкального web-сервиса «SoundScape». Это позволит сосредоточиться на создании приложения, которое будет предлагать улучшенный пользовательский опыт и решать основные проблемы, выявленные в ходе анализа.

3. Технологические аспекты и инструменты разработки web-приложений

3.1. Понятие web-приложения

Web-приложение — это программное решение, которое позволяет пользователям взаимодействовать с различными сервисами, данными или функциональными возможностями прямо через web-браузер, без необходимости установки дополнительных программ на свои устройства. Они представляют собой динамические web-страницы или набор взаимосвязанных web-страниц, которые обладают интерактивными элементами, позволяющими пользователям взаимодействовать с контентом, вносить изменения и получать результаты в реальном времени [6]).

Структура web-приложения зависит от его конкретного назначения и функционала, но обычно включает в себя клиентскую часть (frontend) и серверную часть (backend). Клиентская часть отвечает за визуальное представление приложения и взаимодействие с пользователем, в то время как серверная часть обрабатывает запросы пользователя, осуществляет доступ к базе данных и обеспечивает функциональность приложения [6]).

Существует несколько подходов к созданию web-приложений, каждый из которых имеет свои особенности и преимущества. Традиционный подход включает использование серверных технологий, таких как PHP, Ruby on Rails или Node.js. Эти технологии позволяют создавать приложения, в которых весь контент формируется на сервере и затем отправляется на клиентские устройства. Данный подход обеспечивает хорошую SEO-оптимизацию и поддерживает рендеринг на стороне сервера (SSR), что позволяет быстрее загружать web-страницы для пользователей.

Современные подходы, такие как одностраничные приложения (SPA), основанные на JavaScript фреймворках, таких как React, Angular или Vue.js, становятся все более популярными. Данные фреймворки позволяют создавать более динамичные и интерактивные пользовательские интерфейсы, где весь контент загружается один раз, а затем динамически обновляется без

перезагрузки страницы. Это создает более быстрые и отзывчивые web-приложения, что повышает удобство использования для пользователей [7]).

Каждый из этих подходов имеет свои сильные и слабые стороны, и выбор конкретного подхода зависит от требований проекта, а также от опыта и предпочтений разработчиков. Хорошее web-приложение должно обеспечивать не только функциональность, но и удобство использования, эффективность и масштабируемость.

Хорошее web-приложение должно иметь следующие отличительные черты:

- доступность. Web-приложения могут быть запущены на любом устройстве с доступом к интернету и web-браузером;
- обновление. Изменения и обновления в web-приложении могут быть внесены на серверной стороне и становятся доступными пользователям сразу, без необходимости установки обновлений на устройство;
- масштабируемость. Приложение должно иметь способность к расширению и готовность к высоким нагрузкам;
- гибкость. Приложение должно адаптироваться к изменяющимся потребностям;
- универсальность. Благодаря тому, что web-приложения не привязаны к определенной операционной системе или устройству, они могут быть использованы на различных платформах.

Как и у любого технологического решения, у web-приложений есть свои плюсы и минусы. Среди плюсов можно выделить доступность, обновляемость и масштабируемость, однако недостатки включают в себя ограниченную функциональность без доступа к интернету, ограничения в производительности и безопасности, а также зависимость от стабильной работы серверов и соединения с интернетом.

Кроме того, web-приложения часто требуют дополнительных усилий для обеспечения защиты данных и безопасности пользователей, а также для

оптимизации производительности и масштабируемости при росте числа пользователей.

3.2. Основы разработки web-приложений

Разработка web-приложений включает в себя несколько основных аспектов, каждый из которых играет важную роль в создании функционального и эффективного продукта.

Frontend (клиентская часть) отвечает за то, как пользователь видит и взаимодействует с web-приложением. Он включает в себя разработку пользовательского интерфейса (UI) и его логику, а также интеграцию с бэкендом. Во frontend-разработке используют языки и фреймворки, такие как HTML, CSS и JavaScript, а также современные инструменты, такие как React, Angular или Vue.js, для создания динамичных и отзывчивых пользовательских интерфейсов [8]).

Backend (серверная часть) является скрытой стороной web-приложения, которая отвечает за обработку запросов пользователя, взаимодействие с базой данных и обеспечение функциональности, доступной через API. Бэкенд-разработчики используют различные языки программирования, такие как Java, Python, или Node.js, а также фреймворки, такие как Spring Boot, Django или Nest.js, для создания надежных и масштабируемых серверных приложений [9]).

SPA (Single Page Application) – это тип web-приложения, в котором все необходимые ресурсы загружаются один раз при первой загрузке страницы, а затем контент обновляется динамически без перезагрузки страницы. Это позволяет создавать более быстрые и отзывчивые приложения, которые предлагают пользователю более плавный и приятный опыт использования [10]).

SSR (Server-Side Rendering) – это метод, при котором web-страница создается на сервере и отправляется пользователю как готовый HTML-код. Это позволяет улучшить производительность и SEO-оптимизацию web-приложения [11]).

Базы данных играют ключевую роль в хранении и управлении данными, необходимыми для работы web-приложения. Разработчики могут использовать различные типы баз данных, такие как SQL (PostgreSQL, MySQL) или NoSQL (MongoDB, Redis), в зависимости от требований проекта.

API (Application Programming Interface) – это набор инструкций и структур данных, которые позволяют взаимодействовать между различными компонентами программного обеспечения. В web-приложениях API используется для обмена данными между клиентской и серверной частями, а также для интеграции с внешними сервисами и сторонними приложениями.

3.3. Обзор и описание используемых технологий и инструментов разработки

При разработке web-приложения использовались разнообразные инструменты и технологии, которые позволили эффективно и быстро создать готовое приложение.

Одним из основных инструментов разработки был Visual Studio Code (VScode) – удобный редактор кода, предоставляющий широкий набор функций и инструментов для комфортной работы. Его удобство использования, быстроедействие и наличие разнообразных расширений делают его одним из самых популярных редакторов среди разработчиков. Также его интеграция с Git позволяет легко управлять версиями кода, проводить слияния и коммиты прямо из редактора, что упрощает процесс разработки.

Для хранения и управления данными использовалась Supabase – открытая платформа для разработки web-приложений на основе PostgreSQL. Supabase предоставляет пользователям облачную инфраструктуру и готовые инструменты для работы с базами данных, аутентификации пользователей и автоматизации некоторых задач.

Для реализации монетизации через подписки была интегрирована платежная система Stripe. Stripe предоставляет простой и удобный API для приема платежей онлайн, обеспечивая безопасность и надежность операций.

Его инструменты позволяют легко настраивать различные виды платежей и управлять подписками пользователей.

В процессе разработки были использованы различные языки, фреймворки и библиотеки, каждая из которых отвечает за определенные аспекты разработки приложения:

- HTML (HyperText Markup Language) – язык разметки, используемый для создания структуры web-страниц. Он предоставляет набор элементов и тегов, с помощью которых можно определить содержимое и структуру страницы, такие как заголовки, параграфы, изображения, ссылки и другие разнообразные элементы [12]);

- CSS (Cascading Style Sheets) – язык таблиц стилей, который позволяет задавать внешний вид и оформление web-страниц. С помощью CSS можно определять цвета, шрифты, размеры, расположение элементов, обеспечивая единый и стильный дизайн для всего web-приложения [13]);

- Tailwind CSS – это CSS-фреймворк, который позволяет создавать пользовательские интерфейсы при помощи набора готовых классов. Он был разработан для ускорения процесса разработки и облегчения поддержки кода. С помощью данного фреймворка можно писать CSS-стили прямо в файлах компонентов, не создавая отдельные файлы для стилизации. Основное преимущество Tailwind CSS заключается в том, что он предоставляет большой набор готовых классов. Это значительно сокращает время на создание пользовательского интерфейса, так как не нужно писать каждый раз новые стили для каждого элемента. Кроме того, использование готовых классов позволяет легко и быстро изменять внешний вид элементов, не затрагивая их структуру [14]);

- JavaScript (JS) – это многофункциональный язык программирования, который используется для добавления интерактивности и динамичности на web-страницах. Он позволяет, к примеру, создавать

анимации, обрабатывать события, взаимодействовать с пользователем, делая web-приложения более функциональными и привлекательными [15]);

- TypeScript (TS) – это надстройка (синтаксическое расширение) для JavaScript, которое обеспечивает статическую типизацию для улучшения разработки приложений и избегания ошибок. Он позволяет выявлять ошибки на этапе разработки, улучшать поддержку кода и повышать производительность разработки [16]);

- ReactJS – это JavaScript-библиотека для создания реактивных web-приложений, которая обеспечивает эффективную и масштабируемую разработку web-приложений. Она основана на компонентной архитектуре, что делает ее удобной и гибкой для создания сложных пользовательских интерфейсов. Основная идея и концепция ReactJS – это переиспользуемые компоненты, которые всегда можно взять и вставить в той части приложения, где этот компонент необходим [17]);

- Next.js – это фреймворк React для создания статических и динамических web-приложений, который обеспечивает удобную разработку и оптимизацию производительности. Он предоставляет многофункциональные инструменты для рендеринга и маршрутизации, а также интеграцию с серверным рендерингом (SSR) для распределения нагрузки и реализации модели клиент-сервер, что делает приложение более быстрым за счет распределения нагрузки [18]).

4. Разработка web-сервиса «SoundScape»

4.1. Интерфейс

При разработке интерфейса были учтены все главные аспекты, которые удалось выявить при анализе существующих web-приложений по прослушиванию музыки. Таким образом, получилось создать наглядный и интуитивно понятный интерфейс приложения, который не должен отталкивать потенциальных пользователей. Первые наброски интерфейса можно увидеть на рисунке 1.

В первую очередь была продумана боковая панель, разделенная на два блока. Первый блок отвечает непосредственно за навигацию по web-приложению. На нем расположены кнопки Home, Search и News, при нажатии на которые пользователь будет перенаправляться в соответствующий раздел. Второй блок отвечает за загрузку музыкальных композиций на платформу. В верхнем правом углу располагается кнопка добавления, при нажатии на которую должно всплывать модальное окно для загрузки музыкальной композиции на платформу. Все остальные элементы второго блока – это уже загруженные на платформу композиции.

Справа от боковой панели находится основная страница, которая интуитивно так же поделена на два блока – верхний и нижний.

Верхний блок представляет «шапку» основной страницы. На ней находятся кнопки переключения страниц. Ниже находится надпись с динамическими приветствиями, которые случайным образом выбираются и показываются при заходе в приложение. Под приветствиями находится кнопка Favourites, при нажатии на которую пользователь попадает в свою библиотеку. В верхнем правом углу находятся кнопки входа и регистрации, а также кнопка личного кабинета.

В нижнем блоке отображаются альбомы и музыкальные композиции, которые были загружены или созданы в приложении.

Прототип плейлиста включал в себя кнопку воспроизведения, кнопки переключения музыкальных композиций, ползунков для регулировки

громкости воспроизведения музыки, а также отображение изображения и названия, прикрепленных к текущей проигрываемой композиции. Помимо этого, плеер включает в себя еще две функциональные кнопки, первая отвечает за открытие модального окна с текстом, относящимся к текущей проигрываемой композиции, а вторая – за перемешивание порядка воспроизведения музыки.

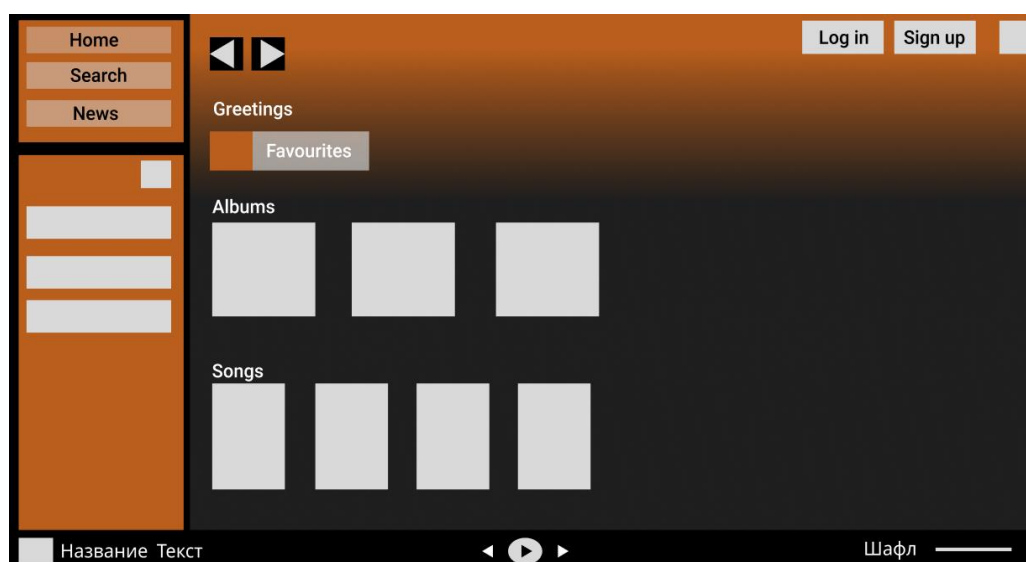


Рисунок 1 – Разработка интерфейса

Конечная реализация интерфейса главной страницы показана на рисунке 2. Все задуманные элементы полностью удалось реализовать, даже с небольшими улучшениями.

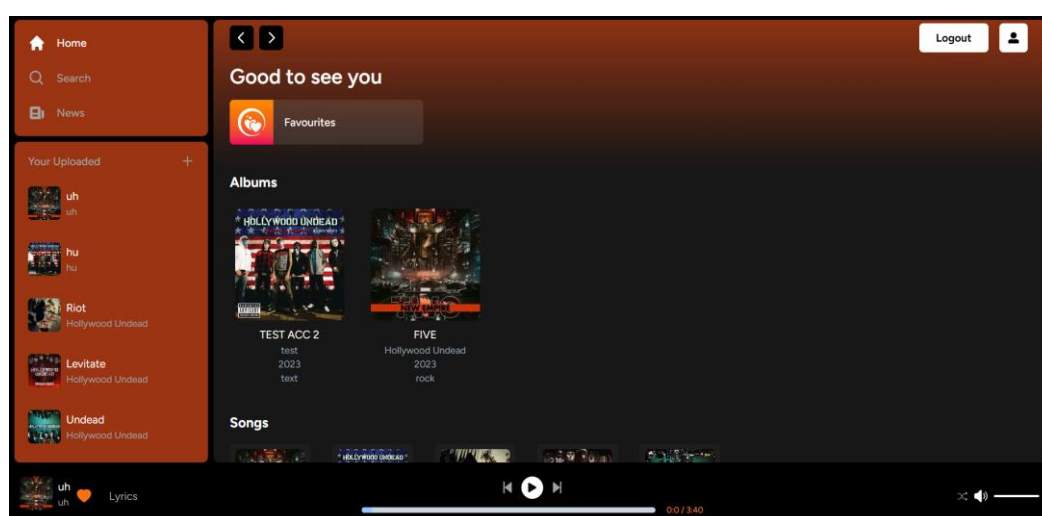


Рисунок 2 – Интерфейс главной страницы

Интерфейс страницы Search показан на рисунке 3. На данной странице реализовано поле поиска музыкальных композиций. Изначально на странице выводятся все композиции, загруженные в приложение. Также любую композицию можно добавить к себе в библиотеку, нажав на знак «сердечка».

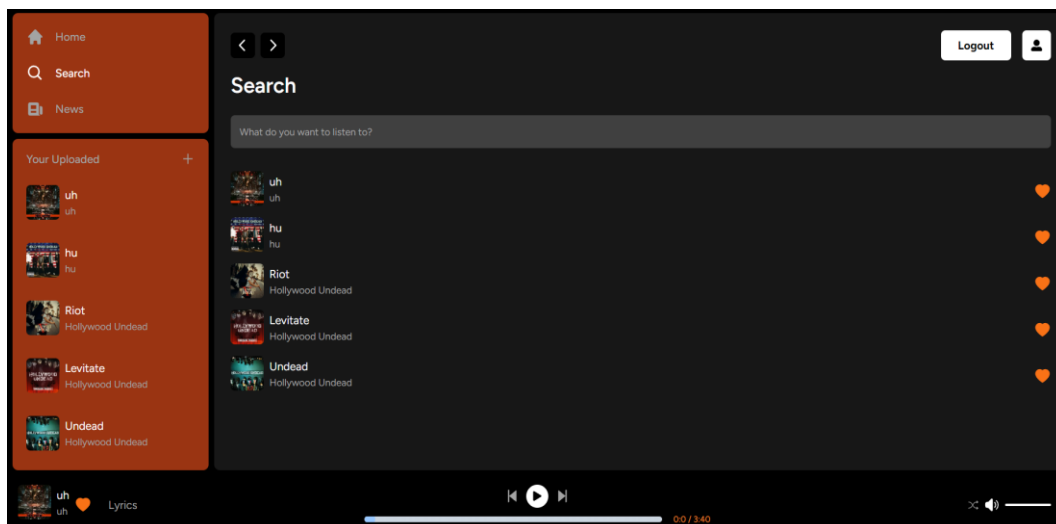


Рисунок 3 – Интерфейс страницы Search

Интерфейс страницы News показан на рисунке 4. На данной страницы отображаются последние новости из категории музыки и шоу-бизнеса, полученные через News API.

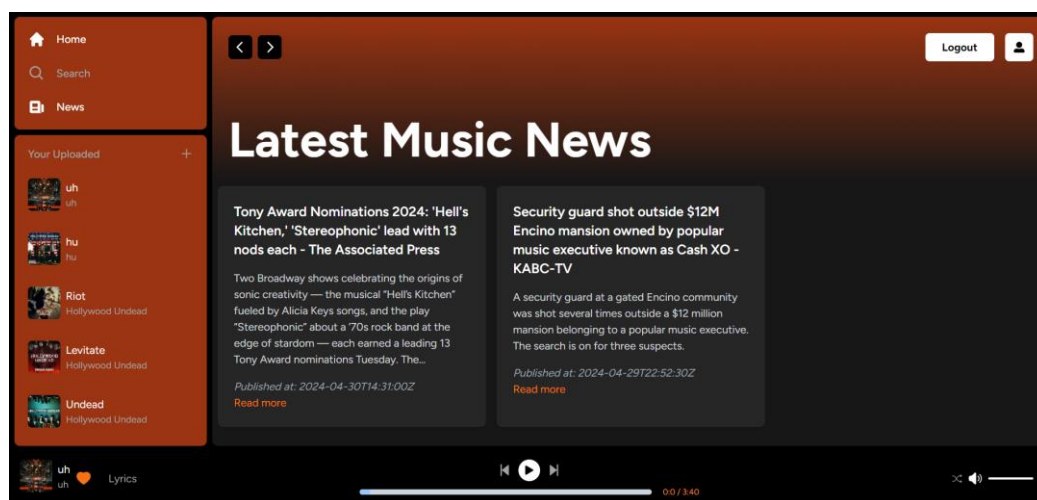


Рисунок 4 – Интерфейс страницы News

Интерфейс личного кабинета показан на рисунке 5. Здесь есть одна единственная кнопка, которая отвечает за перенаправление пользователя на портал для оплаты подписки.

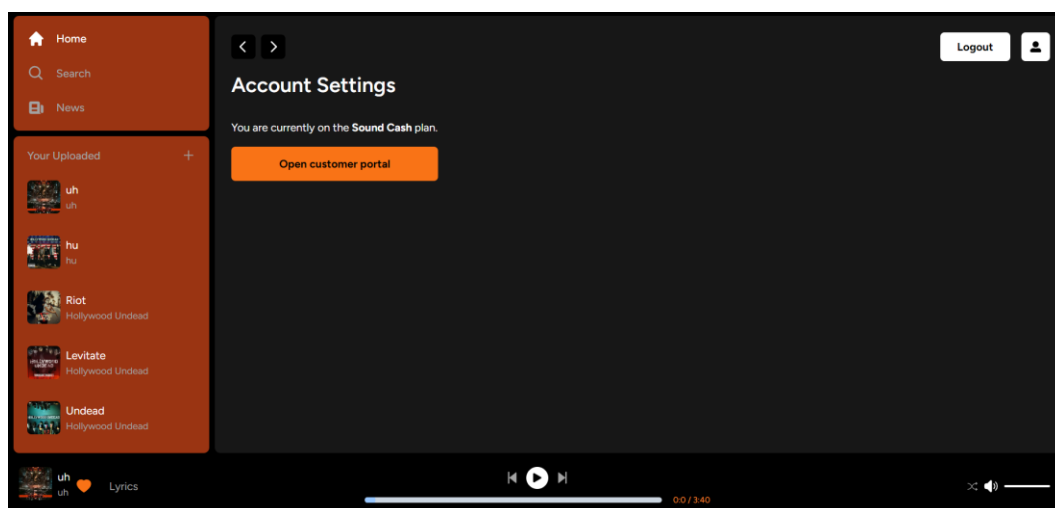


Рисунок 5 – Интерфейс личного кабинета

4.2. Структура приложения

Структура приложения представлена на рисунке 6.

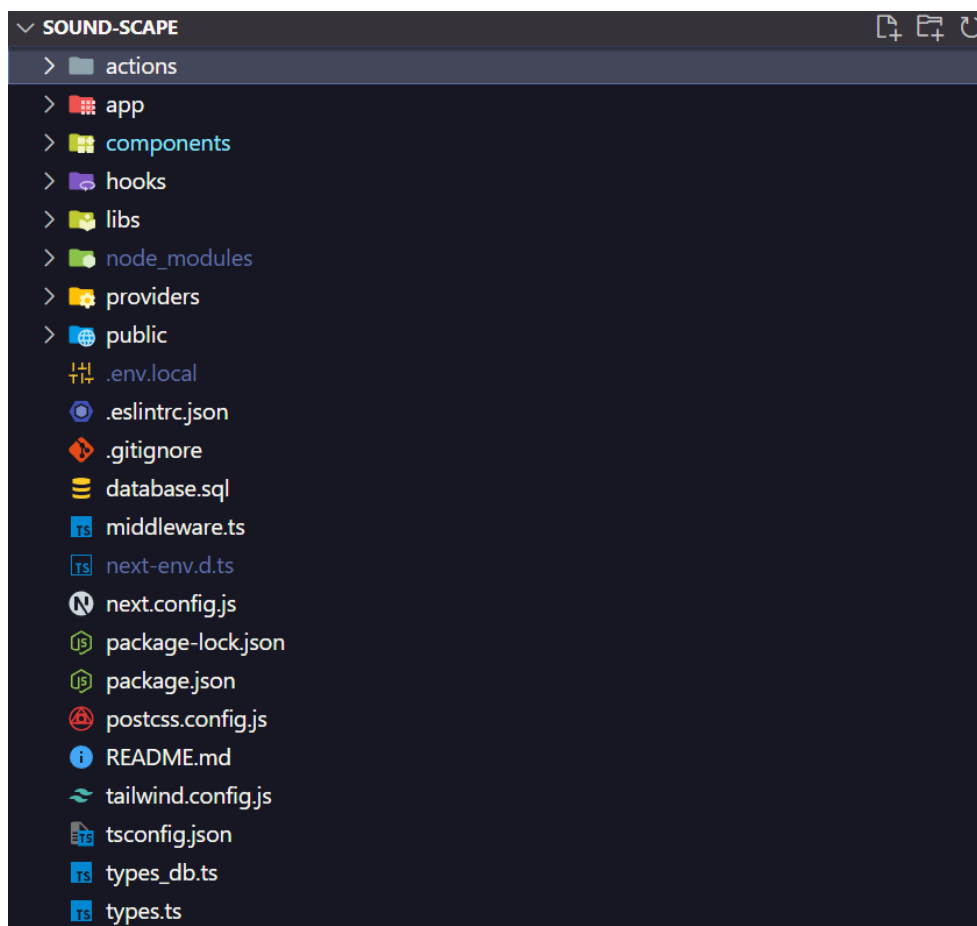


Рисунок 6 – Структура приложения

Краткое описание структуры приложения:

- в папке `actions` содержатся файлы, отвечающие за действия и операции, выполняемые в приложении, в частности через «экшены» происходит взаимодействие с базой данных;
- в папке `app` находятся основные компоненты и файлы, связанные с логикой и отображением приложения, такие как страницы, макеты, и компоненты, относящиеся непосредственно к одной странице;
- в папке `components` расположены переиспользуемые компоненты интерфейса, которые используются на разных страницах и в разных частях приложения;
- в папке `hooks` содержатся пользовательские хуки, которые предоставляют удобные методы для работы с состоянием и эффектами;
- в папке `libs` находятся вспомогательные библиотеки и утилиты, которые используются в различных частях приложения;
- в папке `node_modules` хранятся зависимости проекта, установленные с помощью `npm`;
- в папке `providers` содержатся файлы, связанные с контекстами и провайдерами данных;
- в папке `public` расположены статические ресурсы, такие как изображения в формате `.png` и `.svg`, доступные непосредственно из браузера;
- файл `env.local` содержит ссылки и секретные ключи для подключения к сервисам `supabase` и `stripe`;
- файл `.eslintrc.json`: является конфигурационным файлом `ESLint`, определяющим правила и настройки для проверки и форматирования кода;
- файл `.gitignore` содержит список файлов и папок, которые должны быть проигнорированы системой контроля версий `Git`;
- файл `database.sql` является `SQL`-скриптом, описывающим структуру и схему базы данных приложения. Генерируется автоматически через запрос к `supabase`;

- файл `middleware.ts` содержит промежуточные функции и операции, выполняемые перед или после обработки запросов web-приложением;
- `next-env.d.ts` – файл, предоставляющий определения типов для среды разработки Next.js;
- `next.config.js` – конфигурационный файл Next.js, определяющий настройки и параметры сборки и развертывания приложения;
- `package-lock.json` – файл, содержащий точные версии зависимостей, установленных с помощью npm;
- `package.json` является основным файлом конфигурации проекта npm, содержащим список зависимостей, скрипты и другие метаданные;
- `postcss.config.js` – конфигурационный файл PostCSS, определяющий настройки и плагины для обработки CSS;
- файл `README.md` содержит описание и инструкции по использованию и настройке проекта;
- `tailwind.config.js` – конфигурационный файл Tailwind CSS, определяющий пользовательские настройки и переменные для фреймворка;
- `tsconfig.json` – конфигурационный файл TypeScript, определяющий настройки компиляции и типизации для проекта;
- `types_db.ts` – файл, содержащий определения типов данных, используемых в базе данных;
- `types.ts` – файл, содержащий пользовательские типы данных и интерфейсы, используемые в приложении.

4.3. Компонент Sidebar

В самом начале необходимо было сделать боковую панель (Sidebar), так как данный компонент должен быть виден всегда и на любой странице.

Сам компонент был подразделен на две части, в верхней части находится навигационная панель, в которой есть ссылки на страницы Home, Search и News. При клике на конкретную ссылку происходит загрузка

соответствующей страницы через ее путь. Активная вкладка проверяется при помощи получения пути из «хука» `usePathname` и подсвечивается.

В нижней части находится библиотека загруженных пользователем музыкальных композиций на платформу. При клике на любую композицию она воспроизводится. Также при наведении на любую композицию справа от нее появляется знак «крестик», при нажатии на который композиция удаляется из приложения при помощи передачи `id` песни в запрос к базе данных на удаление. При нажатии на знак «плюс» открывается модальное окно для добавления композиции, куда пользователь вводит название песни, исполнителя, выбирает обложку и загружает саму композицию. После нажатия на кнопку `Create` формируется запрос на создание записи в базе данных. Код компонента `Sidebar` находится в приложении 1.

4.4. Основная страница приложения

Рендеринг основной страницы происходит на сервере, поэтому основная страница является серверным компонентом. Для данной страницы были созданы компоненты `Header`, `ListItem` и `PageContent`.

В компоненте `Header` были реализованы кнопки пагинации, с помощью которых можно двигаться вперед-назад по страницам в приложении при помощи методов роутинга из `Next.js`. Также были реализованы кнопки авторизации и регистрации, выхода из аккаунта и попадания в профиль. При нажатии на кнопку регистрации или авторизации у пользователя всплывает модальное окно, куда он может ввести свою почту и пароль для входа или регистрации. Также сервис `supabase` предоставляет возможность отправки `magic link` и функционал `forgot password`.

В компоненте `ListItem` реализована ссылка для перенаправления в библиотеку пользователя.

`PageContent` отображает основной контент страницы, в частности альбомы и музыкальные композиции. При первом рендеринге страницы все альбомы и песни подгружаются из базы данных при помощи серверных

«экшенов» `getSongs` и `getAllAlbums`, после чего передаются в `PageContent` для последующего отображения.

Также на странице отображаются приветствия, которые случайным образом выбираются из готового списка.

Код основной страницы находится в приложении 2.

4.5. База данных

Для того, чтобы база данных работала с приложением, необходимо ее с данным приложением связать. Это было сделано в файле `.env.local`, куда были прописаны секретные ключи для подключения к базе данных.

Далее был запущен готовый скрипт, предоставленный сервисом `supabase`, который сгенерировал таблицы `customers`, `prices`, `products`, `subscriptions` и `users`. Все таблицы, кроме `users`, необходимы для дальнейшей интеграции сервиса подписок `stripe`. Таблица `users` отвечает непосредственно за пользователей, зарегистрированных в приложении.

Далее необходимо было создать таблицу для музыкальных композиций. Была создана таблица `songs`, хранящая данные о времени загрузки композиции на платформу, названии композиции, исполнителе, пути к файлу композиции и пути к обложке композиции. Данная таблица привязывается к конкретному пользователю через поле `user_id` таблицы `users`. Так же для таблицы `songs` при удалении был выбран тип `Cascade`, чтобы при удалении пользователя автоматически удалялись все треки, загруженные им на платформу.

Аналогичным образом создавались таблицы `albums` и `playlists`. Для этих таблиц необходимо было также создать связующие таблицы `album_songs` и `playlist_songs`, в которых хранятся `id` музыкальной композиции и `id` альбома или плейлиста, к которому она привязана.

Для добавления функциональности понравившихся альбомов и песен были созданы `liked_albums` и `liked_songs`, в которых хранятся `id` музыкальных композиций и альбомов, которые понравились пользователю.

Для организации хранения файлов музыкальных композиций и файлов их обложек, были созданы два «бакета» с соответствующими названиями songs и images.

Для того, чтобы приложение понимало и определяло базу данных и текущего пользователя, были созданы два провайдера: SupabaseProvider и UserProvider. Они «оборачивают» все приложение, чтобы в любом месте была возможность работы с базой данных и возможность получения текущего пользователя.

Подробную схему базы данных можно увидеть на рисунке 7 и рисунке 8.

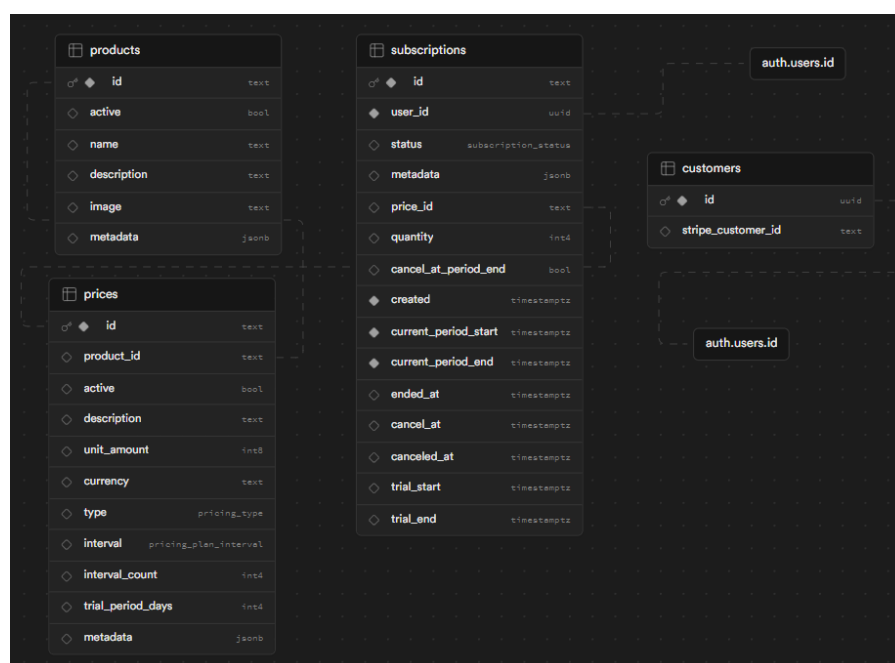


Рисунок 7 – Схема подписок

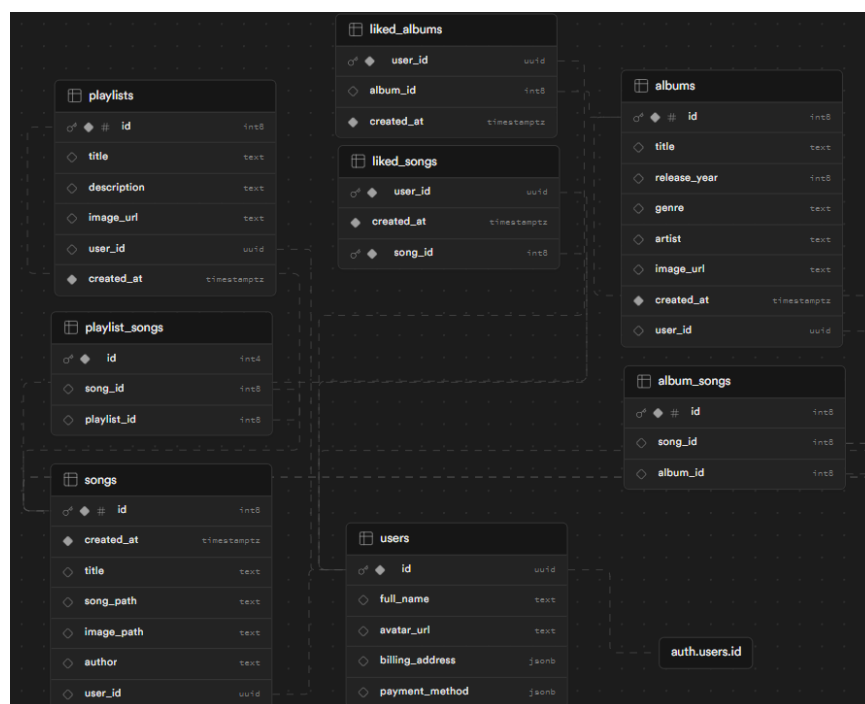


Рисунок 8 – Схема хранения данных

Код провайдеров SupabaseProvider и UserProvider представлен в приложении 3.

4.6. Модальное окно

Для реализации окна авторизации, регистрации, добавления музыкальных композиций и альбомов с плейлистами был разработан компонент модального окна Modal.

Пример модального окна для создания альбома можно увидеть на рисунке 9.

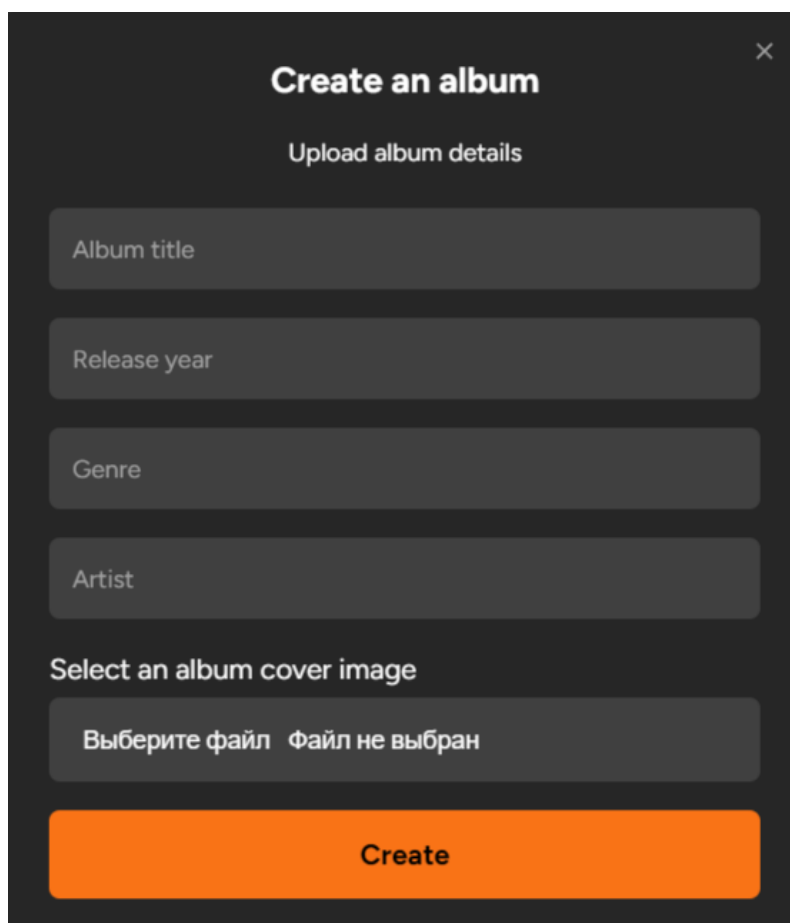


Рисунок 9 – Окно создания альбома

В данный компонент в зависимости от окружения и контекста подаются различные параметры. Все дополнительные поля ввода или, например, загрузки музыкальной композиции передаются через параметр `children`, после чего отображаются в модальном окне. Так же модальное окно реагирует на открытие или закрытие при помощи переменной `isOpen`, а при помощи функции `onChange` изменяет это состояние. У каждого модального окна есть название и описание, которые передаются как параметры из внешнего контекста в переменные `title` и `description` соответственно.

Код данного компонента представлен в приложении 4.

4.7. Страница Search

Самая главная часть страницы Search – это непосредственно сам поиск.

Изначально, при первой загрузке страницы, из базы данных подгружаются все музыкальные композиции при помощи «экшена»

getSongsByTitle, после чего они все отображаются на странице. Код данного «экшена» находится в приложении 5.

Сам поиск песен происходит с помощью компонента SearchInput. Код данного компонента можно найти в приложении 6. В поле ввода вписывается название песни, после небольшой задержки «хук» useEffect отлавливает изменение значения в Input, создает новый url адрес, в котором находится сама страница search и название музыкальной композиции, которую ищет пользователь. После создания нового url-адреса он подается как параметр в метод push объекта router, который отвечает за навигацию и маршрутизацию по приложению. Так что, главная суть поиска состоит в том, чтобы создать новый url-адрес с названием песни и перенаправить по нему пользователя.

Код данной страницы находится в приложении 7.

4.8. Страница News

Страница News состоит из двух основных компонентов – Header и NewsContent. Код данной страницы находится в приложении 8.

Header как компонент имеет такой же функционал как и на всех остальных страницах – это кнопки пагинации, возможность выйти из аккаунта и кнопка профиля. Единственно нововведение на данной странице – это статическая надпись.

При открытии страницы News все новости загружаются с помощью «экшена» getNews, код которого находится в приложении 9. Данный экшен представляет собой загрузку данных с открытого API под названием NewsAPI. Подключение происходит с помощью секретного ключа, полученного в самом сервисе, после чего отправляется axios-запрос на обозначенную категорию, с расширением в виде указания страны. Ответ приходит в виде объекта, который содержит в себе название, описание, время публикации и ссылку на первоисточник.

После загрузки новостей они в виде объекта передаются через параметр в компонент NewsContent, который отображает эти новости в отдельных оформленных блоках. При клике на ссылку с надписью Read more

пользователь попадает в первоисточник. Код данного компонента находится в приложении 10.

4.9. Библиотека пользователя

При открытии страницы Liked первым делом загружаются все понравившиеся альбомы, все понравившиеся песни, все плейлисты и все альбомы пользователя. Происходит это при помощи «экшенов» `getLikedSongs`, `getLikedAlbums`, `getPlaylists`, `getAlbums`. Во всех них реализованы операции `SELECT` из требуемой таблицы при помощи `id` текущего пользователя.

После загрузки всех альбомов и плейлистов загружаются все песни для всех альбомов и плейлистов при помощи «экшенов» `getSongsForPlaylist` и `getSongsForAlbum`, код которых находится в приложении 11. Загрузка происходит поочередно для каждого плейлиста и альбома при помощи их `id`. В итоге возвращается массив объектов, где хранятся `id` и музыкальные композиции конкретного альбома или плейлиста.

После загрузки всех необходимых данных, они в качестве параметров передаются в компонент `LikedContent`, код которого находится в приложении 12. В данном компоненте все песни, альбомы и плейлисты отображаются в соответствующих созданных для этого компонентах. При нажатии на альбом или плейлист открывается модальное окно соответствующего плейлиста или альбома, подробнее про них написано в подразделе 4.10.

На данной странице реализованы две кнопки создания альбома или плейлиста. При нажатии на любую из них открывается компонент модального окна с настройками под альбом или под плейлист, переданными в него. Данный компонент был описан в подразделе 4.6.

Справа от каждой песни отображаются два компонента – это `MusicMenu` и `LikeButton`.

Компонент `MusicMenu` реализует логику добавления песни в альбом или плейлист. При нажатии на данный компонент перед пользователем появляется небольшое меню, где он может выбрать тот альбом или плейлист, в который хочет добавить данную музыкальную композицию. После выбора альбома или

плейлиста через запрос на INSERT добавляется запись в таблицу album_songs или playlist_songs. При этом, при нажатии на компонент MusicMenu, сначала через запрос SELECT проверяется, есть ли данная музыкальная композиция в каком-либо плейлисте или альбоме и если есть – альбом или плейлист не отображаются в меню. Код данного компонента находится в приложении 13.

Компонент LikeButton представляет собой кнопку, при нажатии на которую композиция удаляется или добавляется в библиотеку пользователя. При нажатии на нее в библиотеке, она меняет свой цвет с ярко-оранжевого на прозрачный, после отправляется запрос в базу данных, который по id песни, к которой привязана кнопка удаляет ее из таблицы liked_songs и компонент данной песни пропадает со страницы. Код данного компонента находится в приложении 14.

4.10. Альбомы и плейлисты

AlbumModal представляет собой модальное окно, которое отображает информацию об альбоме, включая его обложку, название, исполнителя, год выпуска, жанр, список треков и различные действия.

Вид модального окна альбома можно увидеть на рисунке 10.

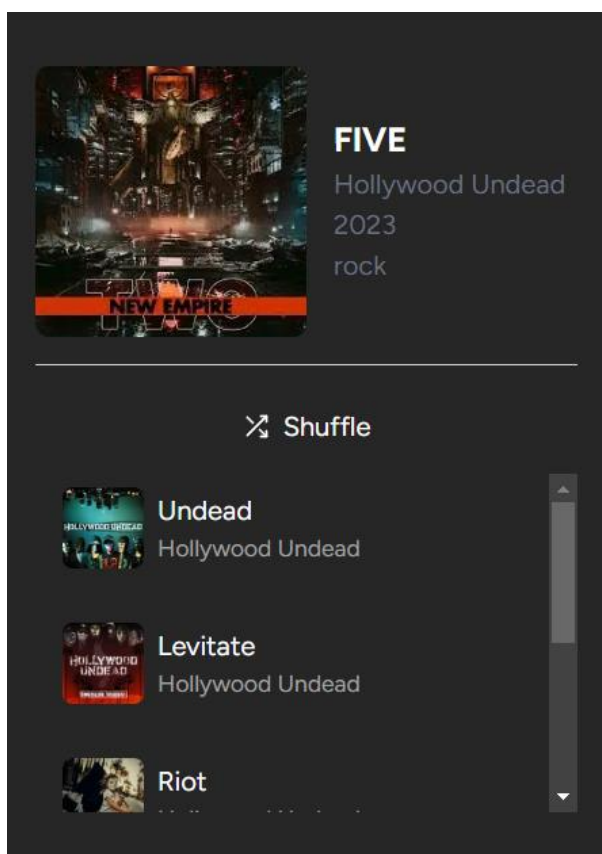


Рисунок 10 – Модальное окно альбома

В данном компоненте было реализовано следующее:

- 1) отображение информации об альбоме:
 - используется компонент Image из библиотеки next/image для отображения обложки альбома;
 - для вывода остальной информации, такой как название альбома, исполнитель, год выпуска и жанр, просто используются соответствующие данные из объекта album, переданного в компонент;
- 2) загрузка треков альбома:
 - при монтировании компонента AlbumModal выполняется SELECT запрос к базе данных с использованием объекта supabase для получения списка треков альбома;
 - запрос выполняется к таблицам album_songs и songs, чтобы получить треки, привязанные к данному альбому;
- 3) управление воспроизведением:

- используется «хук» `useOnPlay`, который позволяет управлять воспроизведением треков;

- при нажатии на кнопку "Play" вызывается функция `onPlay`, которая запускает воспроизведение первого трека из списка `playableSongs`. В данный список попадают все треки открытого альбома;

4) управление списком треков:

- для удаления трека из списка используется функция `handleRemoveSong`, которая удаляет трек из альбома с помощью `DELETE` запроса к базе данных и обновляет список `playableSongs`;

- запрос делается к таблице `album_songs` с помощью `id` музыкальной композиции и `id` альбома;

- при наведении на трек появляется кнопка удаления, которая реагирует на событие `onClick` и вызывает функцию `handleRemoveSong`;

5) действия с альбомом:

- для добавления альбома в избранное используется функция `handleAddAlbum`, которая добавляет запись о альбоме в таблицу `liked_albums`;

- для удаления альбома из избранного используется функция `handleUnFavouriteAlbum`, которая удаляет запись о альбоме из таблицы `liked_albums`;

- при нажатии на кнопку удаления альбома вызывается функция `handleRemoveAlbum`, которая удаляет альбом целиком из таблицы `albums` по `id` альбома и `id` пользователя;

6) проверка наличия альбома в избранном и в коллекции пользователя:

- при монтировании компонента происходит проверка, добавлен ли альбом в избранное пользователя и находится ли он в его коллекции;

- для этого выполняется `SELECT` запрос к таблице `liked_albums`;

- проверка проводится для определения, нужно ли отображать кнопку добавления/удаления альбома из избранного или коллекции;

- 7) управление перемешиванием треков:
 - при нажатии на кнопку "Shuffle" происходит перемешивание списка воспроизведения треков;
 - функция `handleShuffleClick` перемешивает список `playableSongs` и устанавливает первый трек в качестве текущего;
 - перемешивание выполняется с использованием метода `sort` и функции, которая случайным образом меняет порядок элементов в массиве;
- 8) наведение на элементы:
 - при наведении на треки или обложку альбома появляются дополнительные элементы интерфейса, такие как кнопки воспроизведения и удаления;
 - для этого используются состояния `isHovered`, `hoveredSongId` и `isAlbumHovered`, которые устанавливаются при событиях `onMouseEnter` и `onMouseLeave`;
- 9) удаление треков:
 - при нажатии на кнопку удаления трека вызывается функция `handleRemoveSong`;
 - функции выполняют `DELETE` запросы к базе данных для удаления соответствующих записей из таблиц;
 - после удаления данных из базы происходит обновление состояния компонента;
- 10) визуальное отображение подтверждений:
 - при удалении альбома или трека появляются модальные окна с вопросами о подтверждении;
 - для этого используются состояния `showRemoveConfirmation` и `showSongRemoveConfirmation`, которые управляют видимостью модальных окон;
 - подтверждение действия выполняется при нажатии на кнопки "Yes" или "Cancel".

Компонент плейлиста PlaylistModal имеет аналогичное представление, за исключением функционала добавления или удаления его из избранного, так как плейлист создается конкретным пользователем для себя и не отображается на главной странице.

Код компонента AlbumModal представлен в приложении 15.

4.11. Плеер

В первую очередь были разработаны два «хука» для управления состояниями и воспроизведением – useOnPlay и usePlayer. Код useOnPlay представлен в приложении 16, а usePlayer – в приложении 17.

useOnPlay используется для управления воспроизведением треков и работает следующим образом:

1) «хук» useOnPlay принимает список треков songs в качестве аргумента и создает экземпляры других хуков, таких как usePlayer, useSubscribeModal, useAuthModal и useUser, которые понадобятся для управления воспроизведением;

2) главной задачей «хука» useOnPlay является предоставление функции onPlay, которая вызывается при нажатии на кнопку воспроизведения трека. Внутри этой функции происходит следующее:

- проверка аутентификации пользователя. Если пользователь не аутентифицирован, открывается модальное окно аутентификации;
- проверка подписки. Если пользователь аутентифицирован, но не подписан на услугу, открывается модальное окно подписки;
- установка текущего трека и списка треков. Если пользователь аутентифицирован и подписан, вызывается метод setId из «хука» usePlayer, чтобы установить текущий трек для воспроизведения. Также устанавливается список всех треков для воспроизведения с помощью метода setIds;

3) «хук» useOnPlay возвращает функцию onPlay, которая будет вызываться в компоненте для запуска воспроизведения треков.

usePlayer отвечает за управление состоянием плеера и работает следующим образом:

1) «хук» `usePlayer` создает хранилище состояний с помощью библиотеки `Zustand`. В состоянии хранятся следующие данные:

- `Ids` – Массив идентификаторов всех треков;
- `activeId` – Идентификатор текущего активного трека;
- `isShuffle` – Флаг, который указывает, включен ли режим случайного воспроизведения;

2) методы управления состоянием представляют собой:

- `setId` – Устанавливает идентификатор текущего активного трека;
- `setIds` – Устанавливает массив идентификаторов всех треков;
- `toggleShuffle` – Переключает режим случайного воспроизведения;
- `Reset` – Сбрасывает все состояния плеера в исходное состояние;

3) `usePlayer` возвращает экземпляр `Zustand`, содержащий состояние и методы управления этим состоянием. Этот экземпляр используется другими компонентами для получения доступа к состоянию плеера и его управлению.

Сам плеер представляет из себя компонент `PlayerContent`, который обладает следующим функционалом:

1) визуальные элементы и кнопки управления:

- плеер содержит кнопки управления воспроизведением, то есть кнопки `Play/Pause`, `Previous/Next`, а также ползунок времени воспроизведения и кнопки управления громкостью;

- используются иконки из библиотеки `react-icons`, которые представляют собой готовые изображения для визуализации кнопок управления;

2) состояние плеера:

- Для отслеживания состояния плеера используется «хук» `useState`. Например, состояние `isPlaying` отвечает за текущее состояние воспроизведения;

- также используются состояния для хранения информации о текущем времени воспроизведения (`currentTime`), продолжительности трека (`duration`) и уровне громкости (`volume`);

3) управление воспроизведением:

- для воспроизведения аудио используется «хук» `useSound` из библиотеки `use-sound`. Он предоставляет функции `play` и `pause`, которые позволяют управлять воспроизведением звука;

- ползунок времени воспроизведения позволяет пользователю перемещаться по треку. Обработчик `handleSliderChange` обновляет текущее время воспроизведения трека в соответствии с положением ползунка;

4) дополнительные функции:

- кнопка "Lyrics" вызывает функцию `openModal`, которая открывает модальное окно с текстом песни. Для получения текста песни используется `API Lyrics.ovh`;

- пользователь может переключить режим случайного воспроизведения с помощью кнопки "Shuffle", которая вызывает метод `toggleShuffle` из «хука» `usePlayer`.

Помимо всего этого в плеере отображается текущее время трека, обновляемое и получаемое из значения `duration`. Также в плеере отображается обложка текущего проигрываемого трека, его название и исполнитель. Текущий проигрываемый трек можно прямо из плеера добавить к себе в библиотеку, нажав на компонент `LikeButton`.

4.12. Страница аккаунта и интеграция Stripe

Для интеграции с сервисом `stripe` необходимо было зарегистрироваться в данном сервисе, после чего были предоставлены ключи для подключения, которые были прописаны в файле `.env.local`.

Далее были разработаны вспомогательные мини-библиотеки для интеграции сервиса в приложение. Данные мини-библиотеки выполняют следующие функции:

1) stripe.ts:

- в данном файле создается экземпляр объекта Stripe с использованием секретного ключа Stripe, который хранится в переменной окружения STRIPE_SECRET_KEY;

- объект Stripe используется для взаимодействия с API Stripe для выполнения операций по созданию клиента, оформлению подписки и операций, связанных с платежами;

2) stripeClient.ts:

- в данном файле создается клиент Stripe для фронт-части с использованием публичного ключа Stripe, который доступен в переменной окружения файла .env.local;

- клиент Stripe используется для инициализации платежных форм на стороне клиента и для выполнения платежных операций без необходимости передачи секретных ключей на клиентскую сторону;

3) helpers.ts:

- функция getURL используется для получения базового URL-адреса сайта, который используется при создании ссылок и запросов;

- функция postData используется для отправки POST-запросов на сервер с использованием fetch API;

- функция toDateTime используется для преобразования времени в секундах в объект даты JavaScript;

4) supabaseAdmin.ts:

- в данном файле создается клиент Supabase Admin для взаимодействия с базой данных Supabase;

- функция upsertProductRecord используется для вставки или обновления записи о продукте (товаре) в базе данных Supabase на основе данных Stripe;

- функция upsertPriceRecord используется для вставки или обновления записи о цене в базе данных Supabase на основе данных Stripe;

- функция `createOrRetrieveCustomer` используется для создания нового клиента Stripe или получения существующего клиента на основе UUID пользователя;
- функция `copyBillingDetailsToCustomer` копирует информацию о платеже клиента Stripe в объект клиента Supabase;
- функция `manageSubscriptionStatusChange` изменяет статус подписки и делает обновление записей в базе данных Supabase.

После добавления библиотек был создан `api`-файл в папке `webhooks`, отвечающий за обновление информации о продуктах, ценах и статусах подписок в приложении:

- 1) обработка вебхуков Stripe:
 - вебхуки Stripe представляют собой уведомления, которые Stripe отправляет серверу о событиях, таких как создание, обновление или удаление продуктов, цен и подписок;
 - данный файл создан для принятия и обработки POST-запросов от Stripe вебхуков;
- 2) проверка подлинности запроса:
 - в данном файле используется `Stripe-Signature`, чтобы проверить подлинность запроса и убедиться, что запрос действительно пришел от Stripe;
 - это делается с использованием метода `stripe.webhooks.constructEvent`, который проверяет подпись запроса;
- 3) обработка событий:
 - данный файл содержит функцию POST, которая обрабатывает типы событий Stripe, в частности создание или обновление продуктов, цен и подписок;
- 4) вызов функций из других файлов:
 - данный файл вызывает функции из файлов `supabaseAdmin.ts` и `stripe.ts`, чтобы взаимодействовать с базой данных Supabase и объектом Stripe;
- 5) получение ответа от сервера:

- после успешной обработки вебхука сервер возвращает ответ с кодом состояния 200 и сообщением о получении вебхука;
- в случае ошибки сервер возвращает код состояния 400 и сообщение об ошибке.

Далее был создан еще один api-файл `route.ts` в папке `create-checkout-session`. Данный файл отвечает за обработку запросов на создание сеанса оформления заказа Stripe:

- 1) создание сеанса оформления заказа:
 - файл `route.ts` обрабатывает POST-запросы, которые содержат информацию о цене продукта, количестве и метаданных;
 - данный тип запроса используется для создания сеанса оформления заказа Stripe для подписки на продукт;
- 2) получение информации о пользователе:
 - используется функция `createRouteHandlerClient` для создания клиента Supabase для обработки запросов аутентификации;
 - затем извлекается информация о текущем пользователе, включая идентификатор и электронную почту;
- 3) создание или получение клиента Stripe:
 - функция `createOrRetrieveCustomer` из файла `supabaseAdmin.ts` используется для создания нового клиента Stripe или получения существующего клиента на основе идентификатора пользователя и электронной почты;
- 4) создание сеанса оформления заказа Stripe:
 - используется метод `stripe.checkout.sessions.create` для создания сеанса оформления заказа Stripe с указанием типа оплаты, адреса для выставления счетов, товаров, режима (в данном случае подписки) и других параметров;
 - успешный URL перенаправляет пользователя на страницу аккаунта, а отмененный URL перенаправляет на главную страницу;

- 5) получение ответа от сервера:
 - в случае успеха сервер возвращает JSON-ответ с идентификатором сеанса оформления заказа;
 - в случае ошибки сервер возвращает статус 500 и сообщение "Internal Error".

Наконец, был создан последний api-файл `route.ts` в папке `create-portal-link`. Он используется для обработки запросов на создание ссылки на портал выставления счетов Stripe:

- 1) создание ссылки на портал выставления счетов:
 - файл `route.ts` обрабатывает POST-запросы без параметров;
 - данный тип запроса используется для создания ссылки на портал выставления счетов Stripe, который позволяет пользователям управлять своими подписками и платежными методами;
- 2) получение информации о пользователе:
 - используется функция `createRouteHandlerClient` для создания клиента Supabase для обработки запросов аутентификации;
 - затем извлекается информация о текущем пользователе, включая идентификатор и электронную почту;
- 3) создание или получение клиента Stripe:
 - функция `createOrRetrieveCustomer` из файла `supabaseAdmin.ts` используется для создания нового клиента Stripe или получения существующего клиента на основе идентификатора пользователя и электронной почты;
- 4) создание ссылки на портал выставления счетов Stripe:
 - используется метод `stripe.billingPortal.sessions.create` для создания ссылки на портал выставления счетов Stripe с указанием идентификатора клиента и URL-адреса для возврата после завершения процесса;
- 5) получение ответа от сервера:

- в случае успеха сервер возвращает JSON-ответ с URL ссылкой на портал выставления счетов;
- в случае ошибки сервер возвращает статус 500 и сообщение "Internal Error".

После добавления всех необходимых библиотек и сервисов была создана страница аккаунта Account, которая представляет из себя страницу с функционалом одной кнопки, которая выполняет роль перенаправления пользователя на портал текущей подписки:

- 1) перенаправление при отсутствии пользователя:
 - используется «хук» useEffect, чтобы проверить, загрузилась ли информация о пользователе, если пользователь не аутентифицирован происходит перенаправление на главную страницу;
- 2) открытие модального окна подписки:
 - если у пользователя отсутствует активная подписка, отображается кнопка "Подписаться", при нажатии на которую открывается модальное окно для оформления подписки;
- 3) открытие портала клиента:
 - если у пользователя есть активная подписка, отображается сообщение о текущем плане и кнопка "Открыть портал клиента";
 - при нажатии на кнопку "Открыть портал клиента" вызывается функция redirectToCustomerPortal, которая отправляет POST-запрос на сервер для создания ссылки на портал клиента Stripe;
 - затем происходит перенаправление пользователя на страницу портала клиента Stripe;
- 4) интерфейс:
 - компонент отображает информацию о текущем статусе подписки пользователя и предоставляет функциональность для подписки или доступа к portalу клиента в зависимости от этого статуса.

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы было разработано музыкальное web-приложение, обладающее всем необходимым функционалом для комфортного пользования, а также приведено описание процесса разработки. Все задачи, поставленные для достижения этой цели, решены, а именно:

- проведен анализ существующих приложений для выявления недостатков;
- разработан пользовательский интерфейс, который является достаточно наглядным и простым в использовании;
- спроектировано и реализовано музыкальное web-приложение, имеющее в себе весь необходимый функционал для комфортного использования;
- в приложение добавлены дополнительные возможности, которые делают его более привлекательным для пользователей, а также обеспечивают функциональность, ожидаемую большинством современных пользователей.

Созданное web-приложение имеет дальнейшие перспективы развития, ведь web-технологии совершенствуются и изменяются почти каждый день. В первую очередь планируется поменять структуру запросов, усовершенствовав их и переписав с помощью специальной библиотеки Tanstack Query. Также можно улучшить пользовательский интерфейс, обновив вид существующих компонентов, добавить новый функционал в приложение, такой как рекомендации или персональный поток музыки.

Данная тема остается значимой, несмотря на наличие аналогичных приложений, которые, возможно, имеют свои преимущества, но также обладают и недостатками. Всегда существует возможность усовершенствовать программные продукты, учитывая, что идеальных решений не существует.

За время выполнения курсовой работы были реализованы следующие компетенции:

Шифр компетенции	Расшифровка приобретаемой компетенции	Расшифровка освоения компетенции
УК-6	Способен управлять своим временем, выстраивать и реализовывать траекторию саморазвития на основе принципов образования в течение всей жизни	Процесс реализации музыкального web-сервиса был разбит по шагам, которые выполнялись в отведенное для них время.
ПК-4	Разработка требований и проектирование программного обеспечения	Были сформированы требования с подробным описанием функционала, интерфейса и основных характеристик приложения. Затем была разработана структура программного обеспечения и определены взаимодействия между его компонентами.
ПК-5	Оценка и выбор варианта архитектуры программного средства	Архитектура приложения была выбрана исходя из целей и требований проекта, что позволило оптимизировать процесс разработки и успешно реализовать проект.
ПК-6	Разработка тестовых случаев, проведение тестирования и исследование результатов	Были созданы тестовые сценарии для проверки функциональности музыкального web-сервиса и проведено тестирование для обнаружения и исправления ошибок. Это гарантировало высокое качество и надежность приложения.

СПИСОК ЛИТЕРАТУРЫ

- 1) The Spotify Community [Электронный ресурс]. – 2020. — URL: https://community.spotify.com/t5/Ongoing-Issues/idb-p/ongoing_issues (дата обращения 12.03.2024).
- 2) Что не так с Яндекс.Музыкой? [Электронный ресурс]. – 2021. — URL: <https://habr.com/ru/articles/446000/> (дата обращения 12.03.2024).
- 3) Сообщения об ошибках | VK Music [Электронный ресурс]. – 2021. — URL: https://vk.com/topic-86872453_31544393 (дата обращения 12.03.2024).
- 4) Troubleshooting | Deezer Community [Электронный ресурс]. – 2020. — URL: <https://en.deezercommunity.com/troubleshooting-46> (дата обращения 12.03.2024).
- 5) Soundcloud official website [Электронный ресурс]. – 2019. — URL: <https://soundcloud.com/ivegotproblems> (дата обращения 12.03.2024).
- 6) Сан Феликс М.Н. Разработка веб-приложений с Quarkus и React. / Под ред. Сан Феликс М.Н. – Packt, май 2023. – 294 с.
- 7) Как работают веб-приложения [Электронный ресурс]. – 2022. — URL: <https://habr.com/ru/articles/450282/> (дата обращения 14.03.2024).
- 8) Фронтенд-разработка: ключевые технологии и понятия [Электронный ресурс]. – 2021. — URL: <https://habr.com/ru/companies/otus/articles/674748/> (дата обращения 21.01.2024).
- 9) Основы Backend-разработки [Электронный ресурс]. – 2022. — URL: https://github.com/DEBAGanov/interview_questions/blob/main/ (дата обращения 10.02.2024).
- 10) Следующий этап развития Веба [Электронный ресурс]. – 2023. — URL: <https://habr.com/ru/companies/timeweb/articles/695798/> (дата обращения 02.02.2024).
- 11) Server-Side Rendering [Электронный ресурс]. – 2022. — URL: <https://habr.com/ru/articles/527310/> (дата обращения 02.02.2024).
- 12) htmlbook [Электронный ресурс]. – 2017. — URL: <https://htmlbook.ru/html> (дата обращения 17.01.2024).

13) CSS: Cascading Style Sheets [Электронный ресурс]. – 2021. — URL: <https://developer.mozilla.org/en-US/docs/Web/CSS> (дата обращения 17.01.2024).

14) Tailwind [Электронный ресурс]. – 2021. — URL: <https://v2.tailwindcss.com/docs> (дата обращения 20.01.2024).

15) JavaScript [Электронный ресурс]. – 2018. — URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (дата обращения 23.01.2024).

16) Documentation – Typescript for JavaScript Programmers [Электронный ресурс]. – 2020. — URL: <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html> (дата обращения 15.02.2024).

17) Documentation – ReactJS [Электронный ресурс]. – 2019. — URL: <https://react.dev/learn> (дата обращения 15.02.2024).

18) Documentation – Next.js [Электронный ресурс]. – 2021 — URL: <https://nextjs.org/docs> (дата обращения 08.03.2024).

Приложение 1. Код компонента Sidebar

```
"use client";

import { HiHome, HiNewspaper } from "react-icons/hi";
import { BiSearch } from "react-icons/bi";
import { twMerge } from "tailwind-merge";
import { usePathname } from "next/navigation";

import { Song } from "@types";
import usePlayer from "@hooks/usePlayer";

import SidebarItem from "../SidebarItem";
import Box from "../Box";
import Library from "../Library";
import { useMemo } from "react";

interface SidebarProps {
  children: React.ReactNode;
  songs: Song[];
}

const Sidebar = ({ children, songs }: SidebarProps) => {
  const pathname = usePathname();
  const player = usePlayer();

  const routes = useMemo(() => [
    {
      icon: HiHome,
      label: 'Home',
      active: pathname === '/',
      href: '/'
    },
    {
      icon: BiSearch,
      label: 'Search',
      href: '/search',
      active: pathname === '/search'
    },
    {
      icon: HiNewspaper,
      label: 'News',
      href: '/news',
      active: pathname === '/news'
    }
  ], [pathname]);

  return (
    <div
      className={twMerge(`
        flex
        h-full
        `)}
    >
```

```

        player.activeId && 'h-[calc(100%-80px)]'
      )}
    >
    <div
      className="
        hidden
        md:flex
        flex-col
        gap-y-2
        bg-black
        h-full
        w-[300px]
        p-2
      "
    >
    <Box>
      <div className="flex flex-col gap-y-4 px-5 py-4">
        {routes.map((item) => (
          <SidebarItem key={item.label} {...item} />
        ))}
      </div>
    </Box>
    <Box className="overflow-y-auto h-full">
      <Library songs={songs} />
    </Box>
  </div>
  <main className="h-full flex-1 overflow-y-auto py-2">
    {children}
  </main>
</div>
);
}

export default Sidebar;

```

Приложение 2. Код основной страницы приложения

```
import getSongs from "@actions/getSongs";
import Header from "@components/Header";
import ListItem from "@components/ListItem";
import PageContent from "../components/PageContent";
import greetings from "../greetings"
import getAllAlbums from "@actions/getAllAlbums";
import getSongsForAlbum from "@actions/getSongsForAlbum";

export const revalidate = 0;

export default async function Home() {
  const songs = await getSongs();
  const albums = await getAllAlbums();

  const albumSongs = await Promise.all(albums.map(async
(album) => {
    return {
      albumId: album.id,
      songs: await getSongsForAlbum(album.id)
    };
  }));

  const randomGreeting = greetings[Math.floor(Math.random()
* greetings.length)];

  return (
    <div
      className="
        bg-neutral-900
        rounded-lg
        h-full
        w-full
        overflow-hidden
        overflow-y-auto
      "
    >
      <Header>
        <div className="mb-2">
          <h1
            className="
              text-white
              text-3xl
              font-semibold
            ">
            {randomGreeting}
          </h1>
          <div
            className="
              grid
              grid-cols-1
              sm:grid-cols-2
```

```

        xl:grid-cols-3
        2xl:grid-cols-4
        gap-3
        mt-4
    "
  >
    <ListItem
      name="Favourites"
      image="/images/liked2.png"
      href="liked"
    />
  </div>
</div>
</Header>
<div className="mt-2 mb-7 px-6">
  <div className="flex justify-between items-center">
    <PageContent songs={songs} albums={albums}
albumSongs={albumSongs} />
  </div>
</div>
)
}

```


Приложение 3. Код SupabaseProvider и UserProvider

```
"use client";

import { useState } from "react";
import { createClientComponentClient } from "@supabase/auth-
helpers-nextjs";
import { SessionContextProvider } from "@supabase/auth-
helpers-react";

import { Database } from "@/types_db";

interface SupabaseProviderProps {
  children: React.ReactNode;
};

const SupabaseProvider: React.FC<SupabaseProviderProps> = ({
  children
}) => {
  const [supabaseClient] = useState(() =>
    createClientComponentClient<Database>()
  );

  return (
    <SessionContextProvider supabaseClient={supabaseClient}>
      {children}
    </SessionContextProvider>
  );
}

export default SupabaseProvider;

"use client";

import { MyUserContextProvider } from "@/hooks/useUser";

interface UserProviderProps {
  children: React.ReactNode;
}

const UserProvider: React.FC<UserProviderProps> = ({
  children
}) => {
  return (
    <MyUserContextProvider>
      {children}
    </MyUserContextProvider>
  );
}

export default UserProvider;
```

Приложение 4. Код компонента Modal

```
import * as Dialog from '@radix-ui/react-dialog';
import { IoMdClose } from 'react-icons/io';

interface ModalProps {
  isOpen: boolean;
  onChange: (open: boolean) => void;
  title: string;
  description: string;
  children: React.ReactNode;
}

const Modal: React.FC<ModalProps> = ({
  isOpen,
  onChange,
  title,
  description,
  children
}) => {
  return (
    <Dialog.Root open={isOpen} defaultOpen={isOpen}
onOpenChange={onChange}>
      <Dialog.Portal>
        <Dialog.Overlay
          className="
            bg-neutral-900/90
            backdrop-blur-sm
            fixed
            inset-0
          "
        />
        <Dialog.Content
          className="
            fixed
            drop-shadow-md
            border
            border-neutral-700
            top-[50%]
            left-[50%]
            max-h-full
            h-full
            md:h-auto
            md:max-h-[85vh]
            w-full
            md:w-[90vw]
            md:max-w-[450px]
            translate-x-[-50%]
            translate-y-[-50%]
            rounded-md
            bg-neutral-800
            p-[25px]
            focus:outline-none
          ">
          {children}
        </Dialog.Content>
      </Dialog.Portal>
    </Dialog.Root>
  );
}
```

```

">
<Dialog.Title
  className="
    text-xl
    text-center
    font-bold
    mb-4
  "
>
  {title}
</Dialog.Title>
<Dialog.Description
  className="
    mb-5
    text-sm
    leading-normal
    text-center
  "
>
  {description}
</Dialog.Description>
<div>
  {children}
</div>
<Dialog.Close asChild>
  <button
    className="
      text-neutral-400
      hover:text-white
      absolute
      top-[10px]
      right-[10px]
      inline-flex
      h-[25px]
      w-[25px]
      appearance-none
      items-center
      justify-center
      rounded-full
      focus:outline-none
    "
    aria-label="Close"
  >
    <IoMdClose />
  </button>
</Dialog.Close>
</Dialog.Content>
</Dialog.Portal>
</Dialog.Root>
);
}

```

```
export default Modal;
```

Приложение 5. Экшен getSongsByTitle

```
import { createServerComponentClient } from "@supabase/auth-  
helpers-nextjs";  
import { cookies, headers } from "next/headers";  
  
import { Song } from "@types";  
  
import getSongs from "../getSongs";  
  
const getSongsByTitle = async (title: string):  
Promise<Song[]> => {  
  const supabase = createServerComponentClient({  
    cookies: cookies  
  });  
  
  if (!title) {  
    const allSongs = await getSongs();  
    return allSongs;  
  }  
  
  const { data, error } = await supabase  
    .from('songs')  
    .select('*')  
    .ilike('title', `%${title}%`)  
    .order('created_at', { ascending: false })  
  
  if (error) {  
    console.log(error.message);  
  }  
  
  return (data as any) || [];  
};  
  
export default getSongsByTitle;
```

Приложение 6. Код компонента SearchInput

```
"use client";

import qs from "query-string";
import { useEffect, useState } from "react";
import { useRouter } from "next/navigation";

import useDebounce from "@/hooks/useDebounce";

import Input from "../Input";

const SearchInput = () => {
  const router = useRouter();
  const [value, setValue] = useState<string>('');
  const debouncedValue = useDebounce<string>(value, 500);

  useEffect(() => {
    const query = {
      title: debouncedValue,
    };

    const url = qs.stringifyUrl({
      url: '/search',
      query
    });

    router.push(url);
  }, [debouncedValue, router]);

  return (
    <Input
      placeholder="What do you want to listen to?"
      value={value}
      onChange={(e) => setValue(e.target.value)}
    />
  );
}

export default SearchInput;
```

Приложение 7. Код страницы Search

```
import getSongsByTitle from "@/actions/getSongsByTitle";
import SearchInput from "@/components/SearchInput";
import Header from "@/components/Header";

import SearchContent from "../components/SearchContent";

export const revalidate = 0;

interface SearchProps {
  searchParams: { title: string }
};

const Search = async ({ searchParams }: SearchProps) => {
  const songs = await getSongsByTitle(searchParams.title);

  return (
    <div
      className="
        bg-neutral-900
        rounded-lg
        h-full
        w-full
        overflow-hidden
        overflow-y-auto
      "
    >
      <Header className="from-bg-neutral-900">
        <div className="mb-2 flex flex-col gap-y-6">
          <h1 className="text-white text-3xl font-semibold">
            Search
          </h1>
          <SearchInput />
        </div>
      </Header>
      <SearchContent songs={songs} />
    </div>
  );
}

export default Search;
```

Приложение 8. Код страницы News

```
"use client"
import React, { useEffect, useState } from 'react';
import Header from '@components/Header';
import NewsContent from './components/NewsContent';
import getNews from '@actions/getNews';

const NewsPage = () => {
  const [news, setNews] = useState<any[]>([]);

  useEffect(() => {
    async function fetchNews() {
      try {
        const fetchedNews = await getNews();
        setNews(fetchedNews);
      } catch (error) {
        console.error('Error fetching news:', error);
      }
    }

    fetchNews();
  }, []);

  return (
    <div className="bg-neutral-900 rounded-lg h-full w-full
overflow-hidden overflow-y-auto">
      <Header>
        <div className="mt-20">
          <h1 className="text-white text-4xl sm:text-5xl
lg:text-7xl font-bold">Latest Music News</h1>
        </div>
      </Header>
      <NewsContent news={news} />
    </div>
  );
};

export default NewsPage;
```


Приложение 9. Экшен getNews

```
import axios from 'axios';

const getMusicNews = async () => {

  const NEWS_API_KEY = 'de1c6ca593b64f6f9df6d887dc55e853'

  try {
    const musicResponse = await
    axios.get('https://newsapi.org/v2/top-headlines', {
      params: {
        country: 'us',
        category: 'entertainment',
        q: 'music',
        apiKey: NEWS_API_KEY
      }
    });

    const concertResponse = await
    axios.get('https://newsapi.org/v2/top-headlines', {
      params: {
        country: 'gb',
        category: 'entertainment',
        q: 'music',
        apiKey: NEWS_API_KEY
      }
    });

    const allArticles = [...musicResponse.data.articles,
    ...concertResponse.data.articles];

    console.log('Response data:', allArticles);

    return allArticles;
  } catch (error) {
    console.error('Error fetching news:', error);
    throw new Error('Failed to fetch news');
  }
};

export default getMusicNews;
```

Приложение 10. Код компонента NewsContent

```
import React from 'react';

interface NewsItem {
  title: string;
  description: string;
  publishedAt: string;
  url: string;
}

interface NewsContentProps {
  news: NewsItem[];
}

const NewsContent: React.FC<NewsContentProps> = ({ news })
=> {
  return (
    <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-
cols-3 gap-4 m-2">
      {news.map((article: NewsItem, index: number) => {
        if (article.title.includes('Removed')) {
          return null;
        }
        return (
          <div key={index} className="bg-neutral-800 text-
white rounded-lg shadow-md p-6">
            <h2 className="text-xl font-semibold mb-
4">{article.title}</h2>
            <p className="text-gray-300 mb-
4">{article.description}</p>
            <p className="text-gray-400 italic">Published
at: {article.publishedAt}</p>
            <a className="text-orange-500 hover:underline"
href={article.url} target="_blank" rel="noopener
noreferrer">Read more</a>
          </div>
        );
      })}
    </div>
  );
}

export default NewsContent;
```

Приложение 11. Экшены getSongsForPlaylist и getSongsForAlbum

```
import { createServerComponentClient } from "@supabase/auth-  
helpers-nextjs";  
import type { Song } from "@types";  
  
const getSongsForPlaylist = async (playlistId: string):  
Promise<Song[]> => {  
  const { cookies } = await import("next/headers");  
  
  const supabase = createServerComponentClient({  
    cookies: cookies  
  });  
  
  const { data: playlistSongs, error: playlistSongsError } =  
await supabase  
    .from('playlist_songs')  
    .select('song_id')  
    .eq('playlist_id', playlistId);  
  
  if (playlistSongsError) {  
    console.error("Error fetching playlist songs:",  
playlistSongsError.message);  
    return [];  
  }  
  
  const songIds = playlistSongs.map((playlistSong: any) =>  
playlistSong.song_id);  
  
  const { data: songs, error: songsError } = await supabase  
    .from('songs')  
    .select('*')  
    .in('id', songIds)  
    .order('created_at', { ascending: false });  
  
  if (songsError) {  
    console.error("Error fetching songs:",  
songsError.message);  
    return [];  
  }  
  
  const formattedSongs: Song[] = songs.map((song: any) => ({  
    id: song.id,  
    user_id: song.user_id,  
    author: song.author,  
    title: song.title,  
    song_path: song.song_path,  
    image_path: song.image_path  
  }));  
  
  return formattedSongs;  
};
```

```

export default getSongsForPlaylist;

import { createServerComponentClient } from "@supabase/auth-
helpers-nextjs";
import type { Song } from "@types";

const getSongsForAlbum = async (albumId: string):
Promise<Song[]> => {
  const { cookies } = await import("next/headers");

  const supabase = createServerComponentClient({
    cookies: cookies
  });

  const { data: albumSongs, error: albumSongsError } = await
supabase
    .from('album_songs')
    .select('song_id')
    .eq('album_id', albumId);

  if (albumSongsError) {
    console.error("Error fetching album songs:",
albumSongsError.message);
    return [];
  }

  const songIds = albumSongs.map((albumSong: any) =>
albumSong.song_id);

  const { data: songs, error: songsError } = await supabase
    .from('songs')
    .select('*')
    .in('id', songIds)
    .order('created_at', { ascending: false });

  if (songsError) {
    console.error("Error fetching songs:",
songsError.message);
    return [];
  }

  const formattedSongs: Song[] = songs.map((song: any) => ({
    id: song.id,
    user_id: song.user_id,
    author: song.author,
    title: song.title,
    song_path: song.song_path,
    image_path: song.image_path
  }));

  return formattedSongs;
};

```

```
export default getSongsForAlbum;
```

Приложение 12. Код компонента LikedContent

```
"use client"
import { useEffect, useState } from 'react';
import { useRouter } from 'next/navigation';
import Image from 'next/image';
import MediaItem from '@components/MediaItem';
import LikeButton from '@components/LikeButton';
import useOnPlay from '@hooks/useOnPlay';
import useUploadPlaylistModal from
'@/hooks/useUploadPlaylist';
import useUploadAlbumModal from '@/hooks/useUploadAlbum';
import usePlaylistImage from '@/hooks/useLoadPlaylistImage';
import useAlbumImage from '@/hooks/useLoadAlbumImage';
import AlbumModal from '@components/AlbumModal';
import PlaylistModal from '@components/PlaylistModal';
import { Album, Playlist, Song } from '@types';
import { useUser } from '@hooks/useUser';
import MusicMenu from '@components/MusicMenu';

interface LikedContentProps {
  songs: Song[];
  playlists: Playlist[];
  albums: Album[];
  playlistSongs: { playlistId: string, songs: Song[] }[];
  albumSongs: { albumId: string, songs: Song[] }[];
  likedAlbums: Album[];
}

const LikedContent: React.FC<LikedContentProps> = ({ songs,
playlists, playlistSongs, albums, albumSongs, likedAlbums }) =>
{
  const router = useRouter();
  const { isLoading, user } = useUser();
  const [selectedPlaylist, setSelectedPlaylist] =
useState<Playlist | null>(null);
  const [selectedAlbum, setSelectedAlbum] = useState<Album |
null>(null);
  const [playlistSongsState, setPlaylistSongsState] =
useState<Song[]>([]);
  const [albumSongsState, setAlbumSongsState] =
useState<Song[]>([]);
  const [isHovered, setIsHovered] = useState(false);
  const [showMenu, setShowMenu] = useState(false);
  const [selectedSongId, setSelectedSongId] =
useState<string>('');

  const uploadPlaylistModal = useUploadPlaylistModal();
  const uploadAlbumModal = useUploadAlbumModal();
  const onPlay = useOnPlay(songs);

  useEffect(() => {
    if (!isLoading && !user) {
```

```

        router.replace('/');
    }
}, [isLoading, user, router]));

const handleCreatePlaylist = () => {
    return uploadPlaylistModal.onOpen();
};

const handleCreateAlbum = () => {
    return uploadAlbumModal.onOpen();
}

const handleShowMenu = (songId: string) => {
    if (selectedSongId === songId && showMenu ) {
        setShowMenu(false);
    } else {
        setSelectedSongId(songId);
        setShowMenu(true);
    }
};

const handlePlaylistClick = (playlist: Playlist) => {
    setSelectedPlaylist(playlist);
    const songs = playlistSongs.find(item => item.playlistId
=== playlist.id)?.songs || [];
    setPlaylistSongsState(songs);
};

const handleAlbumClick = (album: Album) => {
    setSelectedAlbum(album);
    const songs = albumSongs.find(item => item.albumId ===
album.id)?.songs || [];
    setAlbumSongsState(songs);
};

const handleClosePlaylistModal = () => {
    setSelectedPlaylist(null);
    setPlaylistSongsState([]);
};

const handleCloseAlbumModal = () => {
    setSelectedAlbum(null);
    setAlbumSongsState([]);
};

return (
    <div className="flex flex-col gap-y-8 p-6">
        <div className="text-lg font-semibold text-
white">Playlists</div>
        <button
            className="w-16 h-16 flex items-center justify-
center rounded-lg bg-gray-100 bg-opacity-30 border-2 border-

```

```

gray-200 hover:border-gray-400 text-gray-400 hover:text-gray-600
transition-colors mt-4"
      onMouseEnter={() => setIsHovered(true)}
      onMouseLeave={() => setIsHovered(false)}
      onClick={handleCreatePlaylist}
    >
      <svg
        xmlns="http://www.w3.org/2000/svg"
        className="h-8 w-8"
        fill="none"
        viewBox="0 0 24 24"
        stroke="currentColor"
      >
        <path
          strokeLinecap="round"
          strokeLinejoin="round"
          strokeWidth={2}
          d="M12 6v6m0 0v6m0-6h6m-6 0H6"
        />
      </svg>
    </button>

    <div className="grid grid-cols-2 sm:grid-cols-3
md:grid-cols-4 lg:grid-cols-5 xl:grid-cols-6 gap-6">
      {playlists.map((playlist) => {
        const imageUrl = usePlaylistImage(playlist);
        if (!imageUrl) {
          console.log(`Playlist ${playlist.title} does not
have an image.`);
          return null;
        }

        return (
          <div key={playlist.id} className="flex flex-col
items-center cursor-pointer hover:bg-neutral-800/50 rounded-lg
p-2" onClick={() => handlePlaylistClick(playlist)}>
            <div className="relative w-40 h-40 overflow-
hidden rounded-lg ">
              <Image
                src={imageUrl}
                alt={playlist.title}
                layout="fill"
                objectFit="cover"
                className="object-cover"
              />
            </div>
            <p className="mt-2 text-
white">{playlist.title}</p>
            <p className="text-sm text-gray-
400">{playlist.description}</p>
          </div>
        );
      })}
    </div>
  </div>
</div>

```



```

    }}}
  </div>

  <div className="text-lg font-semibold text-
white">Albums</div>
  <button
    className="w-16 h-16 flex items-center justify-
center rounded-lg bg-gray-100 bg-opacity-30 border-2 border-
gray-200 hover:border-gray-400 text-gray-400 hover:text-gray-600
transition-colors mt-4"
    onMouseEnter={() => setIsHovered(true)}
    onMouseLeave={() => setIsHovered(false)}
    onClick={handleCreateAlbum}
  >
    <svg
      xmlns="http://www.w3.org/2000/svg"
      className="h-8 w-8"
      fill="none"
      viewBox="0 0 24 24"
      stroke="currentColor"
    >
      <path
        strokeLinecap="round"
        strokeLinejoin="round"
        strokeWidth={2}
        d="M12 6v6m0 0v6m0-6h6m-6 0H6"
      />
    </svg>
  </button>

  <div className="grid grid-cols-2 sm:grid-cols-3
md:grid-cols-4 lg:grid-cols-5 xl:grid-cols-6 gap-6">
    {albums.map((album) => {
      const imageUrl = useAlbumImage(album);
      if (!imageUrl) {
        console.log(`Album ${album.title} does not have
an image.`);
        return null;
      }

      return (
        <div key={album.id} className="flex flex-col
items-center cursor-pointer hover:bg-neutral-800/50 rounded-lg
p-2" onClick={() => handleAlbumClick(album)}>
          <div className="relative w-40 h-40 overflow-
hidden rounded-lg ">
            <Image
              src={imageUrl}
              alt={album.title}
              layout="fill"
              objectFit="cover"
              className="object-cover"

```

```

        />
      </div>
      <p className="mt-2 text-
white">{album.title}</p>
      <p className="text-sm text-gray-
400">{album.artist}</p>
      <p className="text-sm text-gray-
400">{album.release_year}</p>
      <p className="text-sm text-gray-
400">{album.genre}</p>
    </div>
  );
  })}
</div>

  <div className="text-lg font-semibold text-
white">Liked Albums</div>
  <div className="grid grid-cols-2 sm:grid-cols-3
md:grid-cols-4 lg:grid-cols-5 xl:grid-cols-6 gap-6">
    {likedAlbums.map((album) => {
      const imageUrl = useAlbumImage(album);
      if (!imageUrl) {
        console.log(`Album ${album.title} does not have
an image.`);
        return null;
      }

      return (
        <div key={album.id} className="flex flex-col
items-center cursor-pointer hover:bg-neutral-800/50 rounded-lg
p-2" onClick={() => handleAlbumClick(album)}>
          <div className="relative w-40 h-40 overflow-
hidden rounded-lg ">
            <Image
              src={imageUrl}
              alt={album.title}
              layout="fill"
              objectFit="cover"
              className="object-cover"
            />
          </div>
          <p className="mt-2 text-
white">{album.title}</p>
          <p className="text-sm text-gray-
400">{album.artist}</p>
          <p className="text-sm text-gray-
400">{album.release_year}</p>
          <p className="text-sm text-gray-
400">{album.genre}</p>
        </div>
      );
    })}

```

```

    </div>

    <div className="text-lg font-semibold text-white">Songs</div>
    <div className="flex flex-col gap-y-4">
      {songs.map((song: any) => (
        <div key={song.id} className="flex items-center">
          <div className="flex-1">
            <MediaItem onClick={(id) => onPlay(id)}
data={song} />
          </div>
          <div className="relative ml-4 mr-4">
            <button
              onClick={() => handleShowMenu(song.id)}
              className="w-8 h-8 flex items-center
justify-center rounded-full bg-transparent hover:bg-gray-200
transition-colors"
            >
              <svg xmlns="http://www.w3.org/2000/svg"
className="h-4 w-4 text-gray-500" viewBox="0 0 20 20"
fill="currentColor">
                <path fillRule="evenodd" d="M2 10C2 9.45
2.45 9 3 9s1 .45 1 1-.45 1-1 1zM7 10c0-.55.45-1 1-1s1 .45 1 1-
.45 1-1 1zM12 10c0-.55.45-1 1-1s1 .45 1 1-.45 1-1 1z"
clipRule="evenodd" />
              </svg>
            </button>
            {showMenu && selectedSongId === song.id && (
              <MusicMenu playlists={playlists}
albums={albums} songId={selectedSongId} onClose={() =>
setShowMenu(false)} />
            )}
          </div>
          <LikeButton songId={song.id} />
        </div>
      )})}
    </div>
    {selectedPlaylist && (
      <PlaylistModal playlist={selectedPlaylist}
songs={playlistSongsState} onClose={handleClosePlaylistModal} />
    )}
    {selectedAlbum && (
      <AlbumModal album={selectedAlbum}
songs={albumSongsState} onClose={handleCloseAlbumModal} />
    )}
  </div>
);
};

export default LikedContent;

```

Приложение 13. Код компонента MusicMenu

```
import { useEffect, useState } from 'react';
import { Album, Playlist } from '@types';
import { useSupabaseClient } from '@supabase/auth-helpers-react';

interface MusicMenuProps {
  playlists: Playlist[];
  albums: Album[];
  songId: string;
  onClose: () => void;
}

const MusicMenu: React.FC<MusicMenuProps> = ({ playlists,
albums, songId, onClose }) => {
  const supabaseClient = useSupabaseClient();
  const [filteredPlaylists, setFilteredPlaylists] =
    useState<Playlist[]>([]);
  const [filteredAlbums, setFilteredAlbums] =
    useState<Album[]>([]);

  useEffect(() => {
    const fetchFilteredItems = async () => {
      try {
        const { data: existingPlaylists, error:
playlistError } = await supabaseClient
          .from('playlist_songs')
          .select('playlist_id')
          .eq('song_id', songId);

        const { data: existingAlbums, error: albumError } =
await supabaseClient
          .from('album_songs')
          .select('album_id')
          .eq('song_id', songId);

        if (playlistError || albumError) {
          throw playlistError || albumError;
        }

        const existingPlaylistIds =
existingPlaylists.map((playlist: any) => playlist.playlist_id);
        const existingAlbumIds = existingAlbums.map((album:
any) => album.album_id);

        const filteredPlaylists =
playlists.filter((playlist) =>
!existingPlaylistIds.includes(playlist.id));
        const filteredAlbums = albums.filter((album) =>
!existingAlbumIds.includes(album.id));

        setFilteredPlaylists(filteredPlaylists);
```

```

        setFilteredAlbums(filteredAlbums);
      } catch (error) {
        console.error('Error checking existing playlists or
albums:', error);
      }
    };

    fetchFilteredItems();
  }, [supabaseClient, playlists, albums, songId]);

  const handleItemClick = async (itemId: string, type:
'playlist' | 'album') => {
    console.log(songId);
    await addToItem(itemId, type);
    onClose();
  };

  const addToItem = async (itemId: string, type: 'playlist'
| 'album') => {
    try {
      const tableName = type === 'playlist' ?
'playlist_songs' : 'album_songs';
      const { error } = await supabaseClient
        .from(tableName)
        .insert([
          { [`${type}_id`]: itemId, song_id: songId
}]);

      if (error) {
        throw error;
      } else {
        console.log('Song added successfully');
      }
    } catch (error) {
      console.error('Error adding song:', error);
    }
  };

  return (
    <div className="absolute top-0 right-0 mt-10 w-48 bg-
neutral-800 shadow-lg rounded">
      <div className="py-1 h-32 overflow-y-auto" role="menu"
aria-orientation="vertical" aria-labelledby="options-menu">
        {filteredPlaylists.map((playlist) => (
          <div key={playlist.id} className="px-4 py-2 text-
sm text-gray-300 hover:bg-neutral-700 cursor-pointer"
onClick={() => handleItemClick(playlist.id, 'playlist')}>
            {playlist.title}
          </div>
        ))}
        {filteredAlbums.map((album) => (

```

```

        <div key={album.id} className="px-4 py-2 text-sm
text-gray-300 hover:bg-neutral-700 cursor-pointer" onClick={()
=> handleItemClick(album.id, 'album')}>
            {album.title}
        </div>
    )}
</div>
</div>
);
};

export default MusicMenu;

```

Приложение 14. Код компонента LikeButton

```
"use client";

import { useEffect, useState } from "react";
import { AiOutlineHeart, AiFillHeart } from "react-
icons/ai";
import { useRouter } from "next/navigation";
import { toast } from "react-hot-toast";
import { useSessionContext } from "@supabase/auth-helpers-
react";

import { useUser } from "@/hooks/useUser";
import useAuthModal from "@/hooks/useAuthModal";

interface LikeButtonProps {
  songId: string;
};

const LikeButton: React.FC<LikeButtonProps> = ({
  songId
}) => {
  const router = useRouter();
  const {
    supabaseClient
  } = useSessionContext();
  const authModal = useAuthModal();
  const { user } = useUser();

  const [isLiked, setIsLiked] = useState<boolean>(false);

  useEffect(() => {
    if (!user?.id) {
      setIsLiked(false);
      return;
    }

    const fetchData = async () => {
      try {
        const { data: likedSongs, error } = await
supabaseClient
          .from('liked_songs')
          .select('song_id')
          .eq('user_id', user.id);

        if (error) {
          throw error;
        }

        const isSongLiked = likedSongs.some(song =>
song.song_id === songId);
        setIsLiked(isSongLiked);
      } catch (error) {
```

```

        console.error('Error fetching liked songs:', error);
        setIsLiked(false);
    }
};

fetchData();
}, [songId, supabaseClient, user?.id]));

const Icon = isLiked ? AiFillHeart : AiOutlineHeart;

const handleLike = async () => {
    if (!user) {
        return authModal.onOpen();
    }

    if (isLiked) {
        const { error } = await supabaseClient
            .from('liked_songs')
            .delete()
            .eq('user_id', user.id)
            .eq('song_id', songId)

        if (error) {
            toast.error(error.message);
        } else {
            setIsLiked(false);
        }
    } else {
        const { error } = await supabaseClient
            .from('liked_songs')
            .insert({
                song_id: songId,
                user_id: user.id
            });

        if (error) {
            toast.error(error.message);
        } else {
            setIsLiked(true);
            toast.success('Success');
        }
    }

    router.refresh();
}

return (
    <button
        className="
            cursor-pointer
            hover:opacity-75

```



```

        transition
        "
        onClick={handleLike}
    >
        <Icon color={isLiked ? '#F97216' : 'white'} size={25}
/>
    </button>
  );
}

export default LikeButton;

```

Приложение 15. Код компонента AlbumModal

```
import { useRouter } from 'next/navigation';
import Image from 'next/image';
import MediaItem from '@components/MediaItem';
import useOnPlay from '@hooks/useOnPlay';
import { FiShuffle, FiX, FiPlay } from 'react-icons/fi';
import { useState, useEffect } from 'react';
import { useSupabaseClient } from '@supabase/auth-helpers-react';

import { Album, Song } from '@types';
import useAlbumImage from '@hooks/useLoadAlbumImage';
import { useUser } from '@hooks/useUser';

interface AlbumModalProps {
  album: Album;
  songs: Song[];
  onClose: () => void;
  isOpenedHome?: boolean;
}

const AlbumModal: React.FC<AlbumModalProps> = ({ album,
songs, onClose, isOpenedHome = false }) => {
  const router = useRouter();
  const imageUrl = useAlbumImage(album);
  const [isShuffleActive, setIsShuffleActive] =
useState(false);
  const [shuffledSongs, setShuffledSongs] =
useState<Song[]>([]);
  const [playableSongs, setPlayableSongs] =
useState<Song[]>([]);
  const [currentIndex, setCurrentIndex] =
useState<number>(0);
  const [albumOpened, setAlbumOpened] =
useState<boolean>(false);
  const [showRemoveConfirmation, setShowRemoveConfirmation]
= useState<boolean>(false);
  const [showSongRemoveConfirmation,
setShowSongRemoveConfirmation] = useState<boolean>(false);
  const [songToRemove, setSongToRemove] =
useState<string>('');
  const [albumToDelete, setAlbumToDelete] =
useState<string>('');
  const [deletedSongs, setDeletedSongs] =
useState<string[]>([]);
  const [hoveredSongId, setHoveredSongId] = useState<string
| null>(null);
  const [isHovered, setIsHovered] =
useState<boolean>(false);
  const [isAlbumHovered, setIsAlbumHovered] =
useState<boolean>(false);
  const [isAlbumInFavourites, setIsAlbumInFavourites] =
useState<boolean>(false);
```

```

    const [isAlbumInUserAlbums, setIsAlbumInUserAlbums] =
useState<boolean>(false);

    const supabase = useSupabaseClient();
    const {user} = useUser();

    const handleCloseModal = (e:
React.MouseEvent<HTMLDivElement, MouseEvent>) => {
    if (e.target === e.currentTarget) {
        onClose();
    }
};

const checkAlbumInFavourites = async () => {
try {
    if (!user) {
        console.log("User is not logged in.");
        return;
    }
    const { data: likedAlbums, error } = await supabase
        .from("liked_albums")
        .select("album_id")
        .eq("user_id", user.id);

    if (error) {
        console.error("Error fetching liked albums:",
error.message);
        return;
    }
    const albumInFavourites = likedAlbums.some(
        (likedAlbum) => likedAlbum.album_id === album.id
    );
    setIsAlbumInFavourites(albumInFavourites);
} catch (error) {
    console.error("Error checking album in favourites:",
error);
}
};

const checkAlbumInUserAlbums = async () => {
try {
    if (!user) {
        console.log("User is not logged in.");
        return;
    }

    const { data: userAlbums, error } = await supabase
        .from("albums")
        .select("id")
        .eq("user_id", user.id);

    if (error) {

```

```

        console.error("Error fetching user albums:",
error.message);
        return;
    }
    const albumInUserAlbums = userAlbums.some((userAlbum) =>
userAlbum.id === album.id);
    setIsAlbumInUserAlbums(albumInUserAlbums);
    } catch (error) {
        console.error("Error checking album in user albums:",
error);
    }
};

useEffect(() => {
    checkAlbumInUserAlbums();
}, []);

const onPlay = useOnPlay(playableSongs);

useEffect(() => {
    const fetchPlaylistSongs = async () => {
        try {
            const { data: albumSongsData, error: albumSongsError
} = await supabase
                .from('album_songs')
                .select('song_id')
                .eq('album_id', album.id);
            if (albumSongsData) {
                const playlistSongIds = albumSongsData.map((item:
any) => item.song_id);
                const { data: newSongsData, error: newSongsError }
= await supabase
                    .from('songs')
                    .select('*')
                    .in('id', playlistSongIds);
                if (newSongsData) {
                    const newSongs: Song[] =
newSongsData.map((songData: any) => ({
                        id: songData.id,
                        user_id: songData.user_id,
                        author: songData.author,
                        title: songData.title,
                        song_path: songData.song_path,
                        image_path: songData.image_path
                    }));
                    setPlayableSongs(newSongs);
                }
            }
        } catch (error) {
            console.error('Error fetching album songs:', error);
        }
    };
};

```

```

        if (album.id) {
            fetchPlaylistSongs();
        }
    }, [album.id]);

    useEffect(() => {
        if (playableSongs.length > 0) {
            console.log('Loaded songs:', playableSongs.map(song =>
song.title));
        }
    }, [playableSongs]);

    useEffect(() => {
        if (isShuffleActive) {
            const shuffled = [...playableSongs].sort(() =>
Math.random() - 0.5);
            setShuffledSongs(shuffled);
            setCurrentIndex(0);
        } else {
            setShuffledSongs([]);
            setCurrentIndex(0);
        }
    }, [isShuffleActive, playableSongs]);

    useEffect(() => {
        if (albumOpened && playableSongs.length > 0) {
            onPlay(playableSongs[currentIndex].id);
        }

        checkAlbumInFavourites();
    }, [currentIndex, playableSongs, albumOpened]);

    const handleShuffleClick = () => {
        const shuffled = [...playableSongs].sort(() =>
Math.random() - 0.5);
        setPlayableSongs(shuffled);
        setCurrentIndex(0);
        setIsShuffleActive(!isShuffleActive);
        setAlbumOpened(true);
        //setPlayableSongs([...songs]);
    };

    const handleNextSong = () => {
        if (isShuffleActive && shuffledSongs.length > 0) {
            setCurrentIndex((prevIndex) => (prevIndex + 1) %
shuffledSongs.length);
        } else if (!isShuffleActive && playableSongs.length > 0)
{
            setCurrentIndex((prevIndex) => (prevIndex + 1) %
playableSongs.length);
        }
    }

```

```

};

const handleRemoveSong = async (songId: string) => {
  setShowSongRemoveConfirmation(true);
  setSongToRemove(songId);
};

const handleRemoveAlbum = async () => {
  setShowRemoveConfirmation(true);
  setAlbumToDelete(album.id);
};

const handleAddAlbum = async () => {
  try {
    if (!user) {
      console.log("User is not logged in.");
      return;
    }
    const { data, error } = await supabase
      .from("liked_albums")
      .insert([
        { user_id: user.id, album_id: album.id }
      ]);

    if (error) {
      console.error("Error adding album to liked_albums:",
error.message);
      return;
    }

    console.log("Album added to liked_albums:", data);
    onClose();
  } catch (error) {
    console.error("Error adding album to liked_albums:",
error);
  }
};

const handleConfirmation = async (confirmed: boolean,
itemId: string) => {
  if (confirmed) {
    try {
      if (itemId === 'album') {
        await supabase
          .from('album_songs')
          .delete()
          .eq('album_id', albumToDelete);

        await supabase
          .from('albums')

```

```

        .delete()
        .eq('id', albumToDelete);

onClose();
router.refresh();
} else {

    await supabase
        .from('album_songs')
        .delete()
        .eq('album_id', album.id)
        .eq('song_id', itemId);

    const updatedSongs = playableSongs.filter(song =>
song.id !== itemId);
    setPlayableSongs(updatedSongs);
    setDeletedSongs(prevDeletedSongs =>
[...prevDeletedSongs, itemId]);
    } catch (error) {
        console.error('Error removing item from album:',
error);
    }
    }
    setShowRemoveConfirmation(false);
    setShowSongRemoveConfirmation(false);
};

const handleUnFavouriteAlbum = async () => {
    try {
        if (!user) {
            console.log("User is not logged in.");
            return;
        }

        const { error } = await supabase
            .from('liked_albums')
            .delete()
            .eq('user_id', user.id)
            .eq('album_id', album.id);

        if (error) {
            console.error("Error removing album from
liked_albums:", error.message);
            return;
        }

        console.log("Album removed from liked_albums");

        setIsAlbumInFavourites(false);

        router.refresh();
    }

```

```

        onClose();
    } catch (error) {
        console.error("Error removing album from
liked_albums:", error);
    }
};

const handlePlayPlaylist = () => {
    onPlay(playableSongs[0]?.id || '');
};

const handleAlbumHover = (hovered: boolean) => {
    setIsAlbumHovered(hovered);
};

const handleSongItemClick = (songId: string) => {
    if (!deletedSongs.includes(songId)) {
        onPlay(songId);
    }
};

return (
    <div className="fixed top-0 left-0 w-screen h-screen
flex justify-center items-center bg-neutral-800 bg-opacity-50"
onClick={handleCloseModal}>
        {showRemoveConfirmation && (
            <div className="fixed top-0 left-0 w-full h-full
flex justify-center items-center z-20">
                <div className="bg-neutral-500 absolute top-1/2
left-1/2 transform -translate-x-1/2 -translate-y-1/2 p-4 rounded
shadow-lg">
                    <p className="text-white">Are you sure you want
to remove this album?</p>
                    <div className="flex justify-end mt-4">
                        <button className="mr-2 text-white"
onClick={() => handleConfirmation(true, 'album')}>Yes</button>
                        <button className="text-white" onClick={() =>
handleConfirmation(false, '')}>Cancel</button>
                    </div>
                </div>
            </div>
        )}
        {showSongRemoveConfirmation && (
            <div className="fixed top-0 left-0 w-full h-full
flex justify-center items-center z-20">
                <div className="bg-neutral-500 absolute top-1/2
left-1/2 transform -translate-x-1/2 -translate-y-1/2 p-4 rounded
shadow-lg">
                    <p className="text-white">Are you sure you want
to remove this song from the album?</p>
                    <div className="flex justify-end mt-4">

```



```

        <button className="mr-2 text-white"
onClick={() => handleConfirmation(true,
songToRemove)}>Yes</button>
        <button className="text-white" onClick={() =>
handleConfirmation(false, '')}>Cancel</button>
    </div>
</div>
</div>
    )}
    <div className="bg-neutral-800 rounded-lg p-8 w-96
relative">
        <div className="flex gap-4 items-center mb-4">
            <div className="w-40 h-40 overflow-hidden rounded-
lg relative" onMouseEnter={() => setIsHovered(true)}
onMouseLeave={() => setIsHovered(false)}>
                {imageUrl && (
                    <>
                        <Image src={imageUrl} alt={album.title}
width={160} height={160} objectFit="cover" className="object-
cover" />
                        {isHovered && (
                            <div className="absolute inset-0 flex
items-center justify-center">
                                <FiPlay className="text-white text-4xl
cursor-pointer" onClick={handlePlayPlaylist} />
                                </div>
                            )}
                        </>
                    )}
                </div>
                <div onMouseEnter={() => handleAlbumHover(true)}
onMouseLeave={() => handleAlbumHover(false)} className="flex-1">
                    <h2 className="text-xl font-
bold">{album.title}</h2>
                    <p className="text-gray-500">{album.artist}</p>
                    <p className="text-gray-
500">{album.release_year}</p>
                    <p className="text-gray-500">{album.genre}</p>
                    {isAlbumHovered && !isOpenedHome &&
!isAlbumInFavourites && (
                        <div className="mt-2 mr-2">
                            <button className="text-red-500"
onClick={handleRemoveAlbum}>Remove</button>
                        </div>
                    )}
                    {isAlbumHovered && isOpenedHome &&
!isAlbumInFavourites && !isAlbumInUserAlbums && (
                        <div className="mt-2 mr-2">
                            <button className="text-orange-500"
onClick={handleAddAlbum}>Add to Favourites</button>
                        </div>
                    )}
                </div>
            </div>

```

```

        {isAlbumHovered && isAlbumInFavourites && (
          <div className="mt-2 mr-2">
            <div className="text-orange-500">In Your
Favourites</div>
              <button className="text-red-500"
onClick={handleUnFavouriteAlbum}>Unfavourite</button>
            </div>
          )}

        </div>
      </div>
      <hr className="border-t border-gray-300 w-full my-4"
/>

      <div className="flex items-center justify-center w-
full cursor-pointer hover:bg-neutral-700 p-2 rounded-lg mb-2"
onClick={handleShuffleClick}>
        <FiShuffle className={`text-white mr-2
${isShuffleActive ? 'text-green-500' : ''}`} />
        <span className="text-white">Shuffle</span>
      </div>
      <div style={{ maxHeight: '200px', overflowY: 'auto',
padding: '0 8px' }}>
        <ul>
          {playableSongs.map((song) => (
            <li key={song.id} className="mb-2 flex
justify-between items-center" onClick={() =>
handleSongItemClick(song.id)}>
              <div className="flex flex-auto items-center
hover:bg-neutral-700 p-2 rounded-lg relative" onMouseEnter={()
=> setHoveredSongId(song.id)} onMouseLeave={() =>
setHoveredSongId(null)}>
                <MediaItem data={song}
deleted={deletedSongs.includes(song.id)}
openedFromPlaylist={true} />
                {(hoveredSongId === song.id &&
!deletedSongs.includes(song.id)) && !isOpenedHome && (
                  <FiX className={`absolute right-2 top-
1/2 -translate-y-1/2 text-neutral-300 opacity-0 cursor-pointer
${hoveredSongId === song.id && !deletedSongs.includes(song.id) ?
'opacity-100' : ''}`} onClick={() => handleRemoveSong(song.id)}
/>
                )}
              </div>
            </li>
          )}
        </ul>
      </div>
    </div>
  </div>
);
};
export default AlbumModal;

```