Федеральное государственное бюджетное образовательное учреждение высшего образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

Кафедра информационных систем и программной инженерии

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту по дисциплине   
"Распределенные программные системы"**

**на тему**

Проектирование и разработка программной системы   
«Музыкальный стриминговый сервис»

Выполнил: ст. гр. ПРИ-123

Майоров М. С.

Принял: Тимофеев А.А.

Владимир, 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc217415772)

[1. Описание предметной области 5](#_Toc217415773)

[1.1 Анализ предметной области 5](#_Toc217415774)

[1.2 Словарь предметной области 5](#_Toc217415775)

[1.3 Пользователи системы и роли 6](#_Toc217415776)

[1.3.1 Характеристика аудитории программной системы 6](#_Toc217415777)

[1.3.2 Расчетное количество пользователей системы 8](#_Toc217415778)

[1.3.3 Перечень ролей пользователей 9](#_Toc217415779)

[2. проектирование системы 10](#_Toc217415780)

[2.1 Диаграмма прецедентов 10](#_Toc217415781)

[2.2 Диаграмма классов 14](#_Toc217415782)

[2.3 Диаграмма состояний 18](#_Toc217415783)

[2.4 Диаграммы видов деятельности и BPMN 20](#_Toc217415784)

[2.5 Описание зон ответственности пользователей системы 23](#_Toc217415785)

[2.6 Функциональные требования к системе 24](#_Toc217415786)

[2.7 Нефункциональные требования к системе 25](#_Toc217415787)

[3. StATE OF ART 27](#_Toc217415788)

[3.1 Обзор актуальных технологий 27](#_Toc217415789)

[3.2 Анализ аналогов 27](#_Toc217415790)

[4. Модель данных 30](#_Toc217415791)

[4.1 Схема базы данных 30](#_Toc217415792)

[4.2 Механизмы обеспечения целостности данных 33](#_Toc217415793)

[4.3 SQL-скрипт базы данных 35](#_Toc217415794)

[5. Макет Программной системы 39](#_Toc217415795)

[5.1 Описание принципов организации интерфейса 39](#_Toc217415796)

[5.2 Диаграмма навигации 41](#_Toc217415797)

[5.3 Макеты страниц 43](#_Toc217415798)

[5.4 Верстка страниц 46](#_Toc217415799)

# ВВЕДЕНИЕ

**Цель проекта:** разработать программную систему музыкального стримингового сервиса, обеспечивающую удобный доступ к многопользовательской библиотеке аудиоконтента и эффективное управление музыкальным каталогом.

**Задачи проекта:**

1. Провести анализ предметной области "Музыкальный стриминговый сервис" и выявить функциональные требования к системе.
2. Спроектировать архитектуру программной системы на основе UML-моделей (диаграммы прецедентов, диаграмм классов и др.).
3. Реализовать back-end и front-end части системы.
4. Провести функциональное тестирование системы.

**Краткая характеристика предметной области:**

Современный музыкальный стриминг — это область, предоставляющая пользователям легальный доступ к обширным библиотекам аудиоконтента через интернет на основе подписки или бесплатной модели с рекламой. Ключевыми процессами в данной области являются управление цифровым каталогом (добавление треков, альбомов, исполнителей), организация потоковой передачи аудио данных а также ведение статистики прослушиваний для анализа предпочтений аудитории. Автоматизация этих процессов позволяет сервису предоставлять мгновенный доступ к миллионам композиций, формировать уникальные плейлисты для каждого.

# 1. Описание предметной области

1.1 Анализ предметной области

**Описание объекта информатизации:** Объектом информатизации является музыкальный стриминговый сервис. Система предоставляет доступ к библиотеке аудиоконтента, позволяет прослушивать музыку в потоковом режиме, создавать персонализированные плейлисты, управлять правами на контент и анализировать статистику прослушиваний. Границы системы включают в себя: каталог музыки, пользовательские аккаунты, систему рекомендаций, панель управления для администраторов и музыкантов.

**Актеры:**

Гость - неавторизованный пользователь;

Пользователь - зарегистрированный пользователь без подписки;

Подписчик - зарегистрированный пользователь с активной подпиской;

Музыкант - контент-мейкер, загружающий музыку;

Администратор - управляет системой, правами, статистикой.

1.2 Словарь предметной области

**Трек (Аудиозапись, Композиция)** – Базовая единица контента в системе, представляющая собой одну музыкальную композицию.

**Исполнитель (Артист, Музыкант, Контент-мейкер)** - Субъект (сольный музыкант, группа, коллектив), создающий музыкальный контент.

**Альбом** – Коллекция треков, выпущенных вместе как единый продукт (студийный альбом, сингл, EP — мини-альбом).

**Плейлист** - Подборка треков, сформированная по определенному принципу (тематика, настроение, пользователем или редакцией сервиса).

**Жанр** - Категория, характеризующая музыкальный контент по стилистическим особенностям (например, рок, поп, классика, электронная музыка).

**Подписка** - Основная бизнес-модель сервиса, при которой пользователь регулярно (ежемесячно) оплачивает доступ к полному функционалу.

1.3 Пользователи системы и роли

1.3.1 Характеристика аудитории программной системы

**1. Демографический портрет**

Возрастные группы:

16-25 лет (45%): Студенты, старшеклассники, молодые специалисты;

26-35 лет (30%): Офисные работники, профессионалы, семейные люди;

36-45 лет (15%): Сложившиеся профессионалы, родители;

46+ лет (10%): Зрелая аудитория с устоявшимися предпочтениями.

География:

Крупные города (60%): Высокая проникновение интернета, мобильность;

Региональные центры (30%): Растущий потенциал, ценовая чувствительность;

Малая населенность (10%): Ограниченный охват, проблемы со связью.

**2. Поведенческие характеристики**

Паттерны использования:

Ежедневное использование (85% пользователей);

Пиковые часы активности: утро (7-10), обед (12-14), вечер (18-22);

Средняя длительность сессии: 45-90 минут.

**3. Техническая грамотность и привычки**

Уровень технической компетенции:

Продвинутые пользователи (40%): активно используют все функции;

Средние пользователи (45%): базовые функции, постепенное освоение;

Начинающие (15%): требуется простота и интуитивность.

Предпочитаемые платформы:

Мобильные устройства (65% прослушиваний);

Десктоп (25% прослушиваний);

Умные колонки и TV (10% прослушиваний).

**4. Потребности и "боли"**

Функциональные потребности:

- Бесперебойный доступ к музыке;

- Персонализированные рекомендации;

- Качественный звук;

- Простота поиска и навигации.

Эмоциональные потребности:

- Открытие новой музыки;

- Создание атмосферы и настроения;

- Социальная идентичность через музыкальные предпочтения;

- Экономия времени на поиск музыки.

Ключевые "боли":

- Необходимость переключаться между разными источниками музыки;

- Ограниченность личной музыкальной библиотеки;

- Сложность поиска новой музыки по вкусу;

- Низкое качество звука в пиратских источниках;

- Отсутствие персонализации.

1.3.2 Расчетное количество пользователей системы

**Методология расчета** основана на анализе:

Рыночной доли в нише музыкального стриминга;

Темпов роста аналогичных сервисов;

Потенциала локального рынка;

Этапов развития проекта.

**Расчет по этапам развития**

Первый год (Запуск и ранний рост):

Квартал 1: 25-100 пользователей (разработчики)

Квартал 2: 1000-1500 пользователей (ранние клиенты)

Квартал 3: 2500-4000 пользователей (локальный маркетинг)

Квартал 4: 5000-8000 пользователей (органический рост)

Итог за первый год: ≈ 7000 пользователей

Второй год (Активный рост):

Квартал 1: 10000-12000 пользователей

Квартал 2: 15000-18000 пользователей

Квартал 3: 20000-24000 пользователей

Квартал 4: 25000-30000 пользователей

Итог за второй год: ≈ 25000 пользователей

Третий год (Стабилизация и масштабирование):

Начало года: 35000 пользователей

Конец года: 100000 пользователей

Итог за третий год: ≈ 100000 пользователей

Данные расчеты позволяют планировать инфраструктуру, маркетинговый бюджет и ресурсы разработки с учетом поэтапного роста аудитории сервиса.

1.3.3 Перечень ролей пользователей

Гость - неавторизованный пользователь;

Пользователь - зарегистрированный пользователь без подписки;

Подписчик - зарегистрированный пользователь с активной подпиской;

Музыкант - контент-мейкер, загружающий музыку;

Администратор - управляет системой, правами, статистикой.

# 2. проектирование системы

## 2.1 Диаграмма прецедентов

На рисунке 1 представлена диаграмма прецедентов для программной системы «Музыкальный стриминговый сервис». Она отражает взаимодействие актеров с системой.

**Прецеденты:**

**Общие для всех (включая Гостя):**

* Просмотр главной страницы
* Поиск треков
* Прослушивание трека
* Регистрация
* Авторизация

**Только для авторизованных пользователей (User):**

* Просмотр профиля
* Выход из системы
* Покупка подписки
* Просмотр условий подписки
* Просмотр своих подписок

**Только для подписчиков (Subscriber):**

* Добавление трека в плейлист
* Управление своими плейлистами
* Прослушивание без рекламы

**Только для музыкантов (Musician):**

* Загрузка нового трека
* Получение информации о загруженных треках

**Только для администраторов (Admin):**

* Управление сервисом
* Модерация контента
* Просмотр статистики
* Просмотр админ-панели

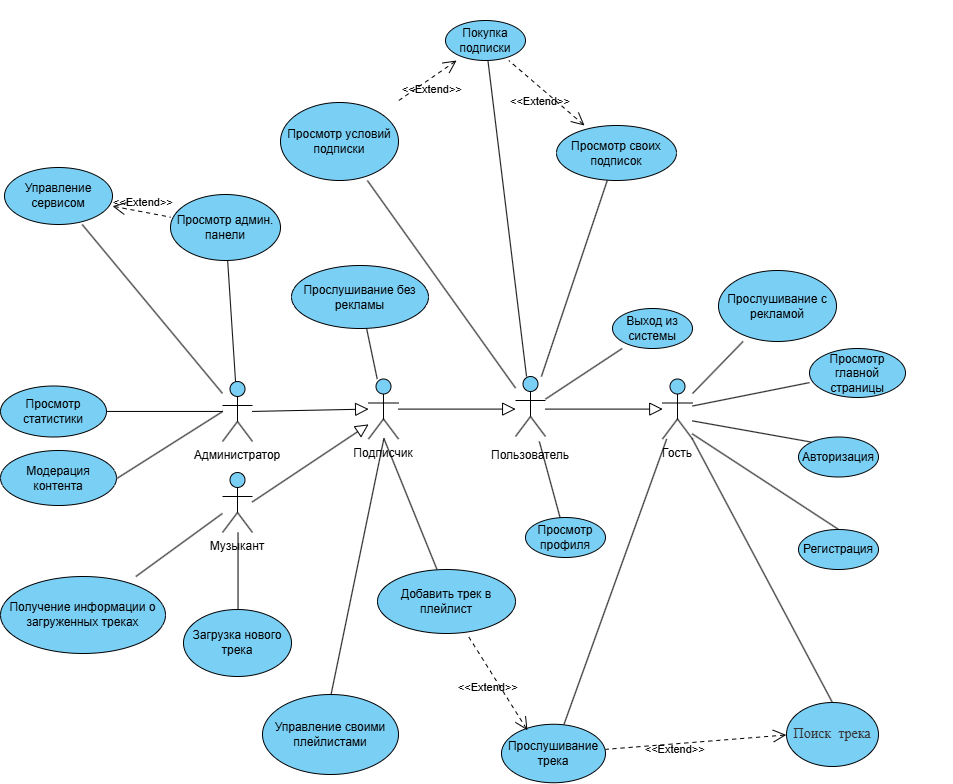


Рисунок 1. Диаграмма прецедентов.

**Спецификация основных прецедентов:**

**Описание прецедента «поиск трека»**

**Название:** «Поиск и прослушивание трека»

**Предусловие:** Пользователь поисковую форму приложения.

**Действующее лицо:** Гость, Пользователь, Подписчик

**Основной поток 1:** Пользователь вводит название трека, альбома или исполнителя в поисковую строку. 2. Система отображает список найденных результатов.

**Альтернативный поток A Трек не найден:** На шаге 2 система не находит результатов. Пользователь может: а) Уточнить поисковый запрос. б) Прервать выполнение прецедента.

**Постусловие:** Трек найден.

**Описание прецедента «покупка подписки»**

**Название:** «Покупка подписки»

**Предусловие:** Пользователь авторизован в системе и имеет роль «Пользователь» (без активной подписки).

**Действующее лицо:** Пользователь

**Основной поток:** 1. Пользователь переходит в раздел «Подписка» или кликает на предложение о её приобретении. 2. Система отображает доступные тарифные планы. 3. Пользователь выбирает тариф и нажимает «Оплатить». 4. Система перенаправляет пользователя на страницу платежного шлюза. 5. Пользователь подтверждает оплату. 6. Платежная система подтверждает успешную транзакцию. 7. Система активирует для пользователя статус «Подписчик» и открывает полный доступ к функционалу.

**Альтернативный поток Ошибка оплаты:** На шаге 6 платежная система сообщает об ошибке. Система уведомляет пользователя и предлагает повторить попытку.

**Постусловие:** Роль пользователя изменена на «Подписчик».

**Описание прецедента «загрузка нового трека»**

**Название:** «Загрузка нового трека»

**Предусловие:** Пользователь имеет роль «Музыкант».

**Действующее лицо:** Музыкант

**Основной поток:** 1. Музыкант переходит в личный кабинет и выбирает опцию «Загрузить новый трек». 2. Система отображает форму для загрузки. 3. Музыкант загружает аудиофайл и заполняет метаданные (название, жанр, принадлежность к альбому). 4. Музыкант нажимает кнопку «Сохранить». 5. Система проверяет данные и сохраняет трек в состоянии «На модерации». 6. Система уведомляет Администратора о новом треке для проверки.

**Альтернативный поток Ошибка валидации:** На шаге 5 система обнаружила ошибку в данных (некорректный формат файла, не заполнены обязательные поля). Система сообщает об ошибке, и Музыкант должен исправить данные.

**Постусловие:** Новый трек создан в системе и ожидает модерации.

**Описание прецедента «Создание плейлиста»**

**Название:** «Создание плейлиста»

**Предусловие:** Пользователь имеет роль "Подписчик".

**Действующее лицо:** Подписчик

**Основной поток:** 1. Пользователь переходит в раздел "Мои плейлисты" 2. Нажимает кнопку "Создать плейлист" 3. Вводит название плейлиста 4. Система создает пустой плейлист.

**Альтернативный поток На шаге 3:** Пользователь отменяет создание. Система возвращает к списку плейлистов.

**Постусловие:** Создан новый плейлист в библиотеке пользователя.

## 2.2 Диаграмма классов

Диаграмма классов представлена на рисунке 2, она отражает основные сущности системы (Пользователь, Трек, Плейлист, Альбом, Исполнитель, Жанр, Подписка, Модерация, Статистика) и их взаимодействия.

**Класс User**

Атрибуты: Id, Username, Email, PasswordHash, Role, DateOfCreated, HasActiveSubscription, CanUploadTracks

Методы: Register(), Login(), Logout(), Profile(), SetPassword(), VerifyPassword()

Описание: Описывает участников системы с различными ролями и правами доступа. Пользователи могут быть: обычными пользователями (User), подписчиками (Subscriber), музыкантами (Musician) или администраторами (Admin). Музыканты и администраторы могут загружать треки, подписчики имеют доступ к созданию плейлистов.

**Класс Track**

Атрибуты: Id, Title, FilePath, Duration, IsModerated, CoverImage, ColorHash, ModerationStatus, UploadedByUserId

Методы: Play(), Pause(), GetStatistics ()

Описание: Является центральной сущностью системы, хранит информацию о музыкальных композициях, их метаданные и статус модерации. Каждый трек имеет аудиофайл, длительность, принадлежность к жанру, альбому и исполнителю.

**Класс Playlist**

Атрибуты: Id, Title, Description, CreatedDate, UpdatedDate, TrackCount, TotalDuration

Методы: Create(), Edit(), AddTrack(), RemoveTrack(), Reorder(), Delete()

Описание: Представляет пользовательские подборки треков. Доступ к созданию плейлистов имеют подписчики, музыканты и администраторы.

**Класс Album**

Атрибуты: Id, Title, ReleaseDate, CoverPath

Методы: AddTrack(), GetStatistics()

Описание: Организует треки в официальные музыкальные релизы исполнителей. Альбом содержит коллекцию треков, имеет обложку и дату выпуска.

**Класс Artist**

Атрибуты: Id, Name, Description, PhotoPath

Методы: UploadTrack(), EditProfile()

Описание: Описывает создателей музыкального контента, управляет их профилями и каталогом произведений. Исполнитель может выпускать альбомы и отдельные треки.

**Класс Genre**

Атрибуты: Id, Name

Методы: GetTracks()

Описание: Категоризирует музыкальные произведения по стилевой принадлежности, обеспечивая навигацию и систему рекомендаций.

**Класс Subscription**

Атрибуты: Id, StartDate, EndDate, IsActivated, Amount, Status, TransactionId

Методы: Activate(), CheckExpiredSubscriptions(), Create()

Описание: Реализует бизнес-модель сервиса, управляя подпиской, ее сроками. Подписка предоставляет доступ к премиум-функциям, включая создание плейлистов.

**Класс Moderation**

Атрибуты: Id, Status, Comment, ModerationDate

Методы: Approve(), Reject(), GetHistory()

Описание: Обеспечивает контроль качества контента через процесс проверки и утверждения загружаемых треков. Каждый трек проходит модерацию администратором перед публикацией.

**Класс TrackStatistics**

Атрибуты: Id, Date, ListenCount

Описание: Аккумулирует данные о прослушиваниях треков, обеспечивая аналитику для администраторов и систему рекомендаций для пользователей.

**Класс AlbumStatistics**

Атрибуты: Id, Date, ListenCount

Описание: Собирает статистику прослушиваний альбомов для аналитики и отчетности.

**Класс PlaylistTrack**

Атрибуты: PlaylistId, TrackId, Position, AddedDate

Описание: Промежуточный класс для реализации связи многие-ко-многим между плейлистами и треками. Хранит позицию трека в плейлисте и дату добавления.



Рисунок 2. Диаграмма классов.

**Описание взаимоотношений между классами:**

**Ассоциация:**

User — Playlist (1 ко многим): Пользователь может создавать несколько плейлистов

User — Subscription (1 ко многим): Пользователь может иметь несколько подписок (в разные периоды)

Artist — Track (1 ко многим): Исполнитель может выпустить множество треков

Album — Track (1 ко многим): Альбом может содержать множество треков

Genre — Track (1 ко многим): Жанр может включать множество треков

Artist — Album (1 ко многим): Исполнитель может выпустить множество альбомов

**Агрегация:**

Playlist содержит коллекцию Track через промежуточный класс PlaylistTrack

Album содержит коллекцию Track

Artist содержит коллекцию Track

Artist содержит коллекцию Album

**Композиция:**

Track связан с Moderation (при удалении трека запись модерации удаляется)

Track связан с TrackStatistics (статистика не существует без трека)

Album связан с AlbumStatistics (статистика не существует без альбома)

**Связь многие-ко-многим:**

Playlist — Track (реализована через промежуточный класс PlaylistTrack)

**Классы ViewModel (для передачи данных):**

RegisterViewModel: Данные для регистрации пользователя

LoginViewModel: Данные для входа в систему

TrackCreateViewModel: Данные для создания трека

PlaylistCreateViewModel: Данные для создания плейлиста

PlaylistEditViewModel: Данные для редактирования плейлиста

SubscriptionViewModel: Данные для оформления подписки

## 2.3 Диаграмма состояний

Диаграмма состояний для объекта «Подписка» представлена на рисунке 3. Она отражает изменение объекта «Подписка» в системе, в ответ на действия пользователя.

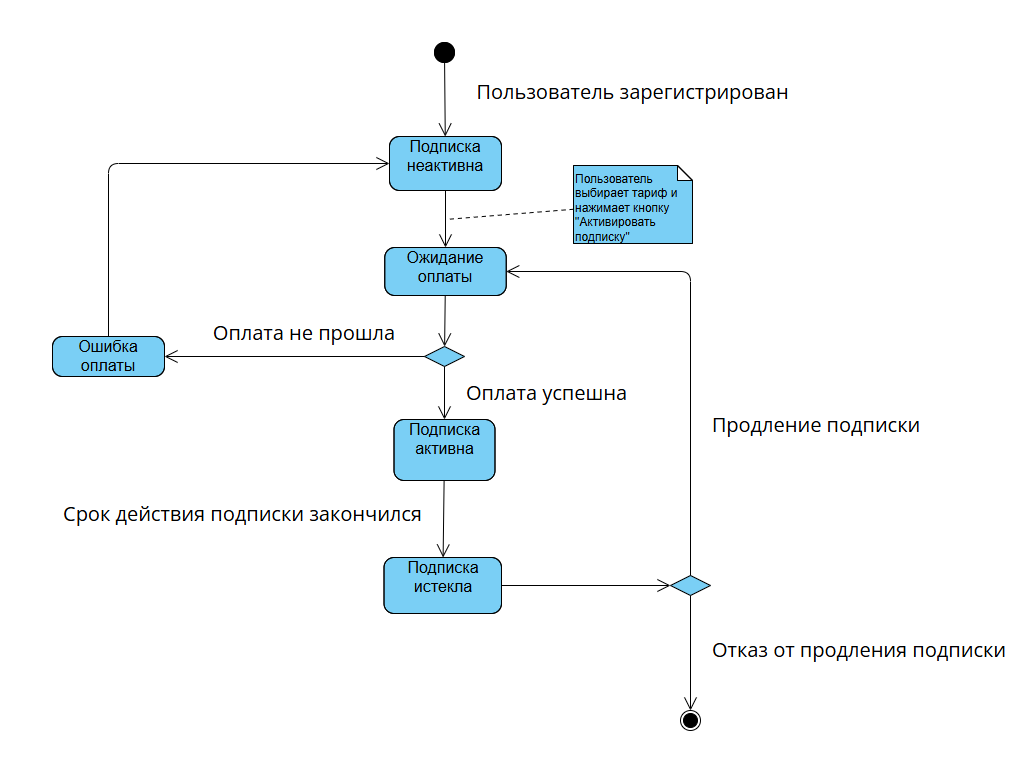


Рисунок 3. Диаграмма состояний.

**Описание состояний и переходов:**

**Начальное состояние:** Пользователь зарегистрирован в системе.

**Подписка неактивна:** У пользователя нет оплаченной подписки.

**Ожидание оплаты:** Пользователь выбрал тариф, ввел данные для оплаты и нажал «Оплатить».

Переход **Оплата не прошла**:

**Ошибка оплаты:** при оплате возникла ошибка, подписка не активировалась, пользователь может попробовать оплатить подписку вновь.

Переход **Оплата успешна**: платежная система провела успешно транзакцию по оплате.

**Подписка активна:** подписка активировалась для данного пользователя.

**Подписка истекла:** истек срок на который была активирована подписка.

Переход **Продление подписки**: по истечению срока подписки, пользователь вновь может оплатить её для активации.

Переход **Отказ от продление подписки**:пользователь не оплачивает подписку повторно.

**Конечное состояние:** подписка неактивна.

## 2.4 Диаграммы видов деятельности и BPMN

Диаграмма видов деятельности для прецедента «Добавить трек в плейлист» представлена на рисунке 4. Она демонстрирует взаимодействие пользователя с системой, для выполнения функции. Диаграмма также отображает динамическую сторону системы. На рисунках 5 и 6 представлены BPMN диаграммы для процессов «Загрузка трека» и «Оформление подписки».

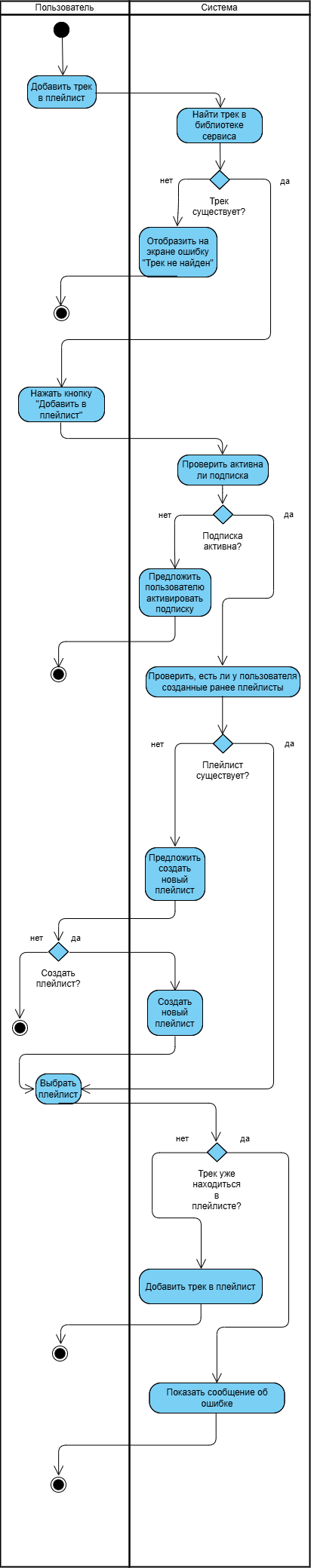


Рисунок 4. Диаграмма видов деятельности.

Рисунок 5. BPMN диаграмма процесса «Загрузка трека».

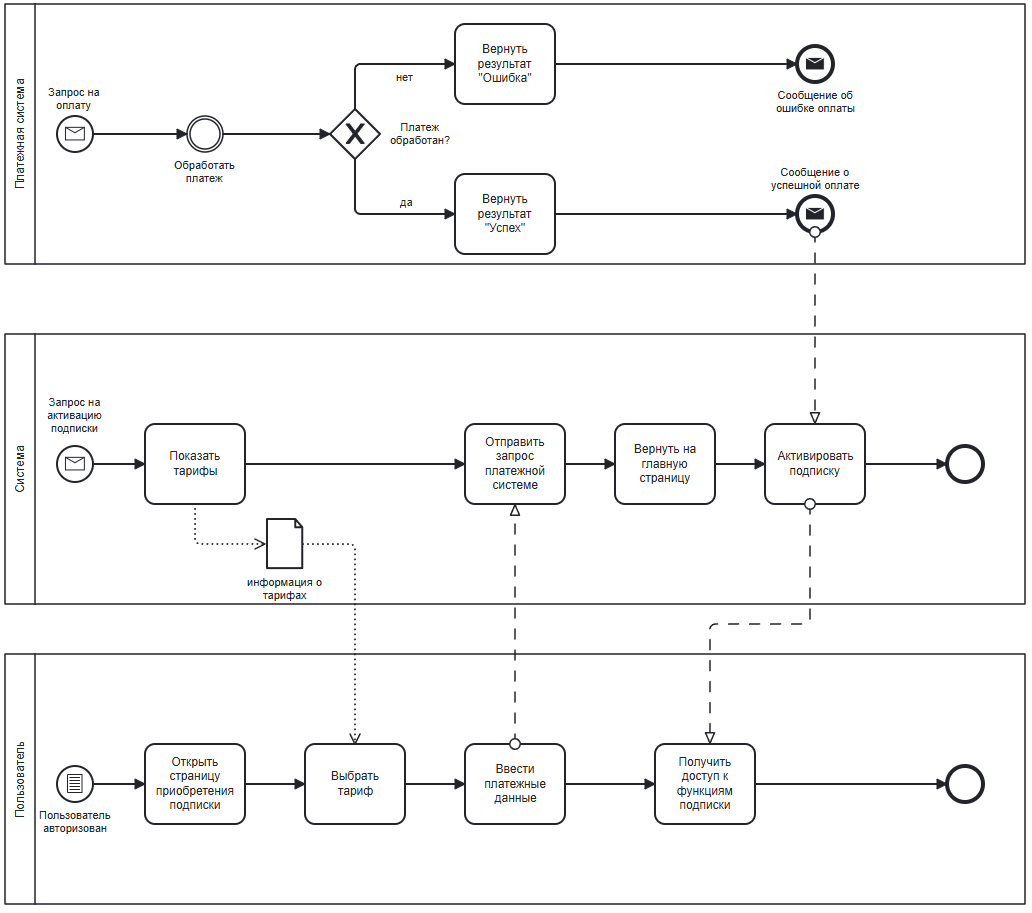
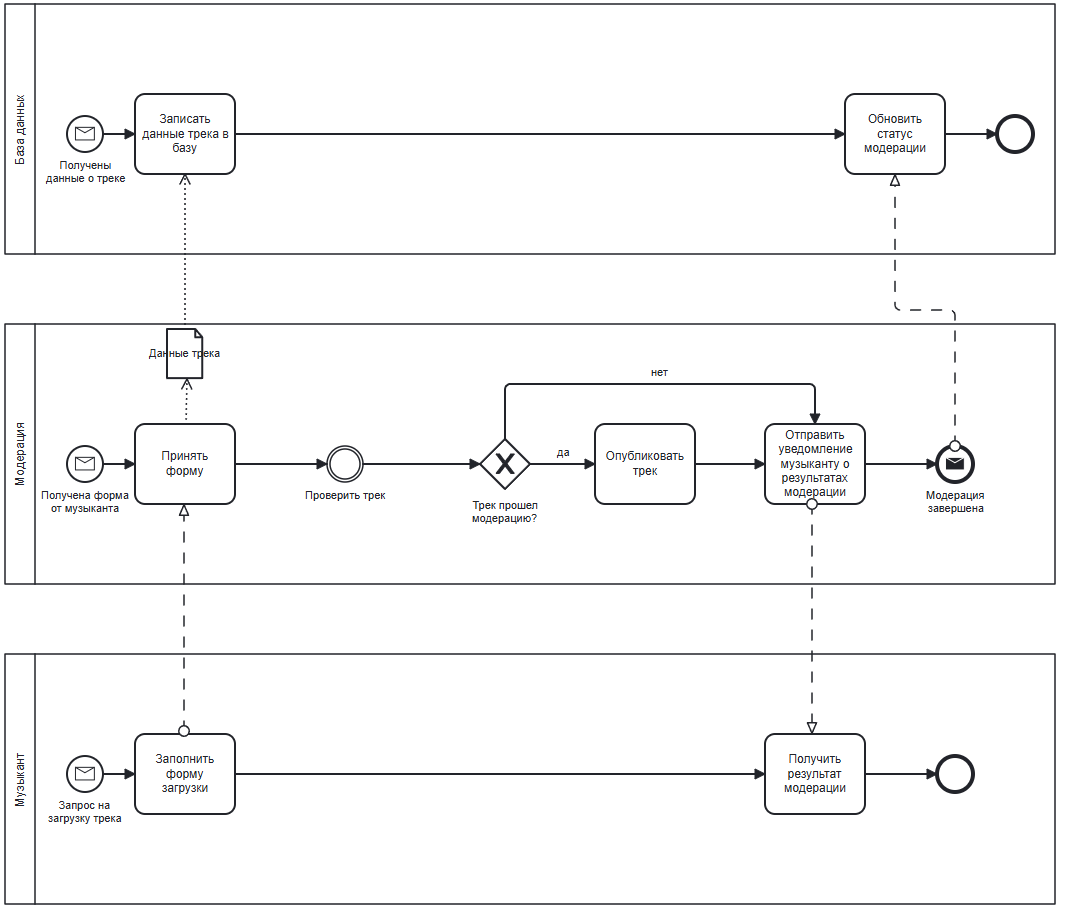


Рисунок 6. BPMN диаграмма процесса «Оформление подписки».

## 2.5 Описание зон ответственности пользователей системы

**1. Гость**

Уровень доступа: Базовый (без регистрации)

Зона ответственности: Ознакомление с базовым функционалом системы; Просмотр контента; Ограниченное прослушивание треков (с рекламой); Поиск и просмотр каталога музыки; Изучение условий использования сервиса.

Ограничения: Прослушивание с рекламой; Невозможность создания плейлистов.

**2. Пользователь**

Уровень доступа: Стандартный (зарегистрированный)

Зона ответственности: Базовые возможности сервиса; Ознакомление с тарифными планами.

**3. Подписчик**

Уровень доступа: Премиум (оплаченная подписка)

Зона ответственности: Полноценное использование сервиса; Создание и редактирование плейлистов.

**4. Музыкант**

Уровень доступа: Контент-мейкер

Зона ответственности: Пополнение музыкального каталога сервиса; Обеспечение качества загружаемого контента; Соблюдение авторских прав.

Функциональные обязанности: Загрузка треков.

**5. Администратор**

Уровень доступа: Системный

Зона ответственности: Общее управление платформой; Обеспечение стабильной работы системы; Контроль качества контента; Администрирование пользователей; Анализ работы сервиса.

Административные функции:

Управление контентом: Модерация загружаемых треков; Проверка на соответствие правилам платформы; Блокировка нарушающего контента

Управление пользователями: Назначение ролей "Музыкант"; Блокировка нарушителей; Техническая поддержка пользователей.

Аналитика и мониторинг: Просмотр системной статистики; Анализ популярности контента; Мониторинг нагрузки системы; Формирование отчетов о работе сервиса.

Системное администрирование: Настройка параметров системы; Управление платежной системой; Резервное копирование данных; Решение технических проблем.

## 2.6 Функциональные требования к системе

**1. Управление пользователями:**

* Система должна предоставлять возможность регистрации новых пользователей и авторизации существующих;
* Пользователь должен иметь возможность восстановить доступ к аккаунту при утере пароля;
* Администраторы должны иметь возможность управлять ролями пользователей;
* Пользователи должны иметь возможность просматривать свой профиль.

**2. Подписки и платежи:**

* Система должна предоставлять возможность ознакомления с тарифами и оформления подписки;
* Пользователи должны получать уведомления об истечении срока подписки.

**3. Музыкальный каталог:**

* Пользователи должны иметь возможность поиска музыкального контента;
* Пользователи должны иметь возможность слушать музыку в браузере;
* Пользователи с ролью Musician должны иметь возможность загружать свои треки;

**4. Плейлисты и библиотека:**

* Подписчики должны иметь возможность создавать и управлять плейлистами;
* Подписчики должны иметь возможность добавлять треки в свои плейлисты;

**5. Модерация:**

* Администраторы должны проверять загруженные треки перед публикацией;
* Администраторы должны принимать решения о публикации треков;
* Музыканты должны получать информацию о статусе модерации их треков;

**6. Статистика:**

* Система должна собирать аналитику о прослушиваниях;

## 2.7 Нефункциональные требования к системе

**1. Производительность:**

* Быстрая загрузка страниц;
* Плавное воспроизведение музыки;
* Быстрый поиск.

**2. Надежность:**

* Работа системы 99.9% времени;
* Регулярное резервное копирование;
* Быстрое восстановление после сбоев;

**3. Безопасность:**

* Защита паролей;
* Безопасные соединения (HTTPS);

**4. Масштабируемость:**

* Возможность роста системы;
* Поддержка увеличения пользователей;

**5. Совместимость:**

* Работа в современных браузерах;
* Поддержка современных форматов;

**6. Удобство использования:**

* Простой и понятный интерфейс;
* Легкая навигация.

**7. Технические требования:**

* Хранение большого каталога музыки;
* Поддержка больших файлов;

# 3. StATE OF ART

## 3.1 Обзор актуальных технологий

**Основная платформа разработки:**

Фреймворк Spring Boot. Данный выбор обусловлен богатой экосистемой библиотек для работы с мультимедийным контентом, высокой производительностью и надежностью платформы Java для создания отказоустойчивых систем.

**Специализированные технологии для работы с аудиоконтентом:**

* JAudiotagger - специализированная библиотека для извлечения и редактирования метаданных аудиофайлов.;
* HTTP Range Requests - реализация поддержки байтовых диапазонов для эффективной потоковой передачи;
* Apache Commons Math - базовые статистические алгоритмы;
* Buffered Streaming - механизм буферизации для обеспечения бесперебойного воспроизведения.

## 3.2 Анализ аналогов

**Перечень критериев оценки**

**Группа 1: Функциональность (вес 40%)**

1.1. Управление музыкальным каталогом (18%)

1.2. Работа с плейлистами (15%)

1.3. Социальные функции (7%)

**Группа 2: Технические характеристики (вес 30%)**

2.1. Качество стриминга (18%)

2.2. Производительность поиска (7%)

2.3. Надежность работы (5%)

**Группа 3: Пользовательский опыт (вес 20%)**

3.1. Удобство интерфейса (8%)

3.2. Персонализация (7%)

3.3. Кроссплатформенность (5%)

**Группа 4: Бизнес-модель (вес 10%)**

4.1. Гибкость монетизации (6%)

4.2. Возможности для артистов (4%)

Сравнение аналогов представлено в таблице 1.

Таблица 1. Сравнительная таблица аналогов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Критерий** | **Вес** | **Spotify** | **Яндекс.Музыка** | **SoundCloud** |
| **1. Функциональность** | 40% |  |  |  |
| 1.1 Управление каталогом | 18% | 5 | 4 | 3 |
| 1.2 Работа с плейлистами | 15% | 5 | 4 | 3 |
| 1.3 Социальные функции | 7% | 4 | 3 | 5 |
| **2. Технические характеристики** | 30% |  |  |  |
| 2.1 Качество стриминга | 10% | 5 | 4 | 3 |
| 2.2 Производительность поиска | 7% | 5 | 5 | 3 |
| 2.3 Надежность работы | 5% | 5 | 4 | 3 |
| **3. Пользовательский опыт** | 20% |  |  |  |
| 3.1 Удобство интерфейса | 8% | 5 | 4 | 3 |
| 3.2 Персонализация | 7% | 5 | 4 | 2 |
| 3.3 Кроссплатформенность | 5% | 5 | 4 | 4 |
| **4. Бизнес-модель** | 10% |  |  |  |
| 4.1 Гибкость монетизации | 6% | 5 | 4 | 5 |
| 4.2 Возможности для артистов | 4% | 4 | 3 | 5 |
| **Итоговый балл** | 100% | **4.7** | **4.1** | **3.4** |

**Ключевые конкурентные преимущества аналогов:**

**Spotify:** передовые алгоритмы рекомендаций, глобальный каталог, кроссплатформенность.

**Яндекс.Музыка:** интеграция с экосистемой Яндекс, качественные рекомендации на основе AI.

**SoundCloud:** социальная ориентированность, поддержка независимых исполнителей.

# 4. Модель данных

## 4.1 Схема базы данных

Физическая ER диаграмма базы данных представлена на рисунке 5. Она демонстрирует отношения между таблицами базы данных.

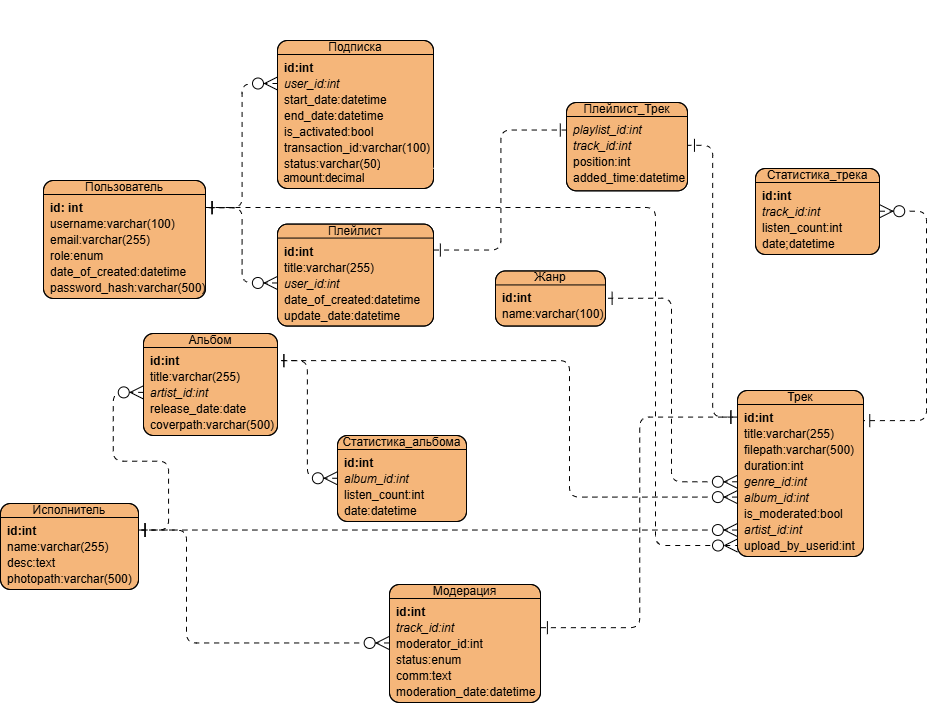


Рисунок 7. Физическая ER диаграмма базы данных.

Описание реляционных таблиц:

Пользователь

* id (INT, PK, AUTO\_INCREMENT)
* username (VARCHAR(100), UNIQUE, NOT NULL)
* email (VARCHAR(255), UNIQUE, NOT NULL)
* password\_hash (VARCHAR(255), NOT NULL)
* role (ENUM('user', 'subscriber', 'musician', 'admin'))
* date\_of\_created (DATETIME)

Подписка

* id (INT, PK, AUTO\_INCREMENT)
* user\_id (INT, FK → users.id)
* start\_date (DATETIME)
* end\_date (DATETIME)
* is\_activated (BOOL)
* transaction\_id (VARCHAR(100))
* amount (DECIMAL)
* status (VARCHAR(50) Default ‘Pending’)

Исполнитель

* id (INT, PK, AUTO\_INCREMENT)
* name (VARCHAR(255), NOT NULL)
* desc (TEXT)
* photopath (VARCHAR(500))

Жанр

* id (INT, PK, AUTO\_INCREMENT)
* name (VARCHAR(100), UNIQUE, NOT NULL)

Альбом

* id (INT, PK, AUTO\_INCREMENT)
* title (VARCHAR(255), NOT NULL)
* artist\_id (INT, FK → artists.id)
* release\_date (DATE)
* coverpath (VARCHAR(500))

Трек

* id (INT, PK, AUTO\_INCREMENT)
* title (VARCHAR(255), NOT NULL)
* file\_path (VARCHAR(500), NOT NULL)
* duration (INT)
* genre\_id (INT, FK → genres.id)
* album\_id (INT, FK → albums.id, NULL)
* is\_moderated (BOOL)
* artist\_id (INT, FK → artists.id, NULL)
* upload\_by\_userid (INT)

Плейлист

* id (INT, PK, AUTO\_INCREMENT)
* title (VARCHAR(255), NOT NULL)
* user\_id (INT, FK → users.id)
* date\_of\_created (DATETIME)

Плейлист\_Трек

* playlist\_id (INT, FK → playlists.id)
* track\_id (INT, FK → tracks.id)
* position (INT Default (0))
* added\_date (DATETIME Default CURRENT TIMESTAMP)
* PRIMARY KEY (playlist\_id, track\_id)

Модерация

* id (INT, PK, AUTO\_INCREMENT)
* track\_id (INT, FK → tracks.id)
* moderator\_id (INT, FK → users.id)
* status (ENUM('approved', 'rejected', 'pending'))
* comm (TEXT)
* moderation\_date (DATETIME)

Статистика\_альбома

* id (INT, PK, AUTO\_INCREMENT)
* album\_id (INT, FK → albums.id)
* date (DATETIME)
* listen\_count (INT)

Статистика\_трека

* id (INT, PK, AUTO\_INCREMENT)
* track\_id (INT, FK → tracks.id)
* date (DATETIME)
* listen\_count (INT)

## 4.2 Механизмы обеспечения целостности данных

**Первичные ключи (PRIMARY KEY)**

Все таблицы имеют первичный ключ id с автоинкрементом для уникальной идентификации записей.

**Внешние ключи (FOREIGN KEY)**

Связи между таблицами обеспечены внешними ключами:

* subscriptions.user\_id → users.id
* albums.artist\_id → artists.id
* tracks.genre\_id → genres.id
* tracks.album\_id → albums.id
* tracks.artist\_id → artists.id
* playlists.user\_id → users.id
* playlist\_track.playlist\_id → playlists.id
* playlist\_track.track\_id → tracks.id
* moderations.track\_id → tracks.id
* moderations.moderator\_id → users.id
* album\_statistics.album\_id → albums.id
* track\_statistics.track\_id → tracks.id
* upload\_by\_userid → users.id

**Ограничения уникальности (UNIQUE)**

* users.username - уникальное имя пользователя
* users.email - уникальный email
* genres.name - уникальное название жанра
* Составной первичный ключ в playlist\_track

**Проверка значений**

* users.role - ENUM с фиксированными значениями ролей
* moderations.status - ENUM с статусами модерации
* is\_activated, is\_moderated - BOOLEAN значения

**NOT NULL ограничения**

Обязательные поля защищены от NULL значений: username, email, password\_hash, role в users; name в artists; name в genres; title в albums и tracks и др.

**Каскадные операции**

* Удаление пользователя → удаление его плейлистов и подписок
* Удаление трека → удаление из плейлистов и статистики

## 4.3 SQL-скрипт базы данных

На рисунке 6, представлен SQL-скрипт создания базы данных в СУБД MySQl.

CREATE DATABASE IF NOT EXISTS MusicStreamingService;

USE MusicStreamingService;

CREATE TABLE IF NOT EXISTS Users (

Id INT PRIMARY KEY AUTO\_INCREMENT,

Username VARCHAR(100) NOT NULL UNIQUE,

Email VARCHAR(255) NOT NULL UNIQUE,

PasswordHash VARCHAR(255) NOT NULL,

Role ENUM('User', 'Subscriber', 'Musician', 'Admin') DEFAULT 'User',

DateOfCreated DATETIME DEFAULT CURRENT\_TIMESTAMP,

INDEX idx\_username (Username),

INDEX idx\_email (Email),

INDEX idx\_role (Role)

);

CREATE TABLE IF NOT EXISTS Artists (

Id INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(255) NOT NULL,

Description TEXT,

PhotoPath VARCHAR(500),

INDEX idx\_name (Name)

);

CREATE TABLE IF NOT EXISTS Genres (

Id INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100) NOT NULL UNIQUE,

INDEX idx\_name (Name)

);

CREATE TABLE IF NOT EXISTS Albums (

Id INT PRIMARY KEY AUTO\_INCREMENT,

Title VARCHAR(255) NOT NULL,

ArtistId INT NOT NULL,

ReleaseDate DATE,

CoverPath VARCHAR(500),

FOREIGN KEY (ArtistId) REFERENCES Artists(Id) ON DELETE CASCADE,

INDEX idx\_title (Title),

INDEX idx\_artist (ArtistId),

INDEX idx\_release\_date (ReleaseDate)

);

CREATE TABLE IF NOT EXISTS Tracks (

Id INT PRIMARY KEY AUTO\_INCREMENT,

Title VARCHAR(255) NOT NULL,

FilePath VARCHAR(500) NOT NULL,

Duration INT NOT NULL,

GenreId INT NOT NULL,

AlbumId INT,

ArtistId INT,

IsModerated BOOLEAN DEFAULT FALSE,

UploadedByUserId INT,

FOREIGN KEY (GenreId) REFERENCES Genres(Id),

FOREIGN KEY (AlbumId) REFERENCES Albums(Id) ON DELETE SET NULL,

FOREIGN KEY (ArtistId) REFERENCES Artists(Id) ON DELETE SET NULL,

FOREIGN KEY (UploadedByUserId) REFERENCES Users(Id) ON DELETE SET NULL,

INDEX idx\_title (Title),

INDEX idx\_genre (GenreId),

INDEX idx\_album (AlbumId),

INDEX idx\_artist (ArtistId),

INDEX idx\_moderated (IsModerated),

INDEX idx\_uploaded\_by (UploadedByUserId)

);

CREATE TABLE IF NOT EXISTS Playlists (

Id INT PRIMARY KEY AUTO\_INCREMENT,

Title VARCHAR(255) NOT NULL,

Description VARCHAR(1000),

UserId INT NOT NULL,

Visibility ENUM('Private', 'Public', 'Unlisted') DEFAULT 'Private',

CreatedDate DATETIME DEFAULT CURRENT\_TIMESTAMP,

UpdatedDate DATETIME,

CoverImagePath VARCHAR(255),

FOREIGN KEY (UserId) REFERENCES Users(Id) ON DELETE CASCADE,

INDEX idx\_user (UserId),

INDEX idx\_visibility (Visibility),

INDEX idx\_created\_date (CreatedDate)

);

CREATE TABLE IF NOT EXISTS PlaylistTracks (

PlaylistId INT NOT NULL,

TrackId INT NOT NULL,

Position INT DEFAULT 0,

AddedDate DATETIME DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (PlaylistId, TrackId),

FOREIGN KEY (PlaylistId) REFERENCES Playlists(Id) ON DELETE CASCADE,

FOREIGN KEY (TrackId) REFERENCES Tracks(Id) ON DELETE CASCADE,

INDEX idx\_position (Position),

INDEX idx\_added\_date (AddedDate)

);

CREATE TABLE IF NOT EXISTS Subscriptions (

Id INT PRIMARY KEY AUTO\_INCREMENT,

UserId INT NOT NULL,

StartDate DATETIME NOT NULL,

EndDate DATETIME NOT NULL,

IsActivated BOOLEAN DEFAULT FALSE,

TransactionId VARCHAR(100),

Amount DECIMAL(10,2) DEFAULT 399.00,

Status VARCHAR(50) DEFAULT 'pending',

FOREIGN KEY (UserId) REFERENCES Users(Id) ON DELETE CASCADE,

INDEX idx\_user (UserId),

INDEX idx\_end\_date (EndDate),

INDEX idx\_is\_activated (IsActivated),

INDEX idx\_status (Status)

);

CREATE TABLE IF NOT EXISTS Moderations (

Id INT PRIMARY KEY AUTO\_INCREMENT,

TrackId INT NOT NULL,

ModeratorId INT NOT NULL,

Status ENUM('Approved', 'Rejected', 'Pending') DEFAULT 'Pending',

Comment TEXT,

ModerationDate DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (TrackId) REFERENCES Tracks(Id) ON DELETE CASCADE,

FOREIGN KEY (ModeratorId) REFERENCES Users(Id),

INDEX idx\_track (TrackId),

INDEX idx\_moderator (ModeratorId),

INDEX idx\_status (Status),

INDEX idx\_moderation\_date (ModerationDate)

);

CREATE TABLE IF NOT EXISTS TrackStatistics (

Id INT PRIMARY KEY AUTO\_INCREMENT,

TrackId INT NOT NULL,

Date DATETIME DEFAULT CURRENT\_TIMESTAMP,

ListenCount INT DEFAULT 1,

FOREIGN KEY (TrackId) REFERENCES Tracks(Id) ON DELETE CASCADE,

INDEX idx\_track (TrackId),

INDEX idx\_date (Date),

INDEX idx\_track\_date (TrackId, Date)

);

CREATE TABLE IF NOT EXISTS AlbumStatistics (

Id INT PRIMARY KEY AUTO\_INCREMENT,

AlbumId INT NOT NULL,

Date DATETIME DEFAULT CURRENT\_TIMESTAMP,

ListenCount INT DEFAULT 1,

FOREIGN KEY (AlbumId) REFERENCES Albums(Id) ON DELETE CASCADE,

INDEX idx\_album (AlbumId),

INDEX idx\_date (Date),

INDEX idx\_album\_date (AlbumId, Date)

);

Рисунок 8. SQL-скрипт создания базы данных.

# 5. Макет Программной системы

## 5.1 Описание принципов организации интерфейса

Интерфейс системы спроектирован с учетом следующих ключевых принципов, основанных на анализе требований и пользователей:

Ролевая адаптивность: Интерфейс динамически изменяется в зависимости от роли пользователя (Гость, Пользователь, Подписчик, Музыкант, Администратор). Отображаются только те функции и разделы, которые доступны для текущей роли.

Максимальная простота и интуитивность: Учитывая широкий охват аудитории с разным уровнем технической грамотности (от начинающих до продвинутых), навигация и основные действия (поиск, воспроизведение, создание плейлиста) должны быть очевидными и выполняться минимальным количеством шагов.

Контент-центричность: Основное пространство экрана отводится под музыкальный контент (обложки альбомов, списки треков, плейлисты). Управляющие элементы (кнопки воспроизведения, добавления в плейлист и тд.) визуально вторичны, но легко доступны.

Единообразие и консистентность: Одинаковые действия (например, добавление трека в плейлист, управление воспроизведением) выполняются одинаково во всех разделах приложения (в поиске, в альбоме, в плейлисте).

Ниже приведен список основных экранных форм (страниц/экранов) системы, сгруппированных по функциональным блокам и ролям.

A. Общие формы (доступны всем ролям, включая Гостя)

* Главная страница (Лента): Новые релизы. Для гостей — общая популярная музыка и призыв к регистрации.
* Страница поиска: Поле ввода и результаты поиска по трекам, альбомам, исполнителям.
* Страница "Сейчас играет" (Плеер): Полноэкранный или всплывающий плеер с управлением (play/pause, next/prev, громкость, прогресс-бар), названием трека, исполнителем, обложкой, кнопкой "Добавить в плейлист".
* Страница альбома: Обложка, название, исполнитель, список треков, кнопка воспроизведения всего альбома.
* Страница исполнителя: Фото/аватар, имя, описание, список популярных треков и альбомов.
* Страница плейлиста (общего/тематического): Обложка, название, описание, список треков.
* Модальное окно "Вход / Регистрация": Формы для авторизации и регистрации нового аккаунта.

B. Формы для авторизованных пользователей (Пользователь, Подписчик)

* Личный кабинет (Мой профиль): Информация о пользователе, аватар, переход к "Моей музыке", настройкам подписки.
* Моя музыка (Библиотека): Вкладки/подразделы: "Плейлисты", "Альбомы", "Исполнители", "Понравившиеся треки", "История прослушиваний".
* Создание / Редактирование плейлиста: Форма для ввода названия, описания, выбора обложки и добавления треков.
* Настройки подписки: Страница с доступными тарифами, статусом текущей подписки.
* Оплата подписки: редирект на платежный шлюз.

C. Формы для Музыканта (наследует все формы Пользователя)

* Кабинет музыканта: Управление профилем исполнителя (имя, описание, аватар).
* Загрузка нового трека: Форма с полями: аудиофайл, название, выбор альбома (или создание нового), выбор жанра, обложка.
* Мои загрузки: Список загруженных треков с указанием статуса модерации ("На проверке", "Одобрено", "Отклонено").

D. Формы для Администратора

* Административная панель: Общая статистика сервиса (кол-во прослушиваний, новые пользователи).
* Панель модерации контента: Список треков со статусом "pending". Возможность прослушать, просмотреть метаданные и нажать кнопки "Одобрить" или "Отклонить" с указанием причины.
* Управление пользователями: Список всех пользователей с возможностью фильтрации по роли, поиска, блокировки, назначения роли "Музыкант".
* Управление каталогом: Просмотр и редактирование информации об альбомах, исполнителях, жанрах (CRUD-интерфейс).

## 5.2 Диаграмма навигации

Диаграмма изображена на рисунке 7 и иллюстрирует основные пути перемещения между экранными формами. Для наглядности она сфокусирована на ключевых сценариях пользователя Подписчика. Элементы, специфичные для других ролей (например, "Кабинет музыканта" или "Панель модерации"), показаны как отдельные ветви.

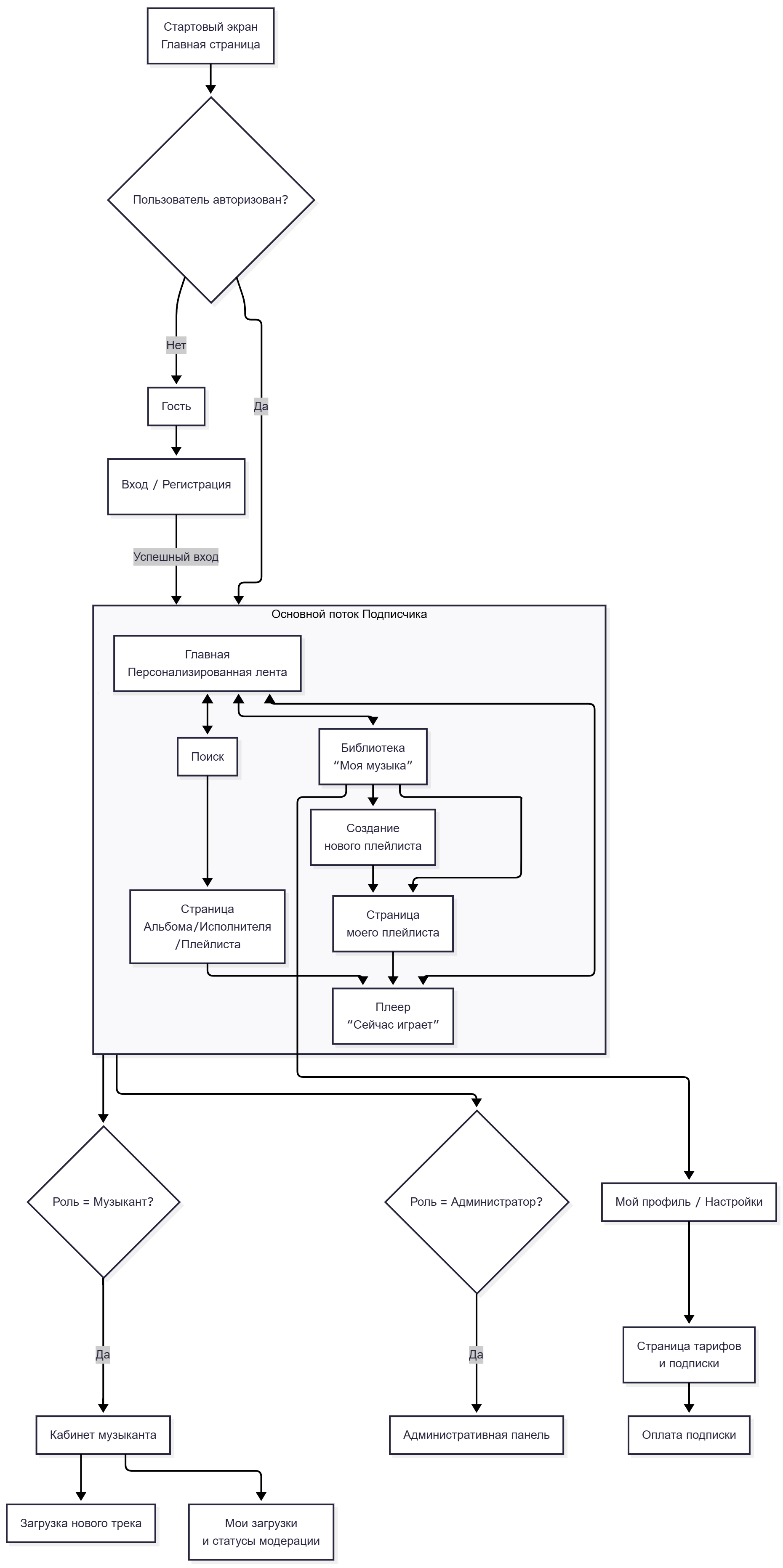


Рисунок 9. Диаграмма навигации.

## 5.3 Макеты страниц

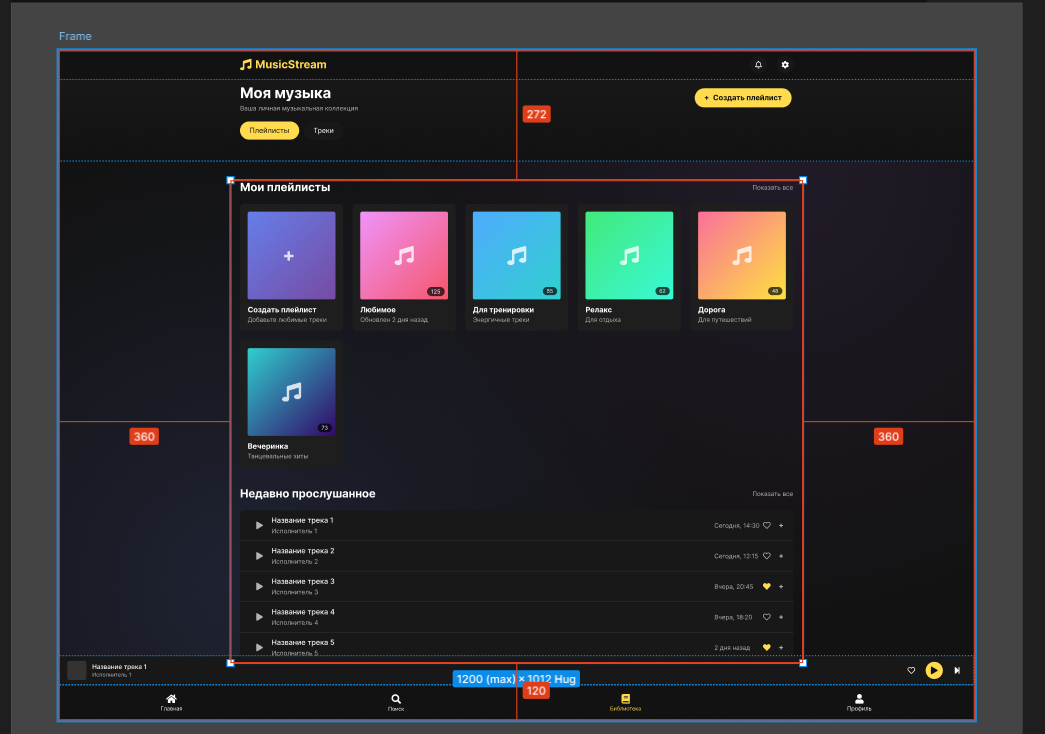


Рисунок 10. Макет страницы «Моя музыка».

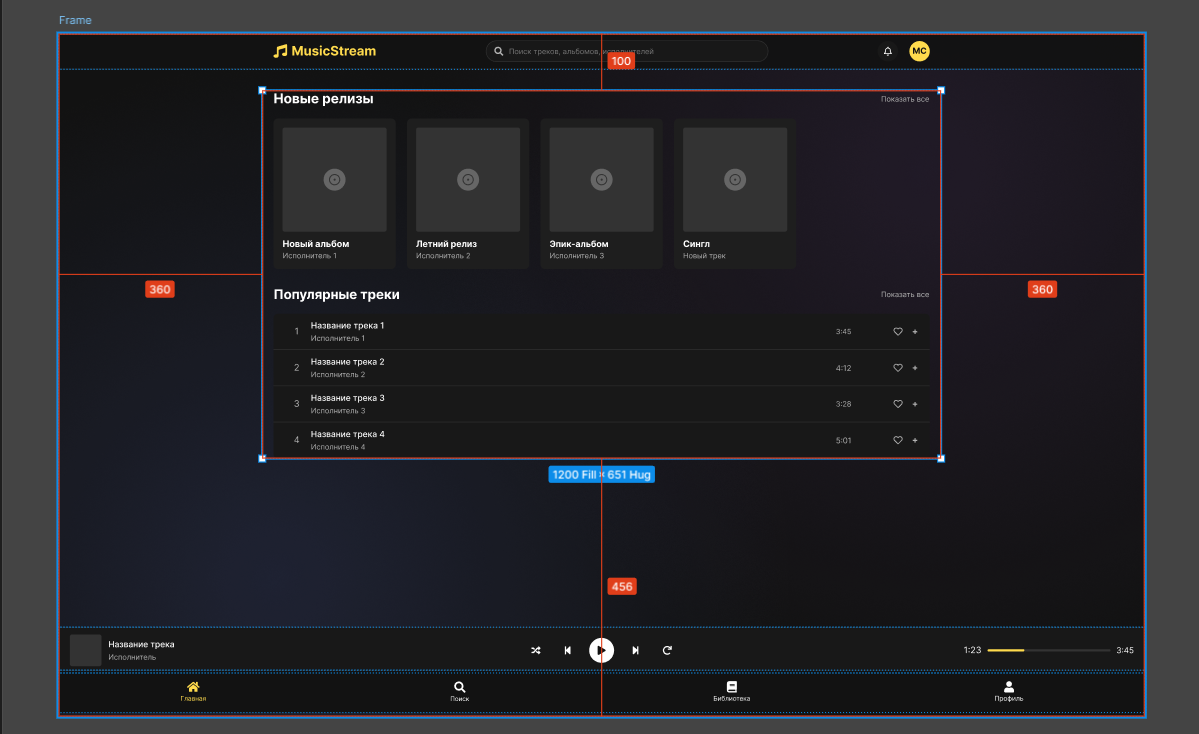


Рисунок 11. Макет главной страницы.

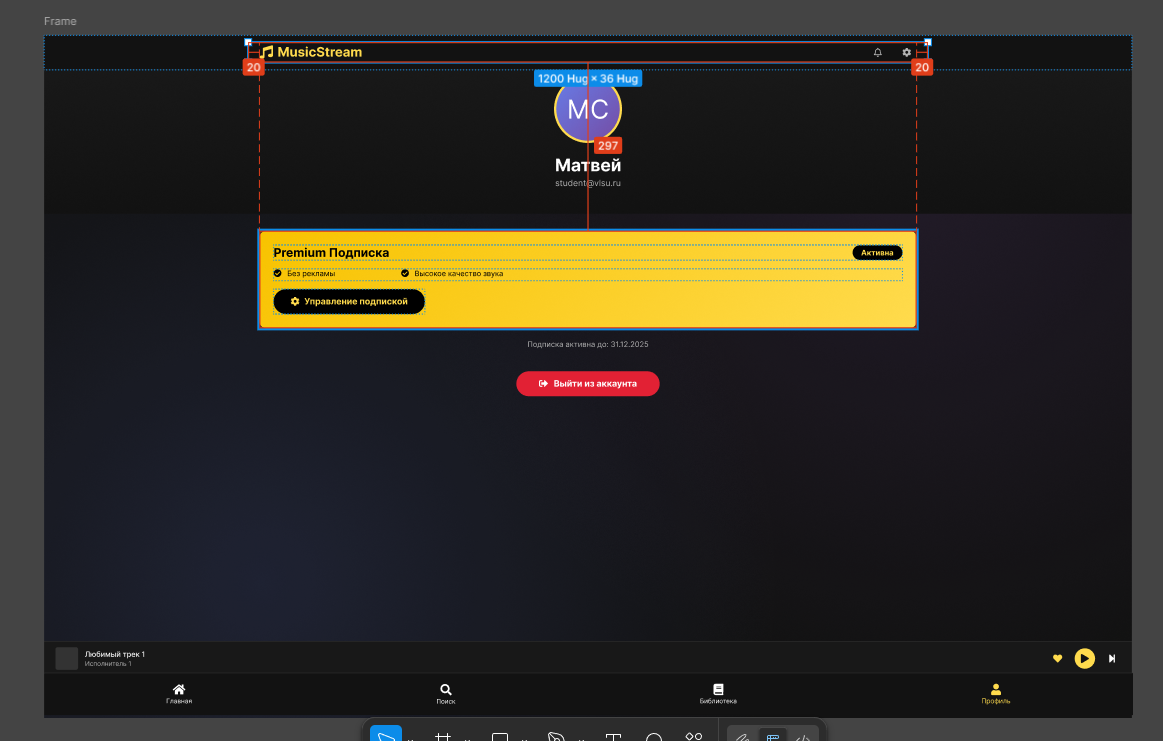


Рисунок 12. Макет страницы профиля.

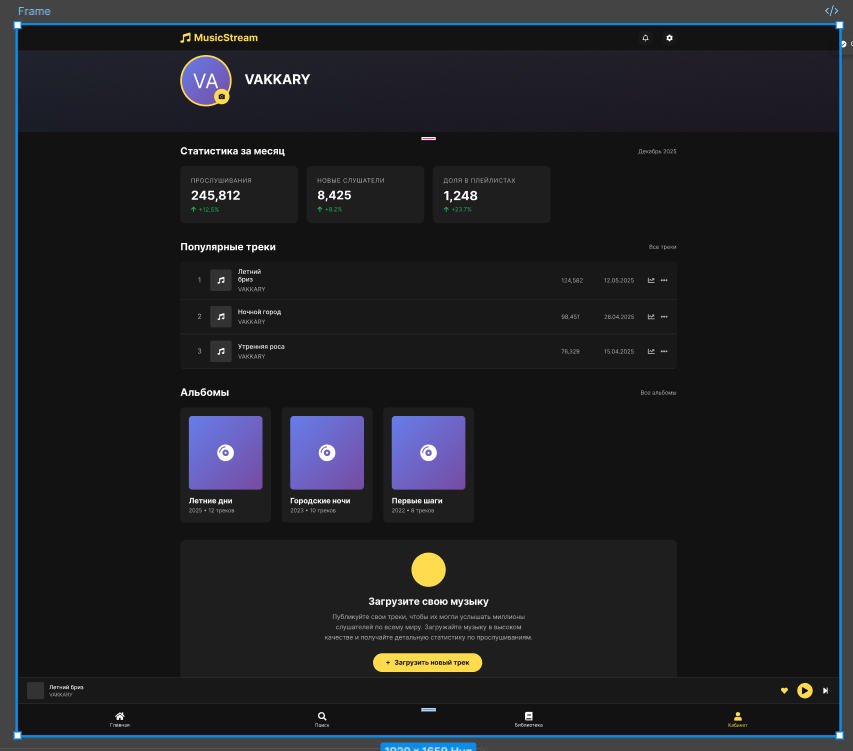


Рисунок 13. Макет страницы профиля музыканта.



Рисунок 14. Макет страницы входа.

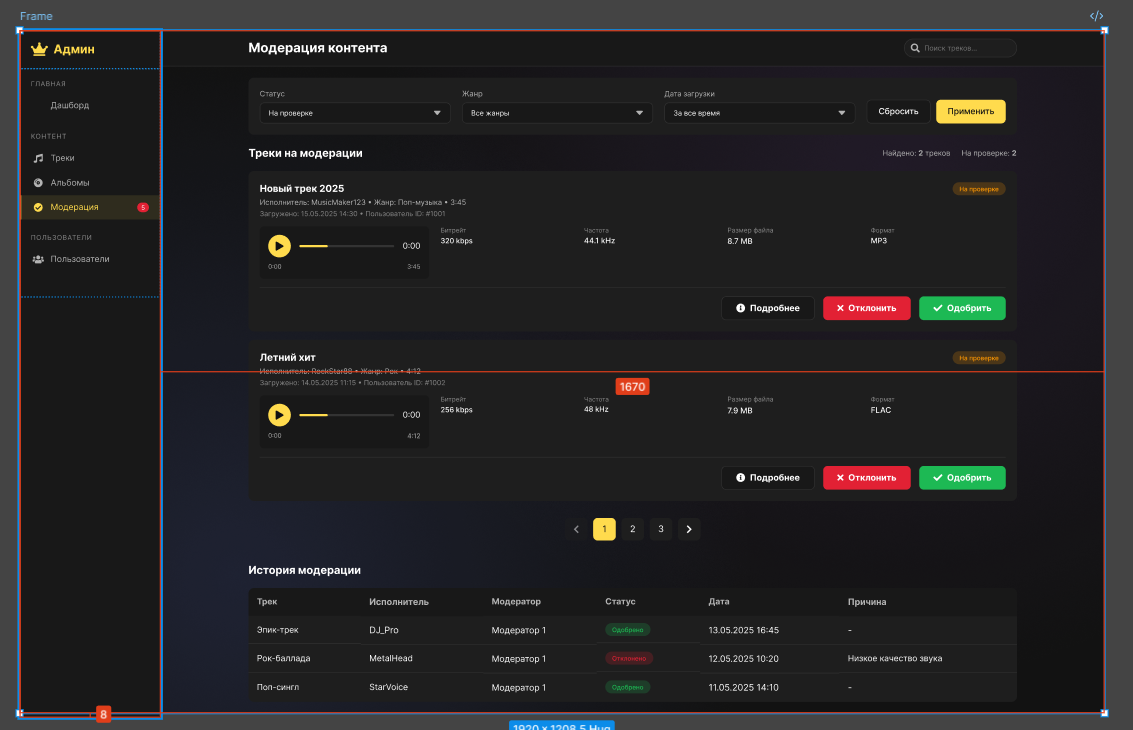


Рисунок 15. Макет страницы модерации.

## 5.4 Верстка страниц

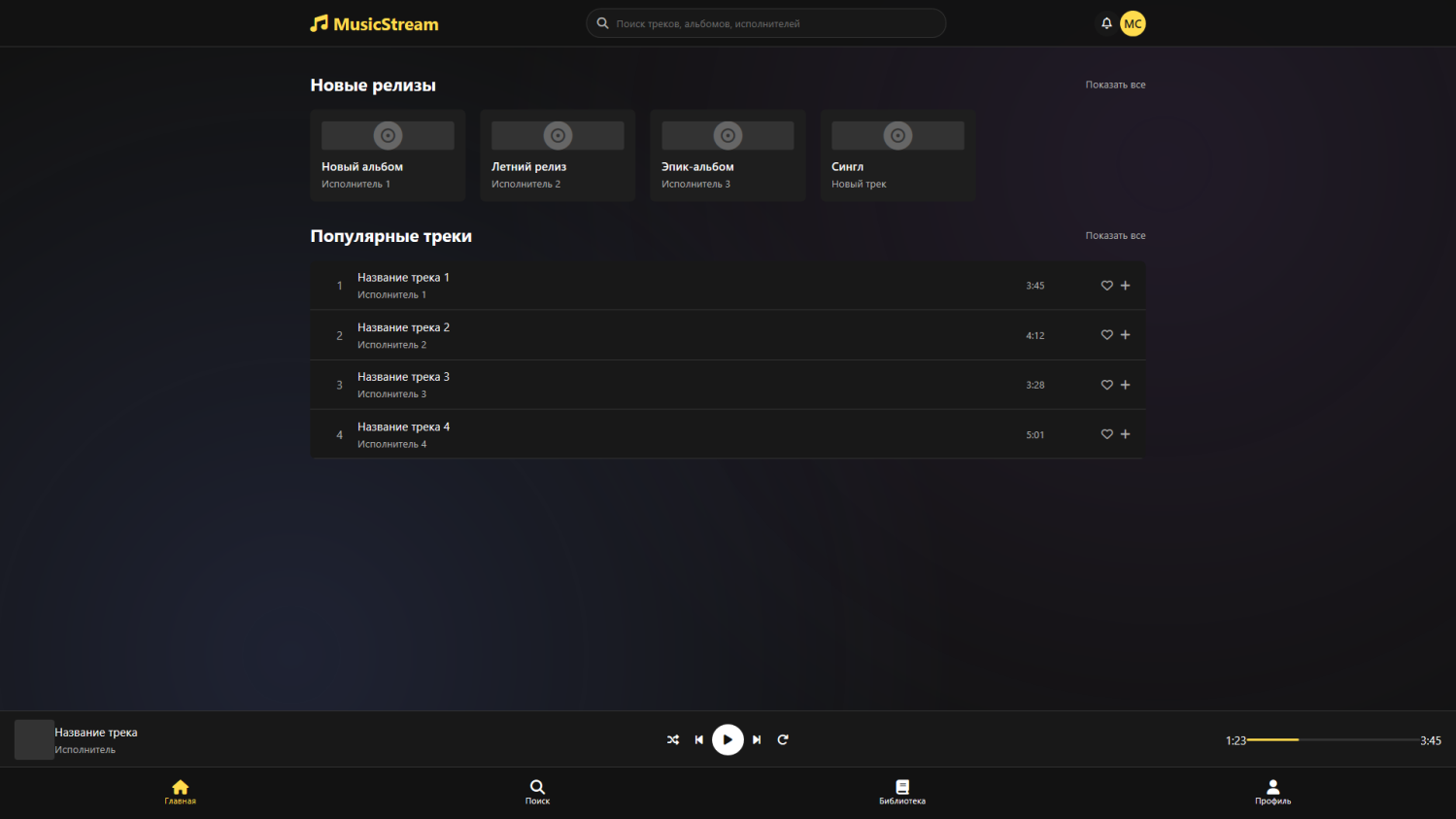


Рисунок 16. Главная страница.

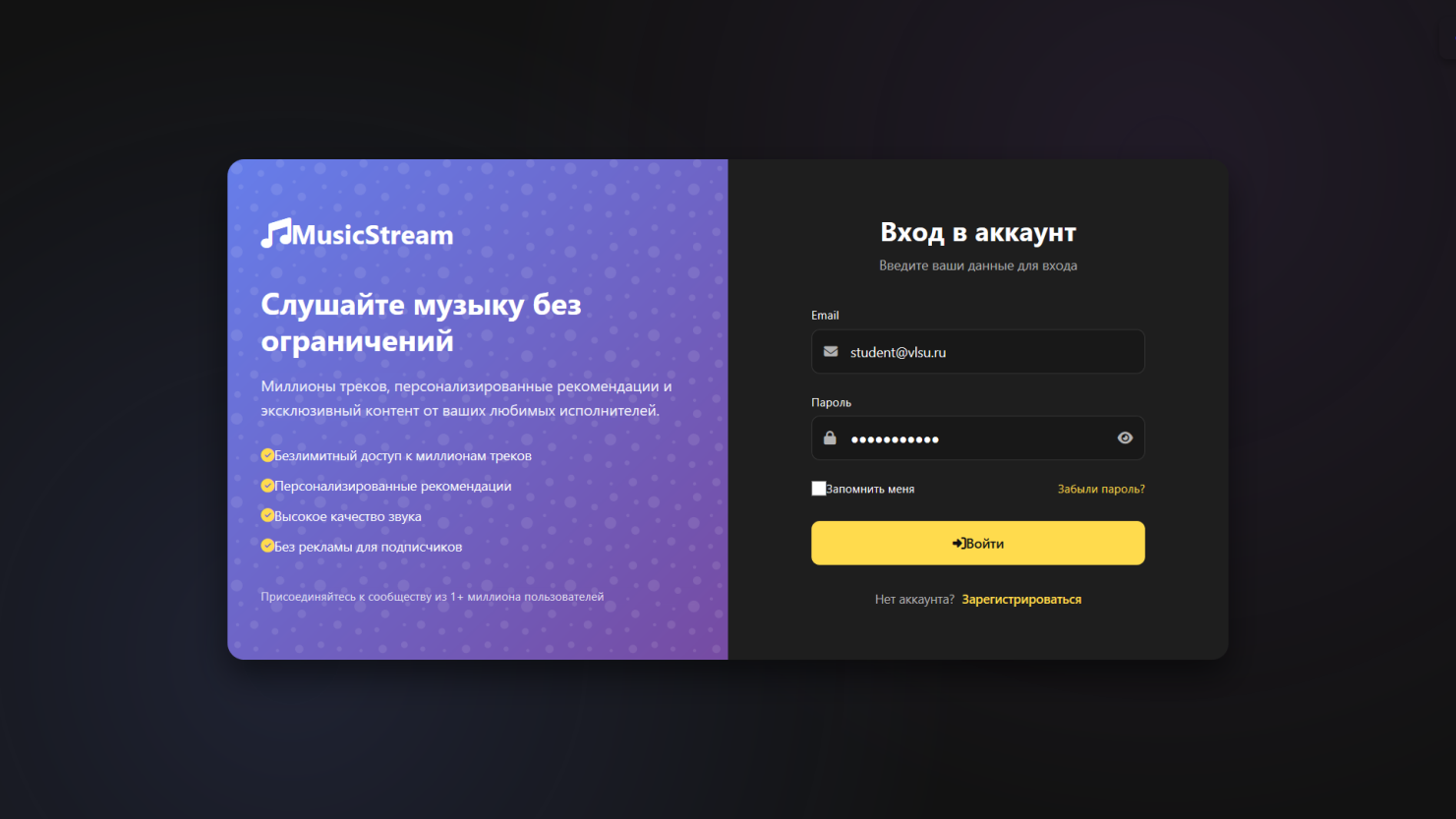


Рисунок 17. Страница входа.

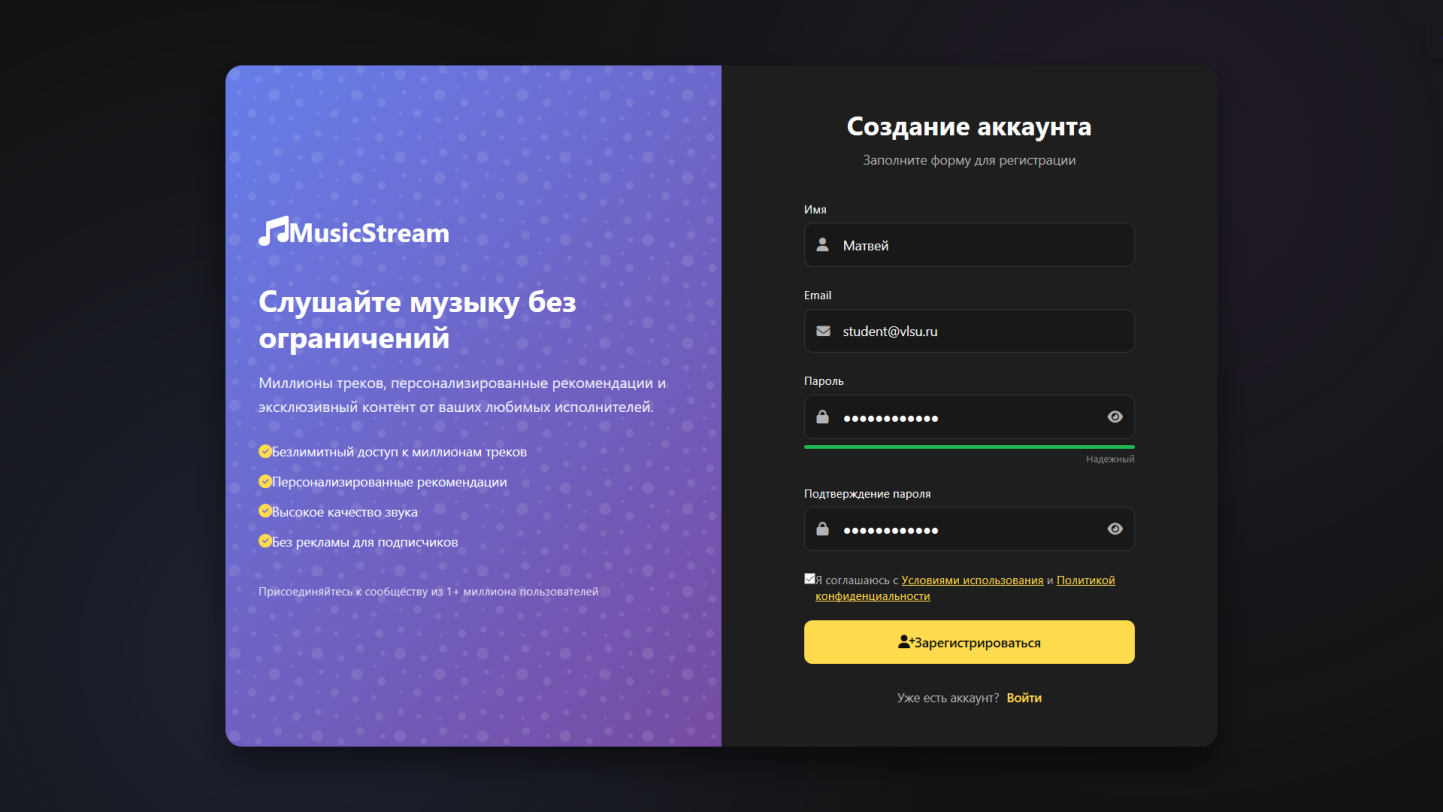


Рисунок 18. Страница регистрации.

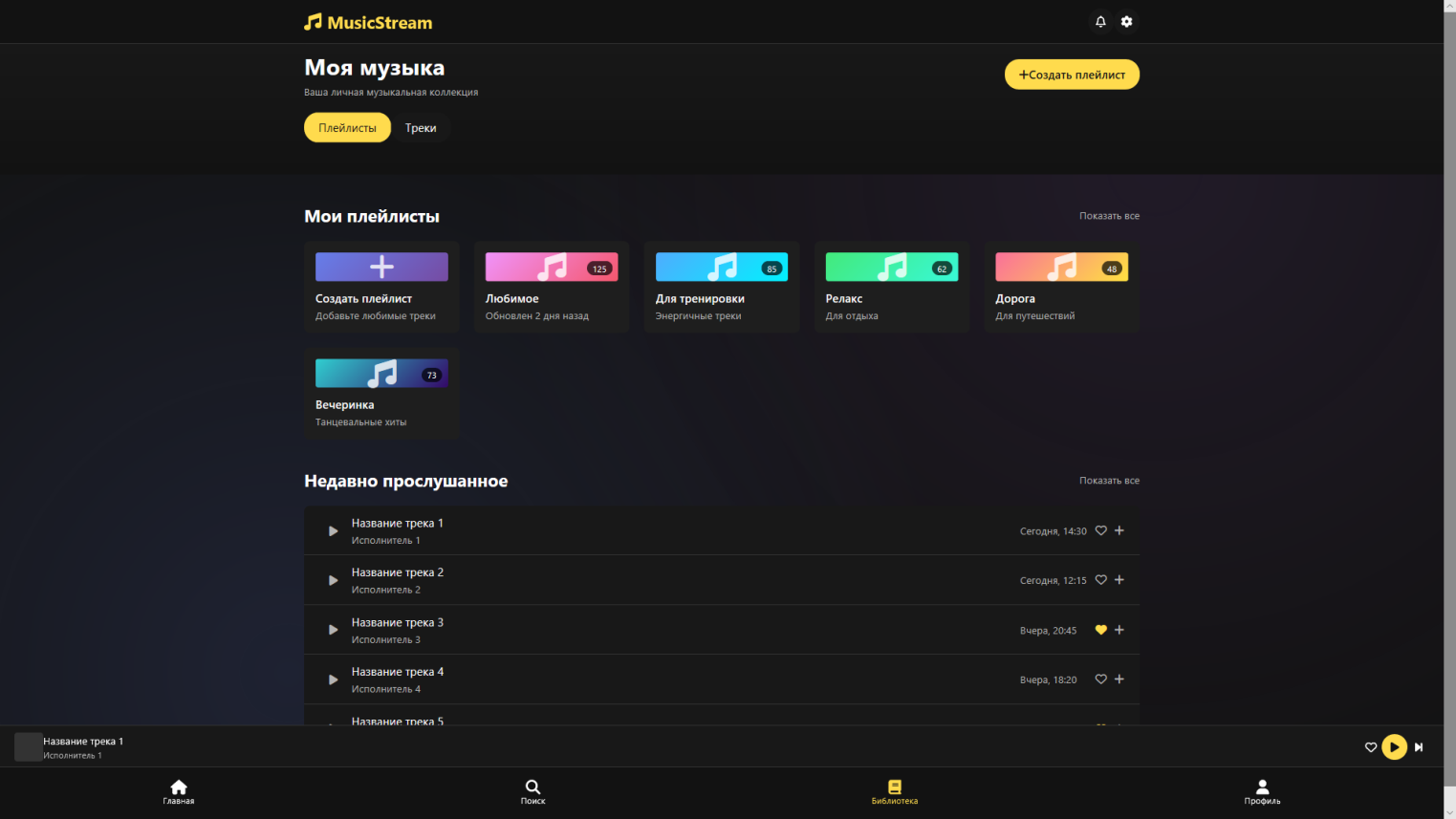


Рисунок 19. Страница «Моя музыка».

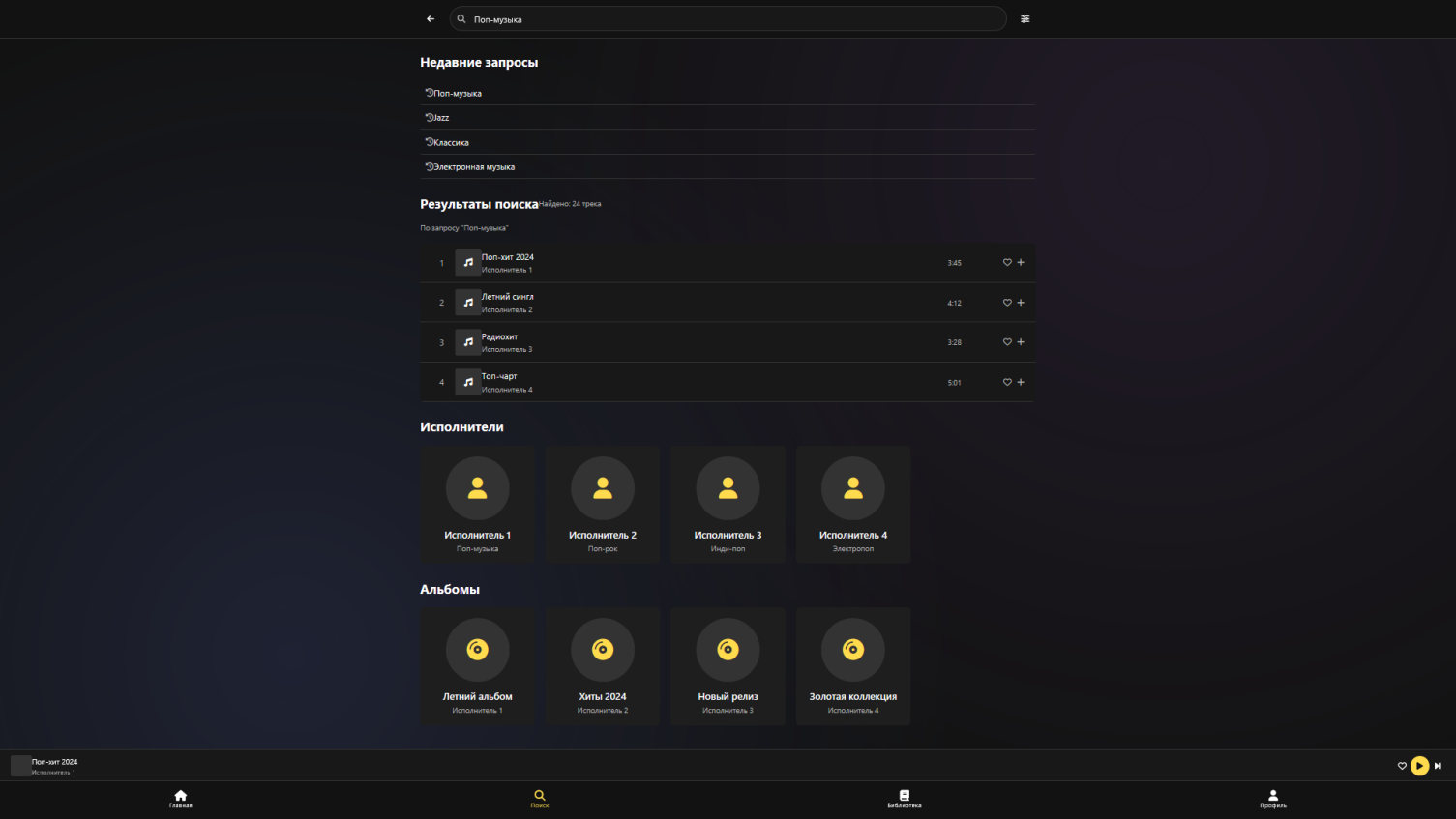


Рисунок 20. Страница поиска.



Рисунок 21. Страница альбома.

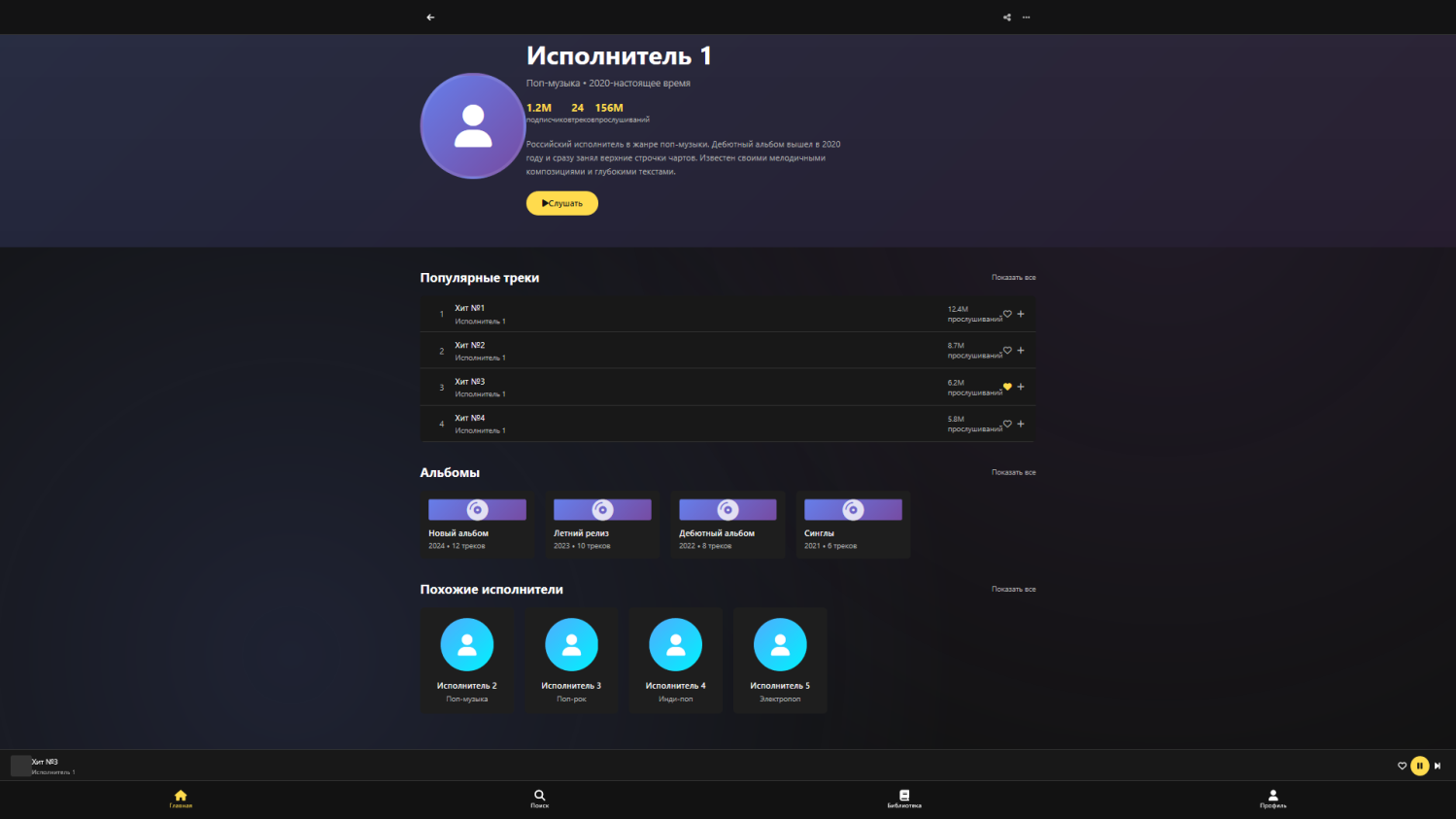


Рисунок 22. Страница исполнителя.



Рисунок 23. Страница плейлиста.

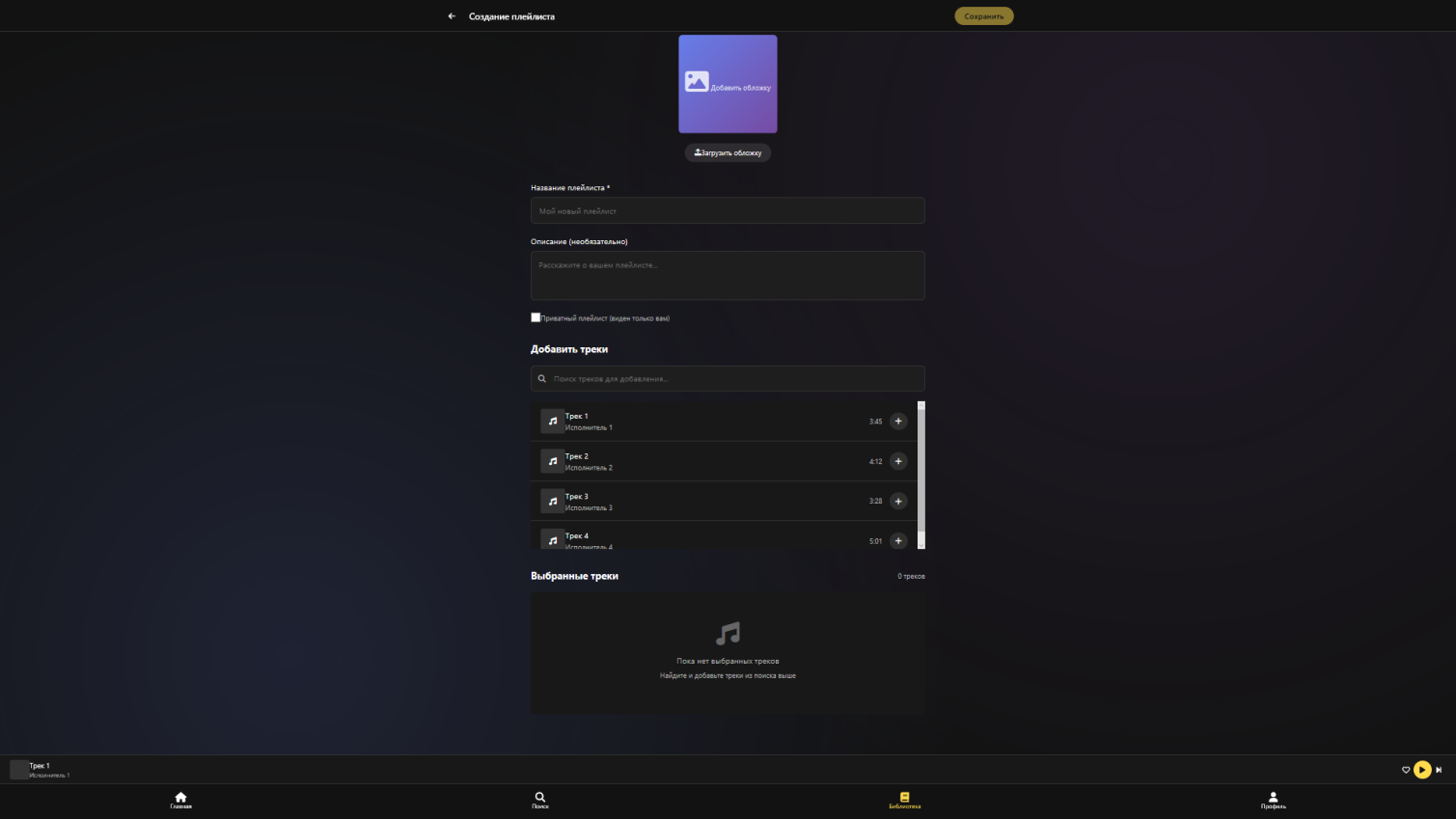


Рисунок 24. Страница создания плейлиста.

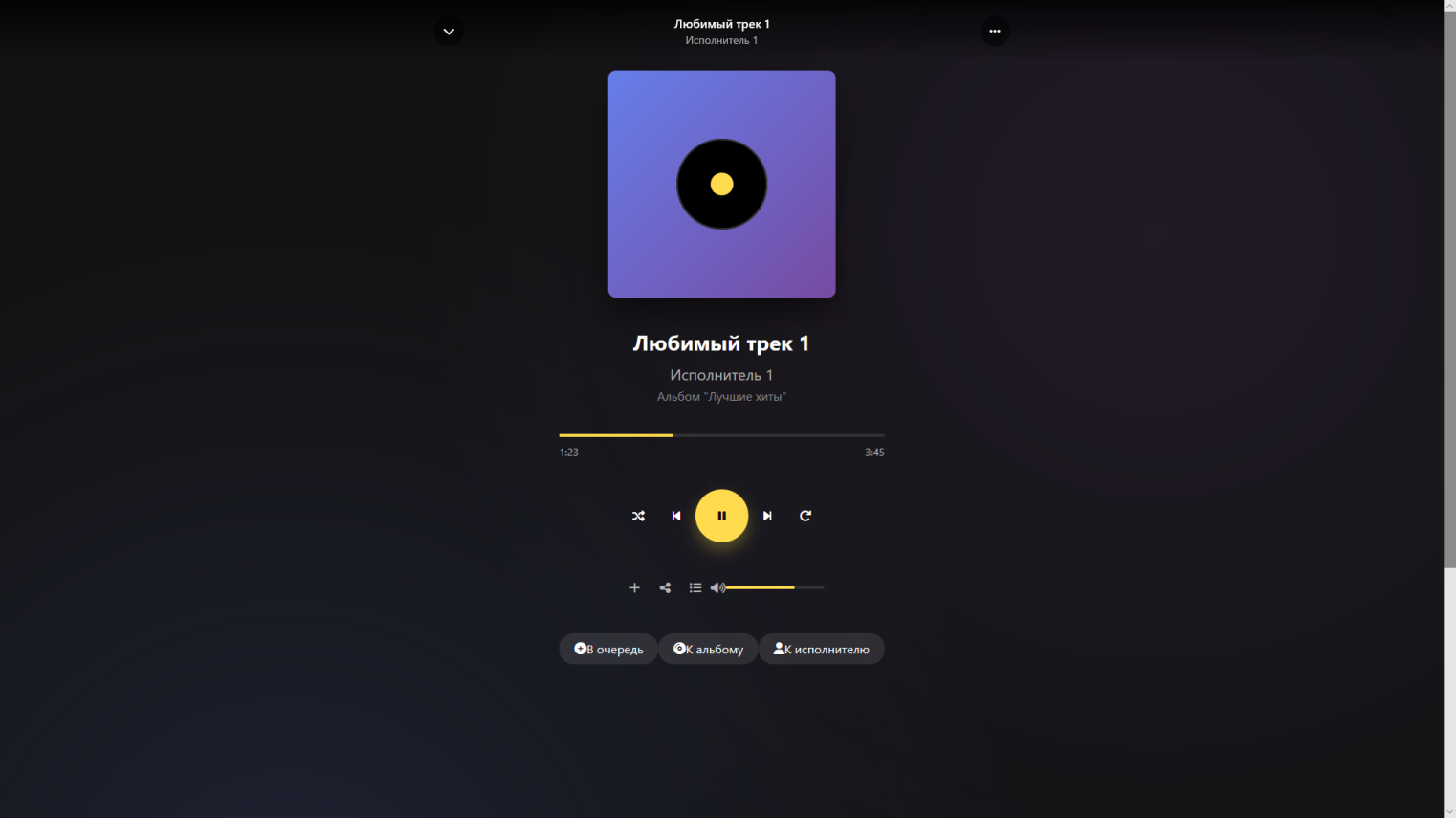


Рисунок 25. Страница плеера.

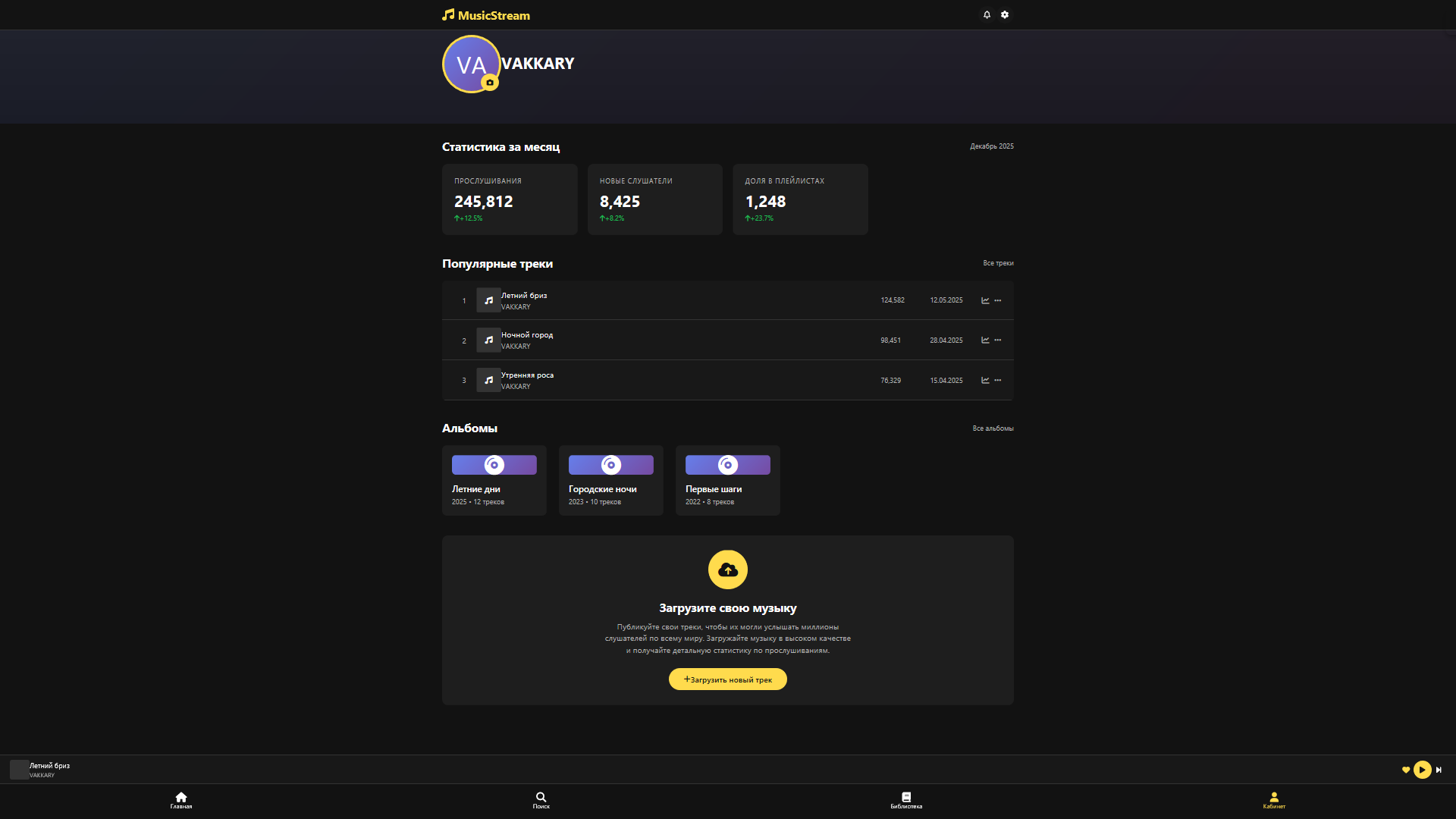


Рисунок 26. Страница «Профиль музыканта».

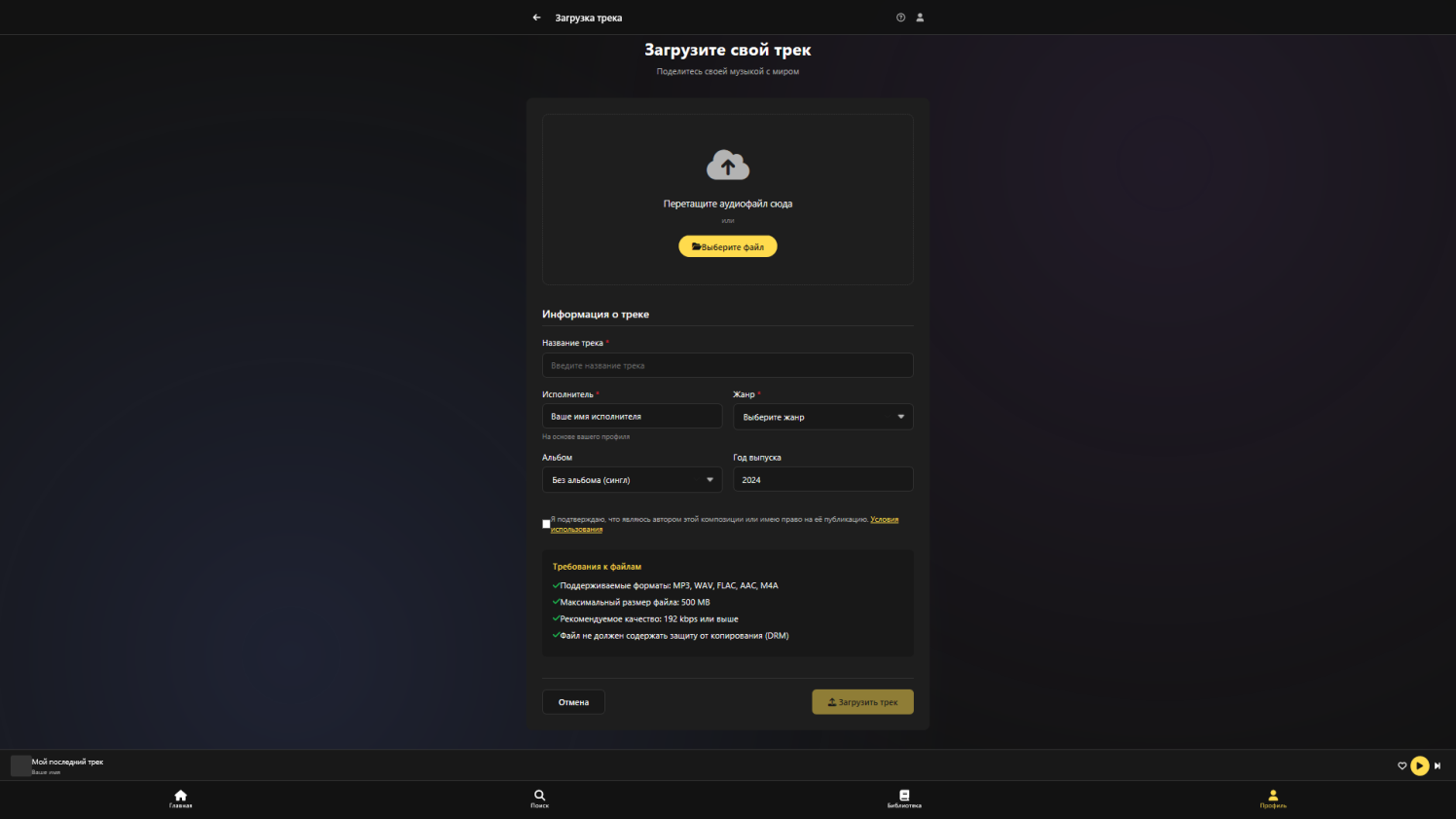


Рисунок 27. Страница загрузки нового трека.

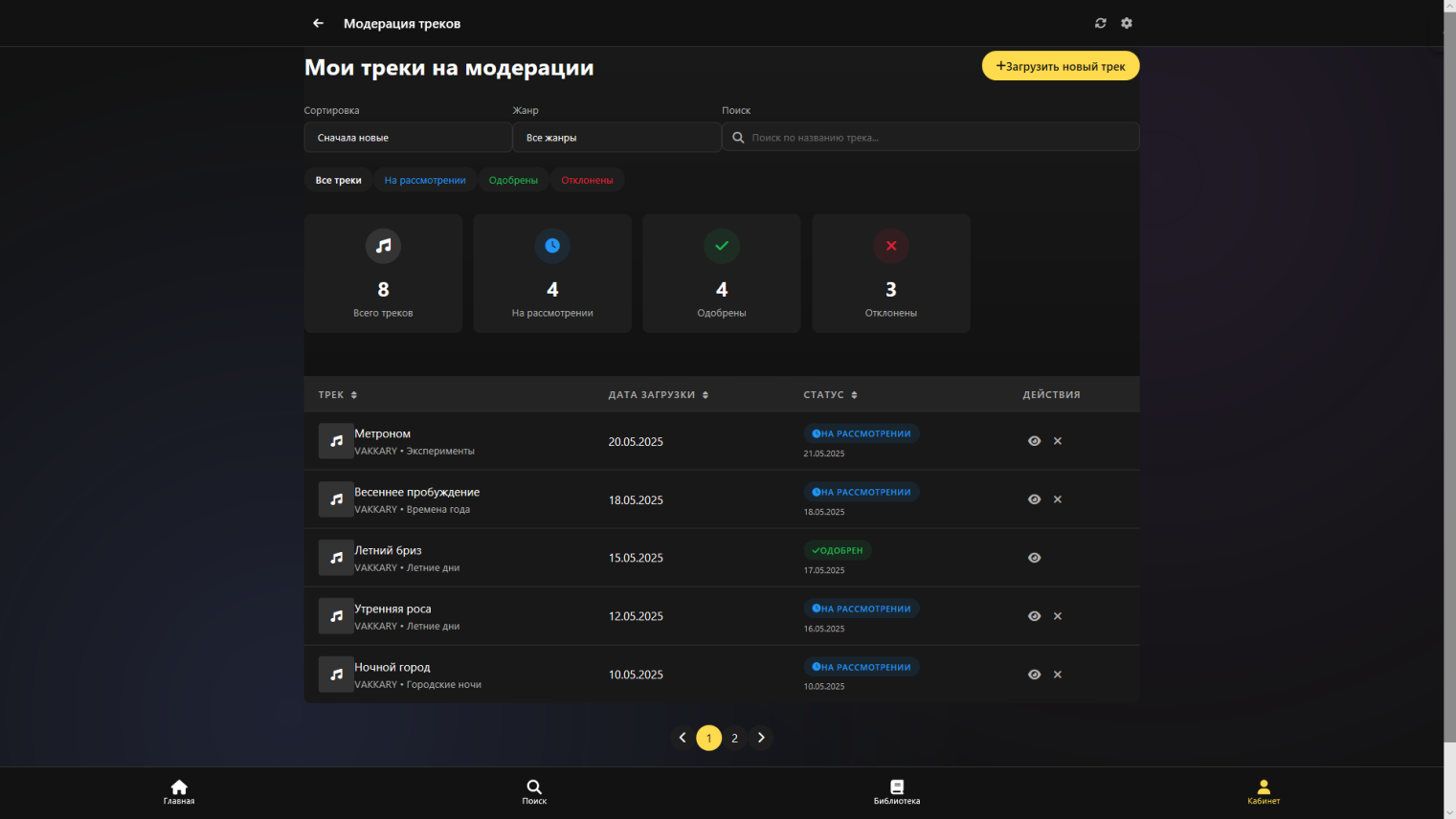


Рисунок 28. Страница загрузок музыканта.

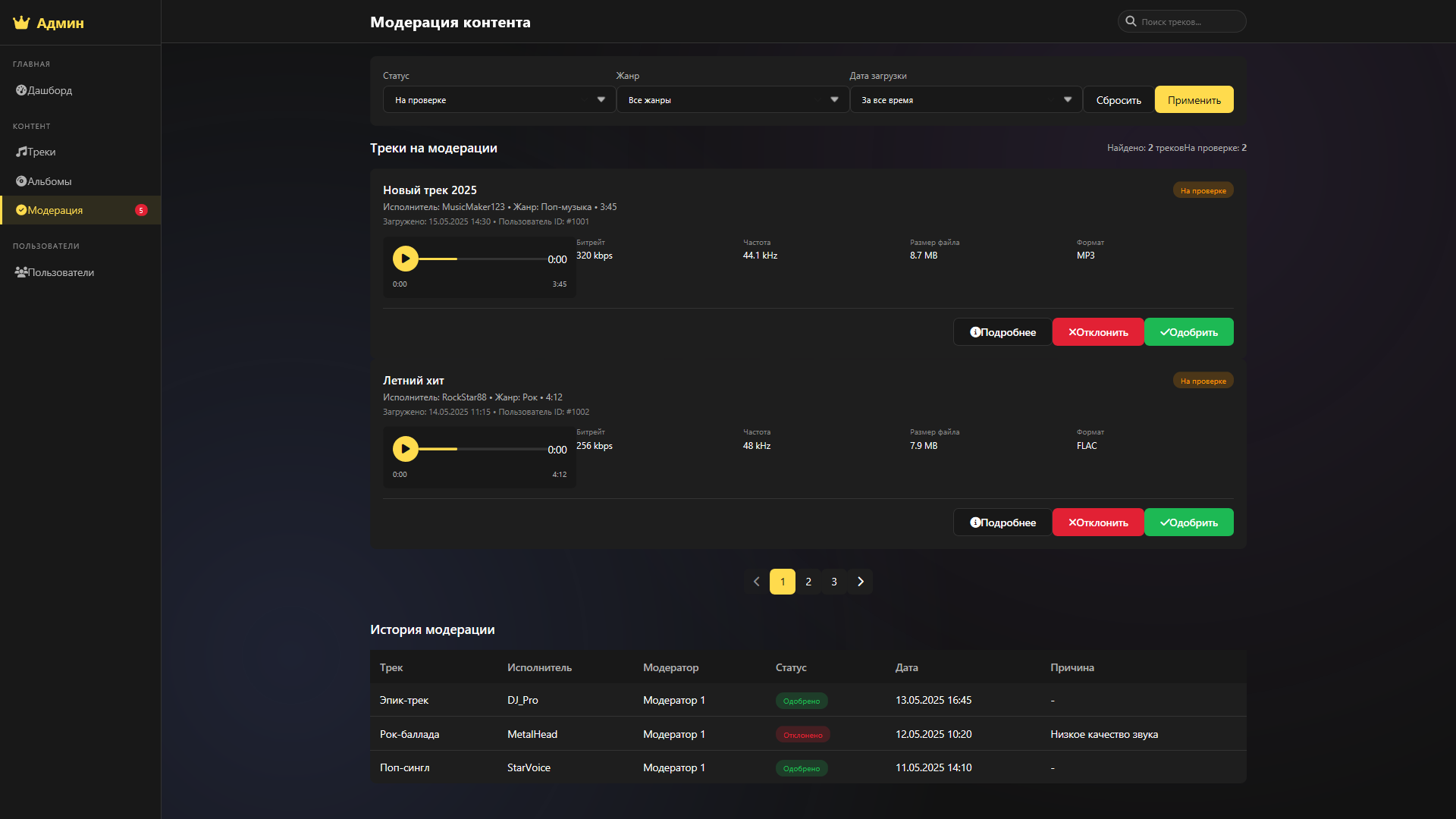


Рисунок 29. Страница админ. панели.

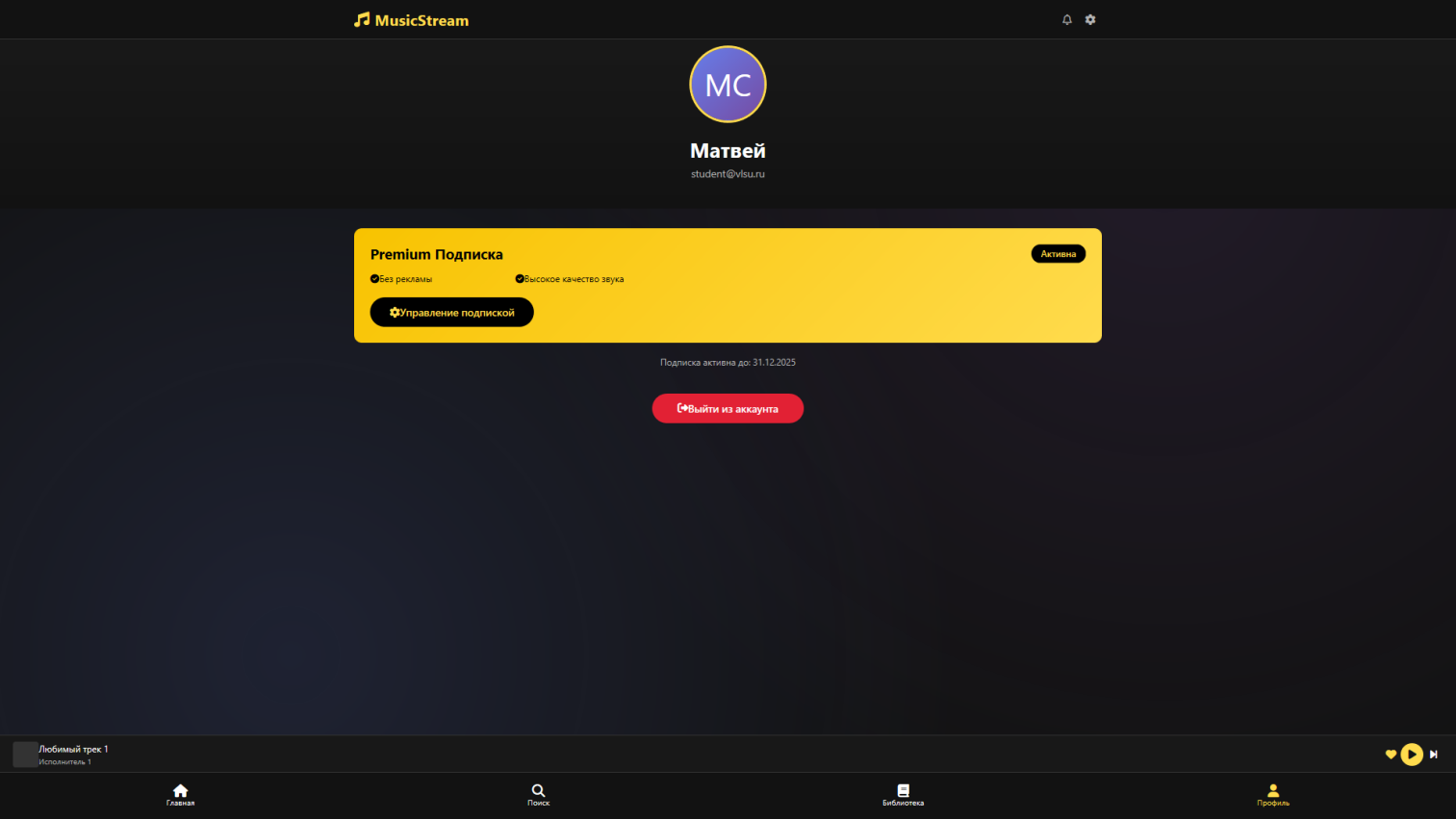


Рисунок 30. Страница профиля.

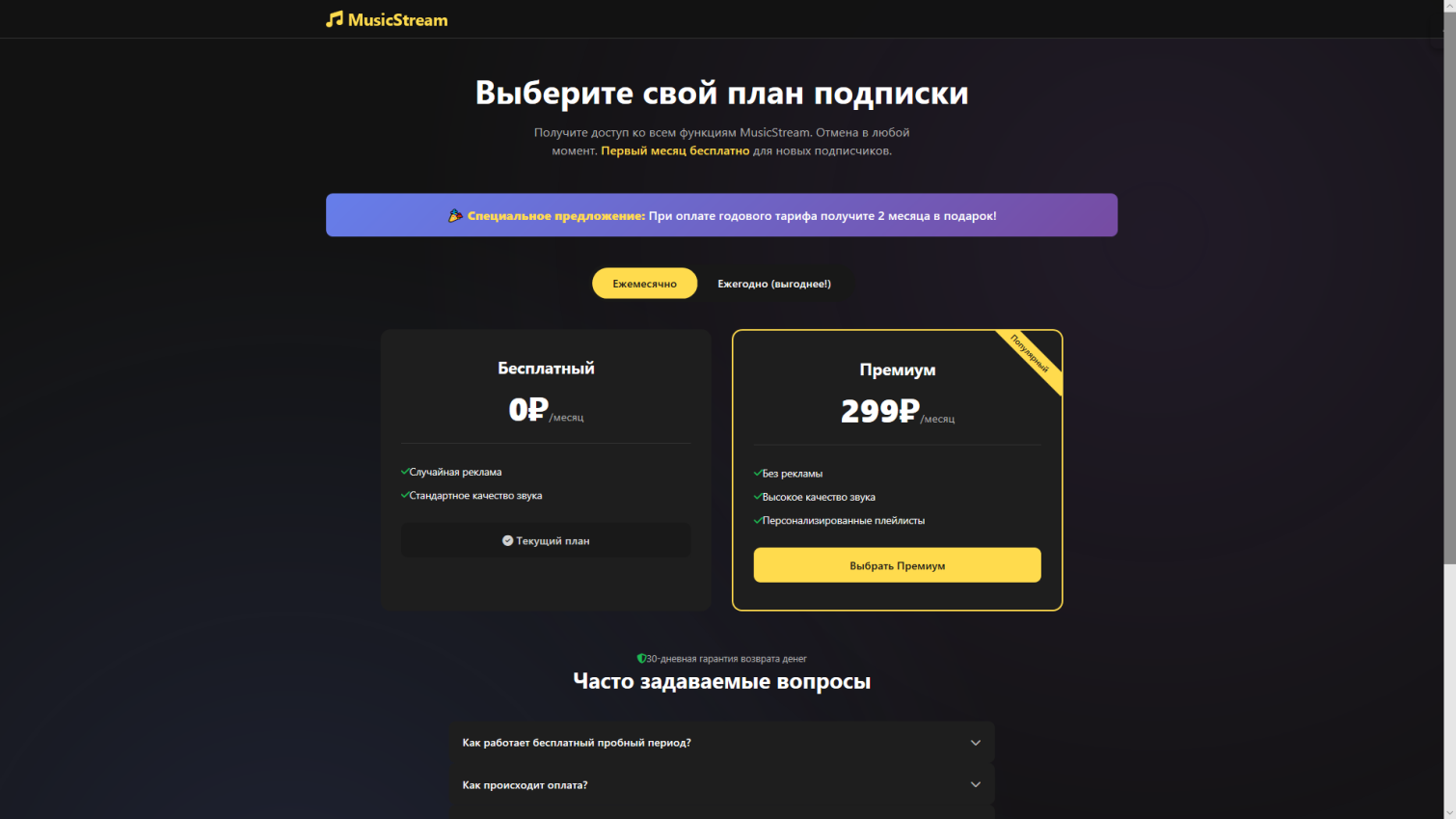


Рисунок 31. Страница тарифов подписки.

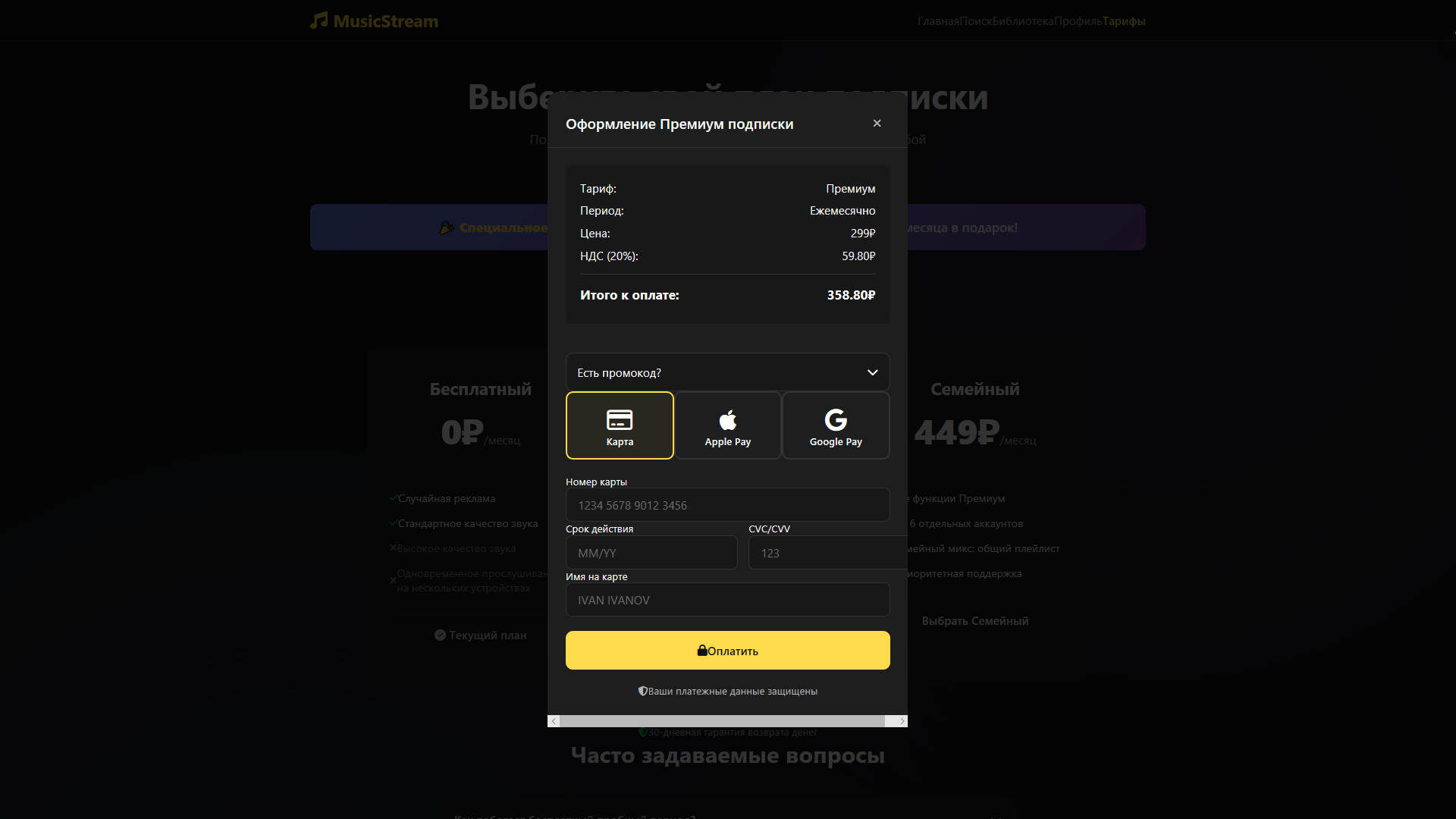


Рисунок 32. Форма оплаты подписки.

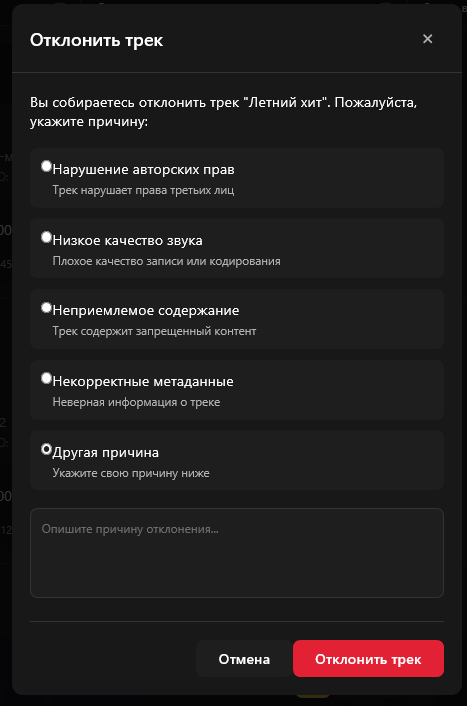


Рисунок 33. Модальное окно выбора причины отклонения трека.



Рисунок 34. Модальное окно детали модерации трека.

# 6. РЕАЛИЗАЦИЯ Программной системы

**6.1 Спецификации компонентов DAO**

**UserDAO**

**Назначение:** Управление данными пользователей системы, включая аутентификацию и управление ролями.

**Методы:**

**Optional<User> findById(Integer id)**

* Назначение: Поиск пользователя по ID
* Входные данные: id - уникальный идентификатор пользователя (целое число, > 0)
* Выходные данные: Optional<User> - объект пользователя или пустой Optional
* Исключения: DataAccessException - ошибка доступа к БД

**Optional<User> findByUsername(String username)**

* Назначение: Поиск пользователя по имени
* Входные данные: username - имя пользователя (строка 3-100 символов)
* Выходные данные: Optional<User> - объект пользователя или пустой Optional
* Исключения: DataAccessException - ошибка доступа к БД

**Optional<User> findByEmail(String email)**

* Назначение: Поиск пользователя по email
* Входные данные: email - электронная почта (валидный формат email)
* Выходные данные: Optional<User> - объект пользователя или пустой Optional
* Исключения: DataAccessException - ошибка доступа к БД

**Optional<User> findByUsernameOrEmail(String usernameOrEmail)**

* Назначение: Поиск пользователя по имени или email (для аутентификации)
* Входные данные: usernameOrEmail - имя пользователя или email
* Выходные данные: Optional<User> - объект пользователя или пустой Optional
* Исключения: DataAccessException - ошибка доступа к БД

**List<User> findAll()**

* Назначение: Получение списка всех пользователей
* Входные данные: отсутствуют
* Выходные данные: List<User> - список всех пользователей (может быть пустым)
* Исключения: DataAccessException - ошибка доступа к БД

**List<User> search(String searchTerm)**

* Назначение: Поиск пользователей по имени, email или роли
* Входные данные: searchTerm - строка поиска (может быть пустой)
* Выходные данные: List<User> - список найденных пользователей
* Исключения: DataAccessException - ошибка доступа к БД

**User save(User user)**

* Назначение: Сохранение нового или обновление существующего пользователя
* Входные данные: user - объект пользователя (не null, поля валидны)
* Выходные данные: User - сохраненный пользователь с заполненным ID
* Исключения:
* DataAccessException - ошибка доступа к БД
* DuplicateKeyException - нарушение уникальности (username/email)

**void updateRole(Integer userId, User.UserRole newRole)**

* Назначение: Обновление роли пользователя
* Входные данные:
* userId - ID пользователя (> 0)
* newRole - новая роль (не null)
* Выходные данные: отсутствуют
* Исключения:
* DataAccessException - ошибка доступа к БД
* EmptyResultDataAccessException - пользователь не найден

**boolean existsByUsername(String username)**

* Назначение: Проверка существования пользователя с указанным именем
* Входные данные: username - имя пользователя
* Выходные данные: true - если существует, false - если нет
* Исключения: DataAccessException - ошибка доступа к БД

**boolean existsByEmail(String email)**

* Назначение: Проверка существования пользователя с указанным email
* Входные данные: email - электронная почта
* Выходные данные: true - если существует, false - если нет
* Исключения: DataAccessException - ошибка доступа к БД

**TrackDAO**

Назначение: Управление данными музыкальных треков, включая поиск, фильтрацию и модерацию.

**Методы:**

**Optional<Track> findById(Integer id)**

* Назначение: Поиск трека по ID
* Входные данные: id - идентификатор трека (> 0)
* Выходные данные: Optional<Track> - объект трека со связанными данными или пустой Optional
* Исключения: DataAccessException - ошибка доступа к БД

**List<Track> findAll()**

* Назначение: Получение всех треков
* Входные данные: отсутствуют
* Выходные данные: List<Track> - список всех треков
* Исключения: DataAccessException - ошибка доступа к БД

**List<Track> findModerated()**

* Назначение: Получение только одобренных треков
* Входные данные: отсутствуют
* Выходные данные: List<Track> - список одобренных треков
* Исключения: DataAccessException - ошибка доступа к БД

**List<Track> findPendingModeration()**

* Назначение: Получение треков, ожидающих модерации
* Входные данные: отсутствуют
* Выходные данные: List<Track> - список треков на модерации
* Исключения: DataAccessException - ошибка доступа к БД

**List<Track> findByArtistId(Integer artistId)**

* Назначение: Получение треков исполнителя
* Входные данные: artistId - ID исполнителя (> 0)
* Выходные данные: List<Track> - список треков исполнителя
* Исключения: DataAccessException - ошибка доступа к БД

**List<Track> findByAlbumId(Integer albumId)**

* Назначение: Получение треков альбома
* Входные данные: albumId - ID альбома (> 0)
* Выходные данные: List<Track> - список треков альбома
* Исключения: DataAccessException - ошибка доступа к БД

**List<Track> findByGenreId(Integer genreId)**

* Назначение: Получение треков по жанру
* Входные данные: genreId - ID жанра (> 0)
* Выходные данные: List<Track> - список треков жанра
* Исключения: DataAccessException - ошибка доступа к БД

**List<Track> findByUploaderId(Integer userId)**

* Назначение: Получение треков, загруженных пользователем
* Входные данные: userId - ID пользователя (> 0)
* Выходные данные: List<Track> - список треков пользователя
* Исключения: DataAccessException - ошибка доступа к БД

**List<Track> search(String query)**

* Назначение: Поиск треков по названию или имени исполнителя
* Входные данные: query - поисковый запрос (минимум 1 символ)
* Выходные данные: List<Track> - список найденных треков
* Исключения: DataAccessException - ошибка доступа к БД

**List<Track> findSimilar(Integer genreId, Integer excludeTrackId, int limit)**

* Назначение: Поиск похожих треков по жанру
* Входные данные:
* genreId - ID жанра (> 0)
* excludeTrackId - ID исключаемого трека
* limit - максимальное количество результатов (1-50)
* Выходные данные: List<Track> - список похожих треков
* Исключения: DataAccessException - ошибка доступа к БД

**Track save(Track track)**

* Назначение: Сохранение нового или обновление существующего трека
* Входные данные: track - объект трека (не null, поля валидны)
* Выходные данные: Track - сохраненный трек с заполненным ID
* Исключения:
* DataAccessException - ошибка доступа к БД
* DataIntegrityViolationException - нарушение целостности данных

**void updateModerationStatus(Integer trackId, boolean moderated)**

* Назначение: Обновление статуса модерации трека
* Входные данные:
* trackId - ID трека (> 0)
* moderated - статус (true - одобрен, false - не одобрен)
* Выходные данные: отсутствуют
* Исключения:
* DataAccessException - ошибка доступа к БД
* EmptyResultDataAccessException - трек не найден

**void delete(Integer id)**

* Назначение: Удаление трека
* Входные данные: id - ID трека (> 0)
* Выходные данные: отсутствуют
* Исключения:
* DataAccessException - ошибка доступа к БД
* DataIntegrityViolationException - нарушение ссылочной целостности

**ArtistDAO**

**Назначение:** Управление данными исполнителей.

**Методы:**

**Optional<Artist> findById(Integer id)**

* Назначение: Поиск исполнителя по ID
* Входные данные: id - ID исполнителя (> 0)
* Выходные данные: Optional<Artist> - объект исполнителя или пустой Optional
* Исключения: DataAccessException - ошибка доступа к БД

**Optional<Artist> findByName(String name)**

* Назначение: Поиск исполнителя по имени
* Входные данные: name - имя исполнителя (не пустая строка)
* Выходные данные: Optional<Artist> - объект исполнителя или пустой Optional
* Исключения: DataAccessException - ошибка доступа к БД

**List<Artist> findAll()**

* Назначение: Получение всех исполнителей
* Входные данные: отсутствуют
* Выходные данные: List<Artist> - список всех исполнителей
* Исключения: DataAccessException - ошибка доступа к БД

**List<Artist> search(String query)**

* Назначение: Поиск исполнителей по имени или описанию
* Входные данные: query - поисковый запрос
* Выходные данные: List<Artist> - список найденных исполнителей
* Исключения: DataAccessException - ошибка доступа к БД

**Artist save(Artist artist)**

* Назначение: Сохранение нового или обновление существующего исполнителя
* Входные данные: artist - объект исполнителя
* Выходные данные: Artist - сохраненный исполнитель
* Исключения:
* DataAccessException - ошибка доступа к БД
* DuplicateKeyException - нарушение уникальности имени

**void delete(Integer id)**

* Назначение: Удаление исполнителя
* Входные данные: id - ID исполнителя
* Выходные данные: отсутствуют
* Исключения:
* DataAccessException - ошибка доступа к БД
* DataIntegrityViolationException - есть связанные записи

**AlbumDAO**

**Назначение:** Управление данными альбомов.

**Методы:**

**Optional<Album> findById(Integer id)**

* Назначение: Поиск альбома по ID
* Входные данные: id - ID альбома (> 0)
* Выходные данные: Optional<Album> - объект альбома со связанным исполнителем
* Исключения: DataAccessException - ошибка доступа к БД

**List<Album> findAll()**

* Назначение: Получение всех альбомов
* Входные данные: отсутствуют
* Выходные данные: List<Album> - список всех альбомов
* Исключения: DataAccessException - ошибка доступа к БД

**List<Album> findByArtistId(Integer artistId)**

* Назначение: Получение альбомов исполнителя
* Входные данные: artistId - ID исполнителя (> 0)
* Выходные данные: List<Album> - список альбомов исполнителя
* Исключения: DataAccessException - ошибка доступа к БД

**List<Album> search(String query)**

* Назначение: Поиск альбомов по названию или имени исполнителя
* Входные данные: query - поисковый запрос
* Выходные данные: List<Album> - список найденных альбомов
* Исключения: DataAccessException - ошибка доступа к БД

**Album save(Album album)**

* Назначение: Сохранение нового или обновление существующего альбома
* Входные данные: album - объект альбома
* Выходные данные: Album - сохраненный альбом
* Исключения:
* DataAccessException - ошибка доступа к БД
* DataIntegrityViolationException - нарушение целостности

**void delete(Integer id)**

* Назначение: Удаление альбома
* Входные данные: id - ID альбома
* Выходные данные: отсутствуют
* Исключения:
* DataAccessException - ошибка доступа к БД
* DataIntegrityViolationException - есть связанные треки

**GenreDAO**

**Назначение:** Управление данными музыкальных жанров.

**Методы:**

**Optional<Genre> findById(Integer id)**

* Назначение: Поиск жанра по ID
* Входные данные: id - ID жанра (> 0)
* Выходные данные: Optional<Genre> - объект жанра или пустой Optional
* Исключения: DataAccessException - ошибка доступа к БД

**Optional<Genre> findByName(String name)**

* Назначение: Поиск жанра по названию
* Входные данные: name - название жанра
* Выходные данные: Optional<Genre> - объект жанра или пустой Optional
* Исключения: DataAccessException - ошибка доступа к БД

**List<Genre> findAll()**

* Назначение: Получение всех жанров
* Входные данные: отсутствуют
* Выходные данные: List<Genre> - список всех жанров
* Исключения: DataAccessException - ошибка доступа к БД

**Genre save(Genre genre)**

* Назначение: Сохранение нового или обновление существующего жанра
* Входные данные: genre - объект жанра
* Выходные данные: Genre - сохраненный жанр
* Исключения:
* DataAccessException - ошибка доступа к БД
* DuplicateKeyException - жанр с таким именем уже существует

**void delete(Integer id)**

* Назначение: Удаление жанра
* Входные данные: id - ID жанра
* Выходные данные: отсутствуют
* Исключения:
* DataAccessException - ошибка доступа к БД
* DataIntegrityViolationException - есть связанные треки

**boolean existsByName(String name)**

* Назначение: Проверка существования жанра по названию
* Входные данные: name - название жанра
* Выходные данные: true - если существует, false - если нет
* Исключения: DataAccessException - ошибка доступа к БД

**PlaylistDAO**

**Назначение:** Управление данными плейлистов пользователей.

**Методы:**

**Optional<Playlist> findById(Integer id)**

* Назначение: Поиск плейлиста по ID
* Входные данные: id - ID плейлиста (> 0)
* Выходные данные: Optional<Playlist> - объект плейлиста с треками
* Исключения: DataAccessException - ошибка доступа к БД

**List<Playlist> findByUserId(Integer userId)**

* Назначение: Получение плейлистов пользователя
* Входные данные: userId - ID пользователя (> 0)
* Выходные данные: List<Playlist> - список плейлистов пользователя
* Исключения: DataAccessException - ошибка доступа к БД

**List<Playlist> findPublicPlaylists()**

* Назначение: Получение публичных плейлистов
* Входные данные: отсутствуют
* Выходные данные: List<Playlist> - список публичных плейлистов
* Исключения: DataAccessException - ошибка доступа к БД

**Playlist save(Playlist playlist)**

* Назначение: Сохранение нового или обновление существующего плейлиста
* Входные данные: playlist - объект плейлиста
* Выходные данные: Playlist - сохраненный плейлист
* Исключения:
* DataAccessException - ошибка доступа к БД
* DataIntegrityViolationException - нарушение целостности

**void delete(Integer id)**

* Назначение: Удаление плейлиста
* Входные данные: id - ID плейлиста
* Выходные данные: отсутствуют
* Исключения: DataAccessException - ошибка доступа к БД

**PlaylistTrackDAO**

**Назначение:** Управление связями между плейлистами и треками (N:M).

**Методы:**

**Optional<PlaylistTrack> findById(Integer playlistId, Integer trackId)**

* Назначение: Поиск связи по ID плейлиста и трека
* Входные данные:
* playlistId - ID плейлиста (> 0)
* trackId - ID трека (> 0)
* Выходные данные: Optional<PlaylistTrack> - объект связи или пустой Optional
* Исключения: DataAccessException - ошибка доступа к БД

**List<PlaylistTrack> findByPlaylistId(Integer playlistId)**

* Назначение: Получение всех треков плейлиста
* Входные данные: playlistId - ID плейлиста (> 0)
* Выходные данные: List<PlaylistTrack> - список связей с треками
* Исключения: DataAccessException - ошибка доступа к БД

**void save(PlaylistTrack playlistTrack)**

* Назначение: Добавление трека в плейлист
* Входные данные: playlistTrack - объект связи (не null)
* Выходные данные: отсутствуют
* Исключения:
* DataAccessException - ошибка доступа к БД
* DuplicateKeyException - трек уже в плейлисте

**void delete(Integer playlistId, Integer trackId)**

* Назначение: Удаление трека из плейлиста
* Входные данные:
* playlistId - ID плейлиста (> 0)
* trackId - ID трека (> 0)
* Выходные данные: отсутствуют
* Исключения: DataAccessException - ошибка доступа к БД

**void deleteByPlaylistId(Integer playlistId)**

* Назначение: Удаление всех треков из плейлиста
* Входные данные: playlistId - ID плейлиста (> 0)
* Выходные данные: отсутствуют
* Исключения: DataAccessException - ошибка доступа к БД

**void updatePositions(Integer playlistId, List<Integer> trackIds)**

* Назначение: Обновление порядка треков в плейлисте
* Входные данные:
* playlistId - ID плейлиста (> 0)
* trackIds - список ID треков в новом порядке
* Выходные данные: отсутствуют
* Исключения:
* DataAccessException - ошибка доступа к БД
* IllegalArgumentException - некорректный список

**SubscriptionDAO**

**Назначение:** Управление данными подписок пользователей.

**Методы:**

**Optional<Subscription> findById(Integer id)**

* Назначение: Поиск подписки по ID
* Входные данные: id - ID подписки (> 0)
* Выходные данные: Optional<Subscription> - объект подписки или пустой Optional
* Исключения: DataAccessException - ошибка доступа к БД

**List<Subscription> findByUserId(Integer userId)**

* Назначение: Получение всех подписок пользователя
* Входные данные: userId - ID пользователя (> 0)
* Выходные данные: List<Subscription> - список подписок пользователя
* Исключения: DataAccessException - ошибка доступа к БД

**Optional<Subscription> findActiveByUserId(Integer userId)**

* Назначение: Поиск активной подписки пользователя
* Входные данные: userId - ID пользователя (> 0)
* Выходные данные: Optional<Subscription> - активная подписка или пустой Optional
* Исключения: DataAccessException - ошибка доступа к БД

**List<Subscription> findActiveSubscriptions()**

* Назначение: Получение всех активных подписок
* Входные данные: отсутствуют
* Выходные данные: List<Subscription> - список активных подписок
* Исключения: DataAccessException - ошибка доступа к БД

**Subscription save(Subscription subscription)**

* Назначение: Сохранение новой или обновление существующей подписки
* Входные данные: subscription - объект подписки
* Выходные данные: Subscription - сохраненная подписка
* Исключения:
* DataAccessException - ошибка доступа к БД
* DataIntegrityViolationException - нарушение целостности

**void delete(Integer id)**

* Назначение: Удаление подписки
* Входные данные: id - ID подписки
* Выходные данные: отсутствуют
* Исключения: DataAccessException - ошибка доступа к БД

**ModerationDAO**

**Назначение:** Управление данными модерации треков.

**Методы:**

**Optional<Moderation> findById(Integer id)**

* Назначение: Поиск записи модерации по ID
* Входные данные: id - ID записи (> 0)
* Выходные данные: Optional<Moderation> - объект модерации или пустой Optional
* Исключения: DataAccessException - ошибка доступа к БД

**Optional<Moderation> findByTrackId(Integer trackId)**

* Назначение: Поиск последней модерации трека
* Входные данные: trackId - ID трека (> 0)
* Выходные данные: Optional<Moderation> - последняя модерация или пустой Optional
* Исключения: DataAccessException - ошибка доступа к БД

**List<Moderation> findAll()**

* Назначение: Получение всех записей модерации
* Входные данные: отсутствуют
* Выходные данные: List<Moderation> - список всех модераций
* Исключения: DataAccessException - ошибка доступа к БД

**List<Moderation> findByModeratorId(Integer moderatorId)**

* Назначение: Получение модераций, проведенных модератором
* Входные данные: moderatorId - ID модератора (> 0)
* Выходные данные: List<Moderation> - список модераций
* Исключения: DataAccessException - ошибка доступа к БД

**List<Moderation> findByStatus(Moderation.ModerationStatus status)**

* Назначение: Получение модераций по статусу
* Входные данные: status - статус модерации
* Выходные данные: List<Moderation> - список модераций
* Исключения: DataAccessException - ошибка доступа к БД

**int findPendingCount()**

* Назначение: Получение количества ожидающих модерации треков
* Входные данные: отсутствуют
* Выходные данные: int - количество ожидающих
* Исключения: DataAccessException - ошибка доступа к БД

**Moderation save(Moderation moderation)**

* Назначение: Сохранение новой или обновление существующей записи модерации
* Входные данные: moderation - объект модерации
* Выходные данные: Moderation - сохраненная запись
* Исключения: DataAccessException - ошибка доступа к БД

**void delete(Integer id)**

* Назначение: Удаление записи модерации
* Входные данные: id - ID записи
* Выходные данные: отсутствуют
* Исключения: DataAccessException - ошибка доступа к БД

**TrackStatisticsDAO**

**Назначение:** Управление статистикой прослушиваний треков.

**Методы:**

**Optional<TrackStatistics> findById(Integer id)**

* Назначение: Поиск записи статистики по ID
* Входные данные: id - ID записи (> 0)
* Выходные данные: Optional<TrackStatistics> - объект статистики или пустой Optional
* Исключения: DataAccessException - ошибка доступа к БД

**List<TrackStatistics> findByTrackId(Integer trackId)**

* Назначение: Получение всей статистики трека
* Входные данные: trackId - ID трека (> 0)
* Выходные данные: List<TrackStatistics> - список записей статистики
* Исключения: DataAccessException - ошибка доступа к БД

**List<TrackStatistics> findByDateRange(LocalDateTime start, LocalDateTime end)**

* Назначение: Получение статистики за период
* Входные данные:
* start - начальная дата
* end - конечная дата
* Выходные данные: List<TrackStatistics> - список записей за период
* Исключения: DataAccessException - ошибка доступа к БД

**int getTotalListensByTrackId(Integer trackId)**

* Назначение: Получение общего количества прослушиваний трека
* Входные данные: trackId - ID трека (> 0)
* Выходные данные: int - общее количество прослушиваний
* Исключения: DataAccessException - ошибка доступа к БД

**Map<Integer, Integer> getTopTracks(int limit)**

* Назначение: Получение самых популярных треков
* Входные данные: limit - максимальное количество результатов (1-100)
* Выходные данные: Map<Integer, Integer> - карта ID трека -> количество прослушиваний
* Исключения: DataAccessException - ошибка доступа к БД

**TrackStatistics save(TrackStatistics statistics)**

* Назначение: Сохранение записи статистики
* Входные данные: statistics - объект статистики
* Выходные данные: TrackStatistics - сохраненная запись
* Исключения: DataAccessException - ошибка доступа к БД

**void recordPlay(Integer trackId)**

* Назначение: Запись одного прослушивания трека
* Входные данные: trackId - ID трека (> 0)
* Выходные данные: отсутствуют
* Исключения: DataAccessException - ошибка доступа к БД

**void delete(Integer id)**

* Назначение: Удаление записи статистики
* Входные данные: id - ID записи
* Выходные данные: отсутствуют
* Исключения: DataAccessException - ошибка доступа к БД

**6.2 Диаграмма компонентов**

Диаграмма компонентов изображена на рисунке 35.

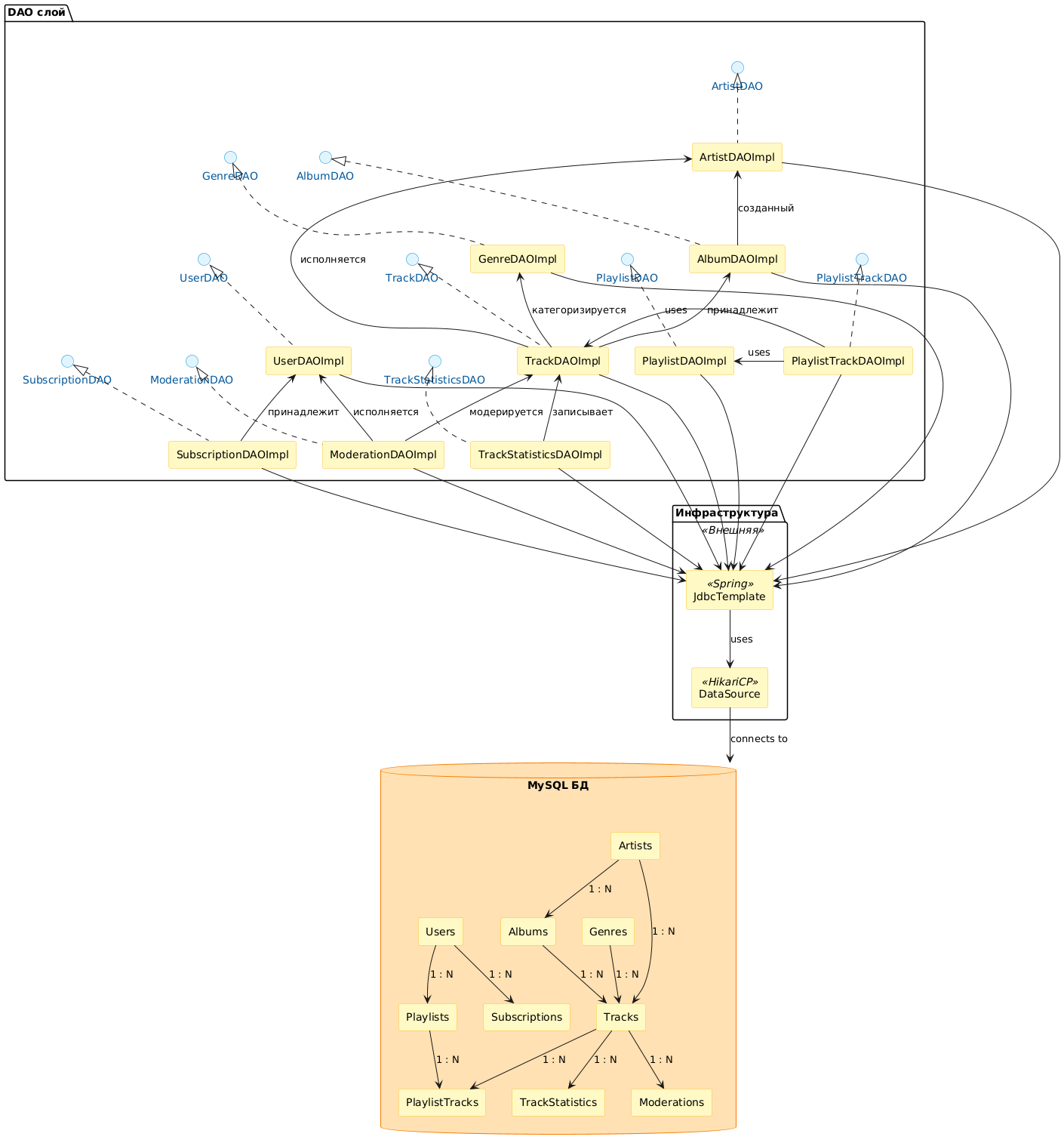


Рисунок 35. Диаграмма компонентов DAO слоя.