Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет Информационных технологий и управления

Кафедра Информационных технологий автоматизированных систем

Дисциплина: Базы и банки данных

|  |
| --- |
|  |
|  |
|  |

Пояснительная записка

к курсовому проекту

на тему

**Автоматизированная система музыкального стримингового сервиса**

БГУИР КП 1-53 01 02 01 031 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент |  | А. В. Максунов |
| Руководитель |  | А. Ф. Трофимович |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Минск 2020

**РЕФЕРАТ**

АВТОМАТИЗИРОВАННАЯ СИСТЕМА МУЗЫКАЛЬНОГО СТРИМИНГОВОГО СЕРВИСА: курсовой проект / А. В. Максунов. – Минск : БГУИР, 2020, – п.з. –65 с.

Курсовой проект на тему «автоматизированная система музыкального стримингового сервиса» была разработана с целью предоставления пользователям возможности прослушивания музыки онлайн, а также скачивания музыки на устройства.

Пояснительная записка к курсовому проекту состоит из введения, 4 разделов, включающих литературный обзор по теме курсового проекта, сравнительный анализ существующих аналогов системы, описание процессов проектирования и программной реализации, заключение и список использованных источников.

Система реализована на языке программирования Java с использованием различных технологий, предназначенных для разработки веб-приложений. В том числе:

* Spring Core
* Spring MVC
* Spring Security
* Hibernate
* JPA (Java persistence API)
* JSP (Java server pages)

Для реализации базы данных приложения была выбрана СУБД *MySQL*.

В ходе работы над курсовым проектом спроектирована и разработана автоматизированная система музыкального стримингового сервиса, также с помощь средств языка графического проектирования *UML* составлена диаграмма вариантов использования, составлена схема базы данных, удовлетворяющая трем начальным формам, и создано краткое руководство пользователя по использованию данной системы.

Данная система будет находится в открытом доступе, и пользователем может быть любой человек. Разработанное приложение будет использоваться в развлекательных целях.

Учреждение образования

«Белорусский государственный университет информатики   
и радиоэлектроники»

Факультет информационных технологий и управления

УТВЕРЖДАЮ

Заведующий кафедрой

––––––––––––––––––––––––

(подпись)

–––––––––––––––––––2020г.

ЗАДАНИЕ

по курсовому проектированию

Студенту *Максунов Александр Владимирович–––––––––––––––––––*

1. Тема проекта *Автоматизированная система музыкального стримингового сервиса*

2. Срок сдачи студентом законченного проекта– –––*20 мая 2020 г.––––*

3. Исходные данные к проекту *Система предоставляет пользователю возможность прослушивать музыку онлайн. Также есть возможность купить понравившуюся аудиозапись, тогда появится доступ к скачиванию данной аудиозаписи на устройство. В системе есть разделение пользователей по ролям. Так, в системе есть администратор и обычный пользователь. Роли определяют доступ к различным функциям приложения. Администратор может добавлять, изменять и удалять записи.*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке):

*Введение*

1. *Анализ технического задание на курсовое проектирование*
2. *Проектирование системы*
3. *Описание процесса реализации*
4. *Руководство пользователя*

*Заключение*

*Список использованных источников*

*Приложение*

5. Консультант по проекту (с обозначением разделов проекта) *А.Ф.Трофимович–––––––– ––––––––––––––––––––––––––––––––––––––––––––––––––––– –*

6. Дата выдачи задания –––– –*2 февраля 2020 г.––––––––––––––––––––––  –*

7. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

*раздел 1 к 15.03 – 15 %; –– –––––––––––––– –––––––––––––––– –*

*раздел 2 к 15.04 – 50 %;–––––––––––––––– ––––––––––––––––––  ––––*

*раздел 3 к 15.05 – 15 %;––––––––––––––––––––– –––––––––––––––––*

*оформление пояснительной записки к 20. 05 – 20 %*

*Защита курсового проекта с* ***23.05*** *по* ***05.06.2020****г.––––––––––––––––––––––*

Руководитель*– –––––––­­­­­-- А. Ф. Трофимович–*

(подпись)

Задание принял к исполнению *–––––––\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Максунов*

(подпись)

# СОДЕРЖАНИЕ

[Введение 6](#_Toc38795064)

[1 Анализ технического задания на курсовое проектирование 7](#_Toc38795065)

[1.1 Существующая ситуация на рынке 7](#_Toc38795066)

[1.2 Аналоги 7](#_Toc38795067)

[1.3 Постановка задачи 10](#_Toc38795071)

[2 Проектирование системы 12](#_Toc38795072)

[2.1 Проектирование бизнес логики 12](#_Toc38795073)

[2.2 Проектирование базы данных 13](#_Toc38795074)

[2.3 Проектирование пользовательского интерфейса 17](#_Toc38795075)

[3 Описание процесса реализации 19](#_Toc38795076)

[3.1 Выбор СУБД 19](#_Toc38795077)

[3.2 Выбор языка программирования 23](#_Toc38795082)

[3.3 Реализация ресурса 27](#_Toc38795086)

[4 Руководство пользователя 51](#_Toc38795087)

[Заключение 60](#_Toc38795088)

[Список использованных источников 61](#_Toc38795089)

[Приложение А (справочное) 62](#_Toc38795090)

# Введение

В последние годы люди все меньше качают из интернета музыку, фильмы и видео. Это связано с появлением стриминговых сервисов. Что же такое стриминговый сервис?

Стриминг (англ. streaming — потоковый) — это способ передачи данных от провайдера к конечному пользователю, при котором контент находится на удаленном сервере, и для его воспроизведения на устройстве ни загрузка, ни установка не требуются. Получается что-то вроде старого-доброго теле- и радиовещания, только с более широкими возможностями: пользователь теперь не зависит от времени трансляции, он сам выбирает наиболее удобный момент для потребления того или иного вида медиа-информации. При этом охват контента в сотни, если не в тысячи раз шире.

Основные преимущества стриминга перед привычными способами потребления медиа-информации заключаются в том, что на устройствах освобождается очень много места, которое раньше было занято разным контентом. Став пользователем сервиса, вы получите доступ к контенту на всех своих устройствах в любое время суток.

Что касается музыкального стримингового сервиса, они сдедержать огромное число аудиозаписей различного направления. Современные технологии позволяют анализировать историю прослушиваний и создавать актуальные подборки с музыкой, которая больше нравится пользователю.

С такими сервисами вам не придётся мучительно подбирать плейлист. Там можно найти множество существующих, созданных автоматически по похожим трекам. Или создать собственные на любой случай жизни. Обычно такие сервисы можно связать через мобильное приложение с умными часами, наушниками нового поколения, домашними музыкальными системами. Поддержка популярных сервисов становится важным элементом для продвижения любого нового аудио-девайса или спортивного браслета. Потому что со стриминговыми сервисами вы можете взять любимую музыку всюду, где есть интернет.

Что же касается перспектив развития, то достаточно рассмотреть пример с музыкальными потоковыми сервисами: по данным Strategy Analytics рост их прибыли за 2013 год составил 40%, в то время как рост прибыли других музыкальных ресурсов не превысил и 9%.

# АНАЛИЗ ТЕХНИЧЕСКОГО ЗАДАНИЯ НА КУРСОВОЕ ПРОЕКТИРОВАНИЕ

## Существующая ситуация на рынке

С появлением стриминговых сервисов прослушивание музыки стало гораздо удобнее и доступнее. А это привело к тому, что мы слушаем ее гораздо чаще. Благодаря этому в последние годы стриминговые сервисы с каждым днем набирают все большую популярность среди людей. Стриминговые сервисы предоставляют нам доступ к огромному количеству музыки различных жанров. Благодаря машинному обучению некоторые сервисы способны анализировать поведенческие данные слушателей, в том числе – количество прослушиваний в разное время суток. Такую аналитику используют для формирования плейлистов — например, считается, что треки в духе «Acoustic Calm» или «Nature Sounds» логично предложить послушать пользователям перед сном. В целом, с появлением стриминга мировой музыкальный рынок постоянно [растет](https://musicbiz.org/news/ifpi-global-music-report-2019-global-music-market-grows-for-fourth-consecutive-year/): так, в 2018 году продажи выросли на 9,7% в сравнении с прошлым годом и составили 19,1 млрд. долларов. Еще один занятный факт: исследования показывают, что благодаря потоковым сервисам доля пиратства на музыкальном рынке снизилась более чем на 50% с 2013 года.

## Аналоги

## Spotify

Spotify — самый популярный в мире музыкальный стриминговый сервис, значительно обгоняющий конкурентов по множеству показателей. К сожалению, он ещё не вышел на российский рынок.

Этот сервис — настоящая музыкальная соцсеть: здесь можно добавлять друзей, обмениваться сообщениями и комментировать музыкальные новости.

Пользовательские плейлисты не только доступны по прямой ссылке, но и отображаются в поиске, что в корне меняет процесс знакомства с новыми аудиозаписями. Рекомендации лишаются выверенности компьютерных алгоритмов, но приобретают человеческий фактор спонтанности.

Главное преимущество Spotify — алгоритмы подбора музыки, которые практически идеально подстраиваются под вкусы слушателей.

Помимо этого, есть тематические плейлисты, составленные другими пользователями и самими музыкантами.

Также в Spotify есть очень крутые персонализированные радиостанции на основе исполнителей, плейлистов, жанров или отдельных треков. А кроме них — кросс-платформенная поддержка и возможность управлять воспроизведением с любых устройств.

* **Преимущества:** высокий битрейт, продвинутые алгоритмы подбора музыки, молниеносная работа, крутой десктопный клиент, множество дополнительных функций.
* **Недостатки:** трудности с использованием в российском сегменте.

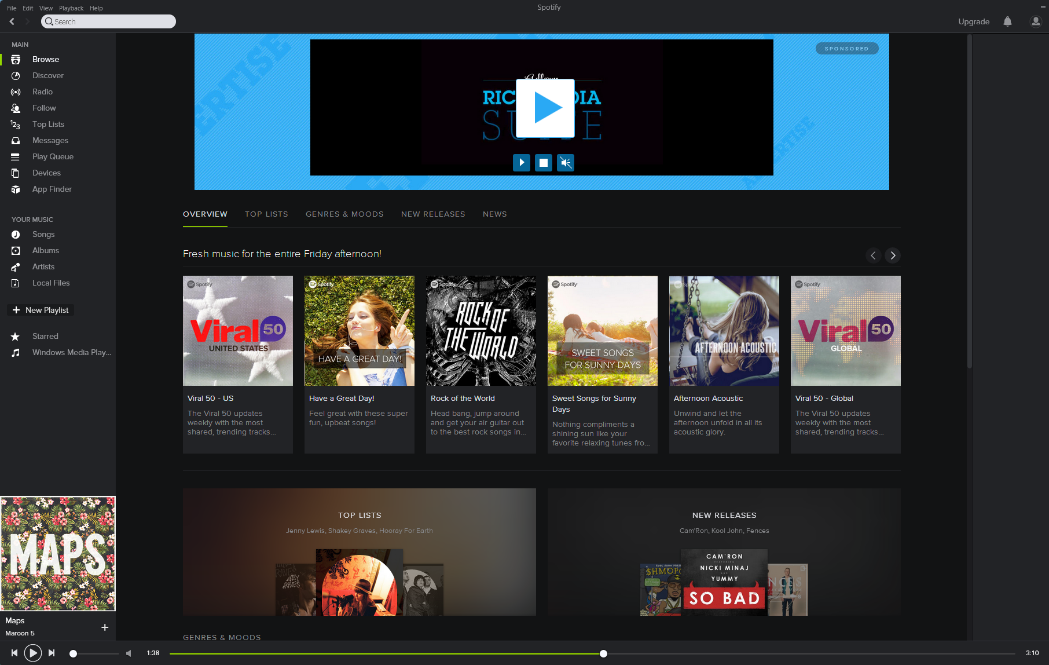


Рисунок 1.1 – Главная страница *Spotify*

## Apple iTunes Store

Apple Music — второй по популярности в мире музыкальный стриминговый сервис после Spotify. Обладает обширной базой в 60 миллионов треков. Сервис предлагает плейлисты для разного настроения и авторские подборки от десятков зарубежных и российских кураторов (музыкальных изданий и лейблов).

[Apple Music](https://lifehacker.ru/10-funkcij-apple-music/) поддерживает функцию составления пользовательских плейлистов, которыми можно делиться с друзьями по ссылке. А ещё сервис умеет рекомендовать подходящую музыку на основе ранее понравившихся вам треков.

* **Преимущества:**совместимость с экосистемой Apple без установки дополнительного ПО, слияние с имеющейся медиатекой, высокий битрейт (256 Кбит/с, AAC), 60 млн треков.
* **Недостатки:**не очень хорошая работа персональных рекомендаций в случае с редкими и специфическими жанрами, обходится дороже, чем конкуренты.

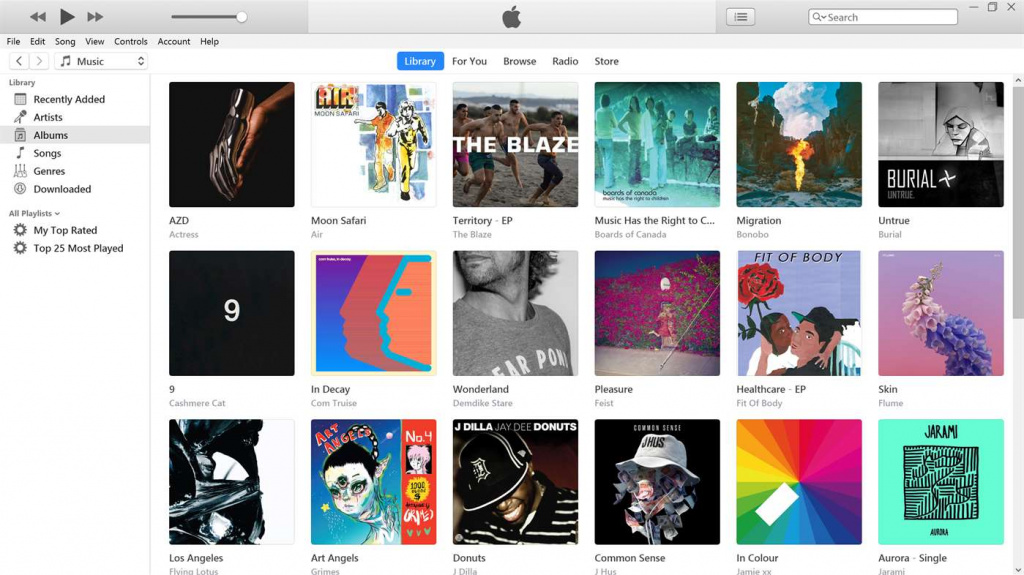


Рисунок 1.2 – Главная страница *Apple iTunes Store*

## Яндекс.Музыка

Стриминговый сервис компании «[Яндекс](https://ru.wikipedia.org/wiki/%D0%AF%D0%BD%D0%B4%D0%B5%D0%BA%D1%81)», позволяющий прослушивать музыкальные композиции, альбомы, подборки музыкальных треков и получать персональные рекомендации.

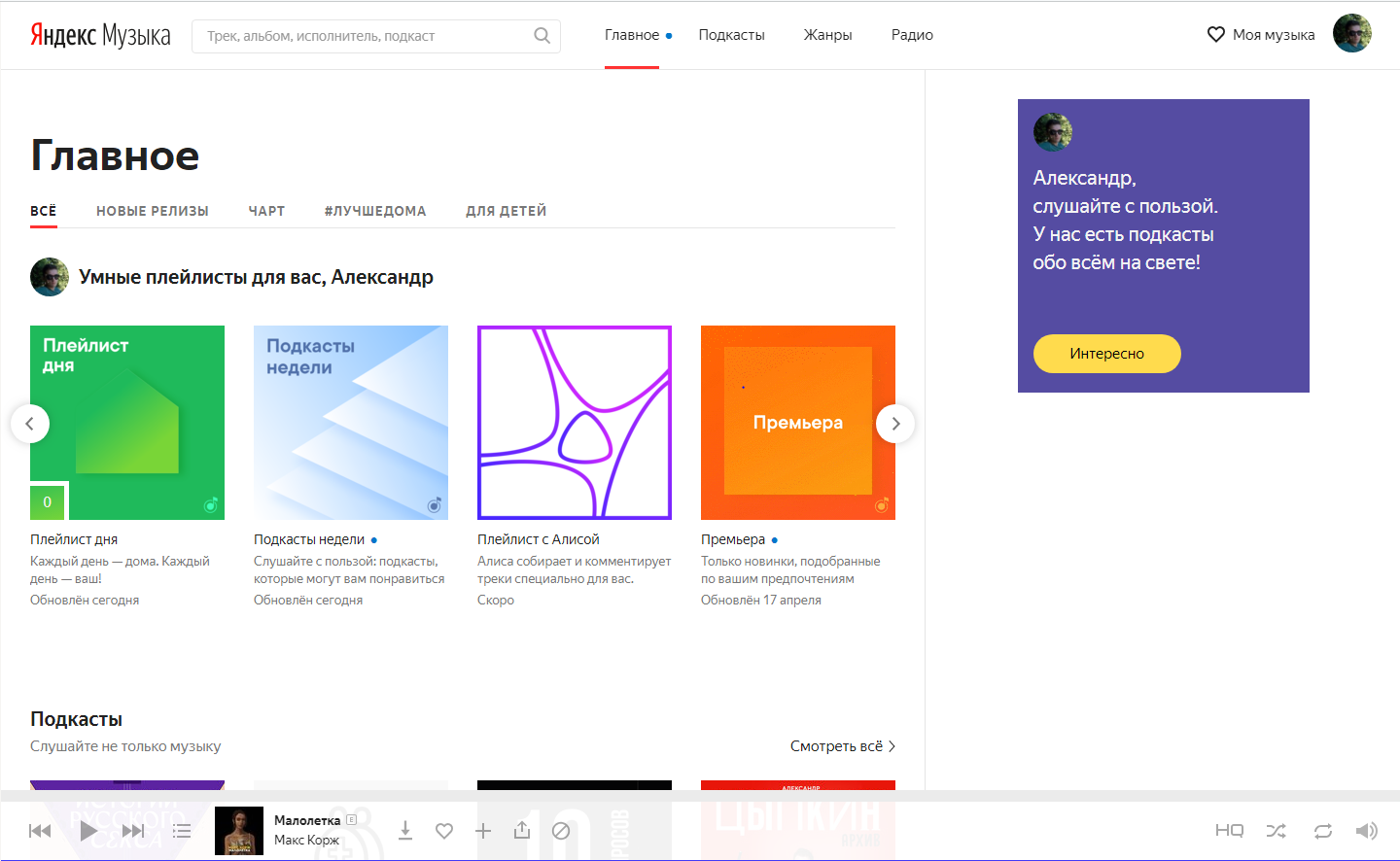
Сервис обладает огромным количеством плейлистов и подборок. Один из них Называется «Плейлист дня». Рекомендации из этого плейлиста обноляются каждый день.

Помимо «Плейлиста дня», куда периодически валятся уже прослушанные вами треки, в Яндекс.Музыке есть ещё два, которые обновляются реже, раз в неделю.

«Премьера» – это новые релизы известных вам исполнителей и новые жанры, которые могут понравиться. Здесь есть место экспериментам со стороны сервиса: например, он может подсунуть немного треков из того, что не соответствует вашим предпочтениям, но внезапно может «зайти».

А «Дежавю» предлагает послушать песни, очень похожие на то, что вы уже добавили в медиатеку. Это могут быть ранее не прослушанные треки любимых вами исполнителей или же совершенно других, но аналогичного жанра.

* **Преимущества:**возможность слушать музыку бесплатно, если вы пользуетесь только десктопным [браузером](https://lifehacker.ru/luchshie-brauzery-dlya-pk/), удобная система рекомендаций, наличие эксклюзивных подкастов.
* **Недостатки:**недостаточно удобный интерфейс и большое количество рекламы в браузерной версии.



**Рисунок 3 –** Главная страница Яндекс.Музыка

## Постановка задачи

Анализ существующих сайтов и приложений позволил определиться с постановкой задачи. Итогом разработки приложения должен стать онлайн сервис, предоставляющий доступ к прослушиванию и скачиванию музыки различных жанров и исполнителей. Для пользователей должна быть добавлена возможность создания своего аккаунта в системе. Пользователь, который вошел в систему через свой аккаунт, должен иметь возможность онлайн прослушивания музыки, покупки как одиночного трека, так и покупки треков через добавление в корзину покупателя. Купленный трек можно скачать на устройство. Пользователь, который не вошел в систему, имеет возможность только онлайн прослушивания музыки без ее сохранения. Также все пользователи должны иметь возможность искать музыку по жанрам, исполнителям и названию треков.

В системе обязательно должен быть пользователь, обладающий правами администратора, который может редактировать аудиозаписи и добавлять новые, может добавлять новых исполнителей и редактировать информации о них. Должна быть реализована работа с плейлистами и подборками.

# Проектирование системы

Приложение состоит из 3 компонентов: серверную часть (*Backend*), пользовательский интерфейс (*Frontend*) и бизнес-логика.

Взаимодействие *frontend* и *backend* происходит по кругу: *frontend* отправляет пользовательскую информацию в *backend*, там она обрабатывается и возвращается обратно, приняв понятную форму.

## Проектирование бизнес логики

Бизнес-логика – в разработке [информационных систем](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) – совокупность правил, принципов, зависимостей поведения объектов предметной области (области человеческой деятельности, которую система поддерживает). Иначе можно сказать, что бизнес-логика – это реализация правил и ограничений автоматизируемых операций. Является синонимом термина «логика предметной области» ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) domain logic).

Проще говоря, бизнес-логика – это реализация предметной области в [информационной системе](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0). К ней относятся, например, формулы расчёта ежемесячных выплат по ссудам (в финансовой индустрии), автоматизированная отправка сообщений электронной почты руководителю проекта по окончании выполнения частей задания всеми подчиненными (в [системах управления проектами](https://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B0%D0%BC%D0%B8)), отказ от отеля при отмене рейса авиакомпанией (в туристическом бизнесе) и т. д.

Для данного проекта наиболее важным этапом в проектировании бизнес-логики является разработка требований к сервису. Ниже приводятся основные функциональные требования, на основе которых будет строиться логика приложения:

* система должна разделять авторизованных и неавторизованных пользователей, а также отдельно выделять администратора системы
* система позволяет пользователю пройти регистрацию или авторизацию с помощью соответствующих форм
* система позволяет покупать аудиозаписи только авторизованным пользователям
* система позволяет добавлять аудиозаписи в корзину только авторизованным пользователям
* система должна проверять все водимые пользователям данные и корректно обрабатывать возникающие ошибки
* система должна корректно проводить все платежные операции
* при удалении из системы какого-либо исполнителя должны автоматически должны удалятся из системы все аудиозаписи и альбомы, связанные с данным исполнителем
* при удалении плейлистов связанные с ними аудиозаписи удалятся не должны
* при покупке отдельной аудиозаписи она должна удалятся из корзины в случае, если она там была
* в система должна быть предусмотрена возможность смены языка интерфейса: русский и английский

Логика приложения строится основе требований, на указанных выше, и вся функциональность реализует их. Основная логика взаимодействия пользователей (посетителей) с системой отображается на рисунке 2.1.

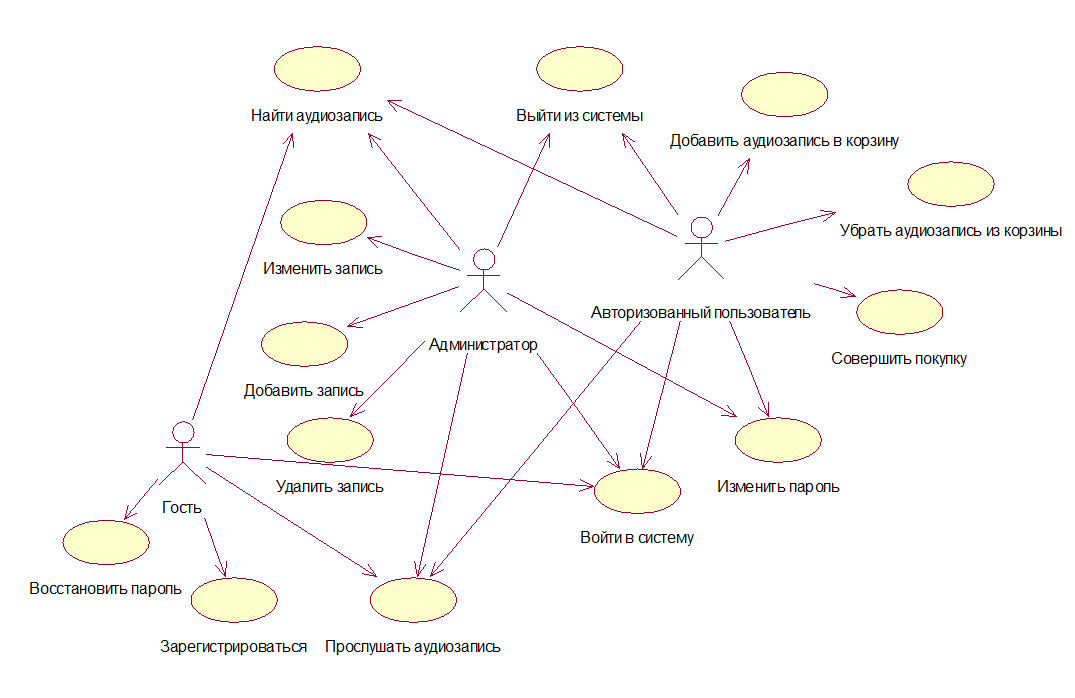


Рисунок 2.1 – Диаграмма вариантов использования

## Проектирование базы данных

Базой данных (БД) называется организованная в соответствии с определенными правилами и поддерживаемая в памяти компьютера совокупность сведений об объектах, процессах, событиях или явлениях, относящихся к некоторой предметной области, теме или задаче. Она организована таким образом, чтобы обеспечить информационные потребности пользователей, а также удобное хранение этой совокупности данных, как в целом, так и любой ее части [2].

Реляционная база данных представляет собой множество взаимосвязанных таблиц (сущностей), каждая из которых содержит информацию об объектах определенного вида. Каждая строка таблицы содержит данные об одном объекте, а столбцы таблицы содержат различные характеристики этих объектов - атрибуты.

В спроектированной базе данных будут храниться данные о пользователях, аудиозаписях, исполнителях, альбомах, плейлистах, а так же ролях пользователей.

Требуется соблюсти соответствие базы данных трём нормальным формам. Нормальная форма – требование, предъявляемое к структуре таблиц в теории реляционных баз данных для устранения из базы избыточных функциональных зависимостей между атрибутами (полями таблиц).

Отношение находится в 1НФ, если все его атрибуты являются простыми, все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице.

Отношение находится во 2НФ, если оно находится в 1НФ, и каждый не ключевой атрибут неприводимо зависит от Первичного Ключа (ПК).

Отношение находится в ЗНФ, когда находится во 2НФ, и каждый не ключевой атрибут не транзитивно зависит от первичного ключа. Проше говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы. Третья нормальная форма соблюдается за счёт создания таблиц с одеждой и поставщиками, чем мы исключаем возможность появления транзитивных связей.

Исходя из функций и задач системы и требований всех трех НФ, создадим следующие сущности базы данных:

* Пользователи (users)
* Исполнители (artists)
* Альбомы (albums)
* Аудиозаписи (tracks)
* Исполнители\_Аудиозаписи (users\_tracks)
* Жанры (genres)
* Плейлисты (playlists)
* Плейлисты\_Аудиозаписи (playlists\_tracks)
* Роли (roles)
* Привилегии (privileges)
* Роли\_Привилегии (roles\_privileges)
* Пользователи\_Роли (users\_roles)
* Корзины (carts)
* Ключи (tokens)

Теперь исходя из требований системы определим атрибуты сущностей:

Сущность «Пользователи»

* Уникальный идентификатор (id)
* Логин (login)
* Пароль (password)
* Имя пользователя (name)
* Фамилия пользователя (surname)
* Электронный адрес (email)
* Статус активности пользователя (active)

Сущность «Исполнители»

* Уникальный идентификатор (id)
* Имя исполнителя (name)
* Страна исполнителя (country)

Сущность «Альбомы»

* Уникальный идентификатор (id)
* Название альбома (title)
* Идентификатор исполнителя (artist\_id)

Сущность «Аудиозаписи»

* Уникальный идентификатор (id)
* Идентификатор исполнителя (artist\_id)
* Название трека (title)
* Идентификатор жанра (genre\_id)
* Стоимость (price)
* Идентификатор альбома (album\_id)
* Путь, указывающий на аудиофайл (url)

Сущность «Пользователи\_Аудиозаписи»

* Уникальный идентификатор (id)
* Идентификатор пользователя (user\_id)
* Идентификатор трека (track\_id)

Сущность «Жанры»

* Уникальный идентификатор (id)
* Название жанра (title)

Сущность «Плейлисты»

* Уникальный идентификатор (id)
* Название плейлиста (title)

Сущность «Плейлисты\_Аудиозаписи»

* Уникальный идентификатор (id)
* Идентификатор плейлиста (playlist\_id)
* Идентификатор трека (track\_id)

Сущность «Роли»

* Уникальный идентификатор (id)
* Название роли (title)

Сущность «Привилегии»

* Уникальный идентификатор (id)
* Название привилегии (title)

Сущность «Роли\_Привилегии»

* Уникальный идентификатор (id)
* Идентификатор роли (role\_id)
* Идентификатор привилегии (privilege\_id)

Сущность «Пользователи\_Роли»

* Уникальный идентификатор (id)
* Идентификатор пользователя (user\_id)
* Идентификатор роли (role\_id)

Сущность «Корзины»

* Уникальный идентификатор (id)
* Идентификатор пользователя (user\_id)
* Идентификатор трека (track\_id)

Сущность «Ключи»

* Уникальный идентификатор (id)
* Идентификатор пользователя (user\_id)
* Ключ (token)
* Дата истечения срока действительности ключа (expirationdate)

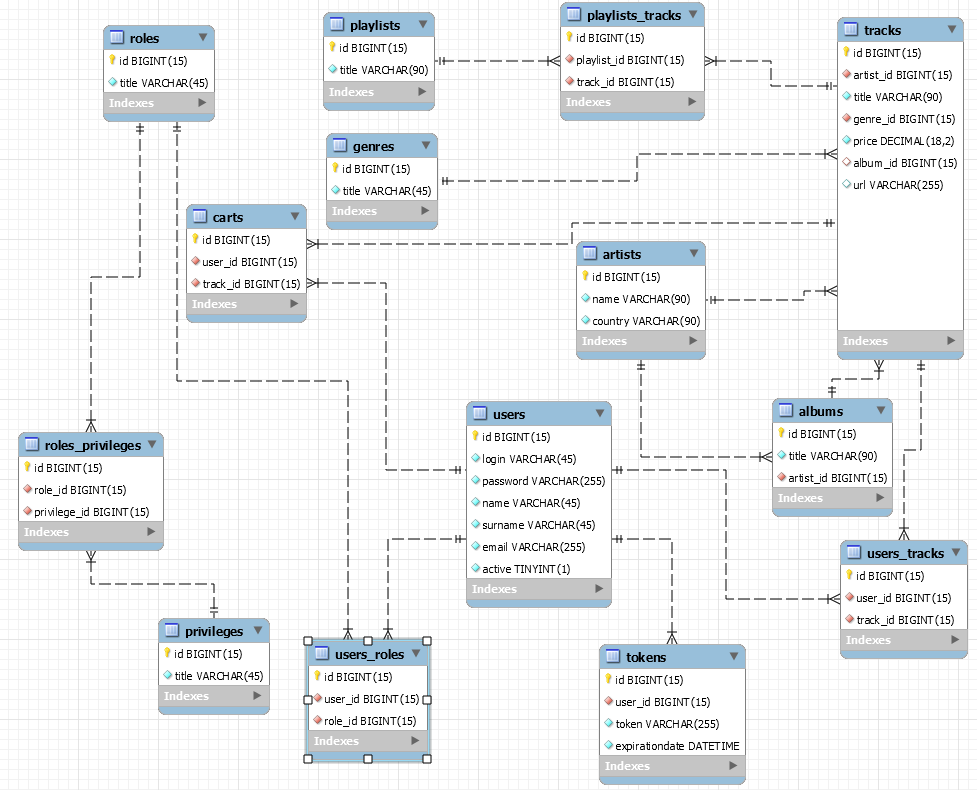


Рисунок 2.2 – Схема базы данных

## Проектирование пользовательского интерфейса

Интерфейсы существуют, чтобы люди могли взаимодействовать с нашим миром. Через интерфейс мы можем прояснить, проиллюстрировать, дать возможность, показать взаимосвязь, объединить людей или разделить, управлять ожиданиями и давать доступ к услугам.

Дизайн пользовательского интерфейса является фактором, оказывающим влияние на три основных показателя качества программного продукта: его функциональность, эстетику и производительность [4].

Функциональность является фактором, на который разработчики приложений зачастую обращают основное внимание. Они пытаются создавать программы так, чтобы пользователи могли выполнять свои задачи и им было удобно это делать. Функциональность важна, но, тем не менее, это не единственный показатель, который должен учитываться в ходе разработки.

Эстетичный внешний вид самого приложения и способа его представления (вплоть до упаковки) позволяет сформировать у потребителя положительное мнение о программе. Однако эстетические характеристики весьма субъективны и описать их количественно гораздо труднее, чем функциональные требования или показатели производительности. Вся эстетика приложения зачастую сводится к простому выбору: соотносятся ли между собой используемые цвета, передают ли элементы интерфейса их назначение и смысл представляемых операций, что ощущает человек при использовании тех или иных элементов управления и насколько успешно он их использует.

Производительность, а равно и надежность, также влияют на перспективу применения программы. Если приложение хорошо выглядит, имеет простое и удобное управление, но, к примеру, медленно прорисовывает экраны, регулярно «подвисает» на десяток-другой секунд или, того хуже, падает с критической ошибкой при некорректных действиях пользователя, у него, вероятно, будет мало шансов на длительную эксплуатацию. В свою очередь, быстрая и стабильная работа приложения могут отчасти компенсировать его не самый стильный дизайн или отсутствие каких-то вторичных функций.

# Описание процесса реализации

## Выбор СУБД

Несмотря на то, что все системы управления базами данных выполняют одну и ту же основную задачу (т.е. дают возможность пользователям создавать, редактировать и получать доступ к информации, хранящейся в базах данных), сам процесс выполнения этой задачи варьируется в широких пределах. Кроме того, функции и возможности каждой СУБД могут существенно отличаться. Различные СУБД документированы по-разному: более или менее тщательно. По-разному предоставляется и техническая поддержка.

При сравнении различных популярных баз данных, следует учитывать, удобна ли для пользователя и масштабируема ли данная конкретная СУБД, а также убедиться, что она будет хорошо интегрироваться с другими продуктами, которые уже используются. Кроме того, во время выбора следует принять во внимание стоимость системы и поддержки, предоставляемой разработчиком.

Сравниваются некоторые из популярных СУБД и на результате данного анализа выбирается подходящая для данного проекта.

## Microsoft SQL Server

Это система управления базами данных, движок которой работает на облачных серверах, а также локальных серверах, причем можно комбинировать типы применяемых серверов одновременно. Вскоре после выпуска Microsoft SQL Server 2016, Microsoft адаптировала продукт для операционной системы Linux, а на Windows-платформе он работал изначально. Одной из уникальных особенностей версии 2016 года является temporal data support (временная поддержка данных), которая позволяет отслеживать изменения данных с течением времени. Последняя версия Microsoft SQL-сервер поддерживает dynamic data masking (динамическую маскировку данных), которая гарантирует, что только авторизованные пользователи будут видеть конфиденциальные данные.

Достоинства:

* Продукт очень прост в использовании
* Текущая версия работает быстро и стабильно
* Движок предоставляет возможность регулировать и отслеживать уровни производительности, которые помогают снизить использование ресурсов.
* Вы сможете получить доступ к визуализации на мобильных устройствах.
* Он очень хорошо взаимодействует с другими продуктами Microsoft.

Недостатки:

* Цена для юридических лиц оказывается неприемлемой для большей части организаций.
* Даже при тщательной настройке производительности корпорация SQL Server способен занять все доступные ресурсы.
* Сообщается о проблемах с использованием службы интеграции для импорта файлов.
* Идеально подходит для: крупных организаций, которые уже используют ряд продуктов Microsoft.

## PostgreSQL

PostgreSQL является одним из нескольких бесплатных популярных вариантов СУБД, часто используется для ведения баз данных веб-сайтов. Это была одна из первых разработанных систем управления базами данных, поэтому в настоящее время она хорошо развита, и позволяет пользователям управлять как структурированными, так и неструктурированными данными. Может быть использован на большинстве основных платформ, включая Linux. Прекрасно справляется с задачами импорта информации из других типов баз данных с помощью собственного инструментария.

Движок БД может быть размещен в ряде сред, в том числе виртуальных, физических и облачных. Самая свежая версия, PostgreSQL 9.5, предлагает обработку больших объемов данных и увеличение числа одновременно работающих пользователей. Безопасность была улучшена благодаря поддержке DBMS\_SESSION.

Достоинства:

* Является масштабируемым и способен обрабатывать терабайты данных.
* Поддерживает формат json.
* Существует множество предопределенных функций.
* Доступен ряд интерфейсов.

Недостатки:

* Документация туманна, поэтому, возможно, ответы на некоторые вопросы придется искать в интернете.
* Конфигурация может смутить неподготовленного пользователя.
* Скорость работы может падать во время проведения пакетных операций или выполнения запросов чтения.

Идеально подходит для организаций с ограниченным бюджетом, но квалифицированными специалистами, когда требуется возможность выбрать свой интерфейс и использовать json.

## MongoDB

Бесплатная база данных, которая имеет коммерческую версию - MongoDB, она предназначена для приложений, которые используют как структурированные, так и неструктурированные данные. Ядро является очень гибким и работает при подключении базы данных к приложениям через драйверы MongoDB. Существует широкий выбор доступных драйверов, поэтому легко найти драйвер, который будет работать с требуемым языком программирования.

Поскольку изначально система MongoDB не была разработана для обработки моделей реляционных данных (хотя может это выполнять), могут возникнуть проблемы производительности, если вы попытаетесь использовать её таким образом. Однако, движок предназначен для обработки различных данных, которые нельзя отнести к реляционным, и может хорошо справляться там, где другие движки работают медленно или бессильны.

MongoDB 3.2 - это последняя версия, и она имеет новую подключаемую систему движков хранения. Документы могут быть проверены в процессе обновления или выполнения вставок, а функции текстового поиска были улучшены. Новая способность частичного индексирования может привести к более высокой производительности, уменьшая размер индексов.

Достоинства:

* Скорость и простота в использовании
* Движок поддерживает json и другие традиционные документы NoSQL.
* Данные любой структуры могут быть сохранены/прочитаны быстро и легко.

Недостатки:

* SQL не используется в качестве языка запросов.
* Инструменты для перевода SQL-запросов в MongoDB доступны, но их следует рассматривать именно как дополнение.
* Программа установки может занять много времени.
* Подходит для организаций, работающих с разнородными данными, которые тяжело поддаются классификации. Для внедрения потребуются высококлассные специалисты.

## MySQL

MySQL - одна из самых популярных баз данных для веб-приложений. Фактически, является стандартом de facto для веб-серверов, которые работают под управлением операционной системы Linux. MySQL - это бесплатный пакет программ, однако новые версии выходят постоянно, расширяя функционал и улучшая безопасность. Существуют специальные платные версии, предназначенные для коммерческого использования. В бесплатной версии наибольший упор делается на скорость и надежность, а не на полноту функционала, который может стать и достоинством и недостатком - в зависимости от области внедрения.

Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

Эта СУБД позволяет выбирать различные движки для системы хранения, которые позволяют менять функционал инструмента и выполнять обработку данных, хранящихся в различных типах таблиц. Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц. Она также имеет простой в использовании интерфейс, и пакетные команды, которые позволяют удобно обрабатывать огромные объемы данных. Система невероятно надежна и не стремится подчинить себе все доступные аппаратные ресурсы.

Достоинства:

* Распространяется бесплатно.
* Прекрасно документирована.
* Предлагает много функций, даже в бесплатной версии.
* Пакет MySQL включен в стандартные репозитории наиболее распространённых дистрибутивов операционной системы Linux, что позволяет устанавливать её элементарно.
* Поддерживает набор пользовательских интерфейсов.
* Может работать с другими базами данных, включая DB2 и Oracle.

Недостатки:

* Придётся потратить много времени и усилий, чтобы заставить MySQL выполнять несложные задачи, хотя другие системы делают это автоматически, например: создавать инкрементные резервные копии.
* Отсутствует встроенная поддержка XML или OLAP.
* Для бесплатной версии доступна только платная поддержка.
* Идеально подходит для: организаций, которым требуется надежный инструмент управления базами данных, но бесплатный.

База данных данного проекта не требует очень сложных операций над данными, необходим удобный рабочий интерфейс для работы с базой данных, проект предусматривает создание реляционной базы данных. Учитывая приведённый выше анализ различных СУБД и опыт работы с одной из СУБД разработчика системы, что так же является важным фактором, так как от этого зависит качество созданной базы данных и скорость её созданий, выбирается СУБД MySQL.

## Выбор языка программирования

Несмотря на то, что все языки программирования, направленные на работу с веб-технологиями, выполняют практически одинаковые функции, сам процесс выполнения этой задачи варьируется в широких пределах. Кроме того, функции и возможности каждого языка программирования могут существенно отличаться. Различные языки программирования документированы по-разному: более или менее тщательно. По-разному предоставляется и техническая поддержка.

При сравнении различных популярных языков программирования, следует учитывать стоимость системы и поддержки, предоставляемой разработчиком.

Сравниваются некоторые из популярных языков программирования и на результате данного анализа выбирается подходящая для данного проекта.

## Python

[Python](https://www.python.org/) широко применяется как интерпретируемый язык для скриптов различного назначения (хотя существуют и трансляторы языка Python).

Как и Ruby, Python имеет целью приблизить синтаксис реальной программы, написанной на нём, к описывающему задачу псевдокоду, что позволяет программисту уменьшить объём программы. Идея создания данного языка возникла в конце 1980-х и была реализована [Гвидо ван Россумом](http://www.linkedin.com/pub/guido-van-rossum/0/756/4a0).

Элегантный дизайн и эффективный, дисциплинирующий синтаксис этого языка облегчают программистам совместную работу над кодом. Python – мультипарадигмальный язык программирования: он позволяет совмещать процедурный подход к написанию кода с объектно-ориентированным и функциональным.

Достоинства:

* Открытая разработка;
* Довольно прост в изучении, особенно на начальном этапе;
* Особенности синтаксиса стимулируют программиста писать хорошо читаемый код;
* Предоставляет средства быстрого прототипирования и динамической семантики;
* Имеет большое сообщество, позитивно настроенное по отношению к новичкам;
* Множество полезных библиотек и расширений языка можно легко использовать в своих проектах благодаря предельно унифицированному механизму импорта и программным интерфейсам;
* Механизмы модульности хорошо продуманы и могут быть легко использованы;
* Абсолютно всё в Python является объектами в смысле ооп, но при этом объектный подход не навязывается программисту.

Недостатки:

* Не слишком удачная поддержка многопоточности;
* На Python создано не так уж много качественных программных проектов по сравнению с другими универсальными языками программирования, например, с Java;
* Отсутствие коммерческой поддержки средств разработки (хотя эта ситуация со временем меняется);
* Изначальная ограниченность средств для работы с базами данных;
* Бенчмарки показывают меньшую производительность Python по сравнению с основными Java VM, что создаёт этому языку репутацию медленного

## PHP

[PHP](http://www.php.net/) (Hypertext PreProcessor, препроцессор гипертекста) – язык программирования, исполняемый на стороне веб-сервера, спроектированный Расмусом Лердорфом ([Rasmus Lerdorf](http://www.linkedin.com/in/rlerdorf)) в качестве инструмента создания динамических и интерактивных веб-сайтов.

Этот язык оказался достаточно гибким и мощным, поэтому приобрёл большую популярность и используется в проектах любого масштаба: от простого блога до крупнейших веб-приложений в Интернете.

Достоинства:

* Является свободным программным обеспечением, распространяемым под особой лицензией (PHP license);
* Легок в освоении на всех этапах;
* Поддерживается большим сообществом пользователей и разработчиков;
* Имеет развитую поддержку баз данных;
* Имеется огромное количество библиотек и расширений языка;
* Может использоваться в изолированной среде;
* Предлагает нативные средства организации веб-сессий, программный интерфейс расширений;
* Является довольно полной заменой проприетарной среды ASP (Active Server Pages) от Microsoft;
* Может быть развёрнут почти на любом сервере;
* Портирован под большое количество аппаратных платформ и операционных систем.

Недостатки:

* Не подходит для создания десктопных приложений или системных компонентов;
* Имеет слабые средства для работы с исключениями;
* Глобальные параметры конфигурации влияют на базовый синтаксис языка, что затрудняет настройку сервера и разворачивание приложений;
* Объекты передаются по значению, что смущает многих программистов, привыкших к передаче объектов по ссылке, как это делается в большинстве других языков;
* Веб-приложения, написанные на php, зачастую имеют проблемы с безопасностью.

## Java

Java  – это язык программирования общего назначения, который следует парадигме объектно-ориентированного программирования и подходу «Написать один раз и использовать везде». Java используется для настольных, сетевых, мобильных и корпоративных приложений.

Java  –  это не только язык программирования, но и экосистема инструментов, охватывающая почти все, что может понадобиться при программировании на Java.

Достоинтсва:

* Объектно-ориентированное программирование;
* Язык высокого уровня с простым синтаксисом и плавной кривой обучения;
* Стандарт для корпоративных вычислительных систем;
* Безопасность;
* Независимость от платформы;
* Язык для распределенного программирования и комфортной удаленной совместной работы;
* Автоматическое управление памятью;
* Многопоточность;

Недостатки:

* Платное коммерческое использование;
* Низкая производительность;
* Отсутствие родного дизайна;
* Многословный и сложный код.

Функционал данного проекта будет реализован на языке программирования Java. Бизнес логика будет реализована с помощью фреймворка Spring. Уровень представления, или графическая часть, будет написана с помощью технологии JSP. Ниже приведены описания данных технологий, и их преимущества.

Spring Framework (или коротко Spring) – универсальный [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) [с открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) для [Java](https://ru.wikipedia.org/wiki/Java)-платформы. Spring обеспечивает решения многих задач, с которыми сталкиваются Java-разработчики и организации, которые хотят создать информационную систему, основанную на платформе [Java](https://ru.wikipedia.org/wiki/Java). Из-за широкой функциональности трудно определить наиболее значимые структурные элементы, из которых он состоит. Spring не всецело связан с платформой [Java Enterprise](https://ru.wikipedia.org/wiki/J2EE), несмотря на его масштабную интеграцию с ней, что является важной причиной его популярности.

JSP (JavaServer Pages) — технология, позволяющая веб-разработчикам создавать содержимое, которое имеет как статические, так и динамические компоненты. JSP-страницы могут содержать код, написанный на Java.

Код JSP-страницы транслируется в Java-код сервлета с помощью компилятора JSP-страниц [Jasper](https://ru.wikipedia.org/wiki/Apache_Tomcat#Jasper), и затем компилируется в [байт-код](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4) виртуальной машины [Java](https://ru.wikipedia.org/wiki/Java) ([JVM](https://ru.wikipedia.org/wiki/Java_Virtual_Machine)). [Контейнеры сервлетов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D0%B5%D0%B9%D0%BD%D0%B5%D1%80_%D1%81%D0%B5%D1%80%D0%B2%D0%BB%D0%B5%D1%82%D0%BE%D0%B2), способные исполнять JSP-страницы, написаны на платформенно-независимом языке Java. JSP-страницы загружаются на сервере и управляются из структуры специального Java server packet, который называется [Jakarta EE](https://ru.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition) Web Application. Обычно страницы упакованы в файловые архивы [.war](https://ru.wikipedia.org/wiki/WAR_(%D1%82%D0%B8%D0%BF_%D1%84%D0%B0%D0%B9%D0%BB%D0%B0)) и [.ear](https://ru.wikipedia.org/wiki/EAR).

Технология JSP является платформенно-независимой, переносимой и легко расширяемой технологией для разработки веб-приложений.

Также будут использоваться следующие технологии:

* Maven;
* Hibernate;
* Lombok;
* Log4j

## Реализация ресурса

Для реализации проекта сначала необходимо создать базу данных, чтобы прописывать логику приложения по работе с данными.

Код создания таблиц базы данных представлен ниже:

Таблица сущности «Пользователи»

*CREATE TABLE `users` (*

*`id` bigint(15) NOT NULL AUTO\_INCREMENT,*

*`login` varchar(45) NOT NULL,*

*`password` varchar(255) NOT NULL,*

*`name` varchar(45) NOT NULL,*

*`surname` varchar(45) NOT NULL,*

*`email` varchar(255) NOT NULL,*

*`active` tinyint(1) NOT NULL,*

*PRIMARY KEY (`id`),*

*UNIQUE KEY `login\_UNIQUE` (`login`),*

*UNIQUE KEY `email\_UNIQUE` (`email`)*

*)*

Таблица сущности «Исполнители»

*CREATE TABLE `artists` (*

*`id` bigint(15) NOT NULL AUTO\_INCREMENT,*

*`name` varchar(90) NOT NULL,*

*`country` varchar(90) NOT NULL,*

*PRIMARY KEY (`id`)*

*)*

Таблица сущности «Альбомы»

*CREATE TABLE `albums` (*

*`id` bigint(15) NOT NULL AUTO\_INCREMENT,*

*`title` varchar(90) NOT NULL,*

*`artist\_id` bigint(15) NOT NULL,*

*PRIMARY KEY (`id`),*

*KEY `fk\_album\_to\_artist\_idx` (`artist\_id`),*

*CONSTRAINT `fk\_albums\_artists` FOREIGN KEY (`artist\_id`) REFERENCES `artists` (`id`) ON DELETE CASCADE ON UPDATE CASCADE*

*)*

Таблица сущности «Аудиозаписи»

*CREATE TABLE `tracks` (*

*`id` bigint(15) NOT NULL AUTO\_INCREMENT,*

*`artist\_id` bigint(15) NOT NULL,*

*`title` varchar(90) NOT NULL,*

*`genre\_id` bigint(15) NOT NULL,*

*`price` decimal(18,2) NOT NULL DEFAULT '0.00',*

*`album\_id` bigint(15) DEFAULT NULL,*

*`url` varchar(255) DEFAULT NULL,*

*PRIMARY KEY (`id`),*

*KEY `fk\_tracks\_artists\_idx` (`artist\_id`),*

*KEY `fk\_tracks\_genres\_idx` (`genre\_id`),*

*KEY `fk\_tracks\_albums\_idx` (`album\_id`),*

*CONSTRAINT `fk\_tracks\_albums` FOREIGN KEY (`album\_id`) REFERENCES `albums` (`id`) ON DELETE SET NULL ON UPDATE CASCADE,*

*CONSTRAINT `fk\_tracks\_artists` FOREIGN KEY (`artist\_id`) REFERENCES `artists` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,*

*CONSTRAINT `fk\_tracks\_genres` FOREIGN KEY (`genre\_id`) REFERENCES `genres` (`id`) ON DELETE RESTRICT ON UPDATE CASCADE*

*)*

Таблица сущности «Пользователи\_Аудиозаписи»

*CREATE TABLE `users\_tracks` (*

*`id` bigint(15) NOT NULL AUTO\_INCREMENT,*

*`user\_id` bigint(15) NOT NULL,*

*`track\_id` bigint(15) NOT NULL,*

*PRIMARY KEY (`id`),*

*KEY `fk\_users\_tracks\_users\_idx` (`user\_id`),*

*KEY `fk\_users\_tracks\_tracks\_idx` (`track\_id`),*

*CONSTRAINT `fk\_users\_tracks\_tracks` FOREIGN KEY (`track\_id`) REFERENCES `tracks` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,*

*CONSTRAINT `fk\_users\_tracks\_users` FOREIGN KEY (`user\_id`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE*

*)*

Таблица сущности «Жанры»

*CREATE TABLE `genres` (*

*`id` bigint(15) NOT NULL AUTO\_INCREMENT,*

*`title` varchar(45) NOT NULL,*

*PRIMARY KEY (`id`),*

*UNIQUE KEY `title\_UNIQUE` (`title`)*

*)*

Таблица сущности «Плейлисты»

*CREATE TABLE `playlists` (*

*`id` bigint(15) NOT NULL AUTO\_INCREMENT,*

*`title` varchar(90) NOT NULL,*

*PRIMARY KEY (`id`)*

*)*

Таблица сущности «Плейлисты\_Аудиозаписи»

*CREATE TABLE `playlists\_tracks` (*

*`id` bigint(15) NOT NULL AUTO\_INCREMENT,*

*`playlist\_id` bigint(15) NOT NULL,*

*`track\_id` bigint(15) NOT NULL,*

*PRIMARY KEY (`id`),*

*KEY `fk\_tracks\_to\_playlists\_tracks\_idx` (`track\_id`),*

*KEY `fk\_playlists\_tracks\_playlists\_idx` (`playlist\_id`),*

*CONSTRAINT `fk\_playlists\_tracks\_playlists` FOREIGN KEY (`playlist\_id`) REFERENCES `playlists` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,*

*CONSTRAINT `fk\_playlists\_tracks\_tracks` FOREIGN KEY (`track\_id`) REFERENCES `tracks` (`id`) ON DELETE CASCADE ON UPDATE CASCADE*

*)*

Таблица сущность «Роли»

*CREATE TABLE `roles` (*

*`id` bigint(15) NOT NULL AUTO\_INCREMENT,*

*`title` varchar(45) NOT NULL,*

*PRIMARY KEY (`id`),*

*UNIQUE KEY `title\_UNIQUE` (`title`)*

*)*

Таблица сущности «Привилегии»

*CREATE TABLE `privileges` (*

*`id` bigint(15) NOT NULL AUTO\_INCREMENT,*

*`title` varchar(45) NOT NULL,*

*PRIMARY KEY (`id`),*

*UNIQUE KEY `title\_UNIQUE` (`title`)*

*)*

Таблица сущности «Роли\_Привилегии»

*CREATE TABLE `roles\_privileges` (*

*`id` bigint(15) NOT NULL AUTO\_INCREMENT,*

*`role\_id` bigint(15) NOT NULL,*

*`privilege\_id` bigint(15) NOT NULL,*

*PRIMARY KEY (`id`),*

*KEY `fk\_roles\_privileges\_privileges\_idx` (`privilege\_id`),*

*KEY `fk\_roles\_privileges\_roles\_idx` (`role\_id`),*

*CONSTRAINT `fk\_roles\_privileges\_privileges` FOREIGN KEY (`privilege\_id`) REFERENCES `privileges` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,*

*CONSTRAINT `fk\_roles\_privileges\_roles` FOREIGN KEY (`role\_id`) REFERENCES `roles` (`id`) ON DELETE CASCADE ON UPDATE CASCADE*

*)*

Таблица сущности «Пользователи\_Роли»

*CREATE TABLE `users\_roles` (*

*`id` bigint(15) NOT NULL AUTO\_INCREMENT,*

*`user\_id` bigint(15) NOT NULL,*

*`role\_id` bigint(15) NOT NULL,*

*PRIMARY KEY (`id`),*

*KEY `fk\_users\_roles\_users\_idx` (`user\_id`),*

*KEY `fk\_users\_roles\_roles\_idx` (`role\_id`),*

*CONSTRAINT `fk\_users\_roles\_roles` FOREIGN KEY (`role\_id`) REFERENCES `roles` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,*

*CONSTRAINT `fk\_users\_roles\_users` FOREIGN KEY (`user\_id`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE*

*)*

Таблица сущности «Корзины»

*CREATE TABLE `carts` (*

*`id` bigint(15) NOT NULL AUTO\_INCREMENT,*

*`user\_id` bigint(15) NOT NULL,*

*`track\_id` bigint(15) NOT NULL,*

*PRIMARY KEY (`id`),*

*KEY `fk\_carts\_users` (`user\_id`),*

*KEY `fk\_carts\_tracks\_idx` (`track\_id`),*

*CONSTRAINT `fk\_carts\_tracks` FOREIGN KEY (`track\_id`) REFERENCES `tracks` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,*

*CONSTRAINT `fk\_carts\_users` FOREIGN KEY (`user\_id`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE*

*)*

Таблица сущности «Ключи»

*CREATE TABLE `tokens` (*

*`id` bigint(15) NOT NULL AUTO\_INCREMENT,*

*`user\_id` bigint(15) NOT NULL,*

*`token` varchar(255) NOT NULL,*

*`expirationdate` datetime NOT NULL,*

*PRIMARY KEY (`id`),*

*KEY `fk\_tokens\_users\_idx` (`user\_id`),*

*CONSTRAINT `fk\_tokens\_users` FOREIGN KEY (`user\_id`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE*

*)*

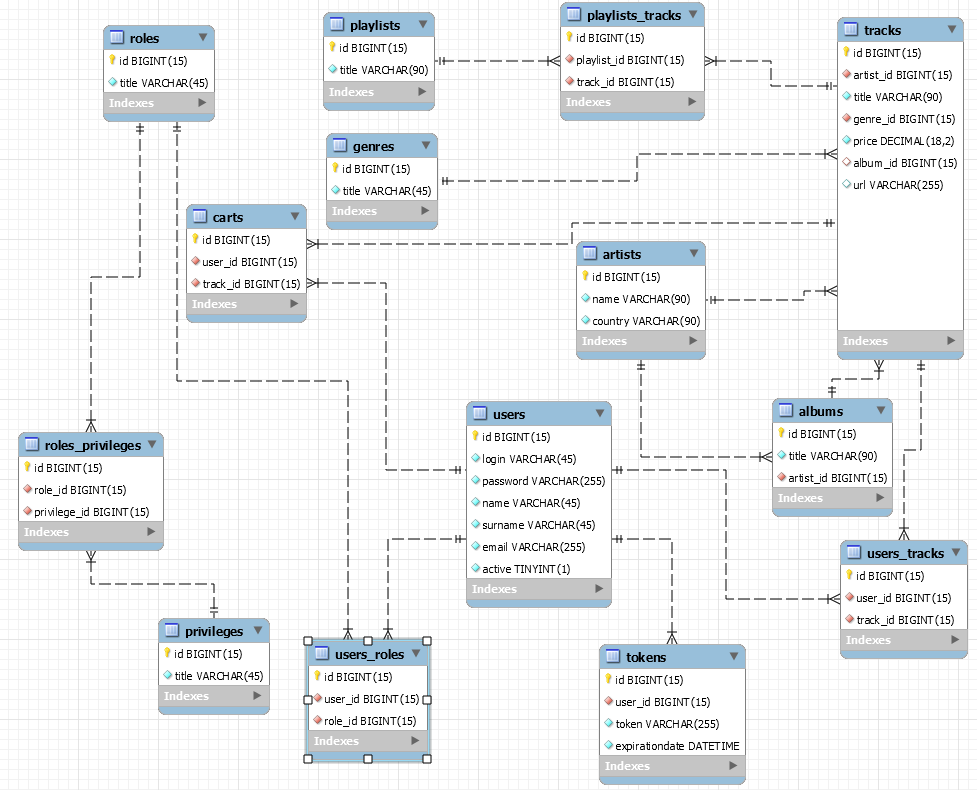
**

Рисунок 3.1 – ER-диаграмма базы данных

База данных готова, можно приступать к созданию самого приложения.

Структура проекта будет соответствовать шаблону проектирования MVC (Model,-View- Controller). В нашем случае Model – это наши сущности (Entity) классы. View – это страницы JSP. Controller – это наша логика, отвечающая на действия пользователя на нашей View.

Проект разобьем на несколько пакетов, которые будут содержать классы.

Структура пакетов проекта представлена на рисунке 3.2.

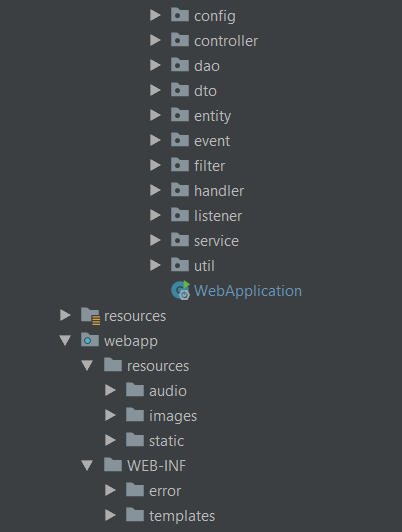


Рисунок 3.2 – Структура пакетов проекта

Дадим краткое описание каждому пакету:

* Config – данный пакет содержит все классы-конфигурации проекта, необходимые для настройки Spring.
* Controller – данный пакет содержит классы-контроллеры, необходимые для обработки HTTP запросов от клиента.
* Dao (Data Access Object) – пакет, содержащий классы, предназначенные для выполнения запросов к базе данных.
* Service – содержит классы, предназначенные для обращения к слою DAO из классов-контроллеров и выполняющие основную бизнес логику приложения.
* Dto (Data Transfer Object) – данный пакет содержит классы почти идентичные классам Entity. Эти классы реализуют шаблон проектирования Data Transfer Object и предназначены для передачи данных об объектах между подсистемами приложения.
* Entity – содержит классы сущности
* Event – содержит классы-события. Spring позволяет объявлять различные события внутри системы. Данные события обрабатываются классами-слушателями.
* Listener – содержит классы-слушатели. Данные классы реагируют на возникающие в системе события и затем обрабатывают их.
* Handler – содержит классы-обработчики. Они также реагируют на различные действия системы.
* Filter – содержит классы-фильтры. Данные фильтры перехватывают HTTP-запрос до того, как он попадет в начнет обрабатываться сервлетом. Данные классы могут изменять запрос, а также принимать решение о том, нужно ли отправлять запрос дальше на обработку.
* Util – содержит классы-утилиты.
* Resources – данный пакет содержит различные файлы конфигурации, файлы сообщений, статические данные, изображения, аудиозаписи и т.д.
* Templates – данный пакет представляет собой уровень View шаблона MVC. В нем содержаться все страницы JSP, представляющие пользовательский интерфейс.

Приступим к рассмотрению основных классов приложения.

Иерархия классов-сущностей представлена на рисунке 3.3.

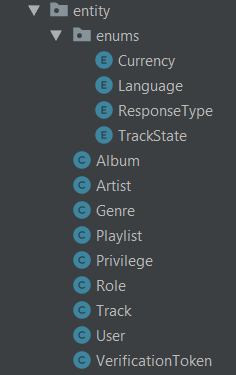


Рисунок 3.3 – Иерархия классов-сущностей

Реализация данных классов представлена ниже:

**Класс *User***

1. @Entity
2. @Getter
3. @Setter
4. @Table(name = "users")
5. **public** **class** User **implements** Serializable {
6. @Id
7. @GeneratedValue(strategy = GenerationType.IDENTITY)
8. **private** Long id;
10. **private** String login;
12. **private** String password;
14. **private** String name;
16. **private** String surname;
18. **private** String email;
20. @Column(name = "active")
21. **private** **boolean** isActive;
23. @ManyToMany
24. @JoinTable(
25. name = "users\_tracks",
26. joinColumns = @JoinColumn(
27. name = "user\_id", referencedColumnName = "id"),
28. inverseJoinColumns = @JoinColumn(
29. name = "track\_id", referencedColumnName = "id"))
30. **private** List<Track> tracks;

33. @ManyToMany
34. @JoinTable(
35. name = "carts",
36. joinColumns = @JoinColumn(
37. name = "user\_id", referencedColumnName = "id"),
38. inverseJoinColumns = @JoinColumn(
39. name = "track\_id", referencedColumnName = "id"))
40. **private** List<Track> cart;
42. @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
43. @JoinTable(
44. name = "users\_roles",
45. joinColumns = @JoinColumn(
46. name = "user\_id", referencedColumnName = "id"),
47. inverseJoinColumns = @JoinColumn(
48. name = "role\_id", referencedColumnName = "id"))
49. **private** List<Role> roles;
51. **public** User() {
52. }
54. **public** BigDecimal getTotalCartPrice() {
55. BigDecimal totalPrice = **new** BigDecimal(0);
56. **if** (cart != **null**) {
57. **for** (Track track : cart) {
58. totalPrice = totalPrice.add(track.getPrice());
59. }
60. }
61. **return** totalPrice;
62. }
64. }

**Класс *Artist***

1. @Entity
2. @Getter
3. @Setter
4. @Table(name="artists")
5. **public** **class** Artist **implements** Comparable<Artist>, Serializable {
7. @Id
8. @GeneratedValue(strategy = GenerationType.IDENTITY)
9. **private** Long id;
11. **private** String name;
13. **private** String country;
15. @OneToMany(mappedBy = "artist")
16. **private** List<Album> albums;
18. @OneToMany(mappedBy = "artist")
19. **private** List<Track> tracks;
21. **public** Artist(String name, String country) {
22. **this**.name = name;
23. **this**.country = country;
24. }
26. **public** Artist() {
27. }
29. @Override
30. **public** **int** compareTo(Artist o) {
31. **return** **this**.name.compareTo(o.name);
32. }
33. }

**Класс *Album***

1. @Entity
2. @Getter
3. @Setter
4. @Table(name = "albums")
5. **public** **class** Album **implements** Serializable {
7. @Id
8. @GeneratedValue(strategy = GenerationType.IDENTITY)
9. **private** Long id;
11. **private** String title;
13. @ManyToOne
14. @JoinColumn(name = "artist\_id")
15. **private** Artist artist;
17. @OneToMany(mappedBy = "album")
18. **private** List<Track> tracks;
20. **public** Album(String title, Artist artist) {
21. **this**.title = title;
22. **this**.artist = artist;
23. }
25. **public** Album() {
26. }
27. }

**Класс *Track***

1. @Entity
2. @Getter
3. @Setter
4. @Table(name = "tracks")
5. **public** **class** Track **implements** Comparable<Track>, Serializable {
7. @Id
8. @GeneratedValue(strategy = GenerationType.IDENTITY)
9. **private** Long id;
11. @ManyToOne
12. @JoinColumn(name = "artist\_id")
13. **private** Artist artist;
15. **private** String title;
17. @ManyToOne
18. @JoinColumn(name = "genre\_id")
19. **private** Genre genre;
21. @ManyToOne
22. @JoinColumn(name = "album\_id")
23. **private** Album album;
25. **private** BigDecimal price;
27. **private** String url;
29. @Transient
30. **private** **transient** TrackState state = TrackState.IN\_STORE;
32. @Transient
33. **private** MultipartFile file;
35. @Transient
36. **private** String fileName;
38. **public** Track(Artist artist, String title, Genre genre, BigDecimal price) {
39. **this**.artist = artist;
40. **this**.title = title;
41. **this**.genre = genre;
42. **this**.price = price;
43. }
45. **public** Track() {
46. }
48. @Override
49. **public** **int** compareTo(Track o) {
50. **int** result = **this**.artist.getName().compareTo(o.artist.getName());
51. **if** (result == 0) {
52. result = **this**.title.compareTo(o.title);
53. }
54. **return** result;
55. }

58. }

**Класс *Genre***

1. @Entity
2. @Getter
3. @Setter
4. @Table(name="genres")
5. **public** **class** Genre **implements** Serializable {
7. @Id
8. @GeneratedValue(strategy = GenerationType.IDENTITY)
9. **private** Long id;
11. **private** String title;
13. @OneToMany(mappedBy = "genre")
14. **private** List<Track> tracks;
16. **public** Genre(String title) {
17. **this**.title = title;
18. }
20. **public** Genre() {
21. }
22. }

**Класс *Playlist***

1. @Entity
2. @Getter
3. @Setter
4. @Table(name = "playlists")
5. **public** **class** Playlist **implements** Serializable {
7. @Id
8. @GeneratedValue(strategy = GenerationType.IDENTITY)
9. **private** Long id;
11. **private** String title;
13. @JsonBackReference
14. @ManyToMany
15. @JoinTable(
16. name = "playlists\_tracks",
17. joinColumns = @JoinColumn(
18. name = "playlist\_id", referencedColumnName = "id"),
19. inverseJoinColumns = @JoinColumn(
20. name = "track\_id", referencedColumnName = "id"))
21. **private** List<Track> tracks;
23. **public** Playlist(String title) {
24. **this**.title = title;
25. }
27. **public** Playlist() {
28. }
29. }

**Класс *Role***

1. @Entity
2. @Getter
3. @Setter
4. @Table(name = "roles")
5. **public** **class** Role **implements** Serializable {
7. @Id
8. @GeneratedValue(strategy = GenerationType.IDENTITY)
9. **private** Long id;
11. **private** String title;
13. @ManyToMany(fetch = FetchType.EAGER)
14. @JoinTable(
15. name = "roles\_privileges",
16. joinColumns = @JoinColumn(
17. name = "role\_id", referencedColumnName = "id"),
18. inverseJoinColumns = @JoinColumn(
19. name = "privilege\_id", referencedColumnName = "id"))
20. **private** List<Privilege> privileges;
22. **public** Role() {
23. }
24. }

**Класс *Privilege***

1. @Entity
2. @Getter
3. @Setter
4. @Table(name = "privileges")
5. **public** **class** Privilege **implements** Serializable {
7. @Id
8. @GeneratedValue(strategy = GenerationType.IDENTITY)
9. **private** Long id;
11. **private** String title;
13. **public** Privilege(String title) {
14. **this**.title = title;
15. }
17. **public** Privilege() {
18. }
19. }

**Класс *VerificationToken***

1. @Entity
2. @Getter
3. @Setter
4. @Table(name = "tokens")
5. **public** **class** VerificationToken {
7. @Id
8. @GeneratedValue(strategy = GenerationType.IDENTITY)
9. **private** Long id;
11. @Column(name = "token")
12. **private** String value;
14. @Transient
15. **private** **int** minutes;
17. @Column(name = "expirationdate")
18. **private** Date expirationDate;
20. @OneToOne(fetch = FetchType.EAGER)
21. @JoinColumn(name = "user\_id")
22. **private** User user;
24. **public** VerificationToken(String value, User user) {
25. **this**.value = value;
26. **this**.user = user;
27. }
29. **public** VerificationToken() {
30. }
32. **public** **void** calculateExpiryDate(**int** expiryTimeInMinutes) {
33. Calendar calendar = Calendar.getInstance();
34. calendar.add(Calendar.MINUTE, expiryTimeInMinutes);
35. expirationDate = **new** Date(calendar.getTime().getTime());
36. }
37. }

Все классы-сущности представляют собой соответствующую таблицу в базе данных. Все поля классов соответствуют атрибутам таблиц.

Для того, чтобы связать таблицы базы данных с классами сущностями, используется фреймворк *Hibernate.* Он позволяет отображать данные из базы данных в виде объектов классов-сущностей. Для этого необходимо пометить класс аннотацией *@Entity,* а также аннотацией *@Table*, указывающей соответствующую таблицу в базе данных. Таким образом, при выполнении запросов к базе будут создаваться объекты соответствующих классов-сущностей, а их поля будут инициализироваться данными из таблицы.

Также *Hibernate* предоставляет возможность объявить в родительской сущности другую дочернюю сущность, которая связана с первой сущностью по внешнему ключу. Таким образом, это позволяет сократить количество явных обращений к базе данных. Для отношения между сущностями «один ко многим» необходимо помечать поля аннотациями *@OneToMany* или *@ManyToOne.* Для отношения «многие ко многим» - *@ManyToMany.*

Далее рассмотрим реализацию классов DAO.

Все запросы к базе данных буду генерироваться с помощью класса *EntityManager*. Этот класс является основным интерфейсом спецификации *JPA. Hibernate* реализует эту спецификацию.

Иерархия классов *DAO* представлена на рисунке 3.4.

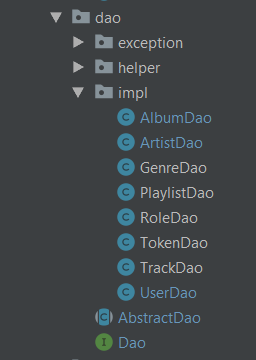


Рисунок 3.4 – Иерархия классов *DAO*

Реализация классов *DAO* представлена ниже.

**Класс *AbstractDao***

1. **public** **abstract** **class** AbstractDao<T> **implements** Dao<T> {
3. @PersistenceContext
4. **protected** EntityManager entityManager;
6. **private** **static** **final** Logger LOGGER = Logger.getLogger(AbstractDao.**class**);
8. **public** AbstractDao() {
9. }
11. @Override
12. **public** **void** save(T t) {
13. entityManager.persist(t);
14. }
16. @Override
17. **public** **void** update(T t) {
18. entityManager.merge(t);
19. }
21. @Override
22. **public** **void** remove(T t) {
23. entityManager.remove(t);
24. }
26. **public** T findById(**long** id) **throws** DaoException {
27. **return** entityManager.find(getClassObject(), id);
28. }
30. @Override
31. **public** List<T> findAll() **throws** DaoException {
32. **return** (List<T>) entityManager.createQuery("FROM " + getTableName()).getResultList();
33. }
35. **protected** **abstract** String getTableName();
37. **protected** **abstract** Class<T> getClassObject();
39. }

Данный класс генерализирован и выполняет для всех сущностей основные *CRUD* операции. В остальных классах будут реализованы специфические операции для сущностей.

**Класс *TrackDao***

1. Repository
2. **public** **class** TrackDao **extends** AbstractDao<Track> {
3. **private** **static** **final** String TABLE\_NAME = "Track";
5. **public** TrackDao() {
7. }
9. **public** List<Track> findByArtistAndTitle(String artist, String title) {
10. CriteriaBuilder cb = entityManager.getCriteriaBuilder();
11. CriteriaQuery<Track> criteriaQuery = cb.createQuery(getClassObject());
12. Root<Track> trackRoot = criteriaQuery.from(getClassObject());
13. criteriaQuery.select(trackRoot);
14. **if** (artist != **null** && !artist.isEmpty()) {
15. criteriaQuery.where(cb.like(trackRoot.join("artist").get("name"), "%" + artist + "%"));
16. }
17. **if**(title!=**null**&&!title.isEmpty()) {
18. criteriaQuery.where(cb.like(trackRoot.get("title"),"%"+title+"%"));
19. }
20. **return** entityManager.createQuery(criteriaQuery).getResultList();
21. }
23. @Override
24. **protected** String getTableName() {
25. **return** TABLE\_NAME;
26. }
28. @Override
29. **protected** Class<Track> getClassObject() {
30. **return** Track.**class**;
31. }
32. }

**Класс *UserDao***

1. @Repository
2. **public** **class** UserDao **extends** AbstractDao<User> {
3. **private** **static** **final** String TABLE\_NAME = "User";
5. **private** **static** **final** Logger LOGGER = Logger.getLogger(UserDao.**class**);
7. @PersistenceContext
8. **private** EntityManager entityManager;
10. **public** UserDao() {
11. }
13. **public** User findByLogin(String login) {
14. **return** (User) entityManager.createQuery("from User u where u.login=:login")
15. .setParameter("login", login)
16. .getSingleResult();
17. }
19. **public** User findByEmail(String email) {
20. **return** (User) entityManager.createQuery("from User u where u.email=:email")
21. .setParameter("email", email)
22. .getSingleResult();
23. }
25. @Override
26. **protected** String getTableName() {
27. **return** TABLE\_NAME;
28. }
30. @Override
31. **protected** Class<User> getClassObject() {
32. **return** User.**class**;
33. }
34. }

**Класс *ArtistDao***

1. @Repository
2. **public** **class** ArtistDao **extends** AbstractDao<Artist> {
4. **private** **static** **final** String TABLE\_NAME = "Artist";
6. **public** ArtistDao() {
8. }
10. **public** List<Artist> findByName(String name) **throws** DaoException {
11. CriteriaBuilder cb = entityManager.getCriteriaBuilder();
12. CriteriaQuery<Artist> criteriaQuery = cb.createQuery(getClassObject());
13. Root<Artist> artistRoot = criteriaQuery.from(getClassObject());
14. criteriaQuery.select(artistRoot);
15. criteriaQuery.where(cb.like(artistRoot.get("name"), "%" + name + "%"));
16. **return** entityManager.createQuery(criteriaQuery).getResultList();
17. }
19. @Override
20. **protected** String getTableName() {
21. **return** TABLE\_NAME;
22. }
24. @Override
25. **protected** Class<Artist> getClassObject() {
26. **return** Artist.**class**;
27. }
28. }

**Класс *AlbumDao***

1. @Repository
2. **public** **class** AlbumDao **extends** AbstractDao<Album> {
4. **private** **static** **final** String TABLE\_NAME = "albums";
6. @Override
7. **protected** String getTableName() {
8. **return** TABLE\_NAME;
9. }
11. @Override
12. **protected** Class<Album> getClassObject() {
13. **return** Album.**class**;
14. }
15. }

**Класс *PlaylistDao***

1. @Repository
2. **public** **class** PlaylistDao **extends** AbstractDao<Playlist> {
4. **private** **static** **final** String TABLE\_NAME = "Playlist";
6. **public** PlaylistDao() {
7. }
9. @Override
10. **protected** String getTableName() {
11. **return** TABLE\_NAME;
12. }
14. @Override
15. **protected** Class<Playlist> getClassObject() {
16. **return** Playlist.**class**;
17. }
18. }

**Класс *GenreDao***

1. @Repository
2. **public** **class** GenreDao **extends** AbstractDao<Genre> {
4. @Override
5. **protected** String getTableName() {
6. **return** "Genre";
7. }
9. @Override
10. **protected** Class<Genre> getClassObject() {
11. **return** Genre.**class**;
12. }
14. **public** Genre findByTitle(String genre) {
15. Query query = entityManager.createQuery("FROM " + getTableName() + " o WHERE o.title=:title");
16. query.setParameter("title", genre);
17. **return** (Genre) query.getSingleResult();
18. }
19. }

**Класс *RoleDao***

1. @Repository
2. **public** **class** RoleDao **extends** AbstractDao<Role> {
4. **public** RoleDao() {
5. }
7. @Override
8. **protected** String getTableName() {
9. **return** "Role";
10. }
12. **public** Role findByTitle(String title) {
13. Query query = entityManager.createQuery("FROM "+getTableName()+" r WHERE r.title=:title");
14. query.setParameter("title", title);
15. **return** (Role)query.getSingleResult();
16. }
18. @Override
19. **protected** Class<Role> getClassObject() {
20. **return** Role.**class**;
21. }
22. }

**Класс *TokenDao***

1. @Repository
2. **public** **class** TokenDao **extends** AbstractDao<VerificationToken> {
4. @PersistenceContext
5. **private** EntityManager entityManager;
7. @Override
8. **protected** String getTableName() {
9. **return** "token";
10. }
12. **public** VerificationToken findByValue(String value) {
13. **return** (VerificationToken) entityManager.createQuery("from VerificationToken vt where vt.value=:value")
14. .setParameter("value", value)
15. .getSingleResult();
16. }
18. @Override
19. **protected** Class<VerificationToken> getClassObject() {
20. **return** VerificationToken.**class**;
21. }
22. }

Рассмотрим процесс регистрации нового пользователя в системы, реализация которого представлена ниже.

* @PostMapping("/registry")
* @ResponseBody
* **public** String registry(@ModelAttribute("user") UserDto user
* , BindingResult result
* , WebRequest request
* , HttpServletResponse response
* , Model model) **throws** IOException {
* String login = user.getLogin();
* String email = user.getEmail();
* **if** (userService.findByLogin(login) != **null**) {
* response.setStatus(HttpStatus.METHOD\_NOT\_ALLOWED.value());
* response.getWriter().write("login is busy");
* **return** **null**;
* }
* **if** (userService.findByEmail(email) != **null**) {
* response.setStatus(HttpStatus.METHOD\_NOT\_ALLOWED.value());
* response.getWriter().write("email is already registered");
* **return** **null**;
* }
* User newUser = **new** User();
* newUser.setLogin(user.getLogin());
* newUser.setPassword(encoder.encode(user.getPassword()));
* newUser.setName(user.getName());
* newUser.setSurname(user.getSurname());
* newUser.setEmail(user.getEmail());
* newUser.setRoles(Arrays.asList(roleService.findByTitle("CLIENT")));
* userService.save(newUser);
* String appUrl = request.getContextPath();
* eventPublisher.publishEvent(**new** OnRegistrationSuccessEvent(newUser, appUrl));
* **return** **null**;
* }

Когда пользователь заполняет форму регистрации, эта информации передается POST-запросом на сервер. Первым делом проверяется наличие в системе зарегистрированных пользователь с такой же почтой или логином. Если такие пользователи существуют, то регистрация прекращается, а на клиент отправляется соответствующая ошибка. Если такого пользователя не существует, то новый пользователь сохраняется в системе. Но аккаунт пользователя будет неактивным, пока пользователь не подтвердит регистрацию. Письмо с подтверждением будет отправлено на указанную электронную почту. Отправление письма происходит объявлением нового события в системе – *OnRegistrationSuccessEvent.* Данное событие обрабатывается обработчиком, код которого представлен ниже.

**Класс *RegistrationEmailListner***

1. @Component
2. **public** **class** RegistrationEmailListener **implements** ApplicationListener<OnRegistrationSuccessEvent> {
4. **private** **static** **final** Logger LOG = Logger.getLogger(RegistrationEmailListener.**class**);
6. @Autowired
7. **private** JavaMailSender sender;
9. @Autowired
10. **private** TokenServiceImpl tokenService;
12. @Override
13. **public** **void** onApplicationEvent(OnRegistrationSuccessEvent event) {
14. **try** {
15. sendConfirmationToken(event);
16. } **catch** (MessagingException e) {
17. LOG.error(e);
18. }
19. }
21. **private** **void** sendConfirmationToken(OnRegistrationSuccessEvent event) **throws** MessagingException {
23. User user = event.getUser();
24. String token = UUID.randomUUID().toString();
25. LOG.info("Create token for user " + user.getEmail() + ": " + token);
26. tokenService.createToken(user, token);
27. String recipient = user.getEmail();
28. String subject = "Registration Confirmation";
29. String url
30. = event.getAppUrl() + "/confirmRegistration/" + token;
31. String message = "Thank you, " + user.getName() + ", for registering. Please <a href='http://localhost:8081" + url + "'>click here</a> to activate your account.\n";
32. MimeMessageHelper helper = **new** MimeMessageHelper(sender.createMimeMessage());
33. helper.setTo(recipient);
34. helper.setSubject(subject);
35. helper.setText(message, **true**);
36. sender.send(helper.getMimeMessage());
37. }
38. }

После подтверждения регистрации аккаунт пользователя будет активирован и готов к работе. Процесс активации аккаунта представлен ниже.

1. @GetMapping("confirmRegistration/{token}")
2. **public** String confirmRegistration(
3. @PathVariable("token") String token, Model model) {
5. VerificationToken verificationToken = tokenService.findByValue(token);
6. LOG.info(verificationToken.getExpirationDate());
7. **if** (!checkToken(verificationToken, model)) {
8. **return** "info";
9. }
10. User user = verificationToken.getUser();
11. userService.enable(user);
12. tokenService.remove(verificationToken);
13. LOG.info("Account successfully activated");
14. **return** "redirect:/activationCompleted";
15. }

Рассмотрим процесс перехода на главную страничку ресурса.

1. @GetMapping("/")
2. **public** String home(@RequestParam(value = "artist", required = **false**) String artist
3. , @RequestParam(value = "title", required = **false**) String title
4. , @RequestParam(required = **false**, name = "genre") String genre
5. , Model model) {
6. List<Genre> genres = genreService.findAll();
7. List<Track> tracks;
8. **if** (genre != **null** && !genre.isEmpty()) {
9. tracks = trackService.findByGenre(genre);
10. } **else** **if** ((artist != **null** && !artist.isEmpty())
11. || (title != **null** && !title.isEmpty())) {
12. tracks = trackService.findByArtistAndTitle(artist, title);
13. } **else** {
14. tracks = trackService.findAll();
15. }
16. Collections.sort(tracks);
17. model.addAttribute("genres", genres);
18. model.addAttribute("tracks", tracks);
19. **return** "home";
20. }

На главной странице реализован поиск аудиозаписей по жанру, исполнителю и названию. Таким образом, поиск необходимых аудиозаписей будет выполнятся на основе параметров *HTTP*-запроса.

Рассмотрим процесс покупки аудиозаписи. Оплата совершается через платежную систему *Paypal*. При совершении платежа сначала создается объект заказа и отправляется на сервер *Paypal* для проверки.

Процесс создания объекта заказа представлен ниже.

1. @GetMapping("/payForCart")
2. **public** String cartPayment(Principal principal) {
3. User user = userService.findByLogin(principal.getName());
4. **if** (user.getTotalCartPrice().doubleValue() == 0.0) {
5. **return** "cart";
6. }
7. **try** {
8. Payment payment = service.createPayment(user.getTotalCartPrice(), CURRENCY, METHOD,
9. INTENT, DESC, "http://localhost:8081/" + CANCEL\_URL,
10. "http://localhost:8081/" + SUCCESS\_URL);
11. **for** (Links link : payment.getLinks()) {
12. **if** (link.getRel().equals("approval\_url")) {
13. **return** "redirect:" + link.getHref();
14. }
15. }
17. } **catch** (PayPalRESTException e) {
19. e.printStackTrace();
20. }
21. **return** "redirect:/paymentFailure";
22. }
24. @GetMapping("/payForTrack/{id}")
25. **public** String trackPayment(Principal principal, @PathVariable("id") **long** id, HttpServletRequest request) {
26. Track track = trackService.findById(id);
27. User user = userService.findByLogin(principal.getName());
28. String referer = request.getHeader("referer");
29. **if** (user.getTracks().contains(track)) {
30. **return** "redirect:" + referer;
31. }
32. **try** {
33. Payment payment = service.createPayment(track.getPrice(), CURRENCY, METHOD,
34. INTENT, DESC, "http://localhost:8081/" + CANCEL\_URL,
35. "http://localhost:8081/" + SUCCESS\_URL + "?trackId=" + id);
36. **for** (Links link : payment.getLinks()) {
37. **if** (link.getRel().equals("approval\_url")) {
38. **return** "redirect:" + link.getHref();
39. }
40. }
42. } **catch** (PayPalRESTException e) {
43. e.printStackTrace();
44. }
45. **return** "redirect:/paymentFailure";
46. }

**Класс *PaypalService***

1. @Service
2. **public** **class** PaypalService {
4. @Autowired
5. **private** APIContext apiContext;
7. **public** Payment createPayment(
8. BigDecimal total,
9. String currency,
10. String method,
11. String intent,
12. String description,
13. String cancelUrl,
14. String successUrl) **throws** PayPalRESTException {
15. Amount amount = **new** Amount();
16. amount.setCurrency(currency);
17. String price = total.setScale(2, RoundingMode.HALF\_UP).toString();
18. amount.setTotal(price);
20. Transaction transaction = **new** Transaction();
21. transaction.setDescription(description);
22. transaction.setAmount(amount);
24. List<Transaction> transactions = **new** ArrayList<>();
25. transactions.add(transaction);
27. Payer payer = **new** Payer();
28. payer.setPaymentMethod(method);
30. Payment payment = **new** Payment();
31. payment.setIntent(intent);
32. payment.setPayer(payer);
33. payment.setTransactions(transactions);
34. RedirectUrls redirectUrls = **new** RedirectUrls();
35. redirectUrls.setCancelUrl(cancelUrl);
36. redirectUrls.setReturnUrl(successUrl);
37. payment.setRedirectUrls(redirectUrls);
39. **return** payment.create(apiContext);
40. }
42. **public** Payment executePayment(String paymentId, String payerId) **throws** PayPalRESTException{
43. Payment payment = **new** Payment();
44. payment.setId(paymentId);
45. PaymentExecution paymentExecute = **new** PaymentExecution();
46. paymentExecute.setPayerId(payerId);
47. **return** payment.execute(apiContext, paymentExecute);
48. }
49. }

При одобрении заказа совершается платеж.

1. @GetMapping(value = SUCCESS\_URL)
2. **public** String successPay(Principal principal, @RequestParam(value = "trackId", required = **false**) String trackId, @RequestParam("paymentId") String paymentId, @RequestParam("PayerID") String payerId) {
3. **try** {
4. Payment payment = service.executePayment(paymentId, payerId);
5. System.out.println(payment.toJSON());
6. **if** (payment.getState().equals("approved")) {
7. User user = userService.findByLogin(principal.getName());
8. **if** (trackId != **null** && !trackId.isEmpty()) {
9. Track track = trackService.findById(Long.valueOf(trackId));
10. user.getCart().remove(track);
11. user.getTracks().add(track);
12. } **else** {
13. **for** (Track track: user.getCart()) {
14. user.getTracks().add(track);
15. }
16. user.getCart().removeAll(user.getCart());
17. }
18. userService.update(user);
19. **return** "redirect:/successfullyPayed";
20. }
21. } **catch** (PayPalRESTException e) {
22. System.out.println(e.getMessage());
23. }
24. **return** "redirect:/paymentFailure";
25. }

После успешного выполнения платежа, купленные аудиозаписи удаляются из корзины и попадают в список приобретенных аудиозаписей пользователя. После попытки совершения платежа пользователь получает соответствующее сообщение.

Итак, рассмотрены основные методы работы с ресурсом. Вся бизнес-логика составлена в соответствии с требованиями, представленными выше.

# Руководство пользователя

При входе в приложение пользователь попадает на главную страницу. На данной странице вверху находится шапка, на которой находятся ссылки для перемещения по сайту. В основной области представлен список аудиозаписей, содержащихся в системе. А также форма для поиска аудиозаписей по имени исполнителя и названии. Слева находится меню выбора аудиозаписей по жанрам. Неавторизованный пользователь не может осуществлять каких-либо действий с аудиозаписями кроме прослушивания. Содержимое главной страницы представлено на рисунке 4.1.

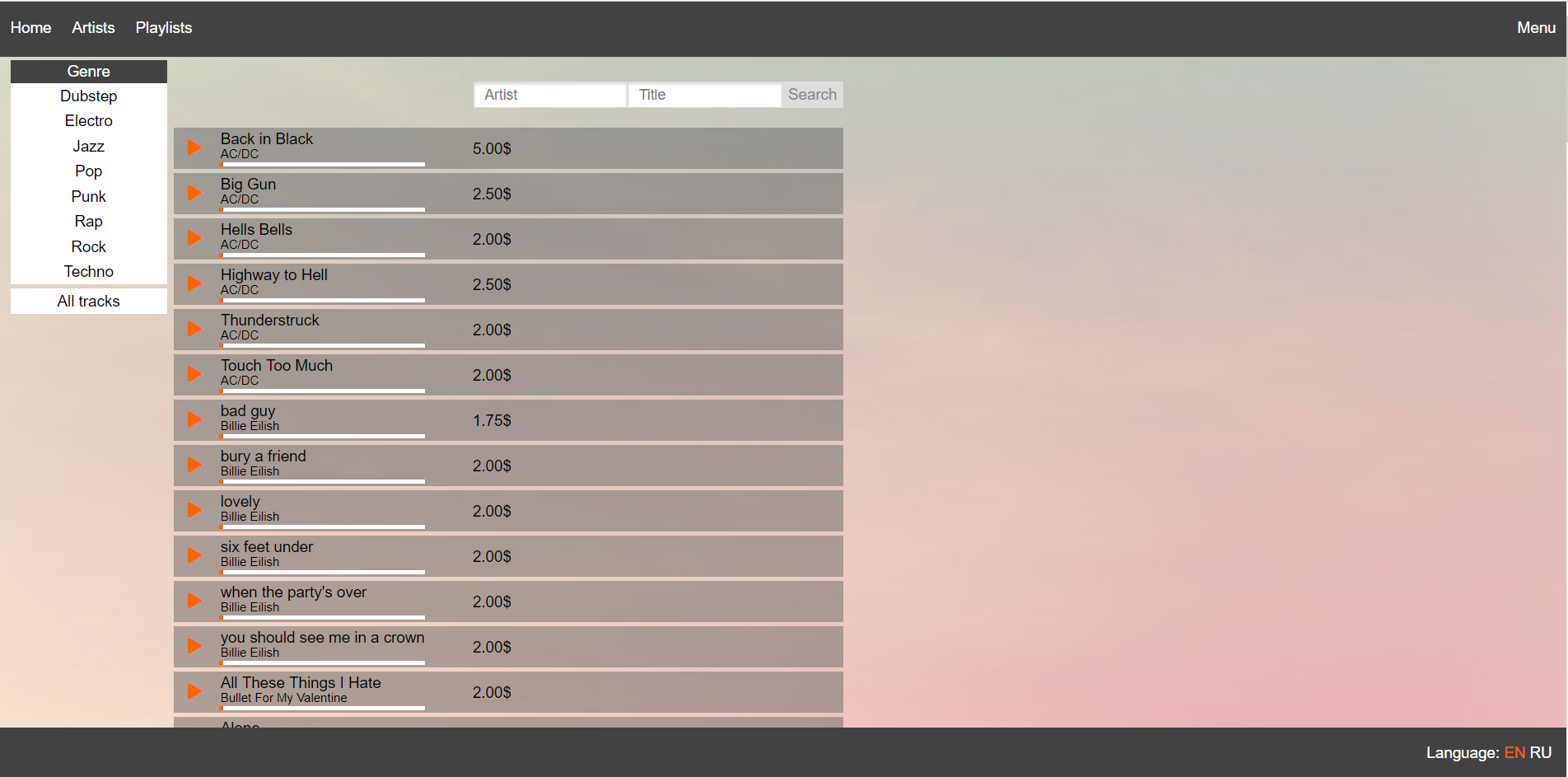


Рисунок 4.1 – Главная страница приложения

Для того, чтобы войти в систему, необходимо нажать на кнопку *«Menu»* в правом верхнем углу и выбрать *«Login»,* как показано на рисунке 4.2.

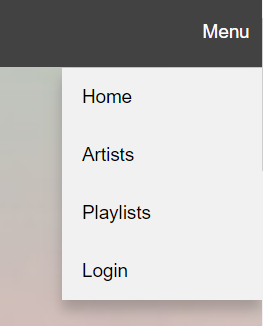


Рисунок 4.2 – Меню сайта

На странице появляется диалоговое окно с формой для ввода данных для входа в систему. Форма для входа представлена на рисунке 4.3. При отсутствии аккаунта, пользователь может создать новый аккаунт (см. рисунок 4.4).

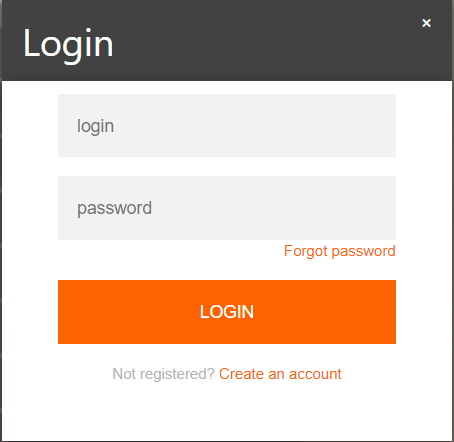


Рисунок 4.3 – Форма входа в систему

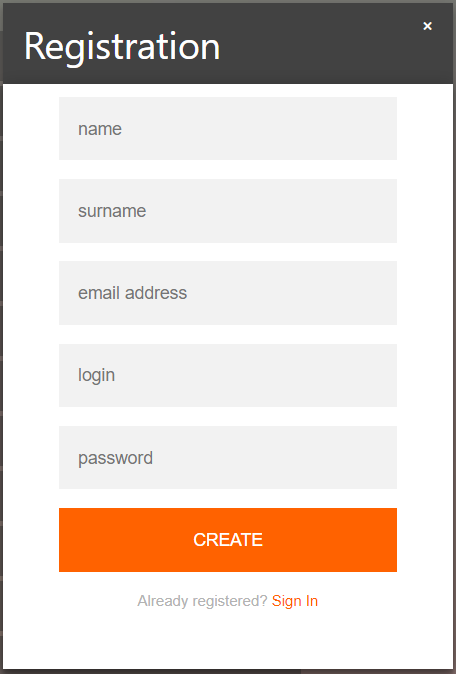


Рисунок 4.4 – Форма для регистрации нового пользователя

После того, как пользователь вошел в систему, он может просмотреть список приобретенных аудиозаписей. Для этого нужно нажать на кнопку *«My music»* на шапке сайта. Содержимое данной страницы представлено на рисунке 4.5.

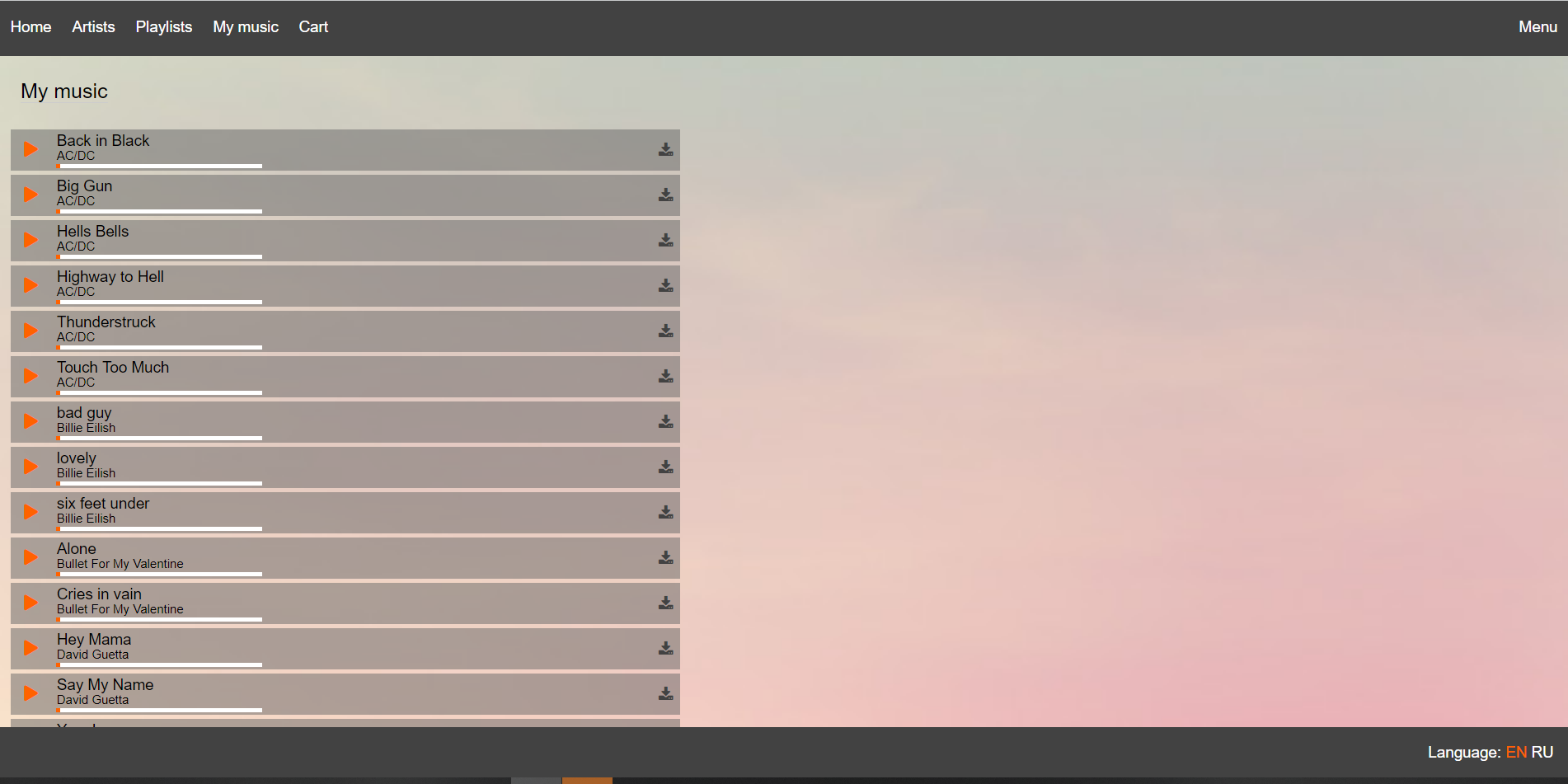


Рисунок 4.5 – Страница с приобретенными аудиозаписями

Также пользователь может просмотреть содержимое своей корзины. Для этого необходимо нажать на кнопку *«Cart»* на шапке сайта. Содержимое данной страницы представлено на рисунке 4.6.

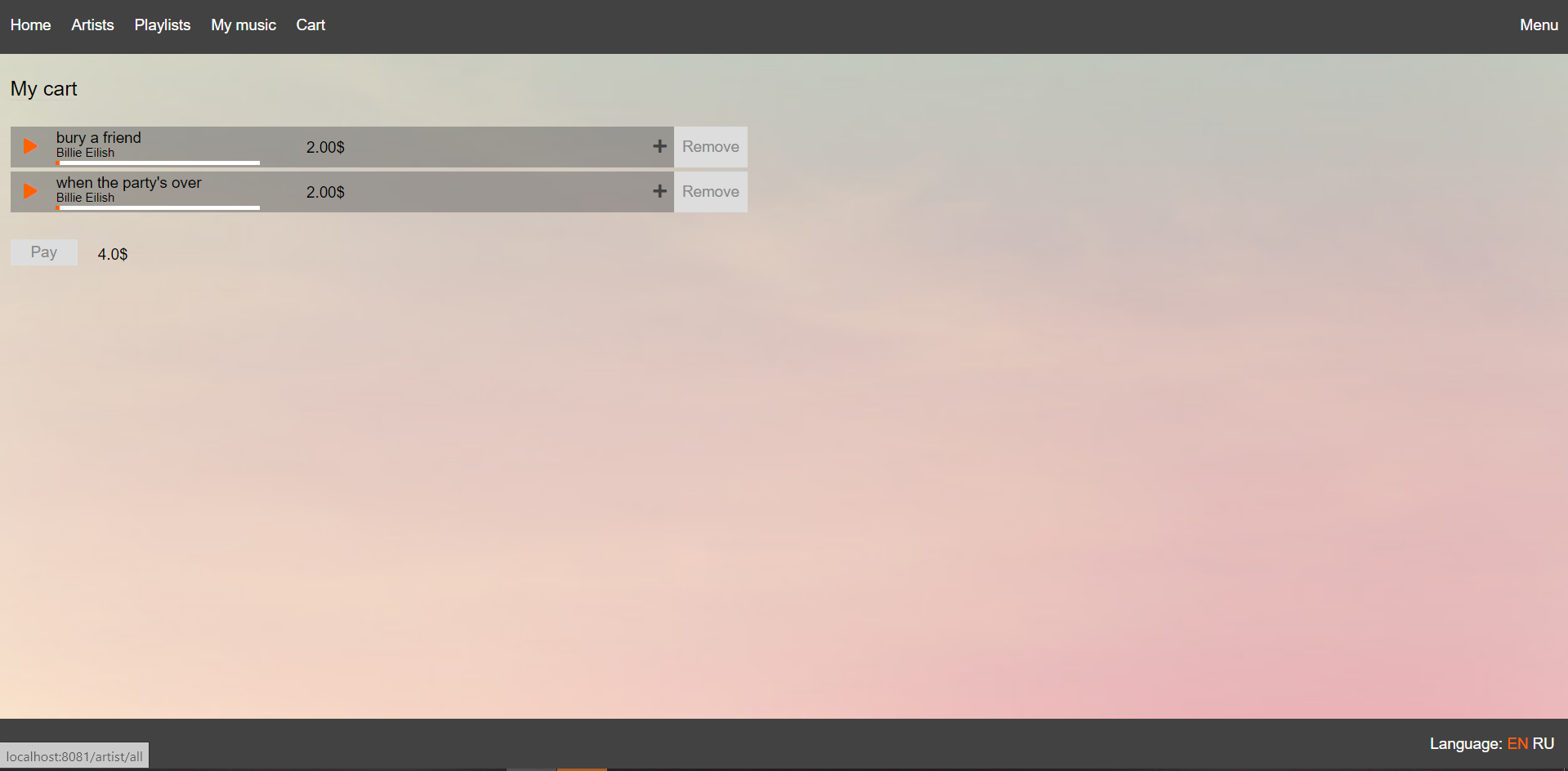


Рисунок 4.6 – Корзина

На данной странице пользователь может оплатить как отдельную аудиозапись, так и всю корзину сразу. Также пользователь может убрать аудиозапись из корзины.

При нажатии на кнопку *«Pay»* пользователь будет перенаправлен на страницу выбора способа оплаты. Данная страница показана на рисунке 4.7.

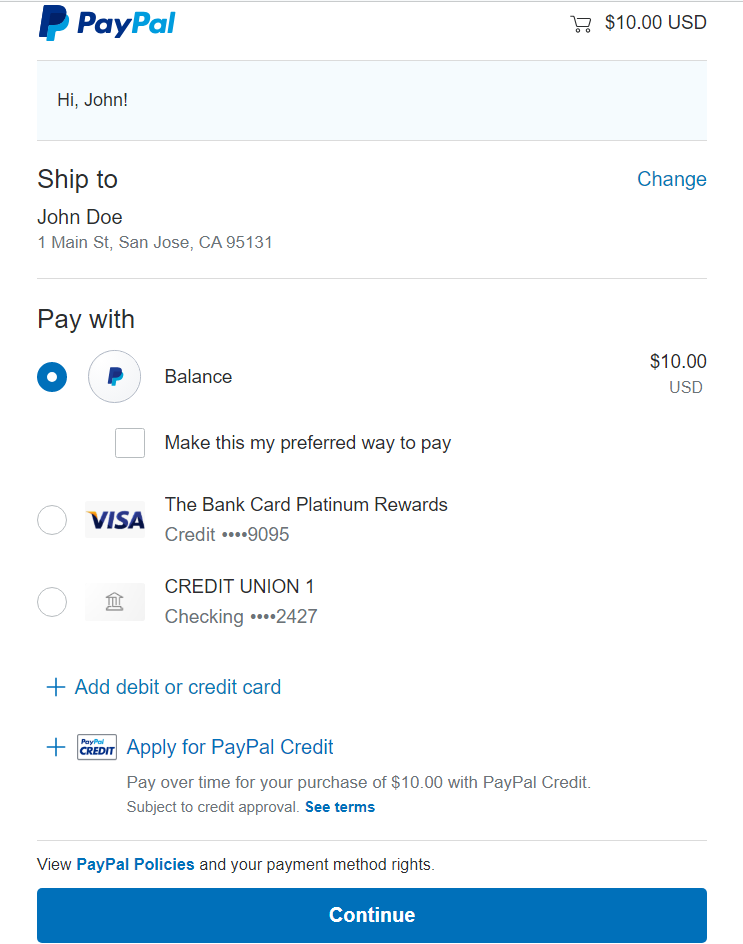


Рисунок 4.7 – Выбор способа оплаты

Если оплата проведена успешна, то пользователь перенаправляется обратно в приложение и получает сообщение об успешно проведенной оплате, как показано на рисунке 4.8.

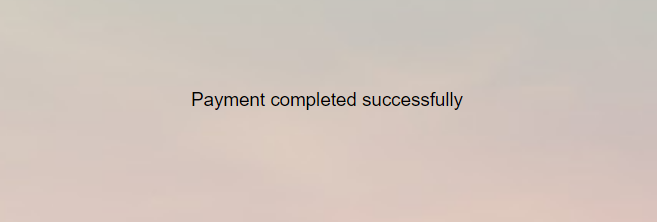


Рисунок 4.8 – Сообщение о проведенной оплате

Чтобы просмотреть список всех исполнителей, необходимо нажать на кнопку *«Artist»* либо на шапке сайта либо в меню. На странице также имеется строка поиска по имени исполнителя.

Содержимое страницы с исполнителями представлено на рисунке 4.9.

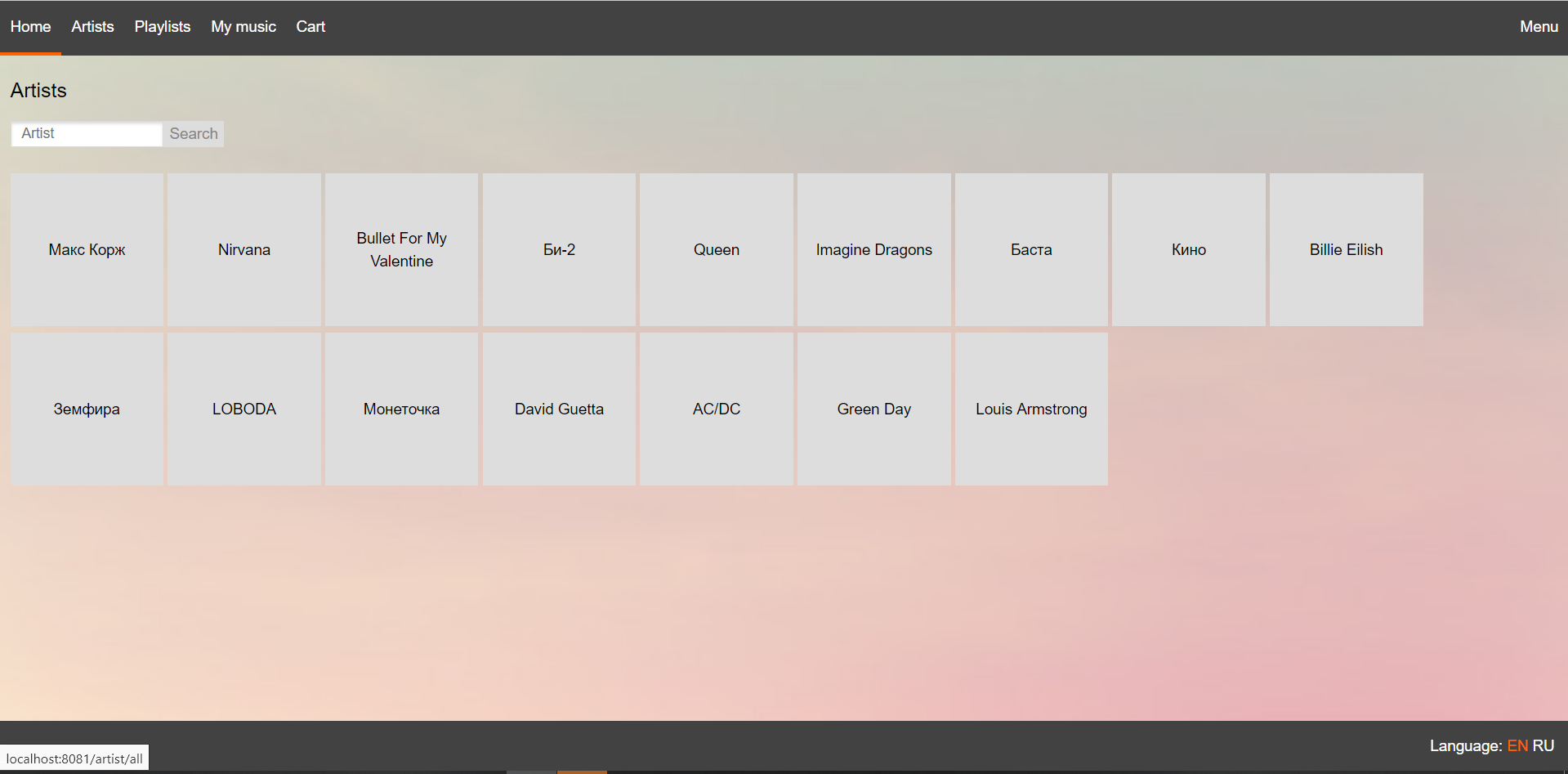


Рисунок 4.9 – Страница с исполнителями

Нажав на нужного исполнителя откроется страница исполнителя. Она будет содержать краткую информацию об исполнителе. А также список всех его аудиозаписей и список альбомов. Содержимое данной страницы приведено на рисунке 4.10.



Рисунок 4.10 – Страница исполнителя

Чтобы открыть список плейлистов, нужно нажать на кнопку *«Playlists»* на шапке сайта или в меню. Страница со всеми плейлистами представлена на рисунке 4.11.

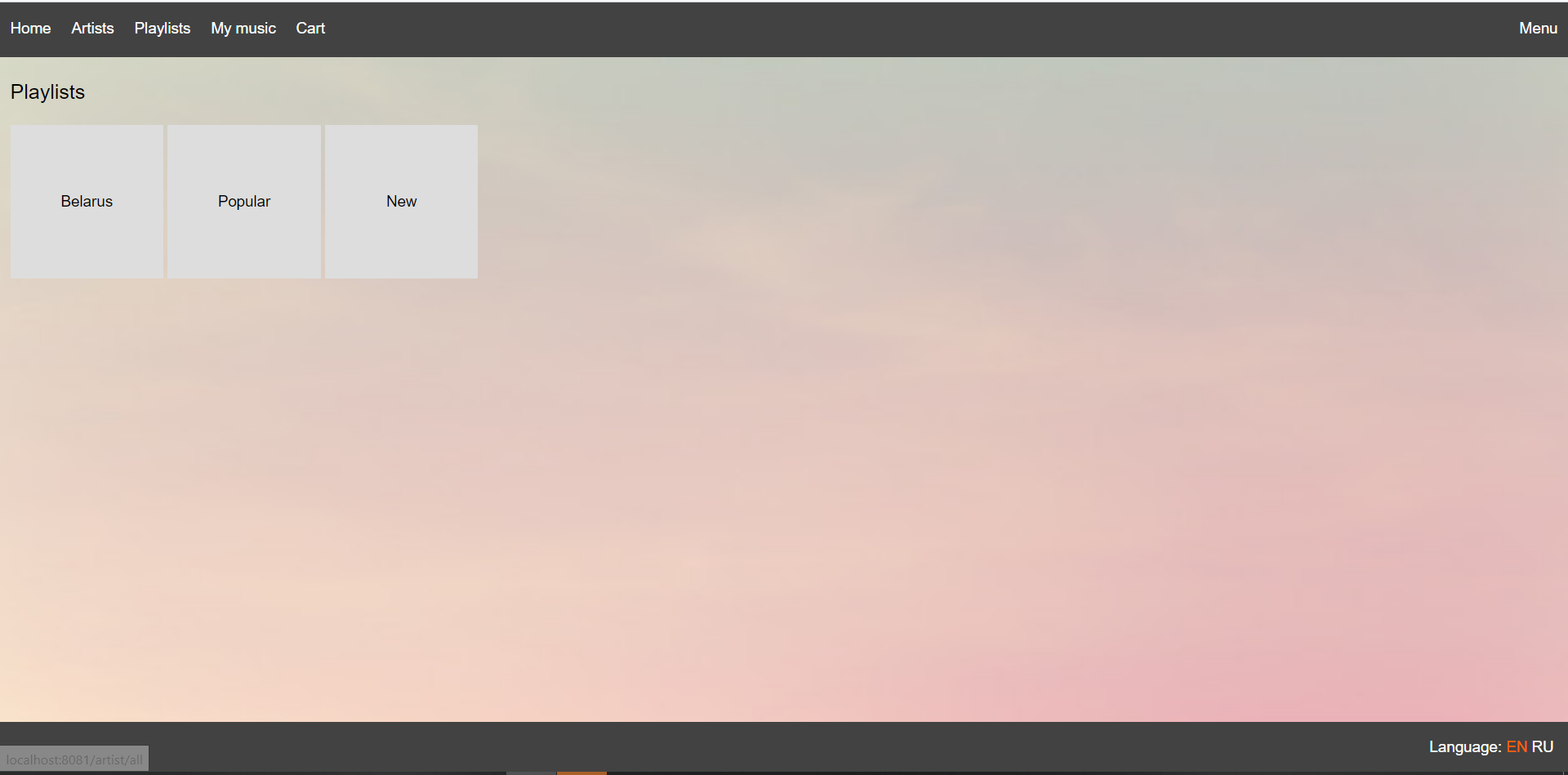


Рисунок 4.11 – Страница с плейлистами

Пользователь может редактировать данные своего профиля. Для этого нужно нажать на кнопку *«My account»* в меню сайта. Откроется страница, показанная на рисунке 4.12. На ней можно изменить имя пользователя, фамилию и почту, к которой привязан аккаунт. Также есть возможность изменить пароль от аккаунта.

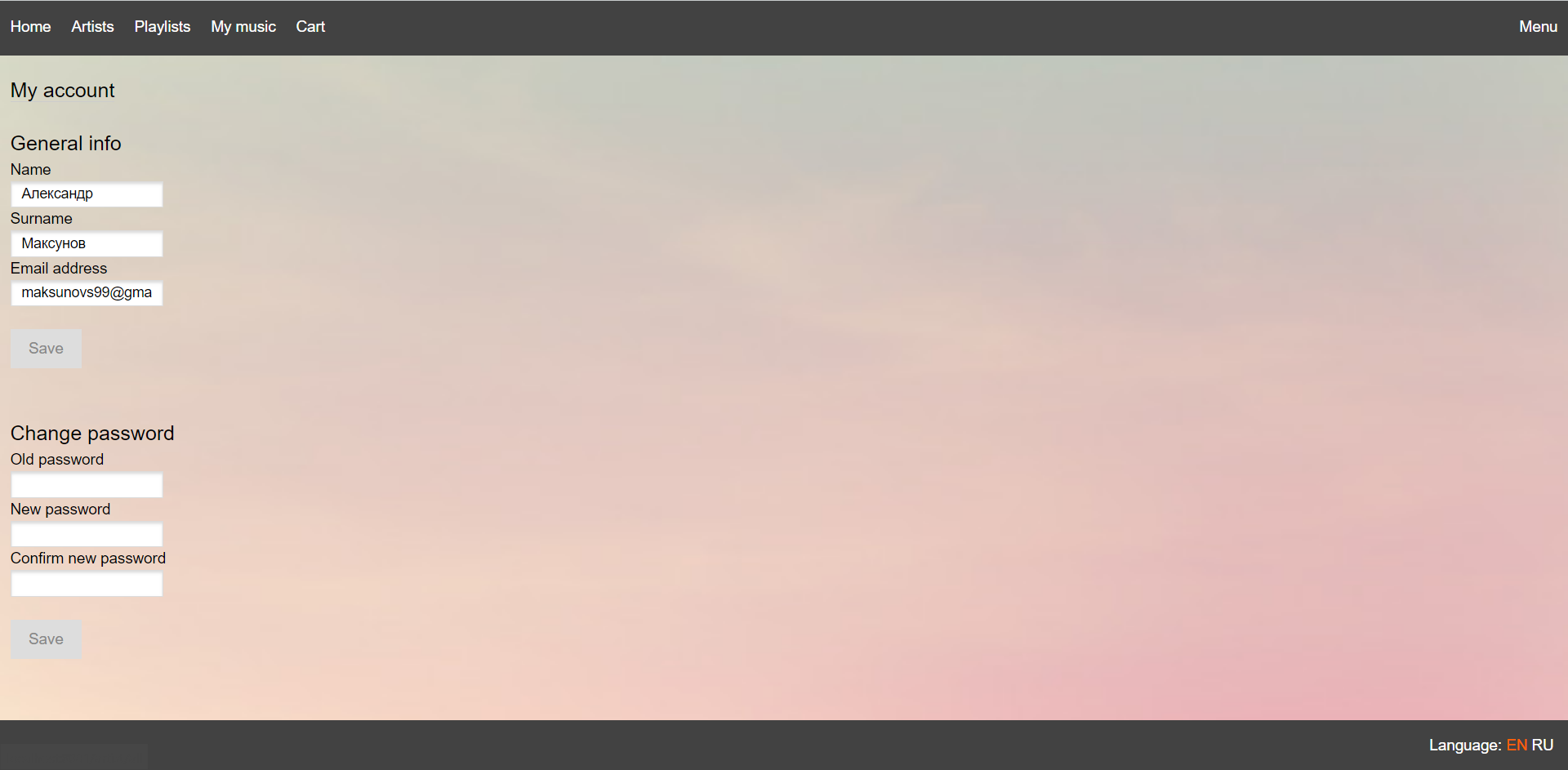


Рисунок 4.12 – Страница редактирования профиля пользователя

Теперь войдем систему под учетной записью пользователя, который обладает правами администратора.

Администратор имеет право на добавления новых записей в систему, а также изменение и удаление.

Форма для добавления нового исполнителя представлена на рисунке 4.13.

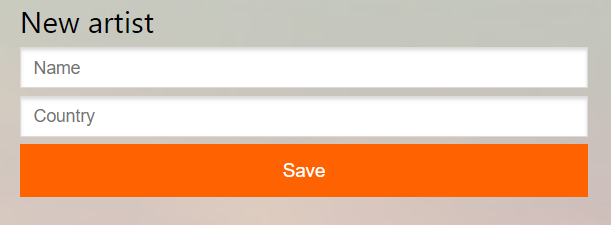


Рисунок 4.13 – Форма добавления нового исполнителя

Имеется возможность добавления новых жанров, а также редактирования и удаления существующих. Данные жанры буду отображаться на главной странице в качестве фильтра для поиска аудиозаписей по жанрам. Страница добавления, удаления и изменения жанров представлена на рисунке 4.14.

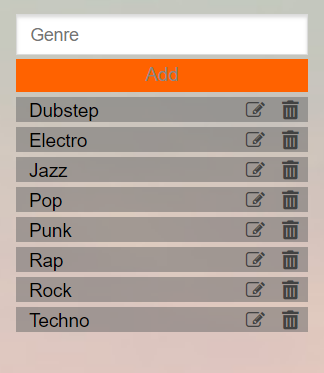


Рисунок 4.14 – Страница добавления, удаления и изменения жанров

На рисунке 4.15 представлена форма для изменения и добавления новых аудиозаписей. В поле «Исполнитель» не нужно писать имя исполнителя самому, так как при нажатии на данное поле появляется список со всеми исполнителями, находящихся в системе. Пользователю остается только выбрать нужно исполнителя. Если нужно исполнителя нет в списке, его необходимо добавить через форму добавления нового исполнителя. Также при на нажатии на поле «Жанр» появляется список всех жанров, и пользователю необходимо выбрать нужный.

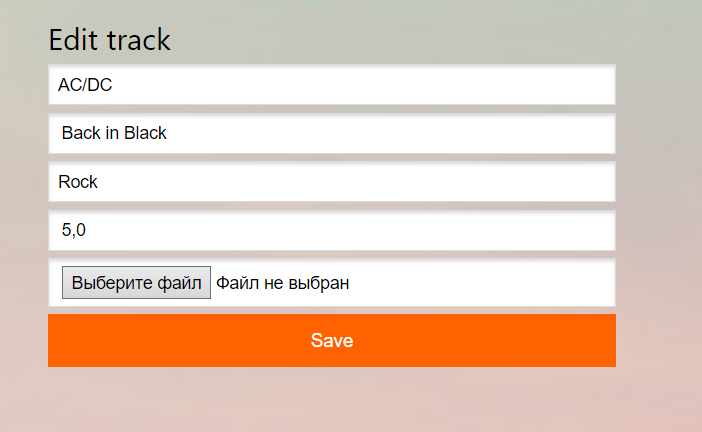


Рисунок 4.15 – Форма изменения данных аудиозаписи

Администратор может редактировать плейлисты и альбомы исполнителей: добавлять и удалять из них аудиозаписи. Пример страницы плейлиста показан на рисунке 4.16. На ней администратор может удалить из плейлиста аудиозапись, нажав на кнопку *«Remove»* напротив нужной аудиозаписи.



Рисунок 4.16 – Страница плейлиста

При нажатии на кнопку *«Edit»* открывается страница редактирования плейлиста. На ней находятся строка для ввода нового названия плейлиста, а также аудиозаписи, не входящие в плейлист. Чтобы добавить аудиозапись в плейлист, необходимо нажать на кнопку *«Add»* напротив нужной аудиозаписи. Пример данной страницы представлен на рисунке 4.17.



Рисунок 4.17 – Страница редактирования плейлиста

# ЗАКЛЮЧЕНИЕ

В результате выполнения данного курсового проекта было разработано приложение, с помощью которого пользователь может прослушивать музыку и покупать ее для получения доступа к скачиванию на различные устройства.

Пользователи данной системы имеют различные роли, которые определяют право доступа к различным функциям приложения. Так, пользователь с правами обычного пользователя может только просматривать контент сайта, прослушивать музыку, покупать ее и скачивать. Тем временем, пользователь с правами администратора может добавлять новых исполнителей, новые аудиозаписи, плейлисты и альбомы. Также он может редактировать и удалять данные записи.

Интерфейс приложения является интуитивно-понятным и не содержит резких контрастов цветов.

Использованы современные технологии для создания веб-приложений (Spring Core, Spring Security, Spring MVC, Hibernate, Maven, JSP, JPA), которые позволяют делать разработку максимально эффективной, а приложение многофункциональным.

# Список использованных источников

[1] Шилдт Г. Java. Полное руководство. 10-е издание/Шилдт г. Издательство Вильямс, 2019. – 1488 с.

[2] Вендров А. М. CASE–технологии. Современные методы и средства проектирования информационных систем. – М.: Финансы и статистика, 1998.

[3] Нормализация отношений. Шесть нормальных форм [Электронный ресурс]. – Режим доступа: https://habr.com/post/254773/, свободный.

[4] [Тереза Нейл](https://oz.by/people/more9036515.html). Проектирование веб-интерфейсов/ [Тереза Нейл](https://oz.by/people/more9036515.html), [Билл Скотт](https://oz.by/people/more9036514.html) – O’Reilly, 2014/ - 352с.

[5] Этапы проектирования данных [Электронный ресурс] – Режим доступа: <http://www.mstu.edu.ru/study/materials/zelenkov/ch_5_1.html>, свободный.

[6] СТП 01-2017. Стандарт предприятия. Дипломные проекты(работы). Общие требования. – Минск: БГУИР, 2017. – 169 с.

# ПРИЛОЖЕНИЕ А (справочное)

**Выдержки из кода**

**Класс *MvcConfig***

1. @Configuration
2. @EnableWebMvc
3. @ComponentScan
4. **public** **class** MvcConfig **implements** WebMvcConfigurer {
6. @Autowired
7. **private** UserService userService;
9. @Autowired
10. **private** TrackStateInitializer initializer;
12. @Override
13. **public** **void** configureViewResolvers(ViewResolverRegistry registry) {
14. InternalResourceViewResolver resolver = **new** InternalResourceViewResolver();
15. resolver.setPrefix("/WEB-INF/templates/");
16. resolver.setSuffix(".jsp");
17. resolver.setViewClass(JstlView.**class**);
18. registry.viewResolver(resolver);
19. }
21. @Override
22. **public** **void** addInterceptors(InterceptorRegistry registry) {
23. registry.addInterceptor(**new** TrackStateHandler(userService, initializer));
24. }
26. @Override
27. **public** **void** configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {
28. configurer.enable();
29. }
30. }

**Класс *PaypalConfig***

1. @Configuration
2. **public** **class** PaypalConfig {
3. @Value("${paypal.client.id}")
4. **private** String clientId;
5. @Value("${paypal.client.secret}")
6. **private** String clientSecret;
7. @Value("${paypal.mode}")
8. **private** String mode;
10. @Bean
11. **public** Map<String, String> paypalSdkConfig() {
12. Map<String, String> configMap = **new** HashMap<>();
13. configMap.put("mode", mode);
14. **return** configMap;
15. }
17. @Bean
18. **public** OAuthTokenCredential oAuthTokenCredential() {
19. **return** **new** OAuthTokenCredential(clientId, clientSecret, paypalSdkConfig());
20. }
22. @Bean
23. **public** APIContext apiContext() **throws** PayPalRESTException {
24. APIContext context = **new** APIContext(oAuthTokenCredential().getAccessToken());
25. context.setConfigurationMap(paypalSdkConfig());
26. **return** context;
27. }
28. }

**Класс WebSecurityConfig**

1. @Configuration
2. @EnableWebSecurity
3. @EnableGlobalMethodSecurity(securedEnabled = **true**, prePostEnabled = **true**)
4. **public** **class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {
6. @Autowired
7. **private** UserDetailsServiceImpl userDetailsService;
9. @Override
10. **protected** **void** configure(HttpSecurity http) **throws** Exception {
12. // Включаем зачщиту от CSRF атак
13. http.csrf()
14. .disable()
15. .authorizeRequests()
16. .antMatchers("/", "/registry", "/info", "/confirmRegistration/\*\*", "confirmPasswordRecovery/\*\*").permitAll()
17. .anyRequest().permitAll()
18. .and()
19. .formLogin()
20. .loginPage("/login")
21. .loginProcessingUrl("/login")
22. .failureHandler(**new** CustomAuthenticationFailureHandler())
23. .usernameParameter("login")
24. .passwordParameter("password")
25. .permitAll()
26. .and()
27. .logout()
28. .logoutUrl("/logout")
29. .logoutSuccessUrl("/")
30. .invalidateHttpSession(**true**)
31. .permitAll();
32. }
34. @Bean
35. **public** BCryptPasswordEncoder passwordEncoder() {
36. **return** **new** BCryptPasswordEncoder();//для кодирования пароля достаточно только объявления этого бина
37. }
38. }

**Файл application.properties**

# =============================================

# SERVER SETTINGS

# =============================================

server.port=8081

# =============================================

# DATASOURCE PROPERTIES

# =============================================

spring.datasource.url=jdbc:mysql://localhost:3306/audiospot?useSSL=false&serverTimezone=UTC

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# =============================================

# HIBERNATE PROPERTIES

# =============================================

spring.jpa.show-sql=false

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.hibernate.naming.naming-strategy=org.hibernate.cfg.ImprovedNamingStrategy

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

# =============================================

# GMAIL SMTP SERVER PROPERTIES

# =============================================

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=trackspot.sup@gmail.com

spring.mail.password=dhjmxzcrabosnhsoohc

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true

verification.expirationTime=5

spring.servlet.multipart.max-file-size=30MB

spring.servlet.multipart.max-request-size=10MB

# =============================================

# PAYPAL CREDENTIALS

# =============================================

paypal.mode=sandbox

paypal.client.id=AdT7bfTC1iWjXSLim0B3aYYnBncEwPWly89BweIb3rv-xjgQ41JlNGorL0NAB3p8DORJsdHjdfsd7Kf456h46kj

paypal.client.secret=EIPsTgrDH4cyMBk\_oC53yjMwBBdDgev57KHUP0P1Eszh1JMUUlEEfhYkZ6sjC-233hfghDJD8dTOJzXLu

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | *Дополнительные сведения* | | | | |
|  | | | | | *Текстовые документы* |  | | | | |
|  | | | | |  |  | | | | |
| *БГУИР ДП 1-53 01 02 01 031 ПЗ* | | | | | *Пояснительная записка* | *65 с.* | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  |  |  |  |  | *БГУИР ДП 1-53 01 02 01 031 Д1* | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| *Изм.* | *Л.* | *№ докум.* | *Подп.* | *Дата* | *Автоматизированная система музыкального стримингового сервиса*  *Ведомость курсового проекта* | *Лит* | | | *Лист* | *Листов* |
| *Разраб.* | | *Максунов* |  |  |  | *1* |  | *65* | *65* |
| *Провер.* | | *Трофимович* |  |  | *Кафедра ИТАС*  *гр. 720601* | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |