Министерство науки и высшего образования Российской Федерации

Муромский институт (филиал)

Федерального государственного бюджетного образовательного учреждения высшего образования

«Владимирский государственный университет   
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Факультет ИТР

Кафедра ПИн

КУРСОВАЯ РАБОТА

По Разработка приложений для мобильных операционных систем

Тема «Музыкальный сервис»

Руководитель

Колпаков А.А

(фамилия, инициалы)

(подпись) (дата)

Студент ПИН - 121

(группа)

Маресев С. А.

(фамилия, инициалы)

(подпись) (дата)

Муром 2024

В данной курсовой работе необходимо было спроектировать приложение музыкального сервиса. В качестве средств разработки приложения была использована среда Android Studio. Язык разработки: Kotlin.

In this course work, it was necessary to design a taxi company application. The Android Studio environment was used as an app development tool. Development language: Kotlin.

Содержание

Введение………………………………………………………………...6

1. Анализ технического задания……………………………….………8

2. Разработка алгоритмов ………………………………………..........11 3. Руководство программиста............................……………….…......16

4. Руководство пользователя……………………...…………………...24

Заключение…………………………………….………………….........26

Список используемой литературы……………………….........………..28

Приложение 1………………………………………......………………...29

Приложение 2. Снимки окон программы................................................31

Введение

Современные музыкальные сервисы играют ключевую роль в сфере цифровых развлечений, предоставляя пользователям удобный доступ к музыкальному контенту. С развитием технологий взаимодействие с музыкой становится не только пассивным, но и активным процессом, включающим создание плейлистов, обмен мнениями и обсуждения различных произведений. Несмотря на то, что слушать музыку — одна из основных функций таких сервисов, не менее важным аспектом является возможность формирования персонализированного контента и взаимодействия пользователей через комментарии.

Программный продукт, представленный в данной курсовой работе, представляет собой музыкальный сервис, основной функционал которого сосредоточен на добавлении музыкальных треков в избранное и оставлении комментариев на них. В отличие от большинства музыкальных платформ, где основное внимание уделяется прослушиванию музыки, наш сервис ориентирован на организацию взаимодействия пользователей с музыкальным контентом через активное участие в формировании личных коллекций и обмене мнениями.

В системе хранятся данные о музыкальных композициях, альбомах и исполнителях. Пользователи могут добавлять музыкальные треки в свои плейлисты избранного и оставлять комментарии, делясь своими впечатлениями о композициях. Это позволяет создать пространство для обсуждения музыки, обмена рекомендациями и создания музыкальной коллекции, которая будет соответствовать индивидуальным предпочтениям каждого пользователя.

Особое внимание в проекте уделяется системе модерации комментариев. Это важный элемент, который позволяет поддерживать высокое качество взаимодействия между пользователями и предотвращать появление неподобающих материалов. Кроме того, предусмотрена возможность добавления новых треков в систему модераторами, что способствует обновлению контента и поддержанию актуальности музыкальной базы данных.

Цель данной курсовой работы — разработка музыкального сервиса, который предоставляет пользователям функционал для добавления треков в избранное и оставления комментариев. Система будет включать в себя эффективные инструменты для модерации контента и добавления новых произведений, обеспечивая тем самым удобство и безопасность для пользователей.

1. Анализ технического задания.

Для успешной реализации проекта необходимо внимательно рассмотреть все ключевые требования и спецификации, чтобы гарантировать правильность разработки и эффективность работы системы. В данном случае проект включает мобильное приложение на платформе Android с использованием языка Kotlin и бекенд, реализованный с помощью Spring и Java, взаимодействующий с базой данных PostgreSQL.

1.1. Функционал

- Сведения о музыкальных композициях: Приложение будет хранить данные о музыкальных треках, альбомах и исполнителях. Для каждого трека будет храниться информация, такая как название, исполнитель, альбом и другие метаданные. Пользователи смогут добавлять треки в избранное и оставлять комментарии к ним.

- Добавление комментариев и модерация: Пользователи могут оставлять комментарии под треками, которые будет проверять модератор. Модерация комментариев позволит обеспечить качество контента и предотвратить размещение неподобающих материалов.

- Управление контентом: Модераторы смогут добавлять новые музыкальные композиции в базу данных, а также редактировать или удалять существующие. Это будет осуществляться через административный интерфейс приложения.

- Регистрация пользователей и создание профилей: Пользователи смогут регистрироваться в системе и создавать персонализированные профили для удобства работы с избранными треками и комментариями.

1.2. Операционная система и языки программирования

- Операционная система Android: Мобильное приложение будет разрабатываться для платформы Android, что обеспечит доступ к большому количеству пользователей. Платформа Android является наиболее распространенной на мобильных устройствах, что делает приложение доступным для широкой аудитории.

- Язык программирования Kotlin: Использование Kotlin в качестве основного языка для разработки мобильного приложения обеспечит современный, лаконичный и безопасный код. Kotlin поддерживает современный подход к разработке приложений, предлагая улучшенные возможности для работы с асинхронным кодом, повышения производительности и безопасности.

1.3. Бекенд и база данных

- Spring и Java: Бекенд будет реализован на платформе Spring с использованием языка программирования Java. Spring предоставляет мощные средства для создания масштабируемых и высокопроизводительных приложений. Он будет обеспечивать обработку запросов от мобильного приложения, взаимодействие с базой данных и выполнение бизнес-логики.

- База данных PostgreSQL: В качестве системы управления базами данных будет использоваться PostgreSQL, что обеспечит надежное хранение данных о пользователях, музыкальных композициях, комментариях и других элементах системы. PostgreSQL отличается высокой производительностью и гибкостью, что идеально подходит для работы с большими объемами данных.

1.4. Документация

- Документирование с помощью Dokka: Для обеспечения прозрачности кода и удобства работы с ним будет использоваться инструмент для документирования — Dokka. Это позволит создавать структурированную и легко читаемую документацию, что упростит дальнейшую поддержку и развитие системы, а также улучшит процесс обучения новых разработчиков.

1.5. Система контроля версий

- Git и GitHub: Для управления версиями исходного кода будет использована система контроля версий Git. Репозиторий будет размещен на платформе GitHub, что обеспечит централизованное хранилище кода, возможность совместной работы над проектом, а также удобный доступ к истории изменений.

6. Этапы разработки

Каждый этап разработки будет тщательно спланирован и согласован с заказчиком, чтобы гарантировать успешную реализацию проекта. Важно будет пройти несколько ключевых этапов:

- Проектирование архитектуры: на этом этапе будут определены структуры базы данных, взаимодействие между бекендом и мобильным приложением, а также механизм модерации контента.

- Разработка мобильного приложения: будет реализован интерфейс для пользователей и модераторов, а также функционал добавления треков в избранное и оставления комментариев.

- Разработка бекенда: на этом этапе будут реализованы API для взаимодействия мобильного приложения с сервером, а также логику управления контентом.

Каждый из этих этапов будет требовать детального планирования, а также постоянного контроля качества разработки, чтобы система функционировала эффективно и соответствовала всем требованиям технического задания.

Таким образом, проект включает все необходимые компоненты для создания надежного музыкального сервиса с удобным пользовательским интерфейсом, эффективной модерацией контента и поддержкой стабильной работы на мобильной платформе Android.

2. Разработка алгоритмов

2.1. Общая структура приложения и клиент-серверное взаимодействие

Приложение состоит из мобильного клиента (разработанного на платформе Android с использованием Kotlin) и бекенд-сервера (реализованного на Spring с использованием Java). Приложение использует клиент-серверную архитектуру, где мобильное приложение (клиент) взаимодействует с сервером для выполнения различных задач, таких как регистрация пользователей, получение информации о треках, добавление комментариев и работа с избранным.

Мобильное приложение и сервер обмениваются данными через REST API, реализованное с помощью Spring Boot. Данные передаются в формате JSON, и взаимодействие осуществляется через стандартные HTTP-запросы (GET, POST, PUT, DELETE). Посмотреть диаграммы базы данных можно в приложении 1.

Основные компоненты системы:

Модель (Model): В мобильном приложении это объекты, которые будут отображаться на UI, такие как SongDto, CommentDto, UserDto и т. д. Они соответствуют DTO сущностям на сервере.

Контроллер (Controller): На клиенте это активити, которые управляют взаимодействием пользователя с приложением, выполняя запросы к серверу и обрабатывая ответы. На сервере это компоненты Spring, которые обрабатывают HTTP-запросы от клиента.

Представление (View): В мобильном приложении это интерфейсы пользователя, представленные через XML-макеты в Android, где отображаются данные, полученные с сервера.

2.2. Логика работы с сервером и базой данных

2.2.1. Сервер (Spring Backend)

Spring Boot служит для реализации серверной части приложения. База данных будет хранить данные о пользователях, музыкальных треках, комментариях и избранных треках. Для работы с базой данных используется PostgreSQL.

На сервере создаются REST API для:

* Регистрации и авторизации пользователей.
* Управления музыкальными композициями.
* Работы с комментариями (добавление, модерация).
* Добавления треков в избранное.

Пример эндпоинта для регистрации с сервера:

@RestController

@RequestMapping("/api/auth")

@RequiredArgsConstructor

public class AuthRestController {

private final AuthenticationService authenticationService;

@PostMapping("/sign-up")

public JwtAuthenticationResponse signUp(@RequestBody @Valid SignUpRequest request) {

return authenticationService.signUp(request);

}

}

2.2.2. База данных (PostgreSQL)

Для хранения данных о пользователях, музыкальных треках, комментариях и избранных треках используется PostgreSQL.

Таблицы в базе данных:

* Users: хранит информацию о пользователях (логин, пароль, email, роль).
* Song: информация о музыкальных композициях (название, исполнитель, альбом, длительность, жанр).
* Comments: комментарии пользователей под треками.
* Playlists: информация о любимых треках пользователя.
* Playlist\_songs: Промежуточная таблица между плейлистами и треками.

Пример SQL-запроса для удаления всех трека из всех плейлистов:

DELETE FROM Playlist p WHERE :song MEMBER OF p.songs

2.3. Алгоритмы работы с клиентом (мобильное приложение)

2.3.1. Регистрация и авторизация

Алгоритм регистрации:

Пользователь вводит данные (логин, email, пароль) в мобильном приложении.

Мобильное приложение отправляет POST-запрос на сервер с данными пользователя.

Сервер обрабатывает запрос, сохраняет данные в базе данных и возвращает ответ о успешной регистрации.

Мобильное приложение сохраняет данные о пользователе в SharedPreferences для дальнейшей работы.

Пример кода для отправки данных с клиента:

fun signUp(request: SignUpRequest, callback: (JwtAuthenticationResponse?) -> Unit) {

authApi.signUp(request).enqueue(object : Callback<JwtAuthenticationResponse> {

override fun onResponse(call: Call<JwtAuthenticationResponse>, response: Response<JwtAuthenticationResponse>) {

if (response.isSuccessful) {

// Если запрос успешен, передаем тело ответа в callback

callback(response.body())

} else {

// Логирование кода ошибки

Log.e("AuthRepository", "Ошибка регистрации: ${response.code()} - ${response.message()}")

callback(null) // Передаем null в случае ошибки

}

}

override fun onFailure(call: Call<JwtAuthenticationResponse>, t: Throwable) {

// Логирование ошибки при запросе

Log.e("AuthRepository", "Ошибка сети: ${t.message}")

callback(null) // Передаем null при сетевой ошибке

}

})

}

Алгоритм авторизации:

Пользователь вводит логин и пароль.

Мобильное приложение отправляет POST-запрос на сервер с данными для авторизации.

Сервер проверяет данные и возвращает ответ (например, токен для дальнейших запросов).

Мобильное приложение сохраняет токен в SharedPreferences для последующих запросов.

2.3.2. Работа с музыкальными треками

Алгоритм получения списка треков:

Мобильное приложение отправляет GET-запрос на сервер для получения списка музыкальных треков.

Сервер возвращает список треков в формате JSON.

Мобильное приложение обрабатывает ответ и отображает список треков в интерфейсе с помощью RecyclerView.

2.3.3. Работа с комментариями и избранным

Алгоритм добавления комментария:

Пользователь вводит комментарий для трека.

Мобильное приложение отправляет POST-запрос на сервер с данными комментария.

Сервер сохраняет комментарий в базе данных и возвращает статус успешного добавления.

Мобильное приложение отображает новый комментарий.

Пример запроса для добавления комментария:

fun addCommentToSong(songId: Long, content: String, callback: (CommentDto?) -> Unit) {

api.addCommentToSong(songId, content).enqueue(object : Callback<CommentDto> {

override fun onResponse(call: Call<CommentDto>, response: Response<CommentDto>) {

callback(response.body())

}

override fun onFailure(call: Call<CommentDto>, t: Throwable) {

callback(null)

}

})

}

Алгоритм добавления трека в избранное:

Пользователь нажимает кнопку "Добавить в избранное".

Мобильное приложение отправляет POST-запрос на сервер для добавления трека в избранное.

Сервер сохраняет информацию о добавленном треке в таблице Playlists.

Мобильное приложение обновляет состояние кнопки и отображает трек в разделе избранного.

3. Руководство программиста

Приложение включает в себя работу с Rest Api, аутентификацию и регистрацию пользователей, а также функциональность для работы с комментариями и плейлистами. Руководство охватывает основные моменты работы с каждым компонентом приложения, начиная с взаимодействия с сервером и заканчивая обработкой пользовательского интерфейса.

Структура проекта подразумевает разделение обязанностей и функционала между различными компонентами:

1. api: обеспечивает доступ к удалённым API для выполнения операций, таких как авторизация, получение данных или выполнение действий (например, модерация, управление песнями).

Используется для работы с REST API через библиотеку Retrofit.

Подпакеты: admin, moderator, user содержат интерфейсы для разных ролей пользователей.

AuthApi: специализируется на запросах, связанных с авторизацией и аутентификацией (вход, регистрация, выход).

2. data: этот пакет предназначен для хранения данных, которые используются в приложении. Содержит DTO (Data Transfer Objects), которые помогают передавать данные между слоями приложения и сервером. DTO используются для сериализации/десериализации данных при работе с API.

3. fragments: отвечает за представление (UI) и взаимодействие с пользователем. Содержит фрагменты (компоненты пользовательского интерфейса), которые связаны с разными типами пользователей приложения.

Фрагменты разделены по ролям:

admin: интерфейс для администраторов (управление пользователями).

moderator: интерфейс для модераторов (модерация комментариев и добавление песен).

user: интерфейс для обычных пользователей (просмотр песен, добавление в избранное, создание комментариев).

4. repo: отвечает за логику работы с данными, слой архитектуры, который управляет данными.

Здесь реализованы методы получения, сохранения и обработки данных из API.

Подпакеты:

admin, moderator, user: содержат классы с бизнес-логикой, относящейся к соответствующим ролям.

AuthRepository:

Управляет процессами аутентификации, такими как вход, регистрация, выход.

5. utils: пакет служит для хранения вспомогательных классов и функций, которые используются в разных частях приложения.

AuthInterceptor: используется для перехвата сетевых запросов и добавления, например, токена аутентификации в заголовки запросов.

PreferencesHelper: упрощает работу с SharedPreferences для сохранения локальных данных (токенов и ролей).

RetrofitClient: отвечает за настройку и создание экземпляра Retrofit для работы с API.

Активности:

MainActivity: главный экран приложения, содержит навигацию между фрагментами.

SignInActivity и SignUpActivity: экраны для входа и регистрации.

3.1 MainActivity: Главная активность

MainActivity отвечает за отображение основного интерфейса приложения и организацию навигации между разными экранами (фрагментами), адаптируя функционал под роль пользователя: администратор, модератор или обычный пользователь.

Описание методов:

onCreate()

Основной метод для инициализации UI.

Настраивает BottomNavigationView, который используется для навигации.

Получает роль пользователя из SharedPreferences (через PreferencesHelper) и вызывает метод setupNavigation() для настройки вкладок навбара в зависимости от роли.

setupNavigation(role: String)

Определяет, какие вкладки и фрагменты будут отображаться в BottomNavigationView в зависимости от роли:

ROLE\_ADMIN: вызывает setupAdminNavigation().

ROLE\_MODERATOR: вызывает setupModeratorNavigation().

ROLE\_USER: вызывает setupUserNavigation().

setupAdminNavigation()

Очищает меню навбара и добавляет вкладки для администраторов:

"Админ" (AdminFragment).

"Пользователи" (AdminUserFragment).

По умолчанию открывается AdminFragment.

setupModeratorNavigation()

Очищает меню навбара и добавляет вкладки для модераторов:

"Модератор" (ModeratorFragment).

"Песни" (ModeratorSongFragment).

"Пользователи" (ModeratorUserFragment).

По умолчанию открывается ModeratorFragment.

setupUserNavigation()

Очищает меню навбара и добавляет вкладки для пользователей:

"Профиль" (UserProfileFragment).

"Песни" (UserSongFragment).

По умолчанию открывается UserProfileFragment.

switchFragment(fragment: Fragment)

Метод для замены текущего фрагмента на новый.

Используется при нажатии на вкладки навбара.

3.2 SignInActivity: Экран авторизации

SignInActivity отвечает за процесс входа пользователя в систему. Здесь реализована проверка учетных данных и взаимодействие с репозиторием для аутентификации.

Описание методов:

onCreate()

Основной метод для инициализации интерфейса экрана.

Настраивает элементы ввода (логин, пароль) и кнопки:

signInButton: обработчик нажатия выполняет проверку введенных данных и отправляет запрос на сервер для входа.

goToSignUpLink: позволяет пользователю перейти на экран регистрации.

Процесс входа:

Собирает логин и пароль из полей ввода.

Если данные введены корректно, формируется объект SignInRequest.

Вызывается метод authRepository.signIn() для отправки запроса аутентификации на сервер.

При успешном входе:

Логирует ответ.

Сохраняет токен и роль пользователя с помощью PreferencesHelper.

Перенаправляет пользователя на MainActivity, сбрасывая стек активностей.

При ошибке:

Логирует сообщение и показывает пользователю уведомление.

Обработка ошибок:

Показывает уведомления, если поля ввода пусты или сервер вернул ошибку.

3.3 SignUpActivity: Экран регистрации

SignUpActivity отвечает за регистрацию новых пользователей. Она предоставляет пользователю интерфейс для ввода данных, а затем отправляет запрос на сервер для создания учетной записи.

Описание методов:

onCreate()

Основной метод для инициализации UI.

Настраивает поля ввода и кнопки:

signUpButton: обработчик регистрации, отправляет запрос с данными пользователя.

goToSignInLink: позволяет вернуться на экран входа (SignInActivity).

Процесс регистрации:

Собирает данные пользователя из полей: имя пользователя, email, пароль.

Если все поля заполнены, создается объект SignUpRequest.

Вызывается метод authRepository.signUp() для отправки запроса регистрации:

При успешной регистрации:

Показывает сообщение об успехе.

Перенаправляет пользователя на экран входа (SignInActivity).

Завершает текущую активность (finish()).

При ошибке:

Показывает сообщение об ошибке.

Обработка ошибок:

Показывает уведомление, если поля ввода пусты или произошла ошибка регистрации.

3.4 RetrofitClient: Клиент для работы с API

RetrofitClient отвечает за настройку и предоставление клиентских интерфейсов API, а также за управление авторизацией через интерцепторы.

Основные функции:

Настройка OkHttpClient:

Создает клиент с использованием токена для авторизации.

Использует AuthInterceptor, чтобы добавлять токен в заголовки запросов.

Создание Retrofit-клиента:

Настраивает базовый URL и конвертер JSON (GsonConverterFactory).

Добавляет OkHttpClient для обработки запросов с учетом авторизации.

Методы для получения API-интерфейсов:

getAuthApi(): Возвращает API для работы с аутентификацией без необходимости токена (например, для входа/регистрации).

getAdminUserApi(context): API для работы с пользователями на стороне администратора.

getModeratorSongApi(context): API для управления песнями на стороне модератора.

getUserSongApi(context): API для взаимодействия с песнями на стороне пользователя.

getModeratorUserApi(context): API для управления пользователями модераторами.

getUserProfileApi(context): API для работы с профилем пользователя.

Детали реализации:

Токен из SharedPreferences:

Использует PreferencesHelper для извлечения сохраненного токена авторизации.

Авторизация через AuthInterceptor:

Добавляет токен в заголовки всех запросов, требующих авторизации.

Гибкость создания API:

API для аутентификации не требует токена, тогда как все остальные API зависят от токена пользователя, предоставленного через AuthInterceptor.

3.5 AuthInterceptor: Интерцептор для авторизации

AuthInterceptor — это класс, отвечающий за добавление токена авторизации в заголовки HTTP-запросов.

Основные функции:

Добавление токена:

Каждый запрос, отправляемый через OkHttpClient, автоматически получает заголовок Authorization с токеном в формате Bearer {token}.

Описание метода:

intercept(chain: Interceptor.Chain): Response

Создает новый запрос с добавлением заголовка Authorization.

Возвращает результат выполнения модифицированного запроса.

3.6 PreferencesHelper: Хранилище пользовательских данных

PreferencesHelper — это объект, отвечающий за управление данными пользователя, сохраненными в SharedPreferences.

Основные функции:

Сохранение и получение токена:

saveToken(context, token): Сохраняет токен авторизации в SharedPreferences.

getToken(context): Извлекает сохраненный токен.

clearToken(context): Удаляет токен из хранилища.

Сохранение и получение роли:

saveRole(context, role): Сохраняет роль пользователя (например, "ROLE\_USER", "ROLE\_ADMIN").

getRole(context): Получает текущую роль пользователя.

clearRole(context): Удаляет роль пользователя.

Вспомогательный метод:

getSharedPreferences(context): Возвращает экземпляр SharedPreferences для текущего приложения.

4. Руковоство пользователя

Руководство пользователя для мобильного приложения

Это руководство поможет вам освоиться с основными функциями и страницами нашего Android-приложения. В нем описаны ключевые экраны и действия, которые доступны пользователю. Просмотреть изображения интерфейса можно перейдя в приложение 2.

4.1. Стартовая страница

При открытии приложения отображается страница входа с двумя основными возможностями:

Вход в аккаунт:

Требуется ввести имя пользователя и пароль. После успешного входа вам будет предоставлен доступ к функционалу, соответствующему вашей роли (Администратор, Модератор или Пользователь).

Регистрация:

Для новых пользователей доступна возможность регистрации с вводом имени, электронной почты и пароля. После успешной регистрации можно авторизоваться в системе.

4.2. Навигация после входа

Интерфейс приложения меняется в зависимости от роли пользователя.

4.2.1 Администратор

Доступные вкладки навигации:

Пользователи: возможность просмотра списка пользователей, добавление нового пользователя, удаление существующих пользователей.

Админ: выход из аккаунта

При нажатии на кнопку "Выход" происходит завершение текущей сессии, и пользователь возвращается на стартовую страницу.

4.2.2 Модератор

Доступные вкладки навигации:

Модератор: возможность выхода из аккаунта.

Песни: поиск песен в библиотеке, добавление новых песен в библиотеку, удаление существующих песен.

Меню трека (при нажатии на трек):

Редактировать трек: возможность изменить название, исполнителя или другие данные.

Комментарии: просмотр комментариев к треку, удаление нежелательных комментариев.

Пользователи: просмотр любимых плейлистов пользователей, просмотр комментариев пользователей с возможностью их удаления.

4.2.3 Пользователь

Доступные вкладки навигации:

Профиль: отображение информации о пользователе.

Меню профиля (при нажатии на имя):

Выход из аккаунта.

Просмотр всех оставленных комментариев.

Просмотр списка избранного с возможностью удаления треков из него.

Песни: поиск песен.

Меню трека (при нажатии на трек):

Добавить/Удалить в избранное: Управление личным списком избранного.

Комментарии:

Просмотр комментариев к треку.

Возможность оставить собственный комментарий.

Заключение

В данной курсовой работе были успешно реализованы следующие задачи:

Основные этапы разработки:

- Проектирование интерфейса: была создана простая и интуитивно понятная структура приложения, обеспечивающая комфортное взаимодействие пользователей с его функционалом.

Стартовая страница предоставляет возможность пользователям войти в систему или зарегистрироваться, заполнив необходимые данные.

После входа в систему интерфейс динамически изменяется в зависимости от роли пользователя (администратор, модератор, пользователь).

Администратор получает доступ к управлению пользователями.

Модератор работает с треками, комментариями и пользователями.

Обычный пользователь может управлять своим профилем, списком избранного и оставлять комментарии.

- Реализация ключевых функций: основной функционал приложения адаптирован под каждую роль:

Администратор может добавлять новых пользователей и удалять существующих.

Модератор имеет доступ к управлению треками (добавление, удаление, редактирование) и комментариями, а также может просматривать любимые плейлисты пользователей.

Пользователь может добавлять и удалять треки из избранного, оставлять комментарии, а также просматривать свои избранные треки и комментарии.

Функционал взаимодействия с сервером включает обработку аутентификации, авторизации и управление данными пользователей.

- Удобство для пользователей: приложение предоставляет удобный и гибкий интерфейс для быстрого выполнения всех действий.

Каждый пользователь получает доступ только к тем разделам и функциям, которые соответствуют его роли.

Возможность оставлять комментарии, добавлять треки в избранное, просматривать плейлисты и управлять ими делает взаимодействие максимально удобным и функциональным.

Разработанное приложение обеспечивает безопасное взаимодействие между пользователями, модераторами и администраторами, предоставляя удобный интерфейс и функционал:

Для пользователей — это средство для работы с треками и их избранным.

Для модераторов — эффективное управление контентом и контроль за пользователями.

Для администраторов — возможность администрирования системы и управления учетными записями.

Учитывая интуитивность интерфейса, безопасность данных и широкий функционал, приложение может стать надежным инструментом для управления музыкальным контентом и взаимодействия с пользователями, обеспечивая удобство, безопасность и эффективность. Просмотреть код продукта можно по ссылке на репозиторий GitHub - https://github.com/focs1t/Lopify/tree/fix-playlists.

Список литературы:

1. Введение в разработку приложений для смартфонов на ОС Android / А.Семакова Национальный открытый Университет «ИНТУИТ», 2016
2. Колисниченко Д.Н. Программирование для Android 5. Самоучитель. — СПб.: БХВ-Петербург, 2015. — 303 с.
3. 2016. Дейтел П., Дейтел X., Уолд A. Android для разработчиков. 3-е изд. — СПб.: Питер, Гриффитс Дэвид, Гриффитс Дон Head First. Программирование для Android. 2-e изд. — СПб.: Питер, 2018. — 912 с.

Приложение

Приложение 1. Модели баз данных

Изображение выглядит как диаграмма, круг, Детское искусство, дизайн

Автоматически созданное описание

Рисунок 1 - Концептуальная модель данных

Изображение выглядит как текст, снимок экрана, диаграмма, число

Автоматически созданное описание

Рисунок 2 – Логическая модель данных

Изображение выглядит как текст, диаграмма, число, снимок экрана

Автоматически созданное описание

Рисунок 3 – Физическая модель данных

Приложение 2. Снимки окон программы

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

Рисунок 1 – Стартовое окно(окно входа)

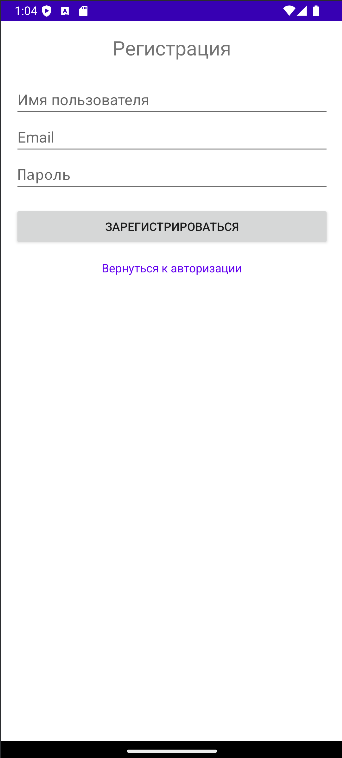


Рисунок 2 – Окно регистрации

Изображение выглядит как текст, снимок экрана, Веб-сайт, программное обеспечение

Автоматически созданное описание

Рисунок 3 – Вид приложения после авторизации

с ролью пользователя

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рисунок 4 – Вкладка песен

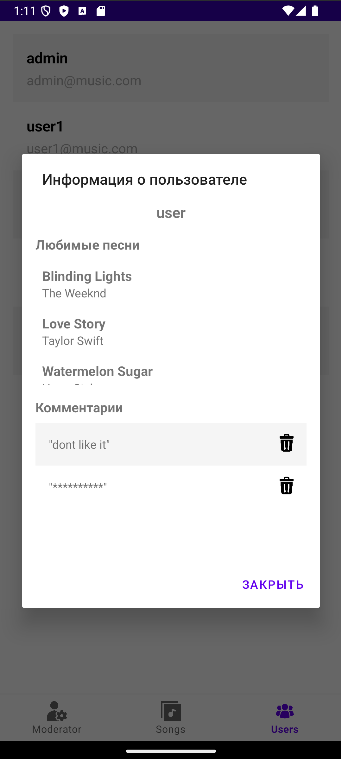


Рисунок 5 – Вкладка пользователей у модератора

с отображением информации о пользователе

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 7 – Меню добавления песни

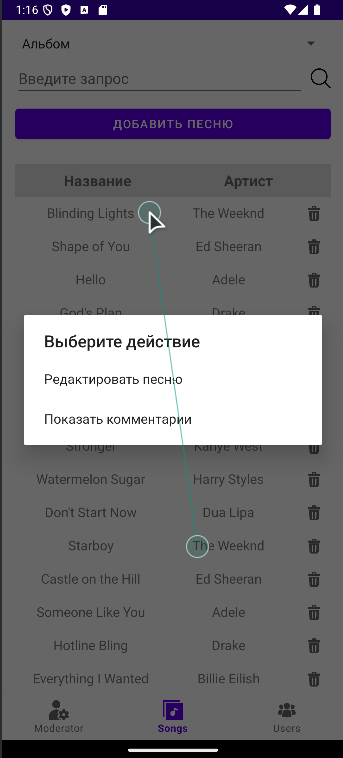


Рисунок 8 – Контекстное меню при нажатии на песню

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

Рисунок 9 – Вкладка пользователей у админа

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 10 – Меню создания пользователя админом