

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SÀI GÒN



ĐỀ CƯƠNG BÁO CÁO

MÔN: Phát triển phần mềm mã
nguồn mở

ĐỀ TÀI: Spotify Clone

Giảng viên hướng dẫn: ThS. Từ Lãng Phiêu

Sinh viên thực hiện:

- Phạm Tấn Đạt – MSSV: 3121410149
- Mộc Nghĩa Tân – MSSV: 3121410441
- Lê Bửu Trí – MSSV: 3121410521

Email liên hệ:

- phamtandat655@gmail.com
- moctan137@gmail.com
- lebuutri89@gmail.com

TP. Hồ Chí Minh, ngày 13 tháng 05 năm 2025

Mục lục

	Trang
1. Giới thiệu dự án	3
1.1 Mô tả tổng quan	3
1.2 Mục tiêu dự án	3
2. Công nghệ sử dụng	5
2.1 Công nghệ backend	5
2.1.1 Modular Monolith	5
2.1.2 RESTful API	10
2.2 Công nghệ frontend	5
2.2.1 Single Page Application	11
2.2.2 Component-Based Architecture	11
2.2.3 State Management và API Integration	11
3. Cấu trúc mã nguồn	7
3.1 Cấu trúc backend	7
3.2 Cấu trúc frontend	8
4. Mô hình ứng dụng	10
4.1 Mô hình backend	10
4.1.1 Modular Monolith	10
4.1.2 RESTful API	10
4.2 Mô hình frontend	10
4.2.1 Single Page Application	11
4.2.2 Component-Based Architecture	11
4.2.3 State Management và API Integration	11
5. Các tính năng	12
5.1 Tính năng chính	12
5.1.1 Frontend	12
5.1.2 Backend	13
5.2 Quy trình hoạt động (Flowchart)	15
5.2.1 Music App	15
5.2.2 User App	20
6. Xây dựng database	34
6.1 Cấu hình database	34
6.2 Mô hình database	34
6.2.1 Mô hình âm nhạc	34
6.2.2 Mô hình người dùng	35
7. Hiện thực	36
7.1 Tính năng đã triển khai	36
7.2 Giao diện minh họa	37

7.2.1 Giao diện dành cho Người Dùng	37
7.2.2 Giao diện dành cho Quản Trị Viên	42
8. Cài đặt và chạy ứng dụng	44
8.1 Cài đặt backend	44
8.2 Cài đặt frontend	45
8.3 Môi trường chạy	46
9. Phân công nhiệm vụ	47
10. Tài liệu tham khảo	48

1. Giới thiệu dự án

1.1 Mô tả tổng quan

Dự án **Spotify Clone** là một hệ thống ứng dụng phát nhạc trực tuyến được phát triển nhằm mô phỏng các chức năng chính của nền tảng Spotify – một trong những dịch vụ âm nhạc kỹ thuật số phổ biến nhất hiện nay. Ứng dụng được xây dựng với mục tiêu tạo ra trải nghiệm người dùng thân thiện, hiện đại, đồng thời đáp ứng được các nhu cầu cơ bản của người nghe nhạc trực tuyến như: phát bài hát theo thời gian thực, quản lý danh sách phát (playlist), tạo album cá nhân, yêu thích bài hát, tìm kiếm nghệ sĩ và bài hát, xem thông tin chi tiết của album và nghệ sĩ, cũng như phát video nhạc nếu có hỗ trợ.

Ngoài ra, hệ thống cũng bao gồm các tính năng liên quan đến quản lý người dùng, nơi người dùng có thể đăng ký tài khoản, đăng nhập, chỉnh sửa hồ sơ cá nhân và theo dõi hoạt động nghe nhạc. Dự án sử dụng kiến trúc client-server với **frontend** được phát triển bằng **ReactJS** và **backend** sử dụng **Django** – một framework mạnh mẽ của Python. Dữ liệu của hệ thống được lưu trữ và quản lý trong **PostgreSQL**. Tất cả các thành phần được triển khai nhằm tạo nên một nền tảng âm nhạc đầy đủ tính năng, vận hành mượt mà và dễ mở rộng trong tương lai.

1.2 Mục tiêu dự án

Mục tiêu chính của dự án **Spotify Clone** là xây dựng một hệ thống phát nhạc toàn diện, nơi người dùng có thể trải nghiệm các tính năng cơ bản đến nâng cao tương tự như một nền tảng âm nhạc thực tế. Cụ thể, dự án hướng đến việc:

- Tạo ra một nền tảng phát nhạc trực tuyến có giao diện người dùng trực quan, dễ sử dụng, hỗ trợ cả thiết bị máy tính và di động.
- Cung cấp chức năng phát nhạc theo danh sách, theo album, hoặc phát từng bài riêng lẻ với các điều khiển như tạm dừng, chuyển bài, phát lại,…
- Hỗ trợ người dùng tạo và quản lý album cá nhân (chỉ người tạo có thể chỉnh sửa hoặc xóa), cũng như tạo danh sách phát để sắp xếp các bài hát yêu thích.
- Cho phép người dùng tìm kiếm bài hát, nghệ sĩ hoặc album thông qua từ khóa; hiển thị kết quả tìm kiếm một cách chính xác và nhanh chóng.
- Tích hợp tính năng quản lý người dùng: đăng ký, đăng nhập, chỉnh sửa thông tin cá nhân, quản lý bài hát yêu thích và lịch sử nghe nhạc.

- Cung cấp giao diện quản trị viên (admin) để kiểm soát nội dung, người dùng và hệ thống thông tin tổng quát nhằm duy trì hoạt động ổn định của nền tảng.
- Hướng đến việc phát triển một ứng dụng có thể mở rộng trong tương lai như thêm tính năng bình luận, nhắm tin giữa người dùng, phát video âm nhạc, hoặc tích hợp API từ các bên thứ ba.

Qua việc hoàn thiện các mục tiêu trên, nhóm dự án không chỉ rèn luyện kỹ năng lập trình và phát triển phần mềm theo nhóm, mà còn tích lũy kinh nghiệm thực tế trong việc triển khai một sản phẩm web hoàn chỉnh từ frontend đến backend.

2. Công nghệ sử dụng

2.1 Công nghệ backend

- **Backend:** Django 4.2.20
- **API:** Django REST Framework
- **Database:** PostgreSQL
- **Xác thực:** JWT (Simple JWT)
- **Test:** Postman
- **Ảnh/Media Upload:** Pillow

2.2 Công nghệ frontend

- **Framework:** ReactJS 18 - Khung chính để xây dựng giao diện người dùng động và có thể tái sử dụng.
- **Quản lý trạng thái và API:** Redux Toolkit Query - Quản lý trạng thái ứng dụng và gọi API đến backend một cách hiệu quả.
- **Định tuyến:** React Router - Xử lý định tuyến phía client cho các trang như tìm kiếm, hồ sơ, và quản trị.
- **Tạo kiểu:** Tailwind CSS - Sử dụng các lớp tiện ích để tạo giao diện responsive và nhất quán.
- **Biểu đồ:** Chart.js và react-chartjs-2 - Hiển thị thống kê (tổng số bài hát, album) trong bảng điều khiển quản trị.
- **Yêu cầu HTTP:** Axios - Client HTTP để thực hiện các yêu cầu API đến backend.
- **Biểu tượng:** @heroicons/react - Cung cấp biểu tượng cho các nút điều khiển (phát, tạm dừng, thu nhỏ, v.v.).
- **Phát video:** HTML5 Video - Hỗ trợ phát các tệp MP4 trong trình phát nhạc.
- **Công cụ xây dựng:** Vite - Công cụ xây dựng nhanh cho phát triển và đóng gói ứng dụng.

2.3 Ứng dụng

Ứng dụng **Spotify Clone** là một nền tảng phát nhạc trực tuyến với các tính năng tương tự như Spotify. Nó hỗ trợ người dùng thực hiện các chức năng cơ bản như phát nhạc, tạo album, và quản lý người dùng. Cụ thể, ứng dụng cung cấp các tính năng sau:

- **Phát nhạc:** Người dùng có thể phát nhạc từ danh sách bài hát yêu thích, album, hoặc danh sách phát cá nhân. Các chức năng điều khiển phát nhạc như tạm dừng, chuyển bài, lặp lại, và điều chỉnh âm lượng đều được tích hợp.
- **Tạo album cá nhân:** Người dùng có thể tạo album và danh sách phát cá nhân, thêm bài hát vào album của mình. Mỗi album chỉ thuộc về một người dùng và người dùng có thể tự chỉnh sửa album của mình.
- **Quản lý người dùng:** Ứng dụng cho phép người dùng đăng ký tài khoản, đăng nhập, và chỉnh sửa hồ sơ cá nhân.
- **Tìm kiếm và khám phá âm nhạc:** Người dùng có thể tìm kiếm bài hát, nghệ sĩ, hoặc album thông qua tính năng tìm kiếm mạnh mẽ. Kết quả tìm kiếm được hiển thị theo từng loại (bài hát, album, nghệ sĩ), giúp người dùng dễ dàng khám phá nội dung âm nhạc mới.
- **Tích hợp video nhạc:** Ứng dụng hỗ trợ phát video âm nhạc định dạng **MP4** cho tất cả các bài hát, mang lại trải nghiệm giải trí phong phú và sinh động hơn cho người dùng, đồng thời giúp nâng cao chất lượng hình ảnh và âm thanh khi thưởng thức nhạc.

Thông qua những tính năng trên, ứng dụng không chỉ cung cấp một nền tảng phát nhạc đơn giản mà còn mang đến một trải nghiệm người dùng phong phú, đầy đủ các tính năng cho việc khám phá và thưởng thức âm nhạc.

3. Cấu trúc mã nguồn

3.1 Cấu trúc backend

Dự án backend được xây dựng bằng Django kết hợp Django REST Framework, tổ chức theo nguyên tắc modular, giúp mở rộng và bảo trì dễ dàng. Chi tiết:

1. **Thư mục gốc spotify_backend/**: chứa các cấu hình toàn cục và các app chức năng.
2. **Thư mục spotify_back_end/**: chứa cấu hình chính của Django:
 - `settings.py`: cấu hình cơ sở dữ liệu, ứng dụng, middleware, static/media.
 - `urls.py`: định tuyến chính của hệ thống.
 - `asgi.py`, `wsgi.py`: phục vụ chạy với server tương ứng.
3. **App api/**: xử lý các chức năng không thuộc về music hay user.
4. **App music/**:
 - `models.py`: các model như Artist, Album, Track, Genre,...
 - `views.py`: trả về danh sách bài hát, album, nghệ sĩ,...
 - `serializers/`: chứa các file:
 - `albums_serializers.py`
 - `artist_serializers.py`
 - `genre_serializers.py`
 - `tracks_serializers.py`
 - `utils.py`: hàm tiện ích phục vụ logic âm nhạc.
5. **App user/**:
 - `models.py`: User, UserFavouriteTrack, UserCreatedAlbum,...
 - `views.py`: đăng ký, đăng nhập, yêu thích,...
 - `serializers/`:
 - `User_Register.py`
 - `User_Serializer.py`
 - `UserCreatedAlbum_Serializer.py`
 - `User_FavouriteTracks.py`

6. Thư mục media/:

- `images/`: lưu ảnh album, nghệ sĩ, avatar,...
- `videos/`: lưu video nếu có.

7. Các thành phần khác:

- `manage.py`, `seed_data.py`, `requirements.txt`, `README.md`, `.gitignore`

3.2 Cấu trúc frontend

Frontend được xây dựng bằng ReactJS, tổ chức theo nguyên tắc modular để dễ dàng mở rộng và bảo trì. Chi tiết:

1. Thư mục gốc `spotify-clone/`: chứa các cấu hình dự án và mã nguồn frontend.

2. Thư mục `public/`: chứa các tài nguyên tĩnh:

- `index.html`: tệp HTML chính để khởi tạo ứng dụng React.
- `favicon.ico`: biểu tượng trang web.

3. Thư mục `src/`: chứa mã nguồn chính của ứng dụng:

- `components/`: các thành phần giao diện có thể tái sử dụng:
 - `Controls.jsx`: nút điều khiển phát/tạm dừng, tiếp theo/trước đó.
 - `Error.jsx`: hiển thị thông báo lỗi.
 - `ErrorBoundary.jsx`: xử lý lỗi trong cây thành phần.
 - `Loader.jsx`: hiển thị trạng thái đang tải.
 - `Player.jsx`: trình phát video HTML5 cho bài hát.
 - `SongCard.jsx`: hiển thị thông tin bài hát trong kết quả tìm kiếm.
 - `Track.jsx`: hiển thị chi tiết bài hát đang phát.
 - `VolumeBar.jsx`: thanh điều chỉnh âm lượng.
- `pages/`: các thành phần cấp trang cho các tuyến đường:
 - `AdminDashboard.jsx`: bảng điều khiển quản trị với thống kê bài hát/album.
 - `ArtistList.jsx`: danh sách nghệ sĩ và bài hát liên quan.
 - `CreateAlbum.jsx`: biểu mẫu tạo album tùy chỉnh.
 - `CreateTrack.jsx`: biểu mẫu tạo bài hát mới cho quản trị viên.
 - `Login.jsx`: giao diện đăng nhập người dùng.
 - `Profile.jsx`: hiển thị và chỉnh sửa hồ sơ người dùng.
 - `Register.jsx`: giao diện đăng ký tài khoản mới.
 - `Search.jsx`: trang tìm kiếm bài hát.
- `redux/`: quản lý trạng thái và API:
 - `features/`:
 - * `playerSlice.js`: quản lý trạng thái trình phát nhạc (bài hát đang phát, âm lượng, v.v.).

- **services/**:
 - * **spotifyApi.js**: định nghĩa các endpoint API (ví dụ: lấy bài hát, nghệ sĩ, tạo album).
- **store.js**: cấu hình Redux store với Redux Toolkit.
- **assets/**: tài nguyên tĩnh:
 - **logo.png**: logo ứng dụng (sử dụng trong giao diện).
- **App.jsx**: thành phần chính với thiết lập định tuyến React Router.
- **index.css**: tệp CSS chính với cấu hình Tailwind CSS.
- **index.js**: điểm vào của ứng dụng, khởi tạo React.

4. Các thành phần khác:

- **package.json**: danh sách phụ thuộc và script (ví dụ: ‘npm start’, ‘npm run build’).
- **tailwind.config.js**: cấu hình Tailwind CSS.
- **README.md**: tài liệu dự án.
- **.gitignore**: loại bỏ các tệp/thư mục không cần commit.

4. Mô hình ứng dụng

4.1 Mô hình backend

4.1.1 Modular Monolith

Ứng dụng đơn thể được chia mô-đun chức năng:

- `music/`: bài hát, album, nghệ sĩ,...
- `user/`: người dùng, cá nhân hóa,...
- `api/`: xử lý logic gateway chung.

Cách tổ chức này vừa dễ mở rộng, vừa duy trì được tính đơn thể.

4.1.2 RESTful API

Hệ thống sử dụng Django REST Framework để:

- Cung cấp API endpoint dạng RESTful.
- Hỗ trợ các HTTP method: GET, POST, PUT, DELETE.
- Trả dữ liệu JSON phục vụ frontend (web/mobile).

Tóm tắt:

- **Kiến trúc:** Modular Monolith
- **Giao tiếp:** RESTful API
- **Framework:** Django + DRF
- **Phân tầng:** models – serializers – views – urls
- **Khả năng mở rộng:** dễ tách thành microservices

4.2 Mô hình frontend

Frontend là một ứng dụng Single Page Application (SPA) được xây dựng bằng ReactJS, tổ chức theo kiến trúc dựa trên thành phần (Component-Based Architecture) và tích hợp chặt chẽ với backend qua RESTful API. Chi tiết:

4.2.1 Single Page Application (SPA)

Ứng dụng sử dụng ReactJS để:

- Tải một trang HTML duy nhất và cập nhật giao diện động mà không cần tải lại trang.
- Cải thiện tốc độ và trải nghiệm người dùng thông qua render phía client.
- Tích hợp với React Router để quản lý các tuyến đường (ví dụ: /search, /profile, /admin).

4.2.2 Component-Based Architecture

Frontend được chia thành các thành phần độc lập:

- **Thành phần giao diện:** Các thành phần tái sử dụng như Player, SongCard, Controls để xây dựng giao diện.
- **Thành phần trang:** Các trang như Search, CreateTrack, AdminDashboard đại diện cho các tuyến đường.
- **Tính mô-đun:** Mỗi thành phần tự quản lý trạng thái cục bộ và nhận dữ liệu qua props, tăng khả năng tái sử dụng và bảo trì.

4.2.3 State Management và API Integration

Quản lý trạng thái và giao tiếp với backend được thực hiện bởi:

- **Redux Toolkit Query:**
 - Quản lý trạng thái ứng dụng (ví dụ: bài hát đang phát, thông tin người dùng).
 - Gọi API RESTful (ví dụ: GET /music/artists/, POST /music/tracks/create/) với caching tự động.
- **Axios:** Thực hiện các yêu cầu HTTP đến backend, tích hợp trong `spotifyApi.js`.
- **Local Storage:** Lưu trữ tạm thời `userId`, `userRole` để xác thực và phân quyền (ví dụ: quản trị viên).

Tóm tắt:

- **Kiến trúc:** Single Page Application với Component-Based Architecture
- **Giao tiếp:** RESTful API thông qua Axios và Redux Toolkit Query
- **Framework:** ReactJS, Redux Toolkit, React Router
- **Phân tầng:** components – pages – redux – assets
- **Khả năng mở rộng:** Dễ dàng thêm thành phần mới hoặc tích hợp với các API bổ sung

5. Các tính năng

5.1 Tính năng chính

5.1.1 Frontend

Frontend cung cấp giao diện người dùng tương tác, được xây dựng bằng ReactJS, tích hợp chặt chẽ với các API backend để hỗ trợ các tính năng phát nhạc, quản lý album, tìm kiếm, và quản trị. Các tính năng chính được triển khai như sau:

- **Phát Nhạc:**

- **Giao diện:** Sử dụng thành phần `MusicPlayer.jsx` và `Player.jsx` để phát các tệp MP4 từ URL backend (ví dụ: `/media/videos/song.mp4`).
- **Điều khiển:** Hỗ trợ phát/tạm dừng, chuyển bài tiếp theo/trước đó, điều chỉnh âm lượng qua `Controls.jsx` và `VolumeBar.jsx`.
- **Chế độ thu nhỏ:** Chuyển đổi giữa giao diện đầy đủ và thanh thu nhỏ cố định ở dưới cùng, hiển thị thông tin bài hát và điều khiển cơ bản.
- **Tích hợp API:** Gọi GET `/tracks/tracksdetail/{track_id}/` để lấy chi tiết bài hát và PATCH `/tracks/{track_id}/play/` để cập nhật lượt xem.

- **Tạo và Quản lý Album:**

- **Giao diện:** Trang `CreateAlbum.jsx` cung cấp biểu mẫu để nhập tên album, tải lên hình ảnh, và chọn bài hát qua checkbox với bộ lọc tìm kiếm.
- **Quản lý:** Người dùng có thể xem và chỉnh sửa album trong `Profile.jsx`, bao gồm thêm/xóa bài hát.
- **Tích hợp API:**
 - * POST `/{id}/albums/create/` để tạo album.
 - * POST `/{id}/albums/{album_id}/add-tracks/` để thêm bài hát.
 - * GET `/{id}/albums/` để lấy danh sách album.
 - * POST `/albums/{album_id}/edit/` và DELETE `/albums/{album_id}/delete/` để chỉnh sửa/xóa.

- **Tìm Kiếm Bài Hát:**

- **Giao diện:** Trang `Search.jsx` cho phép tìm kiếm bài hát theo tên hoặc hiển thị tất cả bài hát khi không có từ khóa.
- **Hiển thị:** Kết quả được hiển thị qua `SongCard.jsx`, bao gồm tên bài hát, nghệ sĩ, và nút phát.

- **Tích hợp API:**

- * GET /tracks/search/ để tìm kiếm bài hát.
- * GET /tracks/genre/{genre_id}/ để lọc bài hát theo thể loại.

- **Quản lý Nghệ Sĩ và Thể Loại:**

- **Giao diện:** Trang ArtistList.jsx hiển thị danh sách nghệ sĩ và bài hát liên quan trong bố cục lưới.

- **Tích hợp API:**

- * GET /artists/ để lấy danh sách nghệ sĩ.
- * GET /artist/details/{artist_id}/ để xem chi tiết nghệ sĩ.
- * GET /genre/ để lấy danh sách thể loại (sử dụng trong CreateTrack.jsx).

- **Quản trị:**

- **Bảng điều khiển:** Trang AdminDashboard.jsx hiển thị thống kê tổng số bài hát và album bằng biểu đồ cột (sử dụng Chart.js).

- **Tạo bài hát:** Trang CreateTrack.jsx cung cấp biểu mẫu để nhập tên bài hát, chọn nhiều thể loại (checkbox với tìm kiếm), chọn một nghệ sĩ (dropdown), và tải lên hình ảnh/video.

- **Phân quyền:** Chỉ người dùng có UserRole: "admin" trong localStorage được truy cập, với chuyển hướng đến / nếu không đủ quyền.

- **Tích hợp API:**

- * GET /stats/tracks/ và GET /stats/albums/ cho thống kê.
- * POST /tracks/create/ để tạo bài hát mới.
- * GET /artists/ và GET /genre/ để lấy danh sách nghệ sĩ/thể loại.

- **Xác Thực và Quản lý Người Dùng:**

- **Giao diện:**

- * Login.jsx và Register.jsx cung cấp biểu mẫu đăng nhập/đăng ký.
- * Profile.jsx cho phép xem, chỉnh sửa thông tin người dùng (tên, email) và quản lý album.

- **Yêu thích bài hát:** Người dùng có thể thêm/xóa bài hát yêu thích trong Profile.jsx.

- **Tích hợp API:**

- * POST /register/ và POST /login/ để đăng ký/đăng nhập.
- * POST /logout/ để đăng xuất.
- * GET /me/ và POST /update/ để quản lý thông tin người dùng.
- * GET /{user_id}/favourites/list/, POST /{user_id}/favourites/, DELETE /{user_id}/favourites/{track_id}/ để quản lý bài hát yêu thích.

5.1.2 Backend

Backend cung cấp các API để quản lý các bài hát, nghệ sĩ, album, thể loại, và thông tin người dùng. Các tính năng này được triển khai thông qua các đường dẫn URL sau:

Quản lý Bài Hát

- Cập nhật lượt xem bài hát: PATCH /tracks/{track_id}/play/
- Tạo bài hát mới: POST /tracks/create/
- Chi tiết bài hát: GET /tracks/tracksdetail/{track_id}/
- Danh sách album: GET /tracks/albums/
- Tải bài hát: GET /tracks/download/{track_id}/
- Tìm kiếm bài hát theo tên: GET /tracks/search/
- Danh sách bài hát theo thể loại: GET /tracks/genre/{genre_id}/

Quản lý Nghệ Sĩ

- Danh sách nghệ sĩ: GET /artists/
- Chi tiết nghệ sĩ: GET /artist/details/{artist_id}/

Quản lý Thể Loại

- Danh sách thể loại: GET /genre/

Thông Kê

- Tổng số album: GET /stats/albums/
- Tổng số bài hát: GET /stats/tracks/

Quản lý Tài Khoản Người Dùng

- Danh sách người dùng: GET /
- Đăng ký người dùng: POST /register/
- Đăng nhập người dùng: POST /login/
- Đăng xuất người dùng: POST /logout/
- Thông tin người dùng hiện tại: GET /me/
- Cập nhật thông tin người dùng: POST /update/
- Danh sách yêu thích của người dùng:
 - GET /{user_id}/favourites/list/
 - POST /{user_id}/favourites/
 - DELETE /{user_id}/favourites/{track_id}/
- Quản lý album người dùng:

- GET /{id}/albums/
- POST /{id}/albums/create/
- POST /albums/{album_id}/edit/
- DELETE /albums/{album_id}/delete/
- POST /{id}/albums/{album_id}/add-tracks/

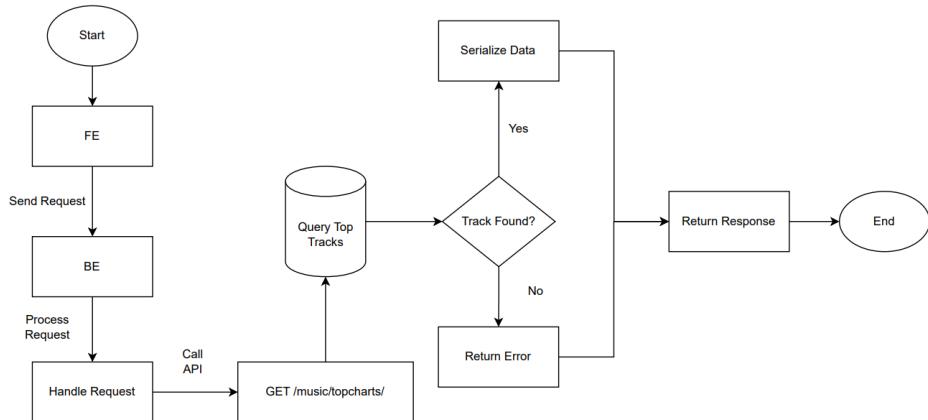
Chức Năng JWT

- Làm mới token: POST /token/refresh/

5.2 Quy trình hoạt động (Flowchart)

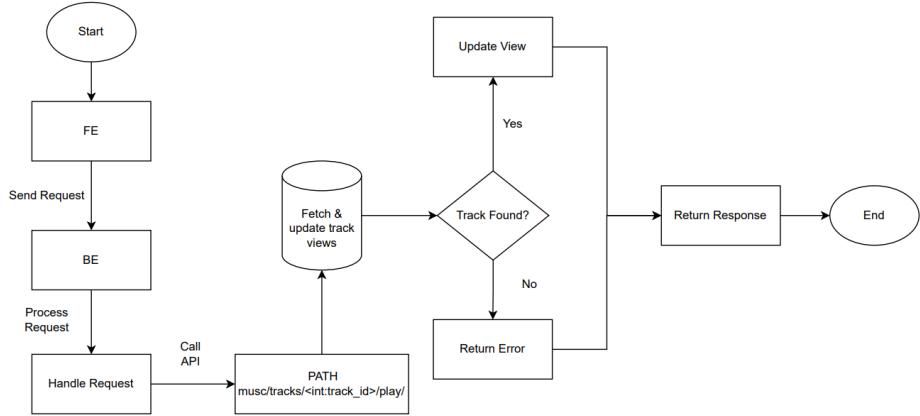
5.2.1 Music App

Lấy danh sách bài hát thịnh hành



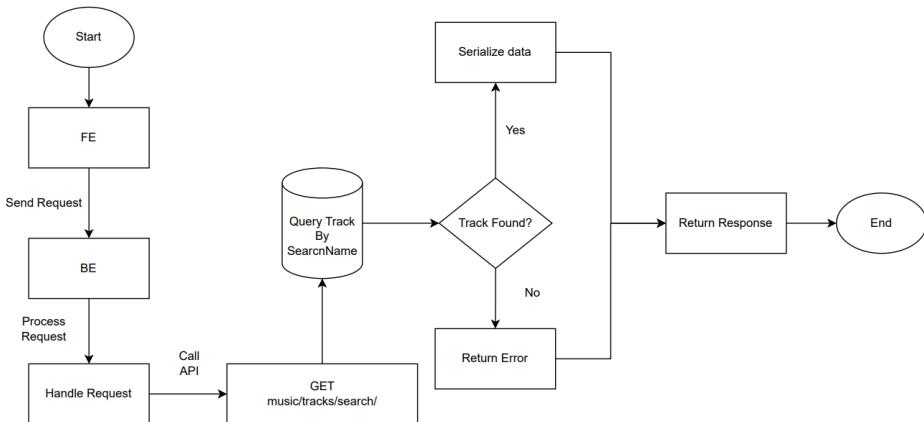
Hình 1: Flowchart quy trình lấy list nhạc hàng đầu

Cập nhật lượt xem



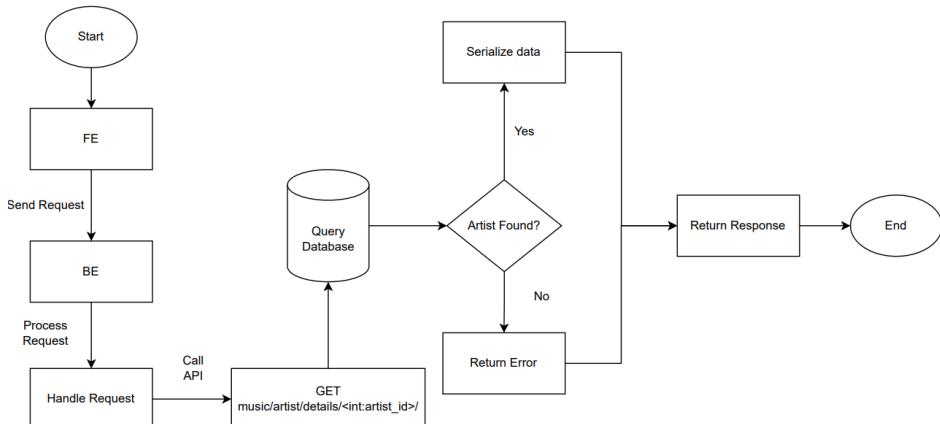
Hình 2: Flowchart quy trình cập nhật lượt xem cho tracks

Tìm kiếm nhạc theo tên



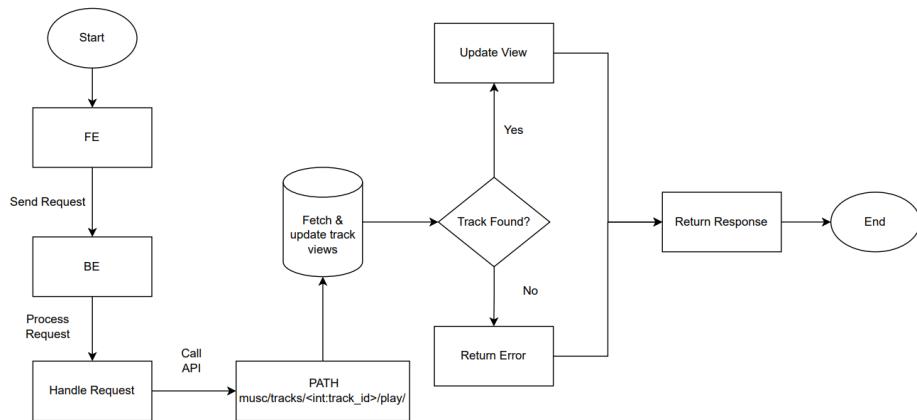
Hình 3: Flowchart quy trình tìm kiếm nhạc theo tên

Thông tin nghệ sĩ



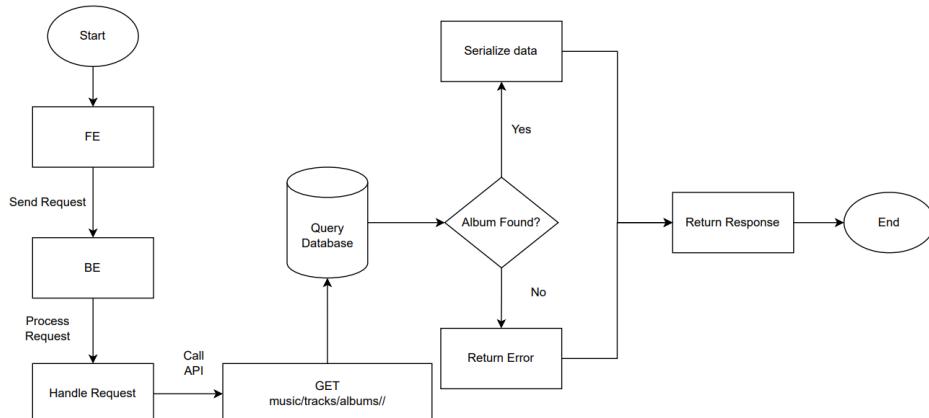
Hình 4: Flowchart quy trình lấy chi tiết nghệ sĩ

Thông tin của nhạc



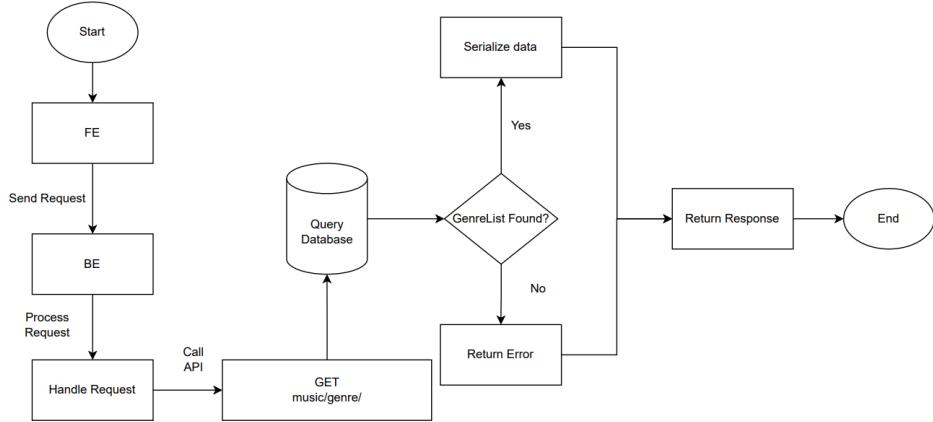
Hình 5: Flowchart quy trình lấy chi tiết thông tin của nhạc bao gồm các bài cùng thể loại

Lấy danh sách Album



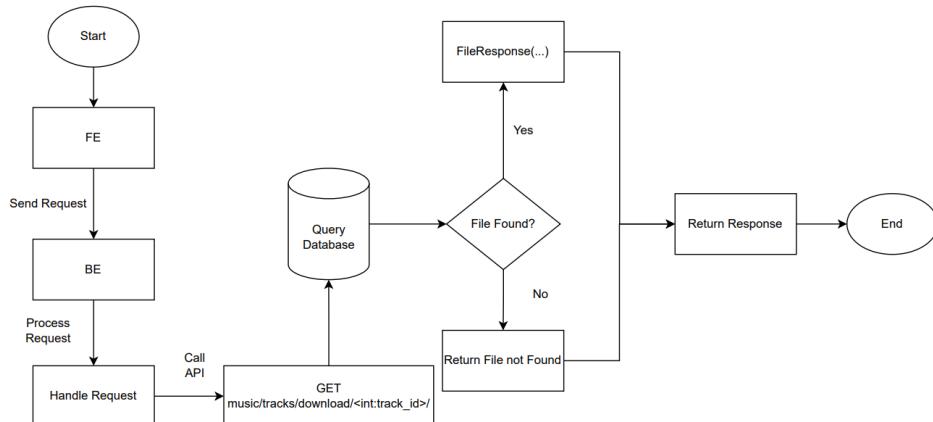
Hình 6: Flowchart quy trình lấy danh sách album

Lấy danh sách Thể loại



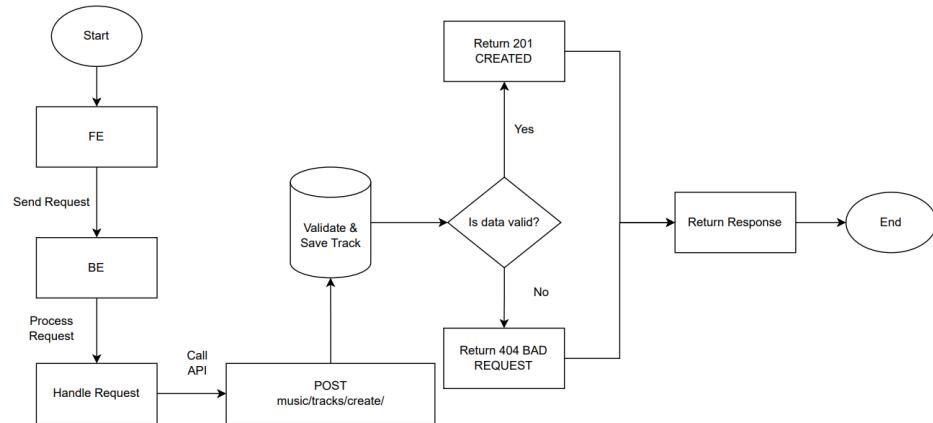
Hình 7: Flowchart quy trình lấy danh sách thể loại

Tải nhạc



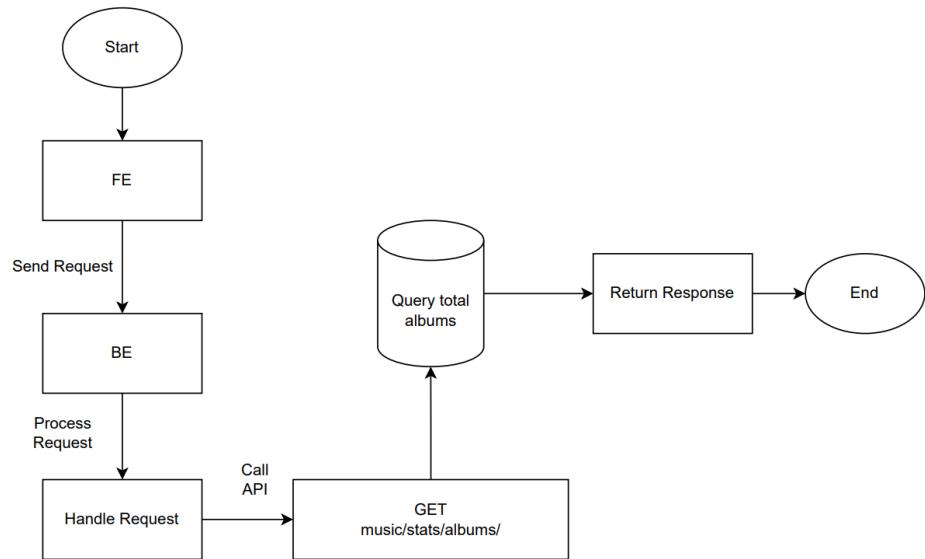
Hình 8: Flowchart quy trình tải nhạc

Thêm nhạc



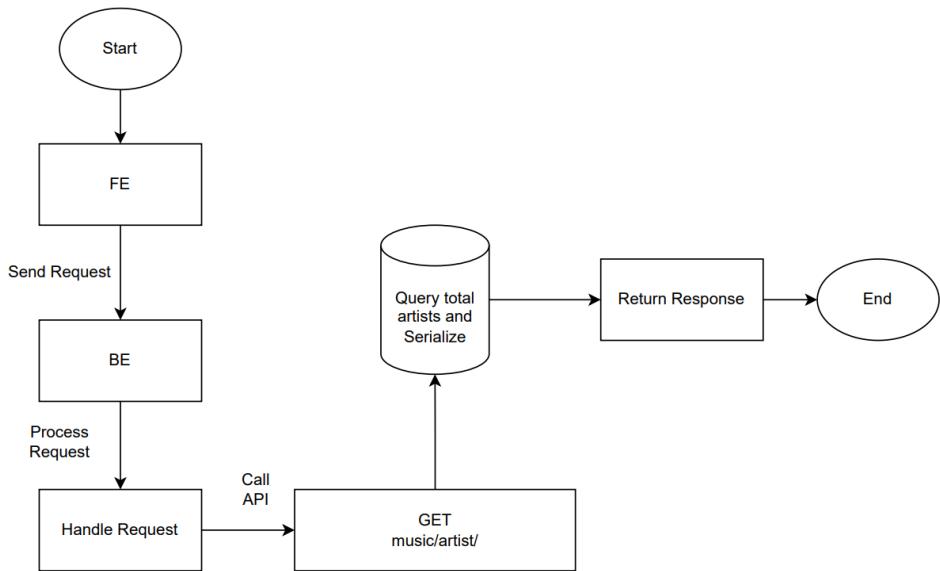
Hình 9: Flowchart quy trình thêm nhạc

Tổng số lượng Albums



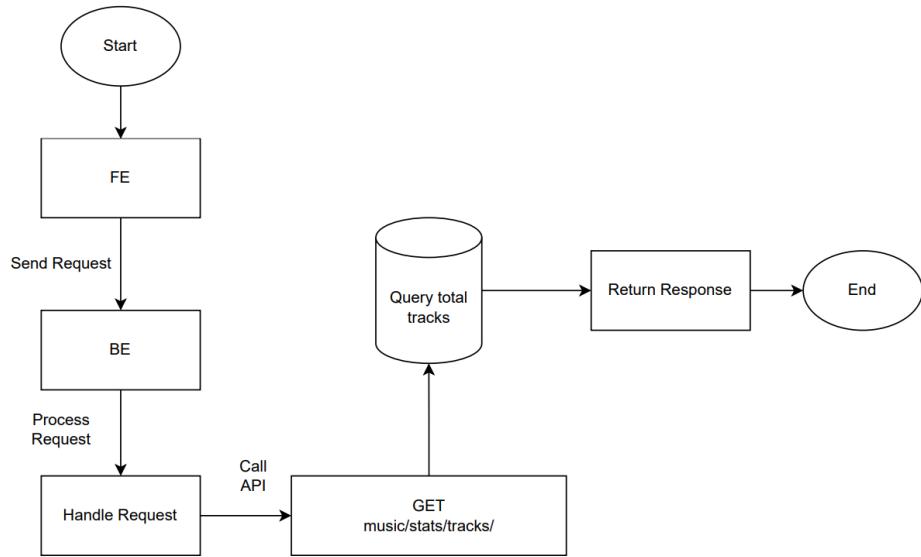
Hình 10: Flowchart quy trình thống kê tổng số lượng albums

Tổng số lượng Nghệ sĩ



Hình 11: Flowchart quy trình thống kê tổng số lượng nghệ sĩ

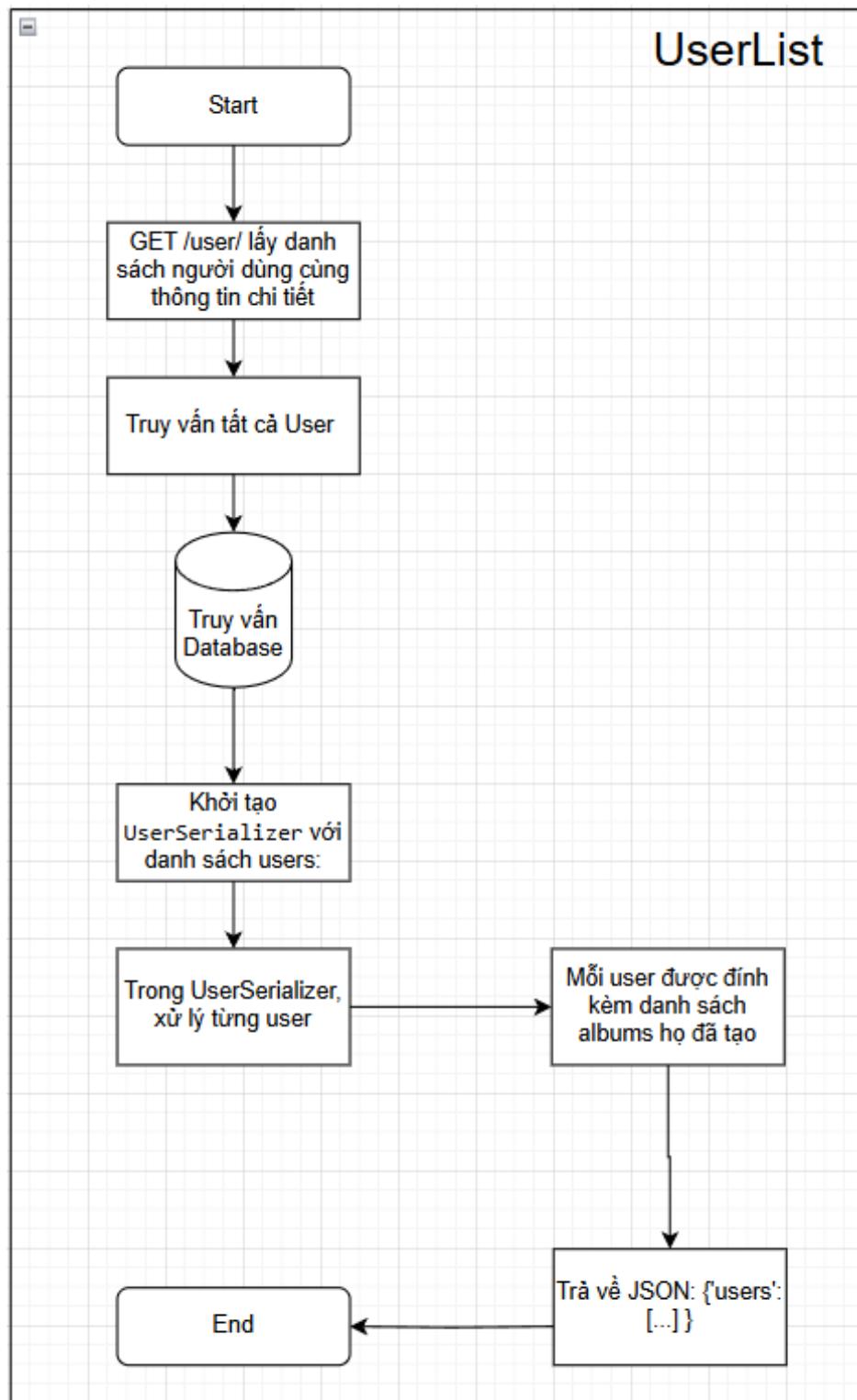
Tổng số lượng Nhạc



Hình 12: Flowchart quy trình thống kê tổng số lượng nhạc

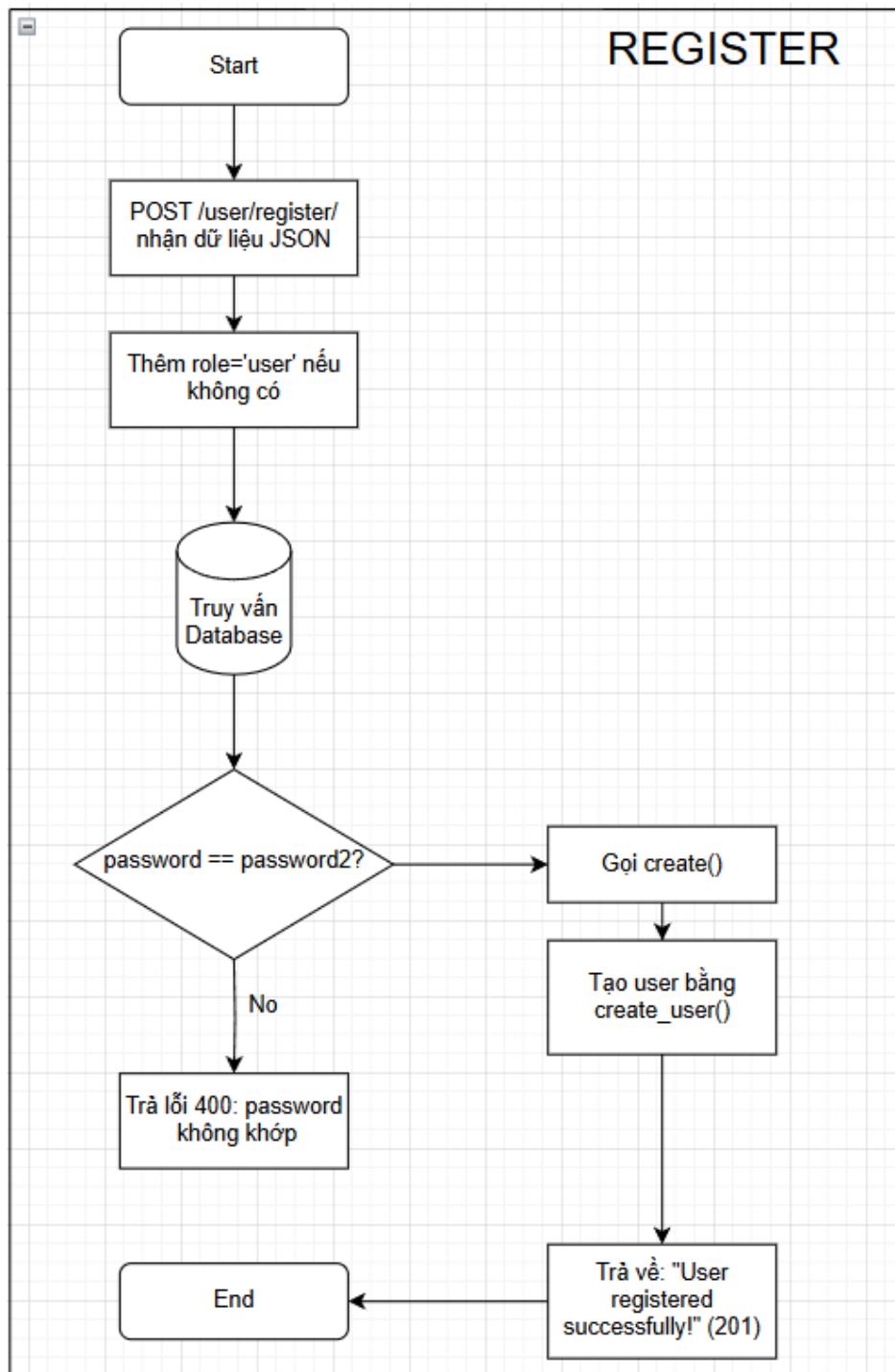
5.2.2 User App

Lấy danh sách người dùng cùng thông tin chi tiết



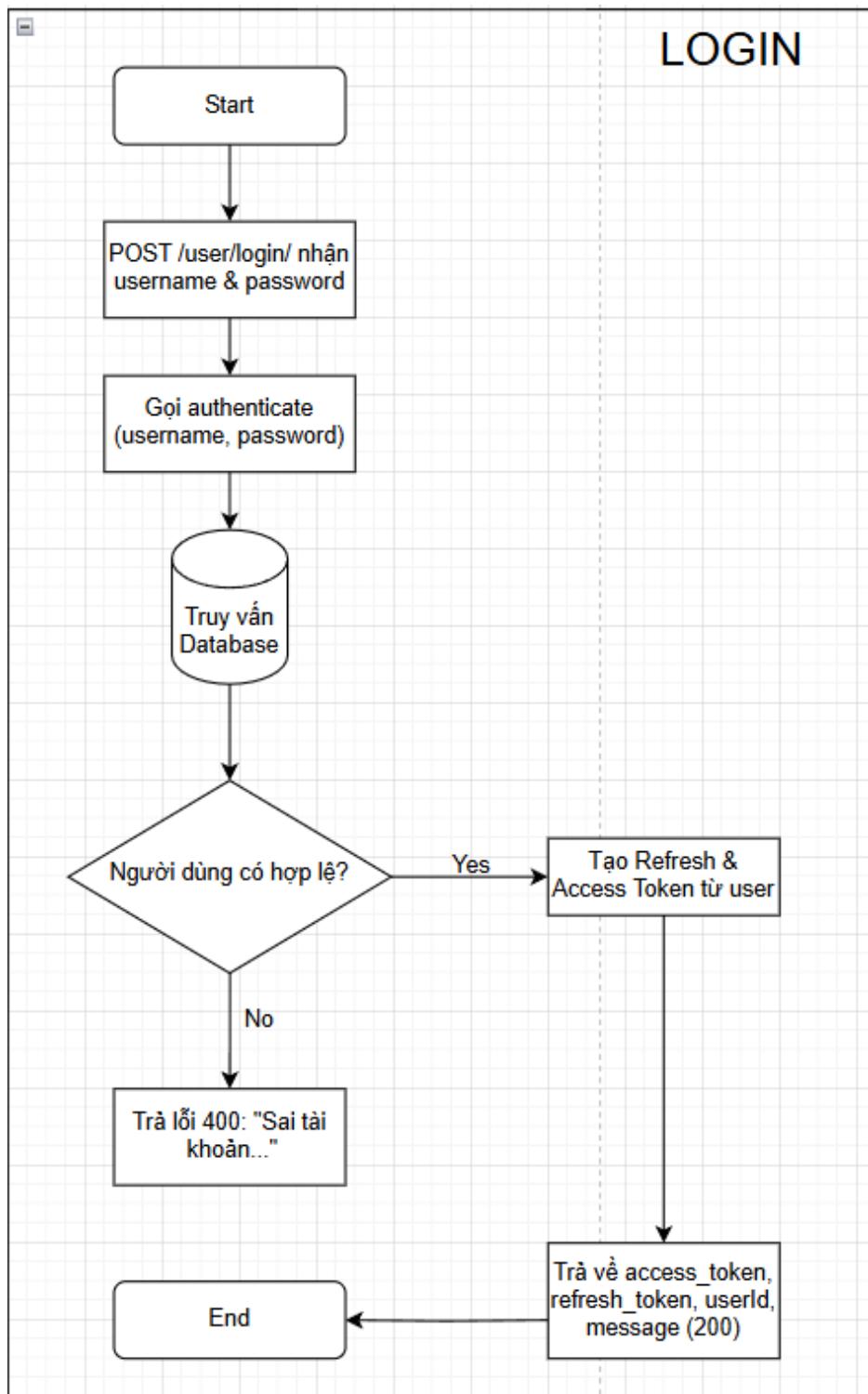
Hình 13: Flowchart quy trình lấy danh sách người dùng

Đăng ký người dùng mới sau khi xác thực thông tin



Hình 14: Flowchart quy trình đăng ký người dùng mới

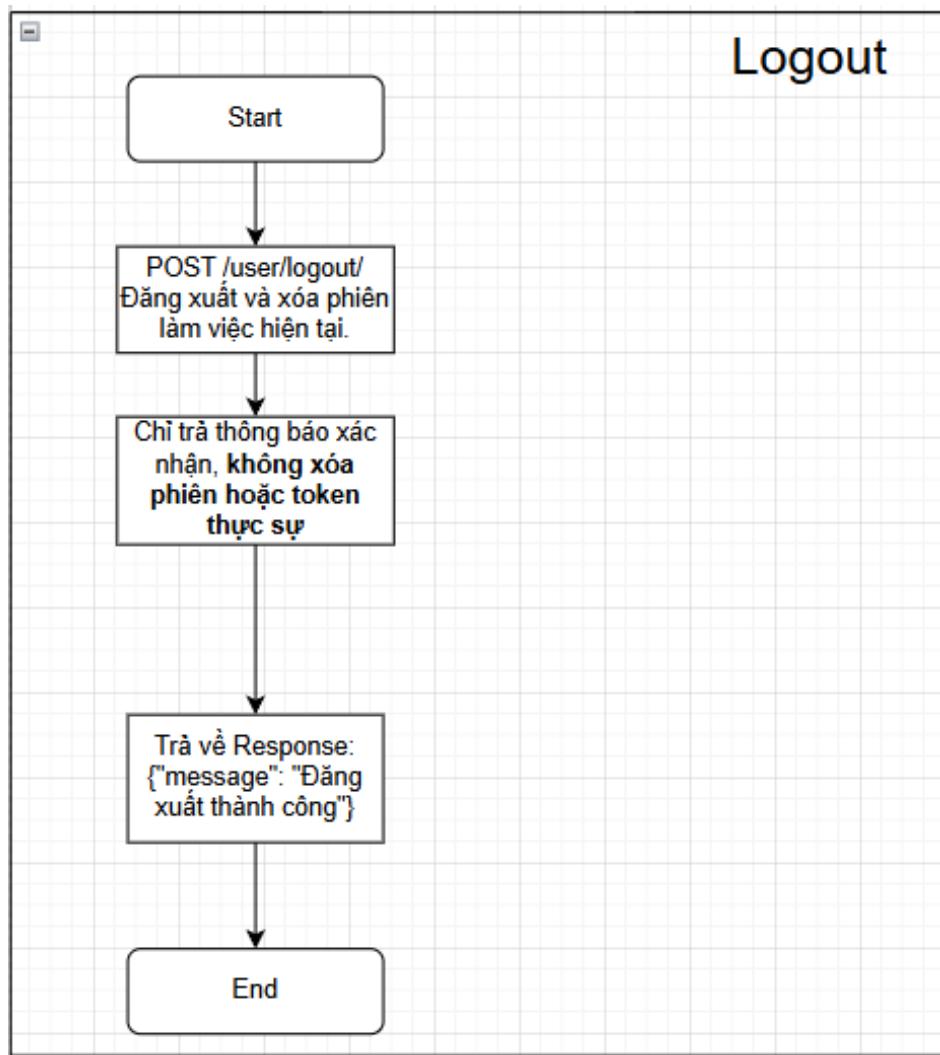
Đăng nhập và tạo phiên làm việc sau khi xác thực thông tin



Hình 15: Flowchart quy trình đăng nhập và tạo phiên làm việc

Đăng xuất và xóa phiên làm việc hiện tại

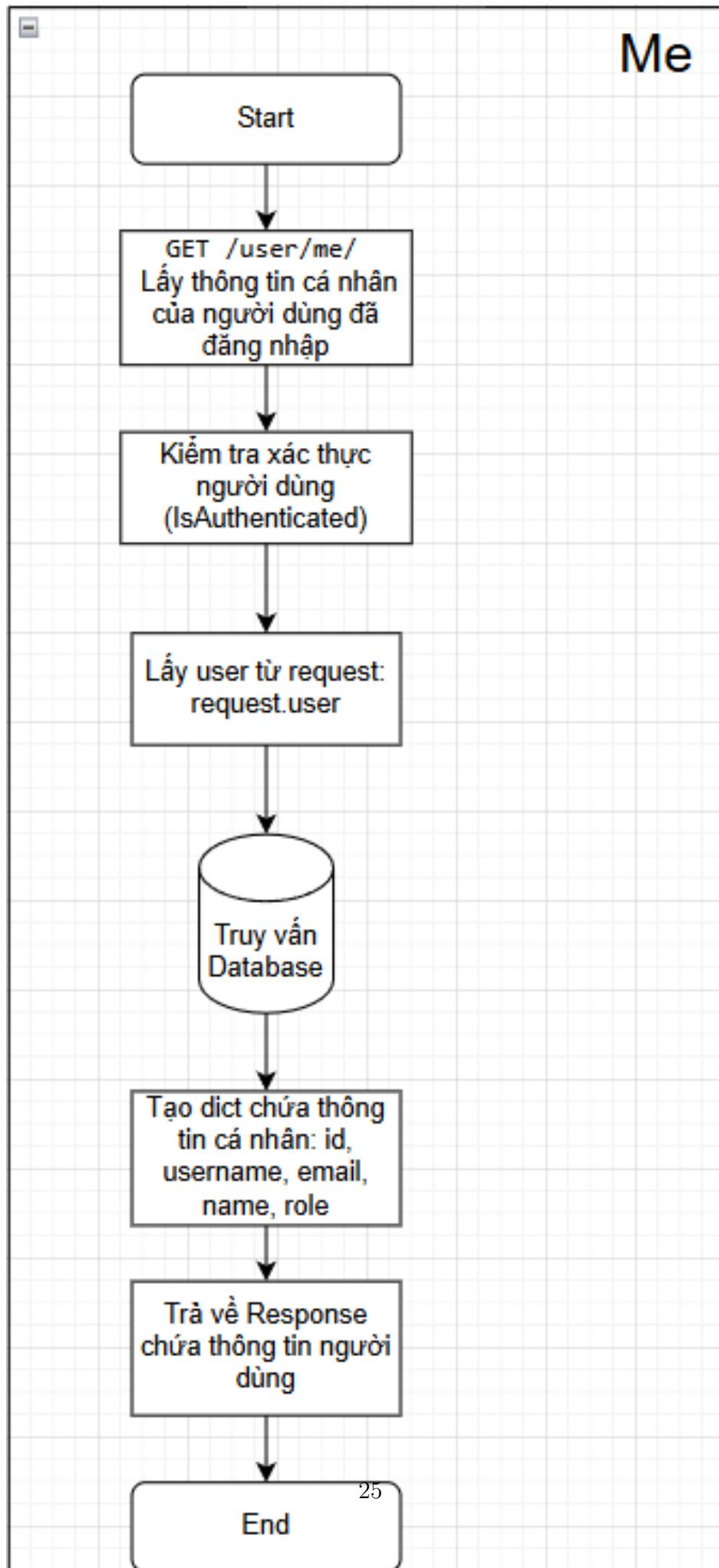
Logout



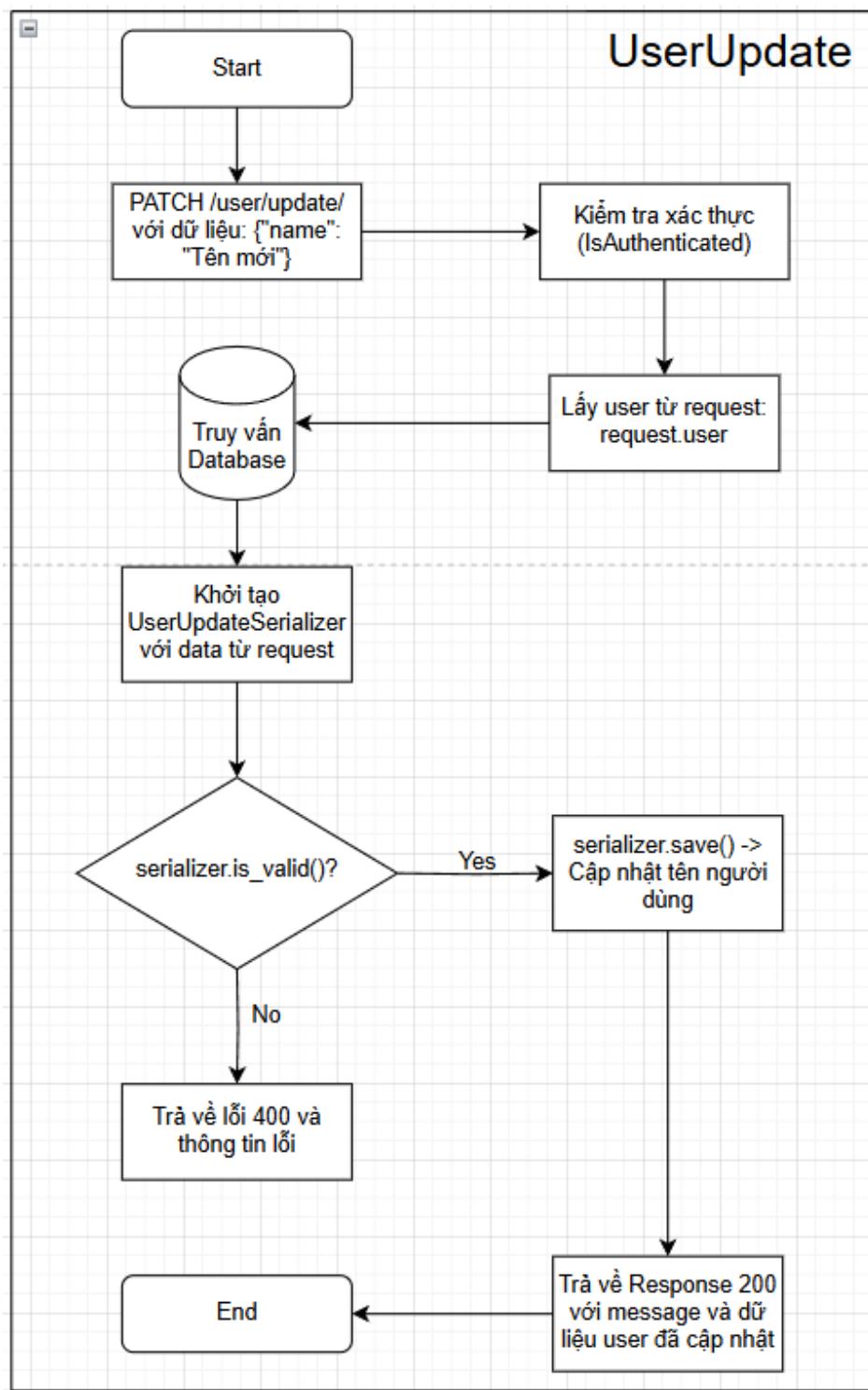
Hình 16: Flowchart quy trình đăng xuất và xóa phiên làm việc hiện tại

Lấy thông tin cá nhân của người dùng đã đăng nhập

Me

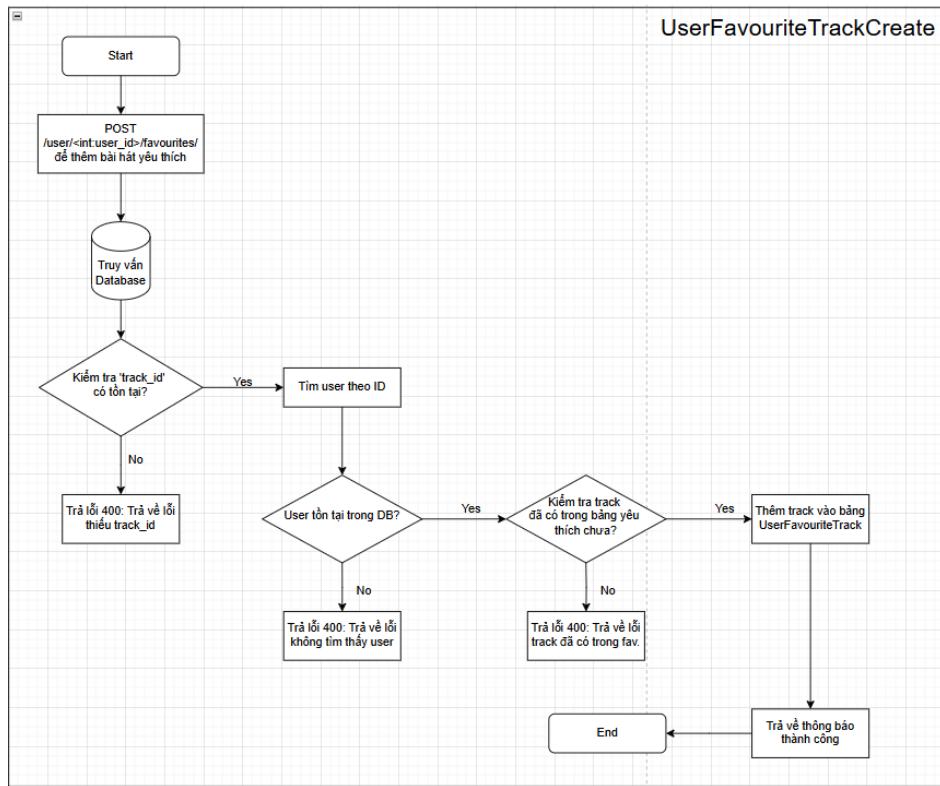


Cập nhật tên người dùng



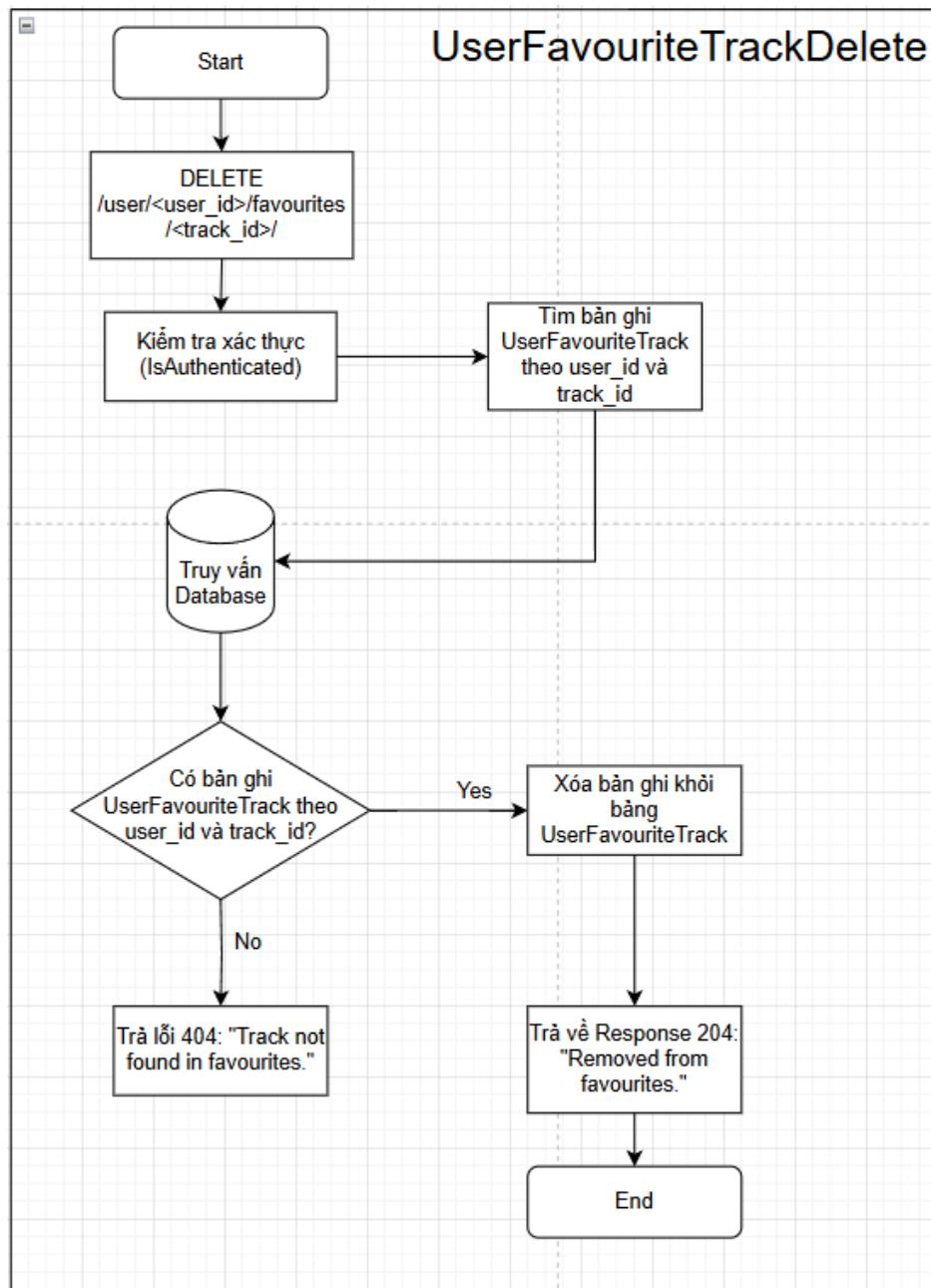
Hình 18: Flowchart quy trình cập nhật tên người dùng

Thêm bài hát vào danh sách yêu thích của người dùng



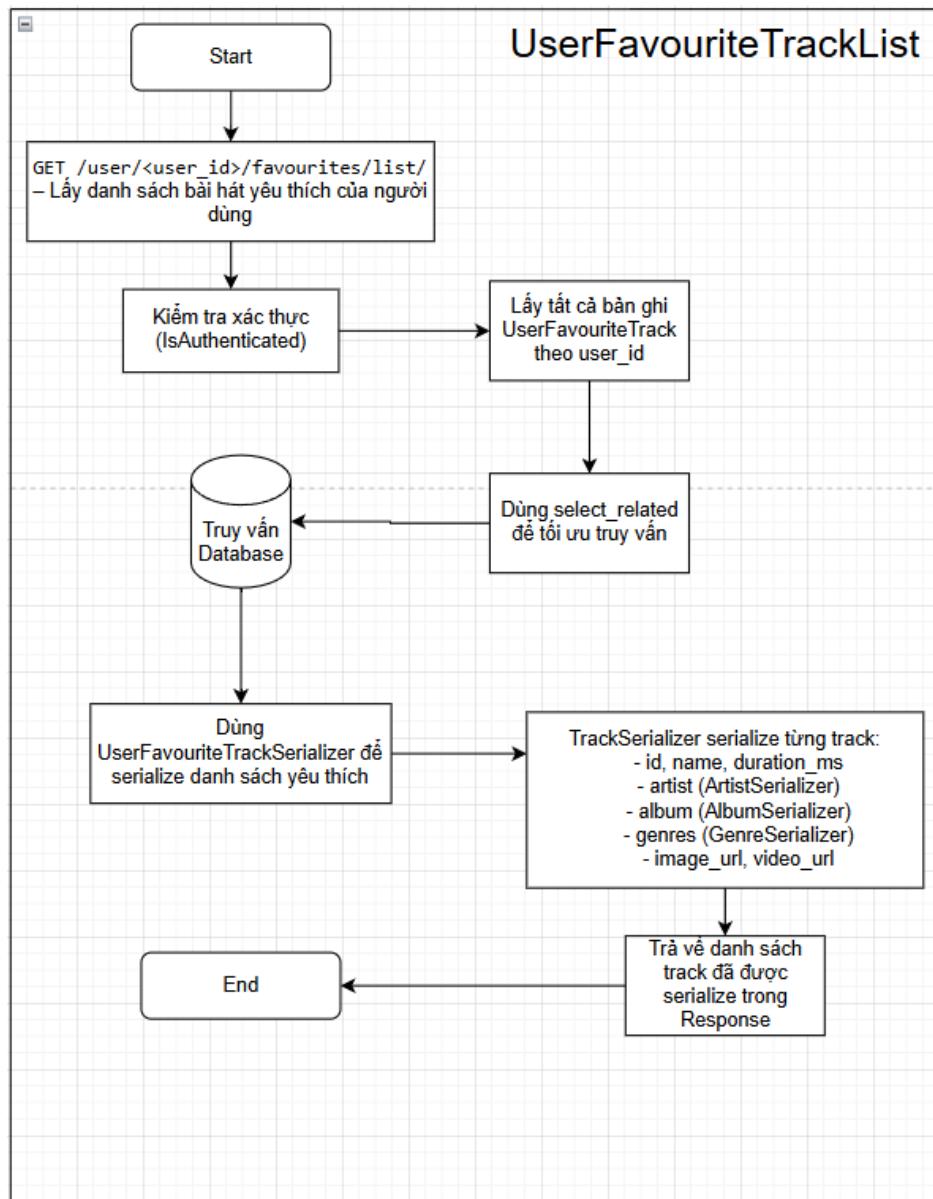
Hình 19: Flowchart quy trình thêm bài hát vào danh sách yêu thích của người dùng

Xóa bài hát khỏi danh sách yêu thích



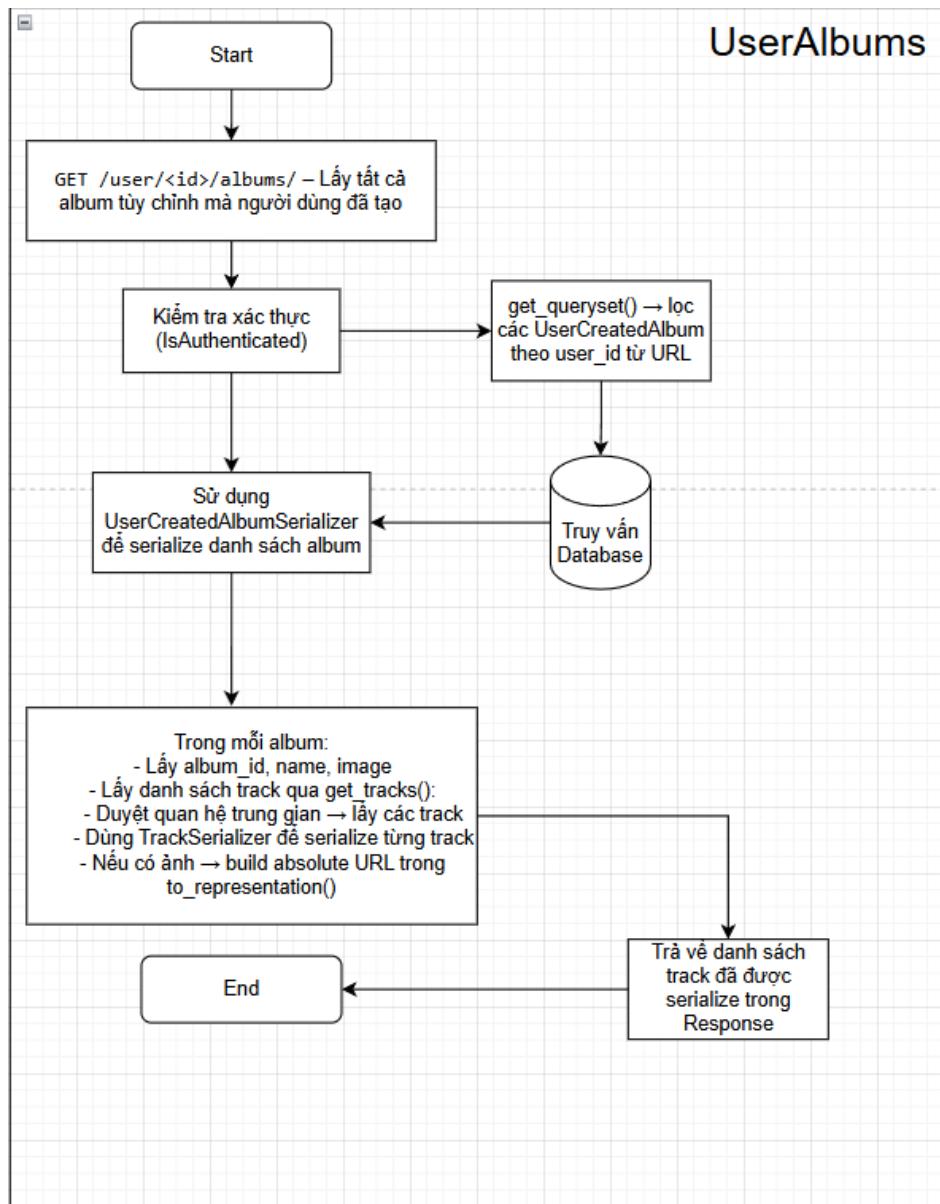
Hình 20: Flowchart quy trình xóa bài hát khỏi danh sách yêu thích

Lấy tất cả bài hát yêu thích của người dùng



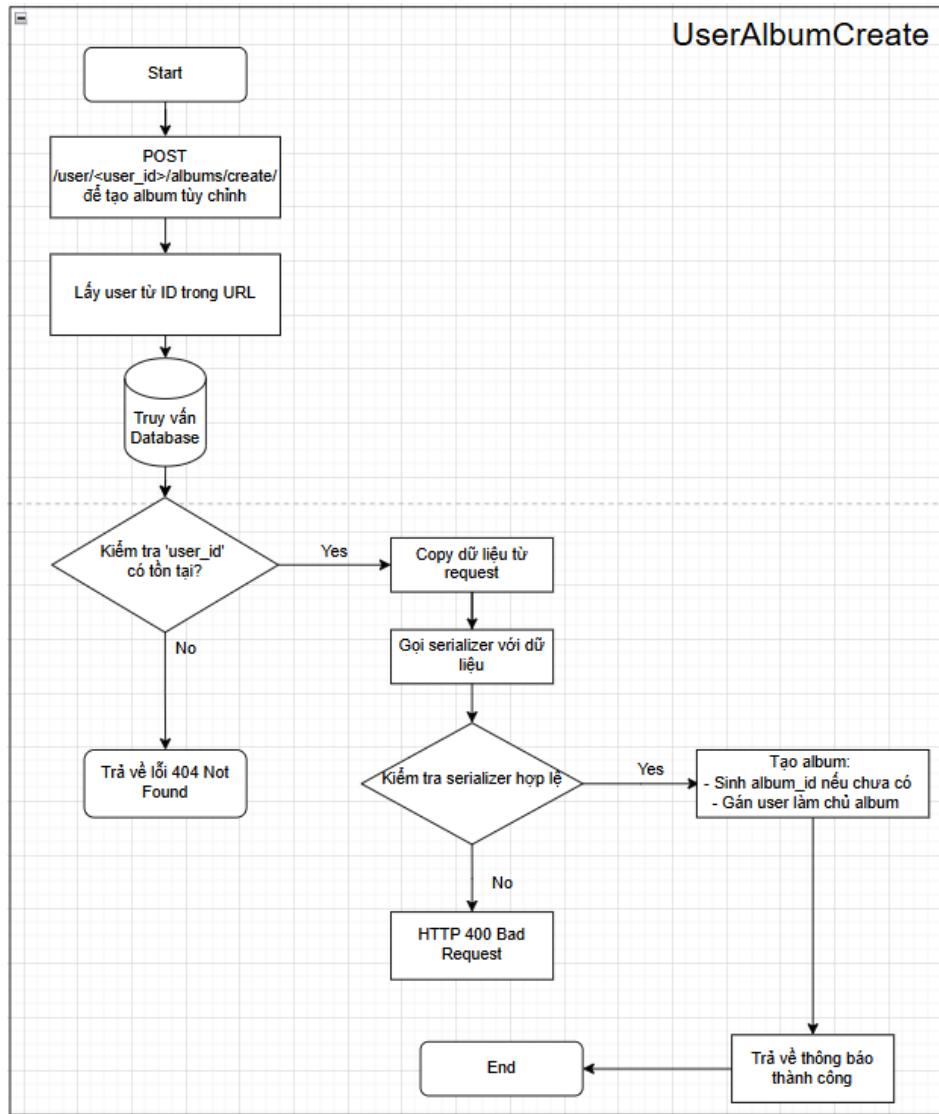
Hình 21: Flowchart quy trình lấy tất cả bài hát yêu thích của người dùng

Lấy tất cả album tùy chỉnh mà người dùng đã tạo



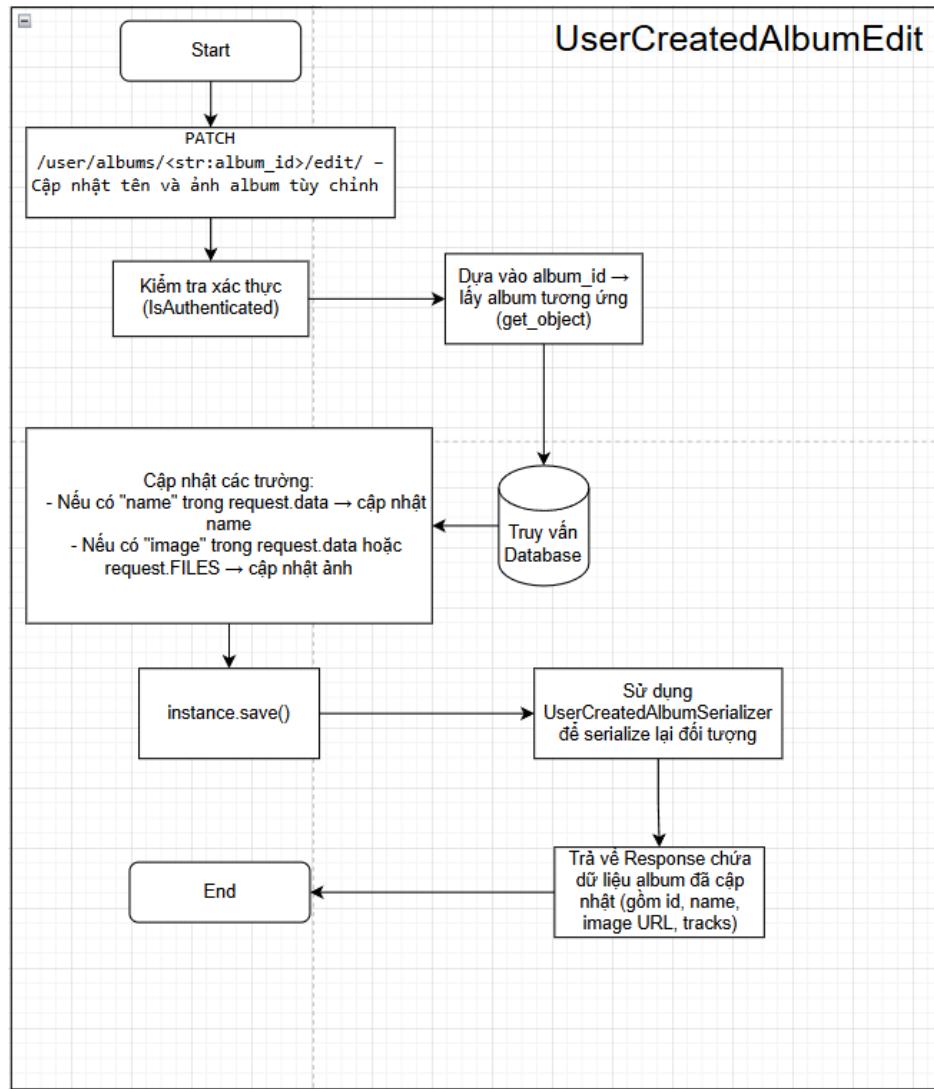
Hình 22: Flowchart quy trình lấy tất cả album tùy chỉnh mà người dùng đã tạo

Tạo album tùy chỉnh mới



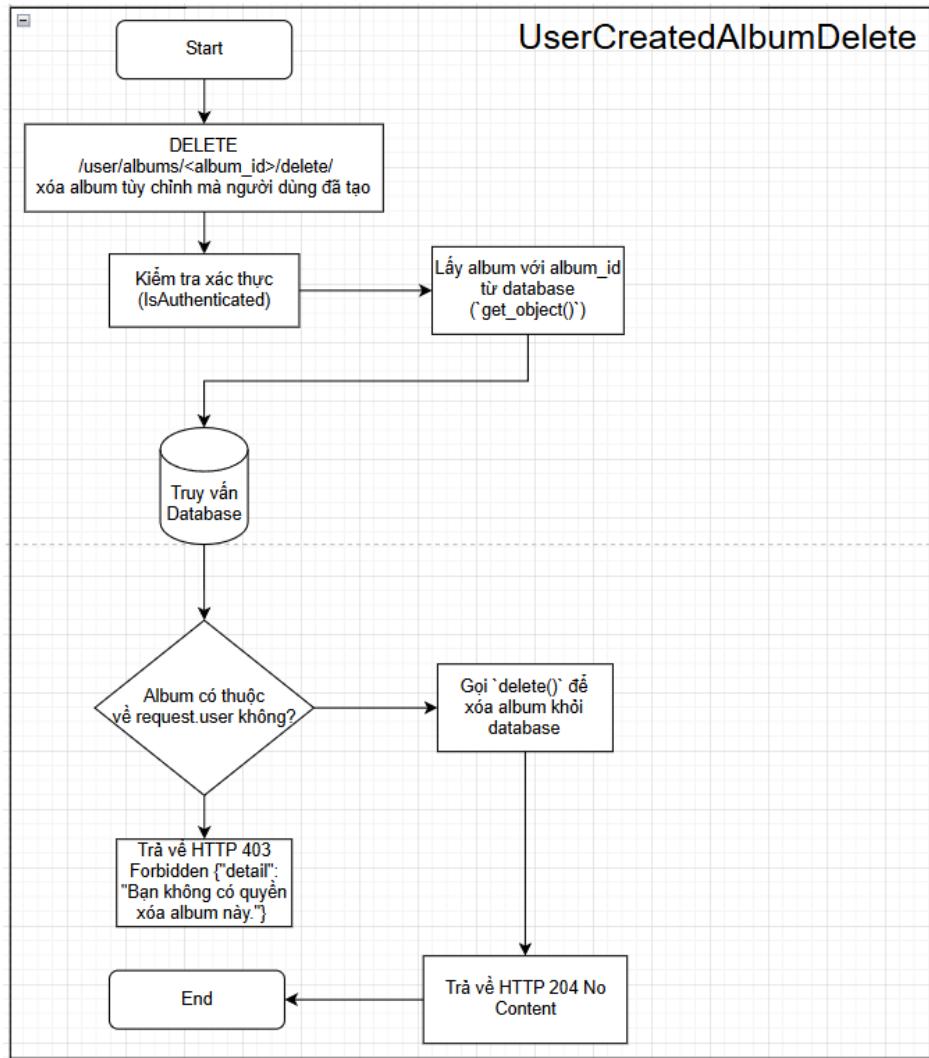
Hình 23: Flowchart quy trình tạo album tùy chỉnh mới

Đổi tên và ảnh album tùy chỉnh



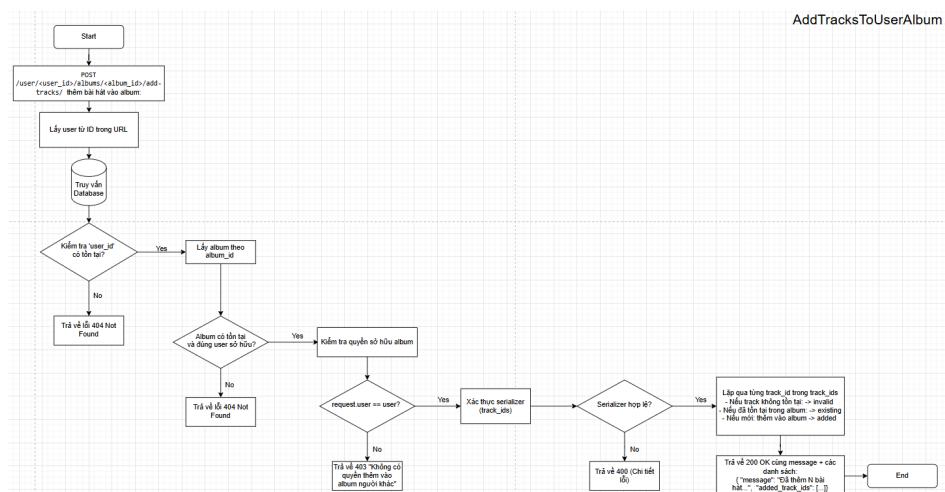
Hình 24: Flowchart quy trình đổi tên và ảnh album tùy chỉnh.

Xóa album tùy chỉnh của người dùng



Hình 25: Flowchart quy trình xóa album tùy chỉnh của người dùng

Thêm các bài hát vào album tùy chỉnh



Hình 26: Flowchart quy trình thêm các bài hát vào album tùy chỉnh

6. Xây dựng database

6.1 Cấu hình database

Dự án sử dụng hệ quản trị cơ sở dữ liệu PostgreSQL. Cấu hình kết nối được thiết lập trong file `spotify_back_end/settings.py` như sau:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql',  
        'NAME': 'spotify_db',  
        'USER': 'postgres',  
        'PASSWORD': 'yourpassword',  
        'HOST': 'localhost',  
        'PORT': '5432',  
    }  
}
```

Các thông tin như NAME, USER, và PASSWORD có thể được điều chỉnh phù hợp với môi trường cài đặt thực tế.

6.2 Mô hình database

Hệ thống được thiết kế với hai nhóm mô hình chính: **mô hình âm nhạc** và **mô hình người dùng**. Các mô hình này được xây dựng bằng Django ORM nhằm đảm bảo khả năng mở rộng và truy vấn hiệu quả.

6.2.1 Mô hình âm nhạc (`music.models.py`)

- **Artist** (Nghệ sĩ):
 - `name`: Tên nghệ sĩ.
 - `image_url`: Hình ảnh đại diện.
 - `followers`: Số lượng người theo dõi.
- **Genre** (Thể loại âm nhạc):
 - `name`: Tên thể loại (duy nhất).
- **Album** (Album):

- **name**: Tên album.
- **artist**: Nghệ sĩ sở hữu album (liên kết khoá ngoại đến Artist).
- **image_url**: Ảnh bìa album.

- **Track** (Bài hát):

- **name**: Tên bài hát.
- **artist**: Nghệ sĩ thể hiện (liên kết khoá ngoại đến Artist).
- **album**: Album mà bài hát thuộc về (có thể để trống).
- **duration_ms**: Thời lượng bài hát (mili giây).
- **genres**: Danh sách thể loại (nhiều-nhiều với Genre).
- **image_url**: Ảnh minh họa bài hát.
- **video_url**: Đường dẫn video liên quan.
- **views**: Số lượt nghe/xem bài hát.

6.2.2 Mô hình người dùng (`user.models.py`)

- **User** (Người dùng):

- Kế thừa từ `AbstractUser`.
- **name**: Tên người dùng.
- **email**: Địa chỉ email (duy nhất).
- **role**: Vai trò (`admin` hoặc `user`, mặc định là `user`).

- **UserFavouriteTrack** (Yêu thích):

- **user**: Người dùng (liên kết khoá ngoại đến User).
- **track**: Bài hát được yêu thích (liên kết khoá ngoại đến Track).

- **UserCreatedAlbum** (Album người dùng tạo):

- **album_id**: Mã album (khóa chính).
- **name**: Tên album.
- **user**: Người tạo album.
- **created_at**: Thời gian tạo album.
- **image**: Ảnh bìa album.

- **UserCreatedAlbumTrack** (Bài hát trong album tự tạo):

- **album**: Album người dùng tạo.
- **track**: Bài hát thuộc album.

7. Hiện thực

7.1 Tính năng đã triển khai

Dự án đã triển khai các tính năng cốt lõi của một ứng dụng phát nhạc tương tự Spotify, tích hợp giao diện frontend (ReactJS) với backend (Django REST Framework) để cung cấp trải nghiệm người dùng liền mạch. Các tính năng chính bao gồm:

- **Phát Nhạc:**
 - Phát các tệp video MP4 từ backend qua trình phát `MusicPlayer.jsx`, hỗ trợ phát/tạm dừng, chuyển bài, và điều chỉnh âm lượng.
 - Hỗ trợ chế độ giao diện đầy đủ và thu nhỏ (thanh cố định dưới cùng).
 - Cập nhật lượt xem bài hát thông qua API PATCH `/tracks/{track_id}/play/`.
- **Tạo và Quản lý Album:**
 - Người dùng tạo album tùy chỉnh với tên, hình ảnh, và danh sách bài hát qua trang `CreateAlbum.jsx`.
 - Quản lý album (xem, chỉnh sửa, xóa) trong trang `Profile.jsx`.
 - Tích hợp API: POST `/{id}/albums/create/`, GET `/{id}/albums/`, POST `/albums/{album_id}/edit/`, DELETE `/albums/{album_id}/delete/`.
- **Tìm Kiếm và Khám Phá:**
 - Tìm kiếm bài hát theo tên hoặc hiển thị tất cả bài hát qua trang `Search.jsx`.
 - Lọc bài hát theo thể loại hoặc xem danh sách nghệ sĩ qua `ArtistList.jsx`.
 - Tích hợp API: GET `/tracks/search/`, GET `/tracks/genre/{genre_id}/`, GET `/artists/`.
- **Xác Thực và Quản lý Người Dùng:**
 - Đăng ký, đăng nhập, và đăng xuất thông qua `Register.jsx`, `Login.jsx`, và `Profile.jsx`.
 - Quản lý thông tin cá nhân (tên, email) và danh sách bài hát yêu thích.
 - Tích hợp API: POST `/register/`, POST `/login/`, POST `/logout/`, GET `/me/`, POST `/update/`, GET/POST/DELETE `/{user_id}/favourites/....`
- **Quản trị:**

- Bảng điều khiển quản trị (`AdminDashboard.jsx`) hiển thị thống kê tổng số bài hát và album bằng biểu đồ cột (`Chart.js`).
- Tạo bài hát mới với tên, thể loại, nghệ sĩ, hình ảnh, và video qua `CreateTrack.jsx`.
- Phân quyền dựa trên `userRole`: "admin" trong `localStorage`.
- Tích hợp API: GET `/stats/tracks/`, GET `/stats/albums/`, POST `/tracks/create/`, GET `/artists/`, GET `/genre/`.

7.2 Giao diện minh họa

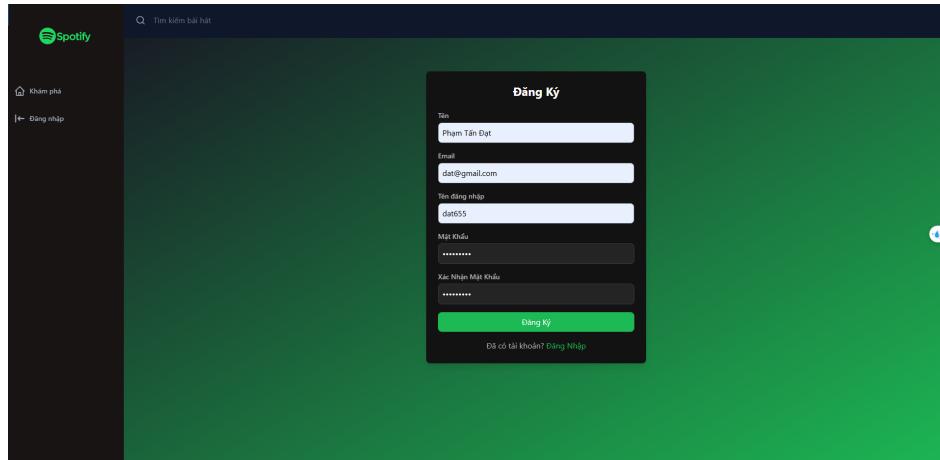
Giao diện được thiết kế responsive với Tailwind CSS, cung cấp trải nghiệm người dùng mượt mà trên cả máy tính và thiết bị di động. Dưới đây là mô tả các giao diện chính dành cho **người dùng** và **quản trị viên**, kèm theo hình ảnh minh họa.

7.2.1 Giao diện dành cho Người Dùng

- **Đăng Ký** (`Register.jsx`):

- Biểu mẫu nhập tên, email, và mật khẩu với kiểm tra hợp lệ.
- Nút *Đăng Ký* gửi yêu cầu đến POST `/register/`.
- Liên kết đến trang đăng nhập cho người dùng hiện có.

Hình ảnh minh họa

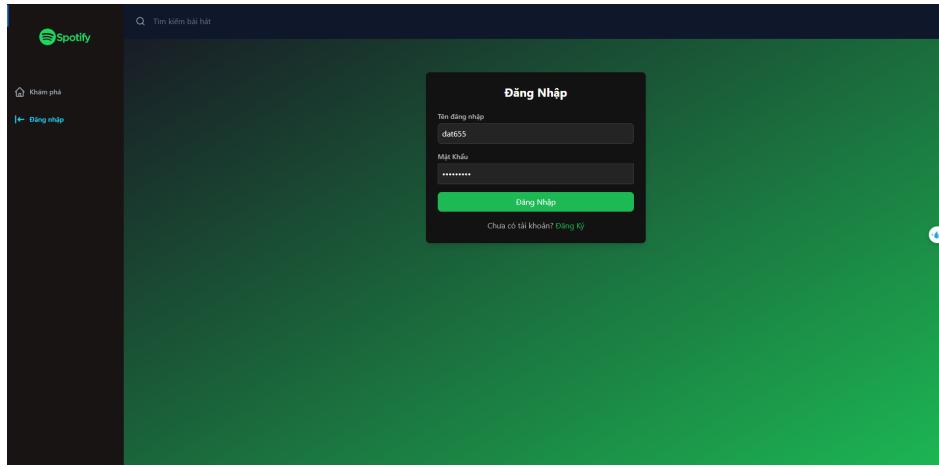


Hình 27: Giao diện trang đăng ký người dùng

- **Đăng Nhập** (`Login.jsx`):

- Biểu mẫu nhập email và mật khẩu.
- Nút *Đăng Nhập* gửi yêu cầu đến POST `/login/`, lưu `userId`, `userRole` vào `localStorage`.
- Liên kết đến trang đăng ký cho người dùng mới.

Hình ảnh minh họa

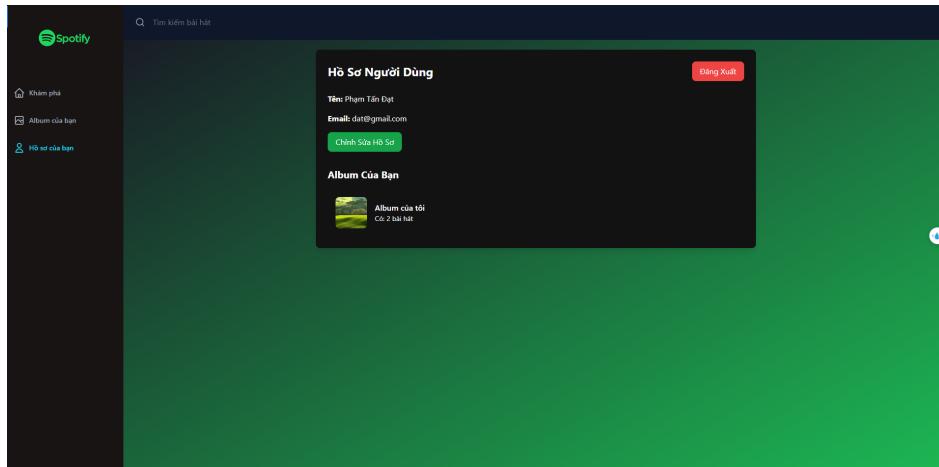


Hình 28: Giao diện trang đăng nhập người dùng

- **Hồ Sơ (Profile.jsx):**

- Hiển thị thông tin người dùng (tên, email) và danh sách album đã tạo.
- Nút *Đăng Xuất* gửi yêu cầu đến POST /logout/.
- Danh sách bài hát yêu thích với tùy chọn thêm/xóa (POST/DELETE /{user_id}/favourite).

Hình ảnh minh họa

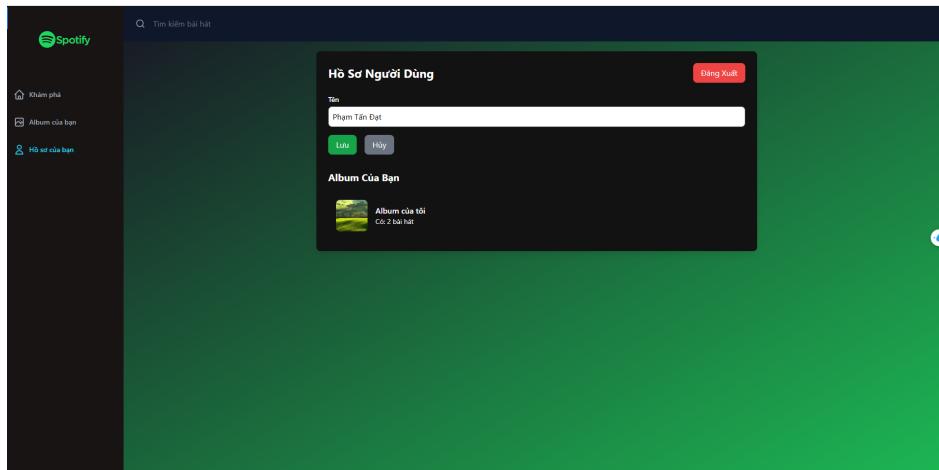


Hình 29: Giao diện trang hồ sơ người dùng

- **Chỉnh Sửa Hồ Sơ (Profile.jsx):**

- Chế độ chỉnh sửa cho phép cập nhật tên và email.
- Biểu mẫu gửi yêu cầu đến POST /update/ để lưu thay đổi.
- Thông báo thành công hoặc lỗi khi cập nhật.

Hình ảnh minh họa

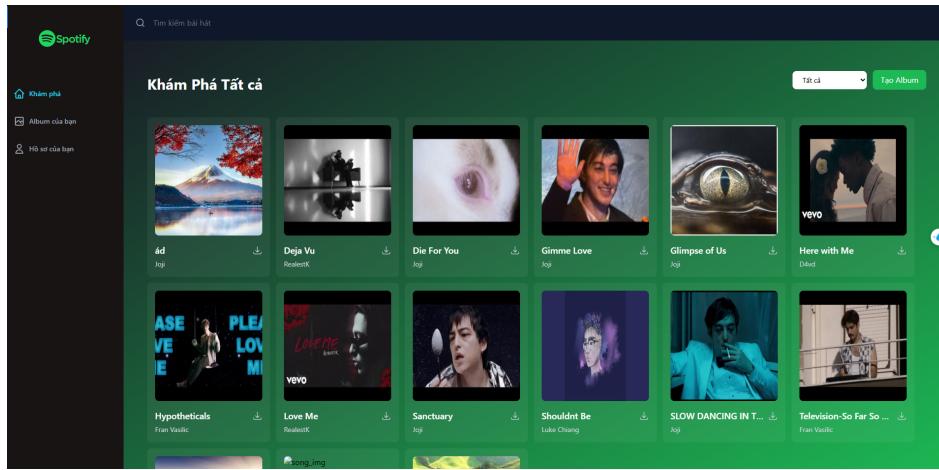


Hình 30: Giao diện chỉnh sửa hồ sơ người dùng

• Khám Phá (ArtistList.jsx):

- Hiển thị danh sách nghệ sĩ trong bố cục lưới, mỗi nghệ sĩ có tên và danh sách bài hát liên quan.
- Gọi GET /artists/ và GET /artist/details/{artist_id}/ để lấy dữ liệu.
- Nhấp vào bài hát để phát trong trình phát.

Hình ảnh minh họa



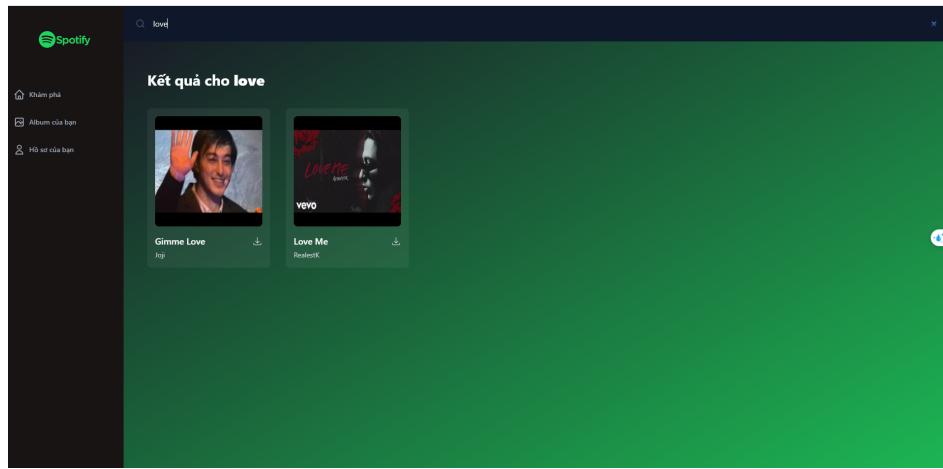
Hình 31: Giao diện trang khám phá nghệ sĩ

• Tìm Kiếm (Search.jsx):

- Ô nhập từ khóa để tìm kiếm bài hát hoặc hiển thị tất cả bài hát khi không có từ khóa.

- Kết quả hiển thị qua SongCard.jsx với tên bài hát, nghệ sĩ, và nút phát.
- Gọi GET /tracks/search/ hoặc GET /tracks/genre/{genre_id}/ để lấy dữ liệu.

Hình ảnh minh họa

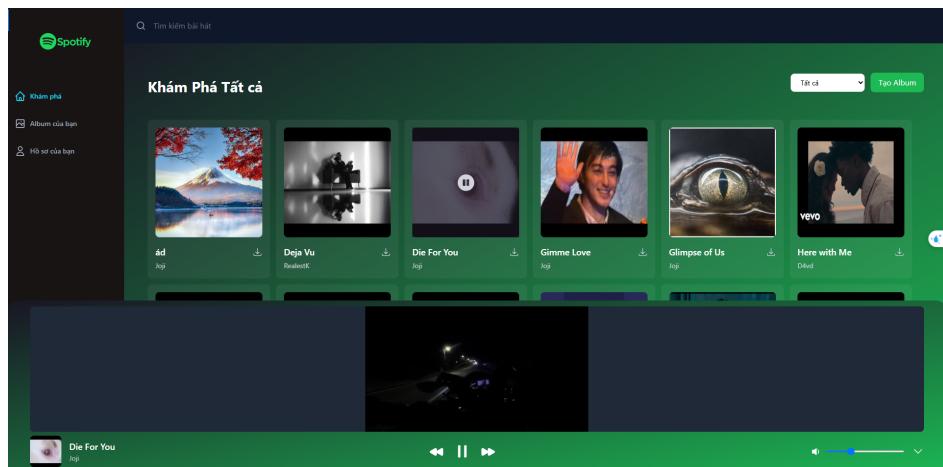


Hình 32: Giao diện trang tìm kiếm bài hát

• Xem Video Âm Nhạc và Nghe Nhạc (MusicPlayer.jsx, Player.jsx):

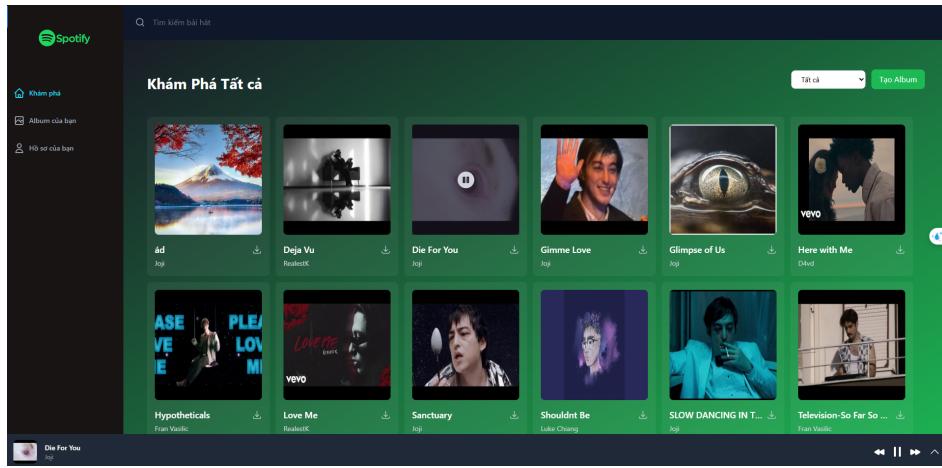
- Trình phát video HTML5 hiển thị video MP4 từ GET /tracks/tracksdetail/{track_id}
- Giao diện đầy đủ: video, thông tin bài hát (Track.jsx), nút điều khiển (Controls.jsx), và thanh âm lượng (VolumeBar.jsx).
- Giao diện thu nhỏ: thanh cố định dưới cùng với thông tin bài hát và điều khiển cơ bản.

Hình ảnh minh họa



Hình 33: Giao diện trình phát video âm nhạc

Hình ảnh minh họa

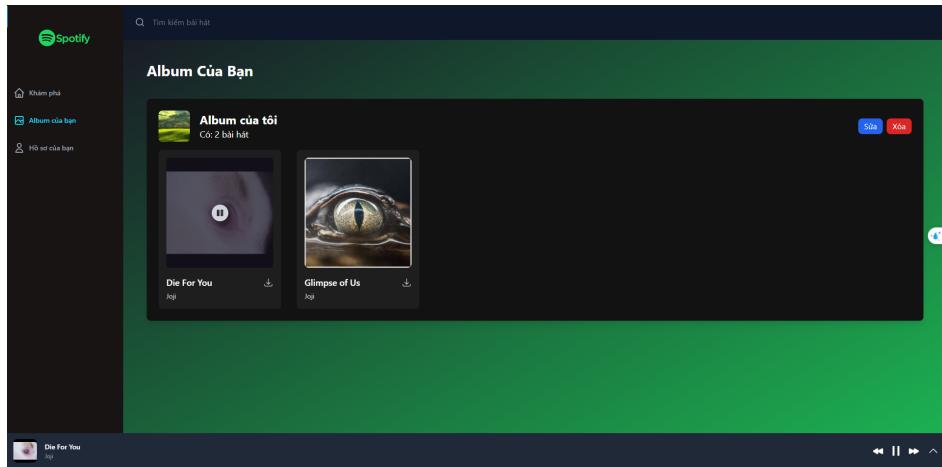


Hình 34: Giao diện nghe nhạc (chế độ thu nhỏ)

- **Album của Bạn (Profile.jsx):**

- Hiển thị danh sách album đã tạo với tên, hình ảnh, và danh sách bài hát.
- Gọi GET /{id}/albums/ để lấy dữ liệu.
- Nút chỉnh sửa hoặc xóa album liên kết với POST /albums/{album_id}/edit/ và DELETE /albums/{album_id}/delete/.

Hình ảnh minh họa



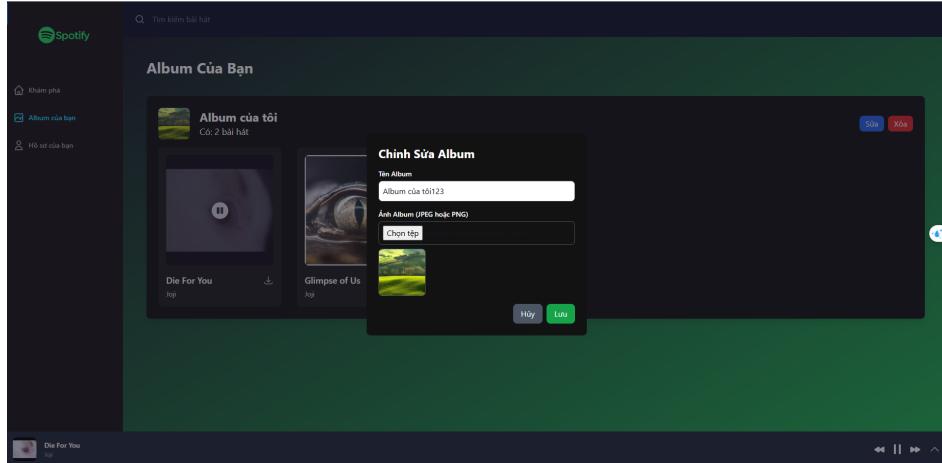
Hình 35: Giao diện danh sách album của bạn

- **Chỉnh Sửa Album (CreateAlbum.jsx):**

- Biểu mẫu chỉnh sửa tên album, hình ảnh, và danh sách bài hát (checkbox với tìm kiếm).

- Gửi yêu cầu đến POST `/id/albums/{album_id}/add-tracks/` hoặc POST `/albums/{album_id}/edit/`.
- Chuyển hướng đến `Profile.jsx` sau khi lưu.

Hình ảnh minh họa



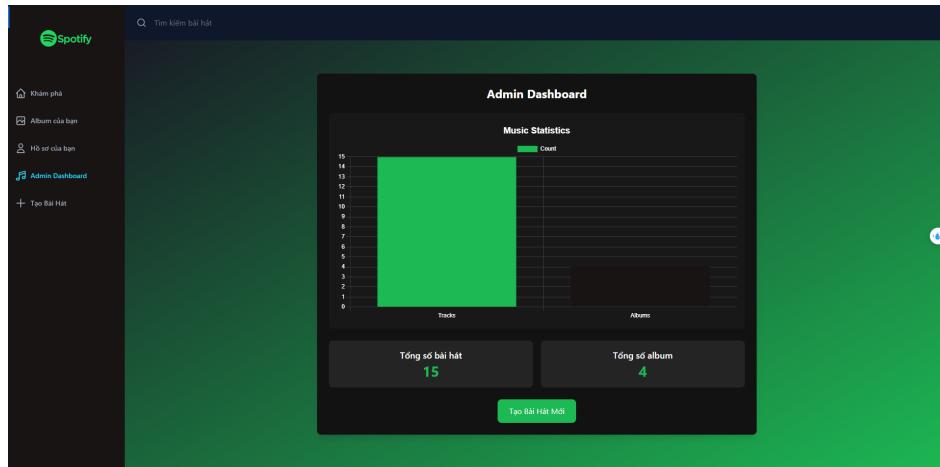
Hình 36: Giao diện chỉnh sửa album

7.2.2 Giao diện dành cho Quản Trị Viên

Quản trị viên có quyền truy cập tất cả giao diện người dùng trên, cộng thêm các giao diện chuyên biệt sau:

- **Admin Dashboard (`AdminDashboard.jsx`):**
 - Hiển thị thống kê tổng số bài hát và album bằng biểu đồ cột (Chart.js).
 - Gọi GET `/stats/tracks/` và GET `/stats/albums/` để lấy dữ liệu.
 - Nút *Tạo Bài Hát Mới* điều hướng đến trang `CreateTrack.jsx`.
 - Chỉ truy cập được khi `userRole: "admin"`, nếu không sẽ chuyển hướng đến `/`.

Hình ảnh minh họa



Hình 37: Giao diện bảng điều khiển quản trị

- **Tạo Bài Hát (CreateTrack.jsx):**

- Biểu mẫu nhập tên bài hát, chọn nhiều thể loại (checkbox với ô tìm kiếm), chọn một nghệ sĩ (dropdown), và tải lên hình ảnh/video.
- Gọi GET /artists/ và GET /genre/ để lấy danh sách lựa chọn.
- Gửi yêu cầu đến POST /tracks/create/ với multipart/form-data.
- Hiển thị thông báo thành công và chuyển hướng đến / sau khi tạo.

Hình ảnh minh họa

The figure is a screenshot of a 'Tạo Bài Hát Mới' (Create New Track) form. The form has several input fields: 'Tên bài hát' (Name) with placeholder 'Nhập tên bài hát', 'Thể loại' (Genre) with placeholder 'Tim kiếm thể loại' and a list of checked checkboxes: 'Alternative R&B', 'Gospel', 'Indie', 'Indie Pop', and 'R&B'; 'Nghệ sĩ' (Artist) with a dropdown menu showing 'Chọn nghệ sĩ'; 'Ảnh bài hát (JPG, PNG)' with a placeholder 'Chọn tệp' and a note 'Không có tệp nào được chọn'; and 'Video bài hát (MP4)' with a placeholder 'Chọn tệp' and a note 'Không có tệp nào được chọn'. At the bottom is a green button labeled 'Tạo Bài Hát'.

Hình 38: Giao diện tạo bài hát mới

8. Cài đặt và chạy ứng dụng

8.1 Cài đặt backend

Hướng dẫn cài đặt backend với Django.

8.1.1 Tạo Môi Trường Áo

```
python -m venv myvenv  
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass  
myvenv\Scripts\activate
```

8.1.2 Cài Thư viện

```
pip install -r requirements.txt
```

8.1.3 Cấu hình Cơ sở dữ liệu trong settings.py

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql',  
        'NAME': 'spotify_db',  
        'USER': 'postgres',  
        'PASSWORD': 'yourpassword',  
        'HOST': 'localhost',  
        'PORT': '5432',  
    }  
}
```

8.1.4 Migration

```
python manage.py makemigrations  
python manage.py migrate
```

8.1.5 Tạo Superuser

```
python manage.py createsuperuser
```

8.1.6 Sinh dữ liệu mẫu

```
py seed_data.py
```

8.1.7 Chạy Server

```
python manage.py runserver
```

Một số lệnh Django khác:

```
django-admin startproject 'project-name' .
python manage.py startapp 'app-name'
```

8.2 Cài đặt frontend

Hướng dẫn cài đặt frontend với ReactJS.

8.2.1 Yêu cầu Hệ thống

- **Node.js**: Phiên bản 16 hoặc cao hơn (kiểm tra bằng `node -v`).
- **npm**: Phiên bản 7 hoặc cao hơn (kiểm tra bằng `npm -v`), hoặc sử dụng Yarn.
- Trình duyệt hiện đại (Chrome, Firefox, v.v.).
- Backend Django đang chạy tại `http://127.0.0.1:8000` (xem phần 8.1).

8.2.2 Cài Đặt Môi Trường

```
npm install
```

8.2.3 Cấu Hình Tailwind CSS

- Đảm bảo tệp `tailwind.config.js` và `index.css` được cấu hình:

```
/* src/index.css */
@tailwind base;
@tailwind components;
@tailwind utilities;
```

- Nếu chưa có, khởi tạo Tailwind:

```
npx tailwindcss init
```

8.2.4 Khởi Chạy Ứng Dụng

```
npm start
```

- Ứng dụng sẽ mở tại <http://localhost:3000>.

8.3 Môi trường chạy

Để cài đặt và chạy thành công ứng dụng **Spotify Clone**, hệ thống cần đáp ứng các yêu cầu về môi trường sau:

- **Hệ điều hành hỗ trợ:**
 - Windows 10/11 (64-bit)
 - macOS 11 trở lên
 - Linux (khuyến nghị Ubuntu 20.04 trở lên)
- **Backend (Django):**
 - Python 3.10 hoặc cao hơn
 - PostgreSQL 13 hoặc cao hơn (đã cài đặt và cấu hình)
 - Virtualenv hoặc module `venv` để tạo môi trường ảo
- **Frontend (ReactJS):**
 - Node.js phiên bản 16 trở lên (kiểm tra với `node -v`)
 - npm phiên bản 7 trở lên hoặc yarn (kiểm tra với `npm -v`)
 - Trình duyệt hiện đại: Google Chrome, Firefox, Microsoft Edge
- **Khác:**
 - Kết nối Internet để cài đặt các gói thư viện từ `pip` và `npm`
 - Trình soạn thảo mã nguồn như VS Code để thuận tiện trong phát triển và cấu hình

Lưu ý: Cần đảm bảo backend Django đang chạy tại địa chỉ <http://127.0.0.1:8000> để frontend React có thể kết nối và hoạt động đúng cách.

9. Phân công nhiệm vụ

- **Phạm Tân Đạt:** Chịu trách nhiệm phát triển toàn bộ phần *frontend* của ứng dụng.
- **Mộc Nghĩa Tân:** Chịu trách nhiệm phát triển phần *backend* liên quan đến *musicapp*, bao gồm việc tạo và xây dựng các API quản lý bài hát, nghệ sĩ, album, và thể loại.
- **Lê Hữu Trí:** Chịu trách nhiệm phát triển phần *backend* liên quan đến *userapp*, bao gồm việc phát triển và chỉnh sửa các tính năng liên quan đến người dùng như đăng ký, đăng nhập, quản lý album người dùng, và các tính năng yêu thích bài hát.

10. Tài liệu tham khảo

1. Django: <https://docs.djangoproject.com/>
2. Django REST Framework: <https://www.django-rest-framework.org/>
3. Django Cors Headers: <https://pypi.org/project/django-cors-headers/>
4. PostgreSQL: <https://www.postgresql.org/docs/>
5. Python: <https://docs.python.org/3/>
6. React Official Documentation, <https://reactjs.org/docs/getting-started.html>
7. Bootstrap Documentation, <https://getbootstrap.com/docs/5.0/getting-started/introduction/>
8. Tailwind CSS Documentation, <https://tailwindcss.com/docs>
9. JavaScript Info, <https://javascript.info/>