

TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO ĐỒ ÁN HỌC PHẦN PHÁT TRIỂN PHẦN MỀM MÃ NGUỒN MỞ

NHÓM 14

Giảng viên phụ trách: Từ Lăng Phiêu

Sinh viên thực hiện:

Nguyễn Anh Khoa	3121410272
Phạm Thị Minh Thư	3120410519
Kim Duy Linh	3121410293
Phạm Duy Trực	3122410439
Huỳnh Tuyết Mai	3120560058

Email liên hệ: akhoa0805@gmail.com

Mục lục

Chương 1: Giới thiệu	3
I. Tổng quan và mục tiêu dự án	3
II. Phát biểu vấn đề	3
III. Phạm vi dự án	4
Chương 2: Cơ sở lý thuyết	5
I. Công nghệ và Thư viện sử dụng	5
II. Kiến trúc hệ thống	6
1. Sơ đồ kiến trúc	6
2. Các mẫu thiết kế đã triển khai	7
III. Thiết kế cơ sở dữ liệu	7
1. Sơ đồ cơ sở dữ liệu (sơ đồ ER)	7
2. Các bảng và mối quan hệ	9
3. Phương pháp mô hình hóa dữ liệu	11
Chương 3: Hiện thực	13
I. Triển khai các tính năng	13
II. Ảnh chụp màn hình và minh họa	13
1. Giao diện người dùng	13
2. Giao diện trang Admin	17
III. Cấu trúc mã nguồn	20
1. Tổ chức thư mục	20
2. Các thành phần chính và tương tác	20
3. Flowchart cho các quy trình phức tạp	20
Chương 4: Cài đặt và triển khai	23
I. Yêu cầu hệ thống	23
II. Các bước cài đặt	23
III. Cấu hình	24
IV. Hướng dẫn triển khai	26
Chương 5: Đóng góp của thành viên	28
I. Phân công nhiệm vụ	28
II. Đóng góp chi tiết của từng thành viên	28
1. Nguyễn Anh Khoa	28
2. Phạm Thị Minh Thư	28
3. Phạm Duy Trực	28
4. Kim Duy Linh	28
5. Huỳnh Tuyết Mai	28

Chương 6: Kết luận	29
I. Tóm tắt dự án	29
II. Hạn chế	29
III. Cải tiến trong tương lai	30

Chương 1: Giới thiệu

I. Tổng quan và mục tiêu dự án

Dự án phát triển ứng dụng nghe nhạc trực tuyến Spotify Clone được thực hiện trong khuôn khổ môn học "Phát triển phần mềm mã nguồn mở". Mục tiêu chính của dự án là thiết kế và xây dựng một ứng dụng web có khả năng cung cấp các tính năng nghe nhạc trực tuyến tương tự như Spotify, sử dụng các công nghệ mã nguồn mở để đảm bảo tính linh hoạt và khả năng tùy chỉnh.

Ứng dụng được phát triển với:

- **Front-end:** React để xây dựng giao diện người dùng hiện đại, thân thiện và tương tác.
- **Back-end:** Django (Python) để xử lý logic phía server và quản lý dữ liệu.
- **Database:** Database: Sử dụng MySQL, một cơ sở dữ liệu mã nguồn mở, để lưu trữ thông tin người dùng, bài hát, album và các dữ liệu liên quan.

Mục tiêu cụ thể của dự án bao gồm:

- Xây dựng một ứng dụng web cho phép phát nhạc trực tuyến với giao diện và trải nghiệm người dùng tương tự Spotify.
- Hỗ trợ phát video âm nhạc (nếu có).
- Quản lý tài nguyên âm nhạc (bài hát, album).
- Cung cấp tính năng quản lý người dùng (tạo album, bài hát yêu thích).
- Xây dựng trang Admin để quản lý toàn bộ hệ thống.
- Tích hợp tính năng chat để tăng tính tương tác trong giao diện web.

Dự án không chỉ nhằm đáp ứng yêu cầu học thuật mà còn giúp nhóm phát triển nâng cao kỹ năng làm việc với các công nghệ mã nguồn mở, từ đó đóng góp vào cộng đồng phát triển phần mềm.

II. Phát biểu vấn đề

Trong bối cảnh nhu cầu giải trí trực tuyến ngày càng tăng, các ứng dụng nghe nhạc trực tuyến như Spotify đã trở thành một phần không thể thiếu trong đời sống của nhiều người. Tuy nhiên, việc tiếp cận các nền tảng như Spotify có thể gặp hạn chế về chi phí hoặc tính năng không hoàn toàn phù hợp với nhu cầu của một số người dùng. Ngoài ra, việc tìm hiểu và xây dựng một ứng dụng tương tự Spotify bằng các công nghệ mã nguồn mở là một bài toán thú vị và mang tính thực tiễn cao.

Dự án Spotify Clone được xây dựng để giải quyết các vấn đề sau:

- Cung cấp một giải pháp thay thế mã nguồn mở cho ứng dụng nghe nhạc trực tuyến, giúp người dùng dễ dàng tùy chỉnh và mở rộng tính năng theo nhu cầu.
- Tích hợp các tính năng cơ bản như phát nhạc, phát video âm nhạc, quản lý bài hát yêu thích và album của người dùng.
- Đảm bảo tính quản lý hiệu quả thông qua trang Admin, giúp quản trị viên dễ dàng kiểm soát nội dung và người dùng.
- Đáp ứng yêu cầu kỹ thuật khi triển khai trên nền tảng Linux, đảm bảo tính ổn định và khả năng mở rộng của hệ thống.

Việc phát triển ứng dụng này không chỉ giúp nhóm hiểu rõ hơn về quy trình phát triển phần mềm mã nguồn mở mà còn tạo ra một sản phẩm có giá trị thực tiễn, có thể được sử dụng hoặc phát triển thêm trong tương lai.

III. Phạm vi dự án

Phạm vi của dự án tập trung vào việc xây dựng một ứng dụng web nghe nhạc trực tuyến với các tính năng cơ bản, phù hợp với yêu cầu của môn học và khả năng của nhóm. Cụ thể:

Phạm vi chức năng

- Phát nhạc trực tuyến: Người dùng có thể nghe các bài hát được lưu trữ trên hệ thống.
- Phát video âm nhạc: Hỗ trợ phát các video âm nhạc (nếu có sẵn dữ liệu).
- Quản lý tài nguyên âm nhạc: Lưu trữ và quản lý bài hát, album.
- Quản lý người dùng: Người dùng có thể tạo danh sách bài hát yêu thích, quản lý album cá nhân.
- Trang Admin: Cung cấp giao diện quản trị để quản lý người dùng, bài hát và nội dung.

Phạm vi kỹ thuật

- Sử dụng các công nghệ mã nguồn mở như React (front-end), Django (back-end), và cơ sở dữ liệu MySQL.
- Triển khai ứng dụng trên hệ điều hành Linux để đảm bảo tính tương thích và hiệu suất.

Giới hạn

- Dự án không tập trung vào việc phát triển các tính năng phức tạp như gợi ý bài hát thông minh dựa trên AI hoặc tích hợp thanh toán trực tuyến.
- Chỉ hỗ trợ các định dạng âm thanh và video phổ biến (MP3, MP4, v.v.), không bao gồm các định dạng đặc thù.

Dự án sẽ được phát triển trong thời gian giới hạn của môn học, với mục tiêu hoàn thiện các tính năng cốt lõi và đảm bảo ứng dụng hoạt động ổn định trên môi trường Linux.

Chương 2: Cơ sở lý thuyết

I. Công nghệ và Thư viện sử dụng

Dự án Spotify Clone được xây dựng dựa trên các công nghệ và thư viện mã nguồn mở hiện đại, đảm bảo hiệu suất và khả năng mở rộng. Dưới đây là tổng quan về các công nghệ và thư viện được sử dụng cho Front-end và Back-end:

1. Front-end (FE)

Front-end của dự án được phát triển bằng **React** (v18.3.1), một thư viện JavaScript phổ biến để xây dựng giao diện người dùng động và tái sử dụng. Các thư viện hỗ trợ bao gồm:

- **React Router DOM** (`react-router-dom`): Quản lý điều tuyến giữa các trang như trang chủ, danh sách phát, hoặc trang quản lý người dùng.
- **Redux và React Redux** (`redux`, `react-redux`, `@reduxjs/toolkit`): Quản lý trạng thái trạng thái phát nhạc.
- **React Hook Form và Zod** (`react-hook-form`, `zod`): Xử lý và xác thực form, như form đăng ký người dùng.
- **Radix UI** (`@radix-ui/*`): Cung cấp các thành phần giao diện không kiểu dáng như Dialog, Dropdown Menu, Select, Slider, Tabs, Toast, và Tooltip, hỗ trợ các tính năng như hiển thị thông báo hoặc chọn bài hát.
- **Bootstrap và React Bootstrap** (`bootstrap`, `react-bootstrap`): Cung cấp các thành phần UI có sẵn như nút, modal, và navbar.
- **Tailwind CSS** (`tailwindcss`, `tailwind-merge`, `tailwindcss-animate`): Thiết kế giao diện nhanh chóng và linh hoạt, hỗ trợ hoạt ảnh.
- **Axios** (`axios`): Gửi yêu cầu HTTP đến Back-end để lấy dữ liệu như danh sách bài hát.
- **React Data Table Component** (`react-data-table-component`): Hiển thị bảng dữ liệu như danh sách bài hát hoặc album.
- **React Day Picker** (`react-day-picker`) và **Date-fns** (`date-fns`): Quản lý ngày tháng, ví dụ: hiển thị ngày phát hành bài hát.
- **React Icons, Lucide React, và Material Symbols** (`react-icons`, `lucide-react`, `material-symbols`): Cung cấp biểu tượng cho giao diện (nút play, pause, menu).
- **React Pro Sidebar** (`react-pro-sidebar`): Tạo sidebar điều hướng cho danh sách phát hoặc bài hát yêu thích.

- **React Spinners** (`react-spinners`): Hiển thị hiệu ứng loading khi tải bài hát.
- **React Toastify** và **Sonner** (`react-toastify`, `sonner`): Hiển thị thông báo, như khi thêm bài hát vào danh sách yêu thích.
- **Tanstack React Table** (`@tanstack/react-table`): Xây dựng bảng dữ liệu động.
- **Lodash** (`lodash`): Thư viện tiện ích hỗ trợ xử lý dữ liệu.
- **Sass** (`sass`): Tiền xử lý CSS để tùy chỉnh giao diện.
- **Công cụ phát triển**: Vite (`vite`) để build và phát triển nhanh, ESLint (`eslint`) và Prettier (`prettier`) để kiểm tra và định dạng mã nguồn.

2. Back-end (BE)

Back-end được phát triển bằng **Django** (v5.1.7), một framework Python mạnh mẽ để xây dựng ứng dụng web. Các thư viện hỗ trợ bao gồm:

- **Django REST Framework** (`rest_framework`, `rest_framework.authtoken`): Xây dựng API RESTful và hỗ trợ xác thực bằng token, cho phép Front-end giao tiếp với Back-end qua các endpoint như lấy danh sách bài hát hoặc xác thực người dùng.
- **Cloudinary** (`cloudinary`, `cloudinary_storage`): Quản lý và lưu trữ file đa phương tiện (hình ảnh, video) trên Cloudinary, ví dụ: lưu trữ ảnh bìa bài hát hoặc video âm nhạc.
- **CORS Headers** (`corsheaders`): Xử lý CORS để cho phép giao tiếp giữa Front-end (React) và Back-end, đảm bảo Front-end có thể gửi yêu cầu từ `localhost:3000`.
- **MySQL** (`django.db.backends.mysql`): Cơ sở dữ liệu chính để lưu trữ thông tin người dùng, bài hát, danh sách phát, và bình luận.
- **Pathlib** (`pathlib`): Thư viện chuẩn của Python để quản lý đường dẫn file, đảm bảo tính tương thích trên các nền tảng.
- **Django tích hợp sẵn**: Các ứng dụng như `django.contrib.auth` (xác thực), `django.contrib.admin` (giao diện admin), và `django.contrib.sessions` (quản lý phiên) được sử dụng để hỗ trợ các tính năng cơ bản.

3. Tổng kết

Sự kết hợp giữa React và Django, cùng với các thư viện hỗ trợ, giúp dự án Spotify Clone đáp ứng đầy đủ các yêu cầu như phát nhạc, quản lý người dùng, và tích hợp tính năng chat (tùy chọn). Các công nghệ mã nguồn mở được chọn đảm bảo tính linh hoạt, dễ bảo trì và khả năng mở rộng trong tương lai.

II. Kiến trúc hệ thống

1. Sơ đồ kiến trúc

Hệ thống Spotify Clone được thiết kế theo mô hình Client-Server, trong đó Front-end (React) đóng vai trò là client và Back-end (Django) đóng vai trò là server.

Mô tả sơ đồ kiến trúc:

- **Client (Front-end):** Người dùng tương tác với giao diện web được xây dựng bằng React, chạy trên trình duyệt. React gửi các yêu cầu HTTP (qua Axios) đến Back-end để lấy dữ liệu hoặc thực hiện các hành động như phát nhạc, thêm bài hát vào danh sách yêu thích.
- **Server (Back-end):** Django xử lý các yêu cầu từ Front-end thông qua API RESTful (Django REST Framework). Dữ liệu được lưu trữ trong MySQL và các file đa phương tiện được quản lý trên Cloudinary.
- **Cơ sở dữ liệu:** MySQL lưu trữ thông tin như người dùng, bài hát, danh sách phát, và bình luận.
- **Lưu trữ file:** Cloudinary được sử dụng để lưu trữ và phục vụ các file đa phương tiện như ảnh bìa và video âm nhạc.
- **Giao tiếp:** CORS Headers cho phép giao tiếp an toàn giữa Front-end và Back-end.

2. Các mẫu thiết kế đã triển khai

Dự án áp dụng các mẫu thiết kế sau để đảm bảo tính mô-đun và dễ bảo trì:

- **Mô hình MVC (Model-View-Controller):** Được áp dụng trong Django, trong đó:
 - **Model:** Định nghĩa cấu trúc dữ liệu (ví dụ: bảng User, Song, Playlist) và giao tiếp với MySQL.
 - **View:** Xử lý logic API (Django REST Framework) để trả về dữ liệu dạng JSON cho Front-end.
 - **Controller:** Điều khiển luồng xử lý yêu cầu thông qua các URL và view.
- **Component-Based Architecture:** Ở Front-end, React sử dụng mô hình dựa trên thành phần (component), cho phép tái sử dụng các thành phần UI như nút phát nhạc, sidebar, hoặc bảng danh sách bài hát.
- **Repository Pattern:** Được áp dụng trong Back-end để tách biệt logic truy cập dữ liệu (MySQL, Cloudinary) khỏi logic nghiệp vụ, giúp dễ dàng thay đổi cơ sở dữ liệu trong tương lai.
- **State Management Pattern:** Redux được sử dụng để quản lý trạng thái ứng dụng ở Front-end, đảm bảo dữ liệu như danh sách bài hát hoặc trạng thái phát nhạc được đồng bộ trên toàn ứng dụng.

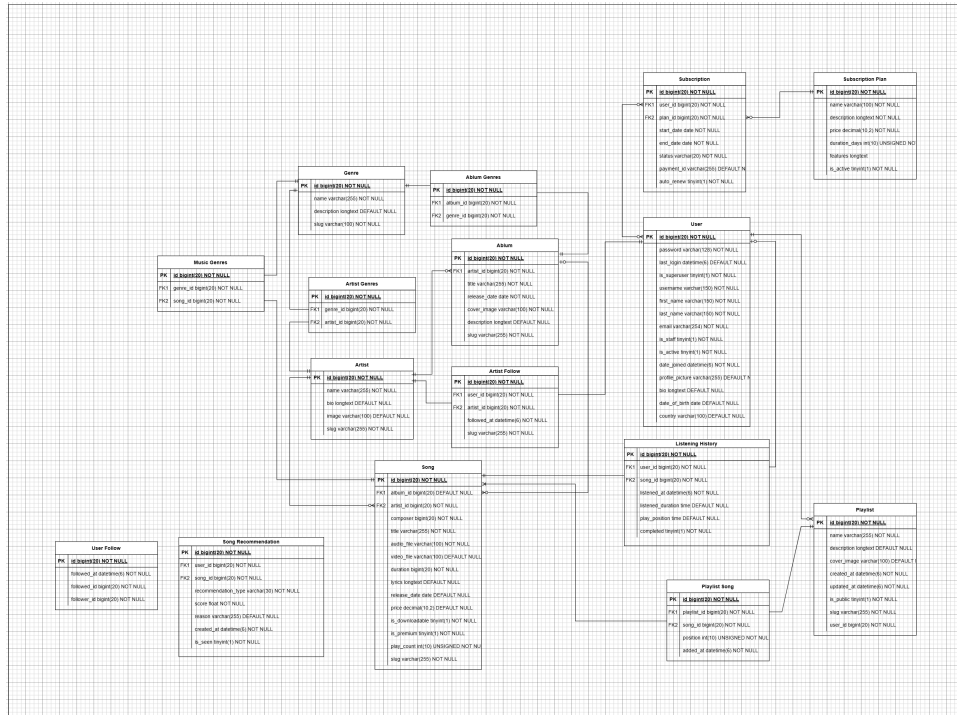
Các mẫu thiết kế này giúp hệ thống dễ dàng mở rộng, bảo trì và tích hợp các tính năng mới như chat hoặc quản lý người dùng nâng cao.

III. Thiết kế cơ sở dữ liệu

1. Sơ đồ cơ sở dữ liệu (sơ đồ ER)

Sơ đồ ER (Entity-Relationship Diagram) của hệ thống Spotify Clone được thiết kế để hỗ trợ các tính năng chính như quản lý người dùng, bài hát, danh sách phát, và lịch sử nghe. Sơ đồ ERD bao gồm các thực thể chính và mối quan hệ giữa chúng, được trình bày trong hình dưới đây:

Mô tả chi tiết về sơ đồ ER:



Hình 2.1: Sơ đồ ERD của hệ thống Spotify Clone

• Thực thể chính (Entities):

- **User**: Đại diện cho người dùng, với các thuộc tính như id, username, email, password, first_name, last_name, is_active, v.v.
- **Artist**: Đại diện cho nghệ sĩ, với các thuộc tính như id, name, bio, image, v.v.
- **Genre**: Đại diện cho thể loại nhạc, với các thuộc tính như id, name, description, v.v.
- **Album**: Đại diện cho album, với các thuộc tính như id, title, artist_id, release_date, cover_image, v.v.
- **Song**: Đại diện cho bài hát, với các thuộc tính như id, title, artist_id, album_id, genre_id, duration, audio_file, v.v.
- **Playlist**: Đại diện cho danh sách phát, với các thuộc tính như id, name, description, user_id, cover_image, v.v.
- **Subscription_Plan**: Đại diện cho gói đăng ký, với các thuộc tính như id, name, description, price, duration_day, v.v.
- **Listening_History**: Đại diện cho lịch sử nghe, với các thuộc tính như id, user_id, song_id, listened_at, v.v.
- **Song_Recommendation**: Đại diện cho gợi ý bài hát, với các thuộc tính như id, user_id, song_id, recommendation_type, v.v.

• Mối quan hệ (Relationships):

- **User - Artist**: Quan hệ nhiều-nhiều qua bảng trung gian Artist_Follow (thuộc tính: user_id, artist_id).

- **User - Playlist:** Quan hệ 1-nhiều, một User có thể tạo nhiều Playlist (`user_id` là khóa ngoại trong bảng Playlist).
- **Playlist - Song:** Quan hệ nhiều-nhiều qua bảng trung gian Playlist_Song (thuộc tính: `playlist_id`, `song_id`, `position`).
- **Song - Album:** Quan hệ nhiều-1, một Album có thể chứa nhiều Song (`album_id` là khóa ngoại trong bảng Song).
- **Song - Genre:** Quan hệ nhiều-1, một Genre có thể áp dụng cho nhiều Song (`genre_id` là khóa ngoại trong bảng Song).
- **Song - Artist:** Quan hệ nhiều-1, một Artist có thể có nhiều Song (`artist_id` là khóa ngoại trong bảng Song).
- **User - Listening_History:** Quan hệ 1-nhiều, một User có nhiều bản ghi lịch sử nghe (`user_id` là khóa ngoại trong bảng Listening_History).
- **User - Subscription_Plan:** Quan hệ nhiều-1, một Subscription_Plan áp dụng cho nhiều User (`subscription_plan_id` là khóa ngoại trong bảng User).
- **User - User:** Quan hệ nhiều-nhiều qua bảng trung gian User_Follow (thuộc tính: `follower_id`, `followed_id`).

2. Các bảng và mối quan hệ

Dưới đây là mô tả chi tiết các bảng, thuộc tính, và mối quan hệ trong cơ sở dữ liệu:

• Bảng User

- `id`: Khóa chính (`bigint`, NOT NULL).
- `username`, `email`: Khóa duy nhất (`varchar`, NOT NULL).
- `password`, `first_name`, `last_name`: Thuộc tính bắt buộc (`varchar`, NOT NULL).
- `subscription_plan_id`: Khóa ngoại tham chiếu đến Subscription_Plan (`bigint`, NULL).
- Các thuộc tính khác: `is_active`, `created_at`, `updated_at`.
- **Mối quan hệ:**
 - * Quan hệ 1-nhiều với Playlist, Listening_History.
 - * Quan hệ nhiều-nhiều với Artist qua Artist_Follow.
 - * Quan hệ nhiều-nhiều với User qua User_Follow.

• Bảng Artist

- `id`: Khóa chính (`bigint`, NOT NULL).
- `name`: Bắt buộc (`varchar`, NOT NULL).
- `bio`, `image`: Không bắt buộc (`varchar`, DEFAULT NULL).
- **Mối quan hệ:**
 - * Quan hệ 1-nhiều với Song, Album.
 - * Quan hệ nhiều-nhiều với User qua Artist_Follow.

- **Bảng Genre**

- id: Khóa chính (**bigint**, NOT NULL).
- name: Bắt buộc (**varchar**, NOT NULL).
- description: Không bắt buộc (**varchar**, DEFAULT NULL).
- **Mối quan hệ:** Quan hệ 1-nhiều với Song.

- **Bảng Album**

- id: Khóa chính (**bigint**, NOT NULL).
- title: Bắt buộc (**varchar**, NOT NULL).
- artist_id: Khóa ngoại tham chiếu đến Artist (**bigint**, NOT NULL).
- release_date, cover_image: Không bắt buộc (**date**, **varchar**, DEFAULT NULL).
- **Mối quan hệ:** Quan hệ 1-nhiều với Song.

- **Bảng Song**

- id: Khóa chính (**bigint**, NOT NULL).
- title: Bắt buộc (**varchar**, NOT NULL).
- artist_id, album_id, genre_id: Khóa ngoại tham chiếu đến Artist, Album, Genre (**bigint**, NOT NULL).
- duration, audio_file: Bắt buộc (**bigint**, **varchar**, NOT NULL).
- lyrics, release_date: Không bắt buộc (**text**, **date**, DEFAULT NULL).
- **Mối quan hệ:**
 - * Quan hệ nhiều-1 với Artist, Album, Genre.
 - * Quan hệ nhiều-nhiều với Playlist qua Playlist_Song.
 - * Quan hệ 1-nhiều với Listening_History, Song_Recommendation.

- **Bảng Playlist**

- id: Khóa chính (**bigint**, NOT NULL).
- name: Bắt buộc (**varchar**, NOT NULL).
- user_id: Khóa ngoại tham chiếu đến User (**bigint**, NOT NULL).
- description, cover_image: Không bắt buộc (**varchar**, DEFAULT NULL).
- **Mối quan hệ:**
 - * Quan hệ nhiều-1 với User.
 - * Quan hệ nhiều-nhiều với Song qua Playlist_Song.

- **Bảng Subscription_Plan**

- id: Khóa chính (**bigint**, NOT NULL).
- name: Bắt buộc (**varchar**, NOT NULL).
- price, duration_day: Bắt buộc (**decimal**, **int**, NOT NULL).

- `description, features`: Không bắt buộc (`varchar, text, DEFAULT NULL`).
- **Mối quan hệ**: Quan hệ 1-nhiều với User.
- **Bảng Listening_History**
 - `id`: Khóa chính (`bigint, NOT NULL`).
 - `user_id, song_id`: Khóa ngoại tham chiếu đến User và Song (`bigint, NOT NULL`).
 - `listened_at`: Bắt buộc (`datetime, NOT NULL`).
 - **Mối quan hệ**: Quan hệ nhiều-1 với User và Song.
- **Bảng Song_Recommendation**
 - `id`: Khóa chính (`bigint, NOT NULL`).
 - `user_id, song_id`: Khóa ngoại tham chiếu đến User và Song (`bigint, NOT NULL`).
 - `recommendation_type, created_at`: Bắt buộc (`varchar, datetime, NOT NULL`).
 - **Mối quan hệ**: Quan hệ nhiều-1 với User và Song.
- **Bảng Artist_Follow**
 - `user_id, artist_id`: Khóa ngoại tham chiếu đến User và Artist (`bigint, NOT NULL`).
 - **Mối quan hệ**: Bảng trung gian cho quan hệ nhiều-nhiều giữa User và Artist.
- **Bảng Playlist_Song**
 - `playlist_id, song_id`: Khóa ngoại tham chiếu đến Playlist và Song (`bigint, NOT NULL`).
 - `position`: Vị trí bài hát trong danh sách phát (`int, NOT NULL`).
 - **Mối quan hệ**: Bảng trung gian cho quan hệ nhiều-nhiều giữa Playlist và Song.
- **Bảng User_Follow**
 - `follower_id, followed_id`: Khóa ngoại tham chiếu đến User (`bigint, NOT NULL`).
 - **Mối quan hệ**: Bảng trung gian cho quan hệ nhiều-nhiều giữa các User.

3. Phương pháp mô hình hóa dữ liệu

Cơ sở dữ liệu của Spotify Clone được thiết kế theo mô hình quan hệ (Relational Model), sử dụng MySQL làm hệ quản trị cơ sở dữ liệu. Phương pháp mô hình hóa dữ liệu được áp dụng như sau:

- **Phân tích yêu cầu**: Dựa trên các tính năng của Spotify Clone (quản lý người dùng, bài hát, danh sách phát, lịch sử nghe, v.v.), các thực thể chính và mối quan hệ giữa chúng được xác định.
- **Chuẩn hóa dữ liệu (Normalization)**:

- **1NF (First Normal Form)**: Đảm bảo tất cả các thuộc tính trong bảng đều là giá trị nguyên tử (atomic). Ví dụ, thuộc tính `duration` trong bảng `Song` là một số nguyên, không chứa giá trị phức hợp.
- **2NF (Second Normal Form)**: Loại bỏ phụ thuộc hàm một phần. Các bảng như `Song` đảm bảo mọi thuộc tính không phải khóa (như `title`, `duration`) phụ thuộc hoàn toàn vào khóa chính `id`.
- **3NF (Third Normal Form)**: Loại bỏ phụ thuộc bắc cầu. Ví dụ, trong bảng `Album`, thuộc tính `artist_id` tham chiếu đến `Artist`, không lưu trữ trực tiếp thông tin nghệ sĩ (như `name`) để tránh dư thừa dữ liệu.
- **Tối ưu hóa truy vấn**:
 - Các khóa ngoại (foreign key) như `user_id`, `song_id` được sử dụng để liên kết các bảng, đảm bảo truy vấn nhanh chóng. Ví dụ, truy vấn lịch sử nghe của một User có thể dễ dàng thực hiện bằng cách join bảng `Listening_History` với `User` và `Song`.
 - Các trường như `id` được đặt làm khóa chính với kiểu `bigint` để hỗ trợ số lượng lớn bản ghi.
- **Quản lý file đa phương tiện**: File âm thanh và hình ảnh (như `audio_file`, `cover_image`) được lưu trữ trên Cloudinary, chỉ lưu đường dẫn (URL) trong cơ sở dữ liệu để giảm tải cho MySQL.

Phương pháp này đảm bảo cơ sở dữ liệu được tổ chức hợp lý, giảm dư thừa dữ liệu, và hỗ trợ hiệu quả các tính năng của ứng dụng Spotify Clone.

Chương 3: Hiện thực

I. Triển khai các tính năng

Dự án được phát triển theo mô hình MVC kết hợp với kiến trúc RESTful và chia frontend - backend rõ ràng. Các tính năng chính đã được hiện thực bao gồm:

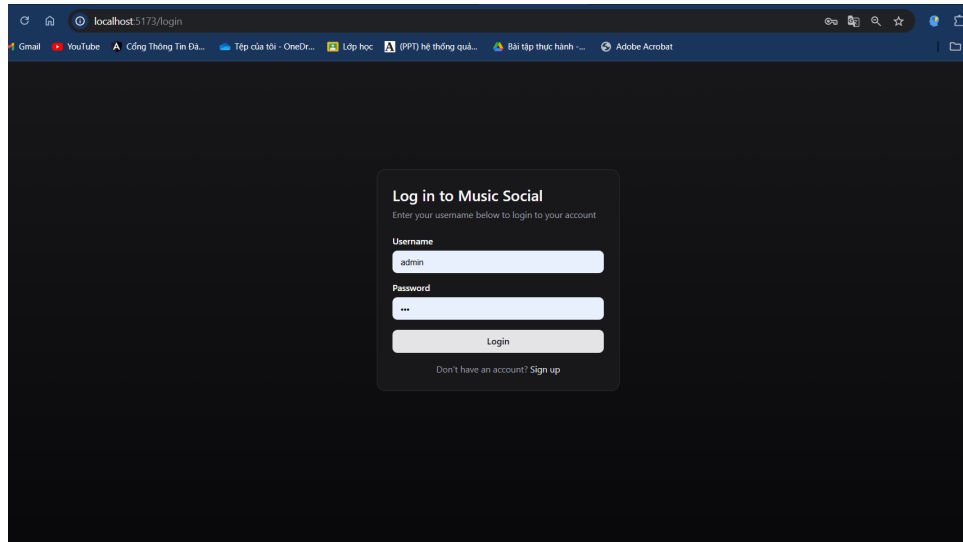
- Đăng ký, đăng nhập người dùng.
- Tìm kiếm bài hát, album, nghệ sĩ.
- Nghe nhạc, thêm vào playlist, đánh dấu yêu thích.
- Xem video bài hát, tải video bài hát.
- Hệ thống quản trị nội dung (admin) để thêm, sửa, xóa dữ liệu nhạc.
- Tương tác với ứng dụng thông qua chức năng chat.
- Quản lý playlist, bộ sưu tập cá nhân.

II. Ảnh chụp màn hình và minh họa

1. Giao diện người dùng

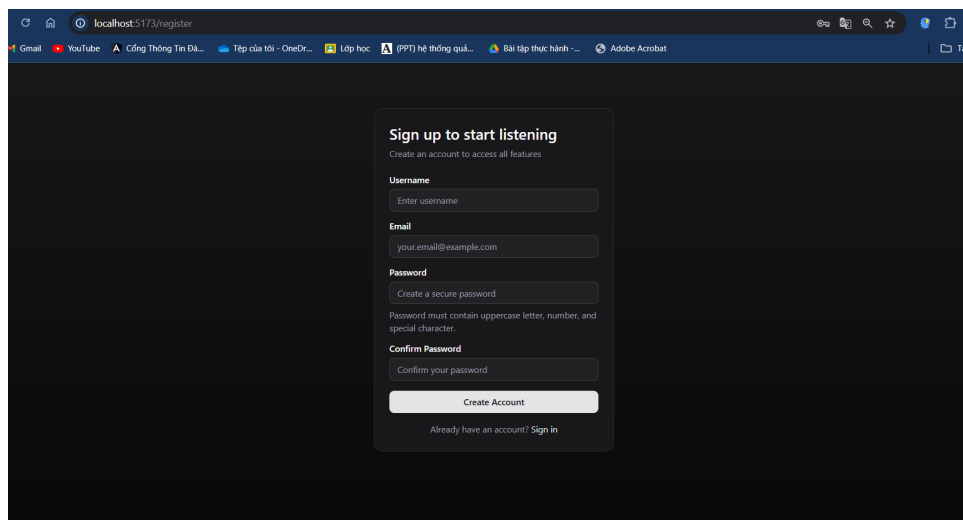
Dưới đây là các ảnh chụp màn hình minh họa giao diện người dùng:

- **Màn hình đăng nhập:** Hiển thị form đăng nhập với các trường username và password.



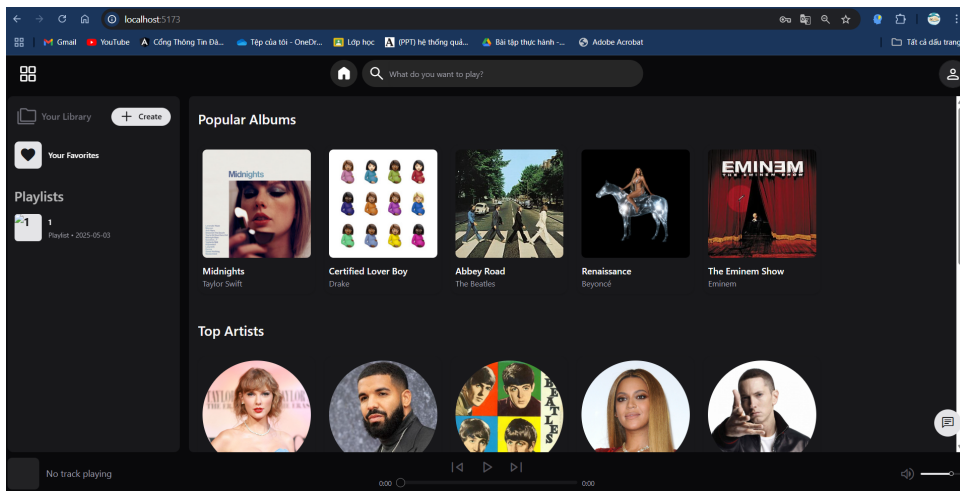
Hình 3.1: Màn hình đăng nhập

- **Màn hình đăng ký:** Hiển thị form đăng ký cho người dùng mới với các trường thông tin cơ bản.



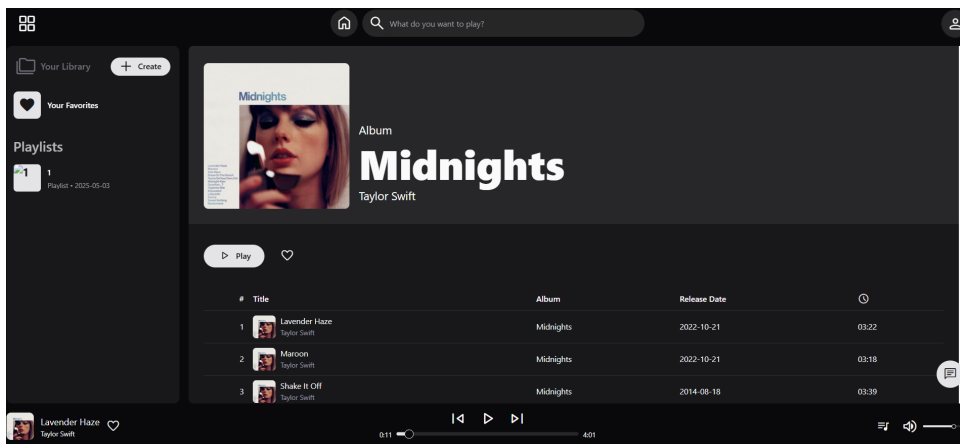
Hình 3.2: Màn hình đăng ký

- **Màn hình trang chủ:** Hiển thị giao diện sau khi đăng nhập thành công, bao gồm danh sách bài hát và nghệ sĩ.



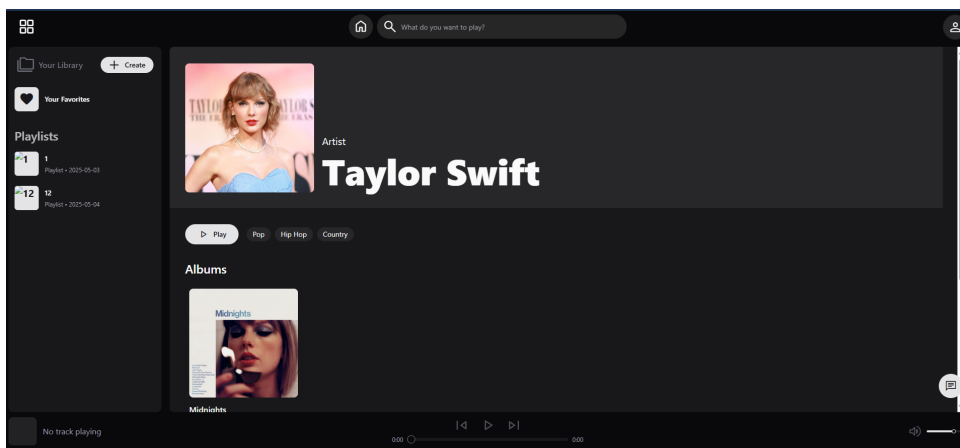
Hình 3.3: Màn hình trang chủ

- **Màn hình album:** Hiển thị giao diện sau khi nhấn chọn một album.



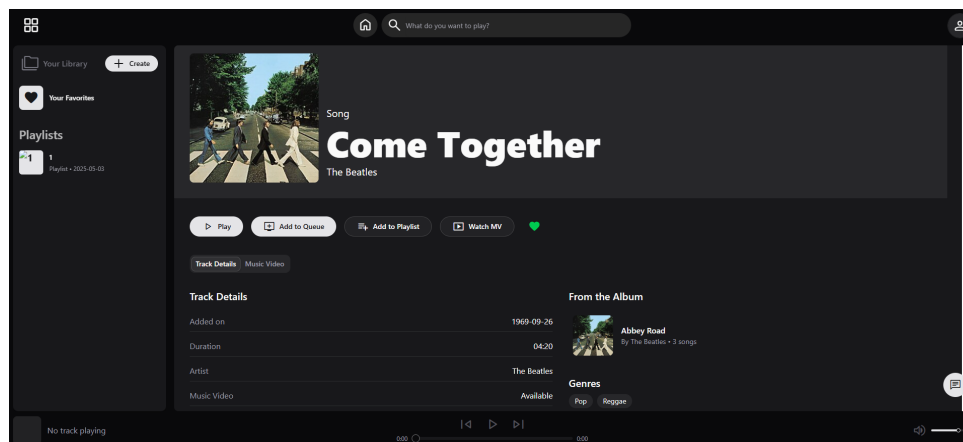
Hình 3.4: Màn hình album

- **Màn hình thông tin Artist:** Hiển thị giao diện thông tin của artist.

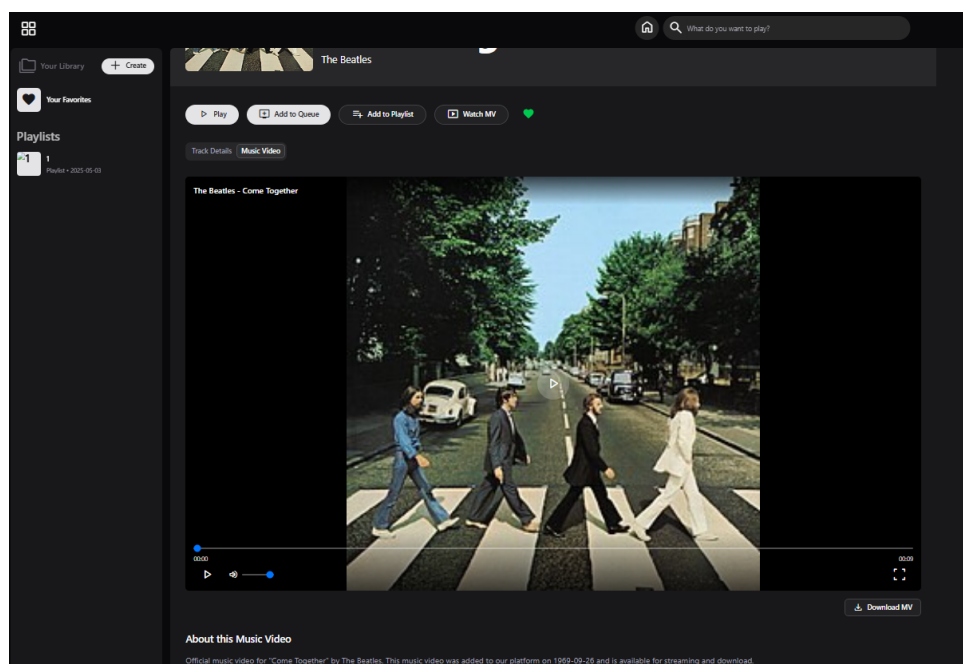


Hình 3.5: Màn hình artist

- **Màn hình thông tin bài hát:** Hiển thị giao diện thông tin bài hát và video âm nhạc.

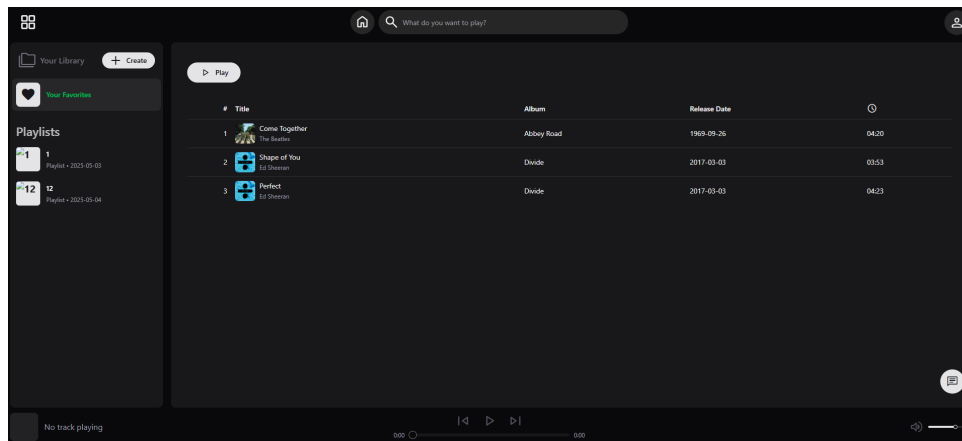


Hình 3.6: Màn hình thông tin bài hát



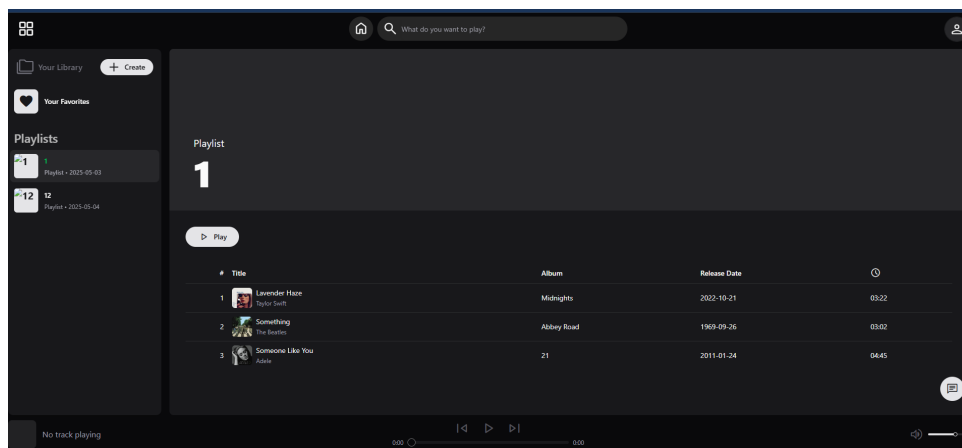
Hình 3.7: Màn hình xem video bài hát

- **Màn hình chứa các bài hát yêu thích:** Hiển thị màn hình chứa các bài hát mà người dùng yêu thích.



Hình 3.8: Màn hình Your Favorites

- **Màn hình playlist:** Hiển thị màn hình có các bài hát mà người dùng cho vào playlist

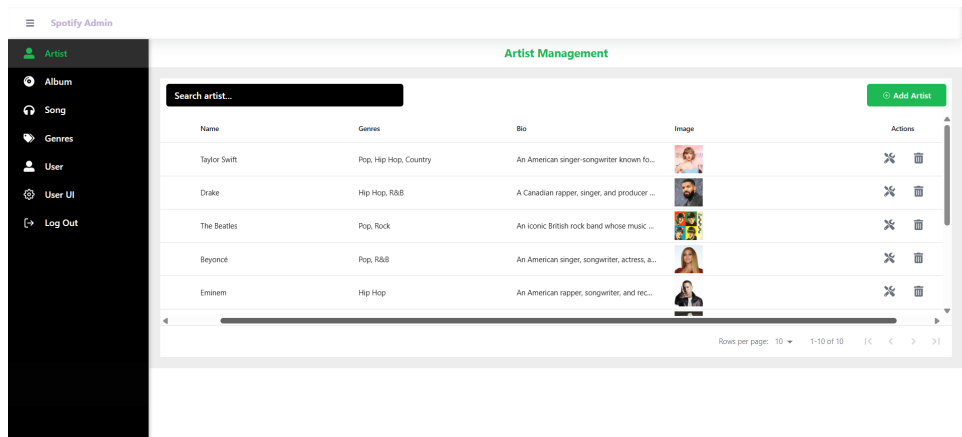


Hình 3.9: Màn hình Playlist

2. Giao diện trang Admin

Dưới đây là các ảnh chụp màn hình minh họa giao diện trang Admin:

- **Màn hình trang admin:** Hiển thị giao diện của trang admin quản lý thêm, sửa, xóa các mục như User, Artist, Album, Song, Genres.



Hình 3.10: Màn hình trang admin (quản lý Artist)

- **Màn hình chức năng thêm mới:** Hiện thị màn hình thêm mới một hạng mục.

Add Artist

Name

Enter artist name

Bio

Enter artist bio

Genres

Select genres

Artist Image

Chọn tệp Không có tệp nào được chọn

Close

Save Artist

Hình 3.11: Chức năng thêm mới

- **Màn hình chức năng chỉnh sửa:** Hiện thị màn hình chỉnh sửa một hạng mục.

Edit Artist

Name

Taylor Swift

Bio


An American singer-songwriter known for narrative songs and genre-span

Genres

Pop x Country x

Artist Image

Chọn tệp Không có tệp nào được chọn



Close

Update Artist

Hình 3.12: Chức năng chỉnh sửa

- **Màn hình chức năng xóa:** Hiển thị màn hình xóa một hạng mục.

Confirm Deletion

Are you sure you want to delete **artist**?

Cancel

Delete

Hình 3.13: Chức năng xóa

III. Cấu trúc mã nguồn

1. Tổ chức thư mục

Dự án gồm 3 phần chính:

- **backend/**: Chứa toàn bộ mã nguồn phía server (Django, REST API, chat, auth...).
- **frontend/**: Giao diện người dùng được xây dựng bằng React + TypeScript.
- **diagrams/**: Chứa sơ đồ ERD và các tài liệu trực quan liên quan đến cơ sở dữ liệu.

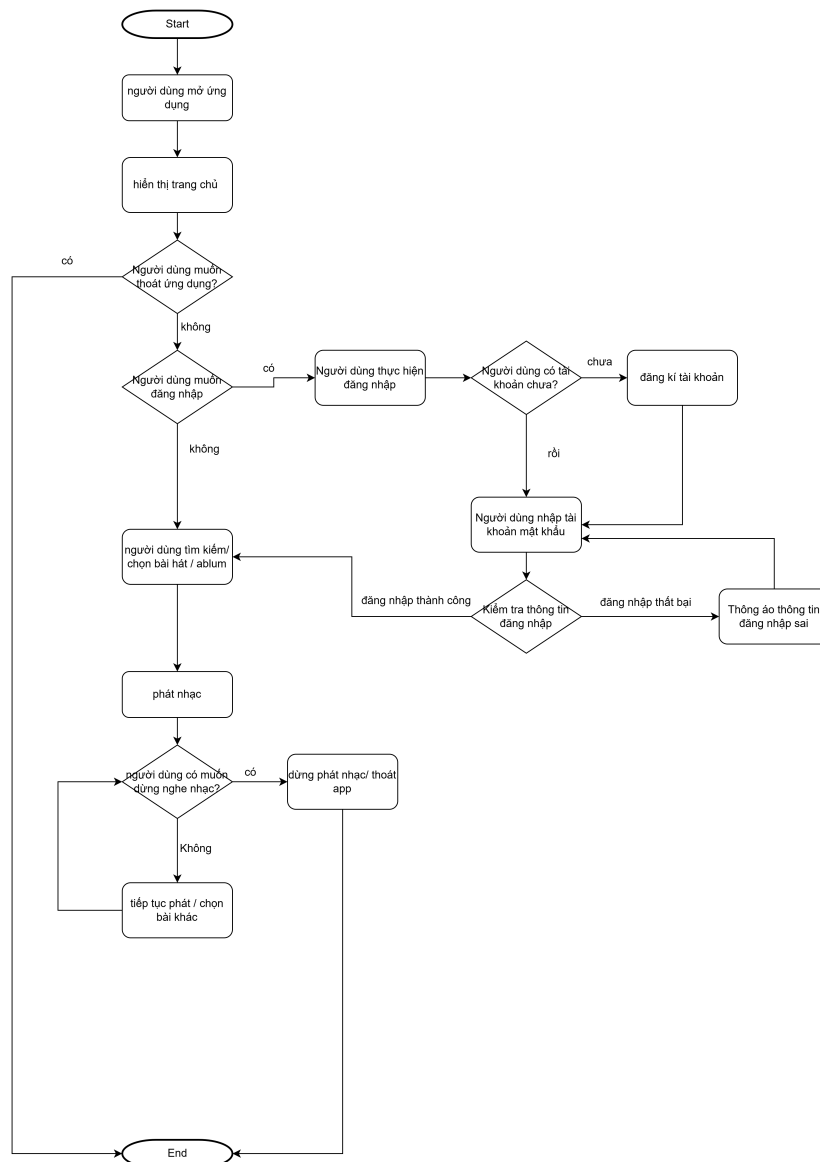
Các thư mục chức năng trong **backend/** được tách riêng như **chat**, **comment**, **music**, **playlist**, **purchase**, **subscription**, **user**, giúp dễ dàng mở rộng và bảo trì.

2. Các thành phần chính và tương tác

- Frontend gửi các yêu cầu HTTP đến Backend thông qua các service tương ứng (**albumService.ts**, **userService.ts**, **authService.ts**,...).
- Backend xử lý yêu cầu, xác thực người dùng, thao tác với cơ sở dữ liệu và trả dữ liệu về Frontend.
- Django Rest Framework là nền tảng chính cho backend API.
- Các modules như **favorite**, **playlist**, **music**, **user** có **models**, **serializers**, **views**, và **urls** riêng biệt để tuân theo nguyên tắc tách biệt và dễ mở rộng.
- Giao diện admin quản lý nội dung sử dụng route riêng trong Frontend (**/admin**).

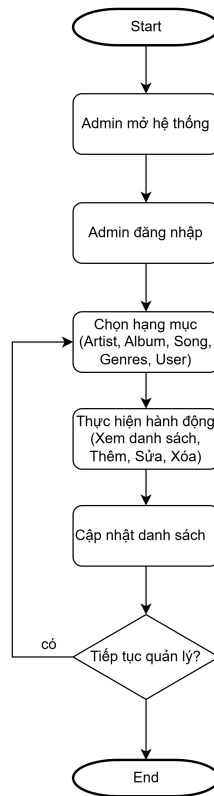
3. Flowchart cho các quy trình phức tạp

Dưới đây là flowchart minh họa quy trình người dùng sử dụng ứng dụng Spotify Clone:



Hình 3.14: Flowchart quy trình người dùng sử dụng ứng dụng Spotify Clone

Dưới đây là flowchart minh họa quy trình người quản trị thực hiện quản lý các hạng mục (Artist, Album, Song, Genres, User):



Hình 3.15: Flowchart quy trình người quản trị thực hiện quản lý các hạng mục

Chương 4: Cài đặt và triển khai

I. Yêu cầu hệ thống

Để chạy ứng dụng Spotify Clone trên máy cục bộ hoặc triển khai lên server, hệ thống cần đáp ứng các yêu cầu phần cứng và phần mềm sau:

1. Yêu cầu phần cứng

- **RAM:** Tối thiểu 4GB, khuyến nghị 8GB để đảm bảo hiệu suất khi chạy Docker và các dịch vụ liên quan.
- **CPU:** Tối thiểu 2GHz, khuyến nghị 3GHz hoặc cao hơn để xử lý nhanh các tác vụ Back-end và Front-end.
- **Dung lượng ổ cứng:** Tối thiểu 10GB để lưu trữ mã nguồn, file âm nhạc và video, và các container Docker.
- **Kết nối mạng:** Cần kết nối Internet ổn định để tải các dependency, truy cập Cloudinary (lưu trữ file âm nhạc và video), và triển khai ứng dụng lên AWS.

2. Yêu cầu phần mềm

- **Hệ điều hành:** Hỗ trợ Windows 10/11 (dựa trên thông tin bạn cung cấp), Linux (Ubuntu 20.04 trở lên), hoặc macOS (12 trở lên). Hướng dẫn cài đặt trong chương này tập trung vào Windows.
- **Phần mềm cần cài đặt trước:**
 - **Docker Desktop:** Dùng để chạy ứng dụng trong container (phiên bản mới nhất, hỗ trợ Windows).
 - **Trình duyệt:** Hỗ trợ các trình duyệt hiện đại như Google Chrome (90+), Firefox (85+), hoặc Microsoft Edge (90+) để truy cập giao diện Front-end.
- **Dịch vụ bên thứ ba:**
 - Tài khoản Cloudinary: Cần API key và secret để lưu trữ và truy xuất file âm nhạc và video.
 - Tài khoản AWS: Cần để triển khai ứng dụng lên AWS (EC2, ECS, hoặc dịch vụ tương tự).

II. Các bước cài đặt

Dưới đây là các bước cài đặt ứng dụng Spotify Clone trên máy cục bộ sử dụng Windows và Docker.

1. Cài đặt Docker Desktop

Truy cập trang chính thức của Docker tại <https://www.docker.com/products/docker-desktop/>. Tải và cài đặt Docker Desktop cho Windows (yêu cầu Windows 10/11 Pro hoặc Enterprise, hoặc bật WSL 2 nếu dùng Windows Home).

Sau khi cài đặt, khởi động Docker Desktop và đảm bảo nó đang chạy (biểu tượng Docker xuất hiện trên thanh taskbar).

2. Clone mã nguồn từ repository

Mở Command Prompt hoặc PowerShell.

Clone repository của dự án bằng lệnh:

```
git clone https://github.com/trucpham04/PTPMNM.git
```

Di chuyển vào thư mục dự án:

```
cd PTPMMNM
```

3. Chạy ứng dụng bằng Docker

Đảm bảo bạn đã có file `Dockerfile` và `docker-compose.yml`.

Mở Command Prompt hoặc PowerShell trong thư mục dự án.

Chạy lệnh sau để build và khởi động ứng dụng:

```
docker-compose up --build
```

Lệnh này sẽ:

- Build các container cho Front-end và Back-end.
- Khởi chạy ứng dụng với cấu hình trong `docker-compose.yml`.

Sau khi chạy thành công, bạn có thể truy cập:

- Front-end tại: <http://localhost:3000> (hoặc cổng được cấu hình trong `docker-compose.yml`).
- Back-end tại: <http://localhost:8000/api> (hoặc cổng được cấu hình).

III. Cấu hình

Dưới đây là các bước cấu hình cần thiết để ứng dụng Spotify Clone hoạt động đúng sau khi cài đặt.

1. Cấu hình Front-end

Proxy: File `vite.config.js` đã được cấu hình proxy để xử lý các yêu cầu API từ Front-end đến Back-end:

```

server: {
  proxy: {
    "/api": {
      target: "http://localhost:8000",
      changeOrigin: true,
      rewrite: (path) => path.replace(/^\/api/, ""),
    },
  },
},
},

```

Cấu hình này đảm bảo mọi yêu cầu `/api` từ Front-end (như `/api/songs`) sẽ được chuyển tiếp đến Back-end tại `http://localhost:8000` mà không gặp lỗi CORS.

2. Cấu hình Back-end

MySQL: Dựa trong `settings.py`, ứng dụng sử dụng MySQL làm cơ sở dữ liệu. Trong môi trường Docker, MySQL được cấu hình qua `docker-compose.yml`. Đảm bảo container MySQL chạy đúng với các thông tin như:

- Tên database: `spotifyclone`.
- User: `root` (hoặc user tùy chỉnh).
- Password: Được khai báo trong `docker-compose.yml`.

Sau khi container MySQL chạy, cần chạy migration để tạo các bảng:

```
docker-compose exec backend python manage.py migrate
```

Cloudinary: Ứng dụng sử dụng Cloudinary để lưu trữ file âm nhạc và video. Cần đăng ký tài khoản Cloudinary tại <https://cloudinary.com> và lấy thông tin cấu hình gồm:

- Cloud name.
- API key.
- API secret.

Thông tin này cần được thêm vào `docker-compose.yml` dưới dạng biến môi trường cho container Back-end:

```
CLOUDINARY_URL=cloudinary://<api_key>:<api_secret>@<cloud_name>
```

Ví dụ:

```
CLOUDINARY_URL=cloudinary://123456789:abc123@mycloudname
```

Sau khi thêm, khởi động lại container Back-end:

```
docker-compose up --build
```

IV. Hướng dẫn triển khai

Dưới đây là hướng dẫn triển khai ứng dụng Spotify Clone lên AWS sử dụng Docker.

1. Chuẩn bị môi trường AWS

Tạo tài khoản AWS: Đăng ký tại <https://aws.amazon.com> nếu chưa có tài khoản.

Tạo EC2 instance:

- Truy cập AWS Management Console, vào dịch vụ EC2.
- Tạo một instance mới:
 - Chọn AMI: **Amazon Linux 2** hoặc **Ubuntu 20.04**.
 - Loại instance: **t2.micro** (miễn phí cho tài khoản mới) hoặc **t2.medium** (khuyến nghị cho ứng dụng).
 - Tạo key pair để SSH vào instance (lưu file **.pem**).
 - Cấu hình Security Group: Mở các cổng 80 (HTTP), 3000 (Front-end), và 8000 (Back-end).
- Kết nối đến instance qua SSH:

```
ssh -i <your-key.pem> ec2-user@<instance-public-ip>
```

2. Cài đặt Docker trên EC2 instance

Cài đặt Docker trên Amazon Linux 2:

```
sudo yum update -y
sudo amazon-linux-extras install docker
sudo service docker start
sudo usermod -a -G docker ec2-user
```

Cài đặt Docker Compose:

```
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)"
sudo chmod +x /usr/local/bin/docker-compose
```

Đăng xuất và đăng nhập lại để áp dụng quyền Docker.

3. Chuẩn bị mã nguồn

Clone mã nguồn lên EC2 instance:

```
git clone https://github.com/trucpham04/PTPMNM.git
cd PTPMMNM
```

4. Triển khai ứng dụng

Build và chạy ứng dụng:

```
docker-compose up --build
```

Sau khi build thành công, ứng dụng sẽ chạy:

- Front-end tại: `http://<instance-public-ip>:3000`.
- Back-end tại: `http://<instance-public-ip>:8000/api`.

Chương 5: Đóng góp của thành viên

I. Phân công nhiệm vụ

Thành viên	Nhiệm vụ chính
Nguyễn Anh Khoa	Phát triển chính phần backend, trang quản trị (admin)
Phạm Duy Trực	Phát triển chính phần UI (frontend)
Phạm Thị Minh Thư	Phát triển chức năng chat (frontend và backend)
Kim Duy Linh	Viết và hoàn thiện tài liệu báo cáo
Huỳnh Tuyết Mai	Viết và hoàn thiện tài liệu báo cáo

Bảng 5.1: Phân công nhiệm vụ của các thành viên

II. Đóng góp chi tiết của từng thành viên

1. Nguyễn Anh Khoa

Chịu trách nhiệm phát triển phần backend của hệ thống, bao gồm các API phục vụ cho các chức năng chính như quản lý người dùng, bài hát, nghệ sĩ, album,... Ngoài ra, Nguyễn Anh Khoa cũng xây dựng và triển khai trang quản trị (admin).

2. Phạm Thị Minh Thư

Phụ trách phát triển tính năng chat của hệ thống, bao gồm cả frontend và backend. Thư đã thiết kế hệ thống gửi và nhận tin nhắn tăng trải nghiệm của người dùng.

3. Phạm Duy Trực

Phát triển phần giao diện người dùng cho toàn bộ hệ thống (frontend). Trực chịu trách nhiệm thiết kế bố cục, giao diện trang chủ, trang chi tiết bài hát, phát nhạc, phát và tải video,... và các chức năng tương tác với backend qua API.

4. Kim Duy Linh

Đảm nhiệm việc viết tài liệu báo cáo của nhóm, bao gồm mô tả kiến trúc hệ thống, phân tích chức năng, và tổng hợp nội dung các chương trong tài liệu hướng dẫn.

5. Huỳnh Tuyết Mai

Đảm nhiệm việc viết tài liệu báo cáo của nhóm, bao gồm mô tả kiến trúc hệ thống, phân tích chức năng, và tổng hợp nội dung các chương trong tài liệu hướng dẫn.

Chương 6: Kết luận

I. Tóm tắt dự án

Dự án Spotify Clone là một ứng dụng web mô phỏng các tính năng cốt lõi của nền tảng nghe nhạc trực tuyến Spotify, được xây dựng nhằm mục tiêu học tập và nghiên cứu về phát triển ứng dụng web full-stack. Dự án sử dụng **React** (v18.3.1) cho Front-end và **Django** (v5.1.7) cho Back-end, kết hợp với cơ sở dữ liệu MySQL và dịch vụ Cloudinary để lưu trữ file âm nhạc và video. Các công nghệ và thư viện hỗ trợ bao gồm Redux (quản lý trạng thái), Axios (gọi API), React Router DOM (điều hướng), Django REST Framework (xây dựng API), và nhiều thư viện khác để hỗ trợ giao diện và tính năng.

Ứng dụng đã triển khai thành công các tính năng chính như:

- Đăng ký và đăng nhập người dùng.
- Phát nhạc, tạm dừng, chuyển bài, và quản lý thanh phát nhạc.
- Tạo, chỉnh sửa và quản lý danh sách phát (Playlist).
- Theo dõi nghệ sĩ (Artist Follow) và xem thông tin nghệ sĩ.
- Tích hợp tính năng chat trong giao diện web Spotify Clone.
- Trang Admin để người dùng admin có thể quản lý các thông tin như User, Artist, Song, Album, Genres.
- Chức năng phát và tải video âm nhạc.

Dự án đã đạt được mục tiêu đề ra, bao gồm việc xây dựng một ứng dụng web có khả năng phát nhạc trực tuyến, quản lý người dùng, và cung cấp trải nghiệm người dùng tương tự Spotify. Ngoài ra, tính năng chat tích hợp và khả năng tải video âm nhạc đã mang lại giá trị gia tăng, giúp ứng dụng trở nên đa dạng và hấp dẫn hơn. Quá trình phát triển cũng giúp nhóm làm quen với các công nghệ hiện đại và quy trình triển khai ứng dụng lên môi trường production.

II. Hạn chế

Mặc dù dự án đã hoàn thành các tính năng cơ bản, vẫn còn một số hạn chế cần được ghi nhận:

- **Hiệu suất tải file đa phương tiện:** Việc sử dụng Cloudinary để lưu trữ và phục vụ file âm nhạc và video có thể gặp vấn đề về tốc độ tải nếu số lượng người dùng tăng cao hoặc băng thông mạng không ổn định.

- **Tính năng chưa đầy đủ:** Một số tính năng nâng cao của Spotify như phát nhạc offline, tích hợp trí tuệ nhân tạo để gợi ý bài hát chính xác hơn, hoặc hỗ trợ phát nhạc trên nhiều thiết bị cùng lúc chưa được triển khai.
- **Khả năng mở rộng:** Hệ thống hiện tại chưa được tối ưu để xử lý số lượng lớn người dùng đồng thời (scalability), đặc biệt là trong việc quản lý cơ sở dữ liệu và lưu trữ file trên Cloudinary.
- **Bảo mật:** Mặc dù đã áp dụng các biện pháp cơ bản như xác thực token (Django REST Framework), ứng dụng chưa triển khai các tính năng bảo mật nâng cao như mã hóa dữ liệu nhạy cảm hoặc chống tấn công DDoS.
- **Giao diện người dùng:** Giao diện tuy đã đáp ứng các chức năng cơ bản, nhưng vẫn cần cải thiện về tính thẩm mỹ và trải nghiệm người dùng (UX) để cạnh tranh với các ứng dụng thương mại như Spotify.

III. Cải tiến trong tương lai

Dựa trên các hạn chế đã nêu, dự án có thể được cải tiến trong tương lai theo các hướng sau:

- **Tối ưu hiệu suất:** Áp dụng các kỹ thuật như caching (sử dụng Redis) để tăng tốc độ truy xuất dữ liệu, và cân nhắc sử dụng CDN (Content Delivery Network) để cải thiện tốc độ tải file âm nhạc và video từ Cloudinary.
- **Thêm tính năng nâng cao:**
 - Hỗ trợ phát nhạc offline bằng cách lưu trữ tạm thời file âm nhạc trên thiết bị người dùng.
 - Tích hợp thuật toán học máy (Machine Learning) để gợi ý bài hát dựa trên sở thích và thói quen nghe nhạc của người dùng.
 - Hỗ trợ đồng bộ phát nhạc trên nhiều thiết bị (cross-device playback) thông qua WebSocket hoặc các giao thức tương tự.
- **Cải thiện khả năng mở rộng:**
 - Sử dụng Kubernetes để quản lý các container Docker, cho phép mở rộng hệ thống theo số lượng người dùng.
 - Chuyển đổi cơ sở dữ liệu sang hệ thống phân tán (như PostgreSQL với sharding) để xử lý số lượng lớn bản ghi.
- **Tăng cường bảo mật:**
 - Mã hóa dữ liệu nhạy cảm (như mật khẩu, thông tin cá nhân) bằng các thuật toán như AES.
 - Thêm các biện pháp chống tấn công DDoS, ví dụ: sử dụng AWS Shield hoặc Cloudflare.
 - Áp dụng xác thực hai yếu tố (2FA) cho người dùng để tăng cường bảo mật tài khoản.
- **Nâng cấp giao diện người dùng:**

- Thiết kế lại giao diện để tăng tính thẩm mỹ, sử dụng các thư viện UI hiện đại như Material-UI hoặc Ant Design.
- Tối ưu trải nghiệm người dùng bằng cách thêm các hiệu ứng chuyển động mượt mà (animation) và cải thiện khả năng điều hướng.
- **Tích hợp thêm dịch vụ bên thứ ba:** Ví dụ, tích hợp hệ thống thanh toán (Stripe, PayPal) để hỗ trợ mua gói đăng ký trực tiếp trên ứng dụng, hoặc tích hợp API của Spotify để lấy dữ liệu bài hát thực tế.

Những cải tiến này sẽ giúp Spotify Clone không chỉ trở thành một dự án học tập mà còn có tiềm năng phát triển thành một sản phẩm thương mại với khả năng cạnh tranh trên thị trường.