Trường Đại Học Sài Gòn

Khoa Công Nghệ Thông Tin

Xây dựng ứng dụng Clone Spotify

3121410071 - Huỳnh Gia Bảo - huynhgiabaodhsg@gmail.com

3121410181 - Nguyễn Huỳnh Tuấn Hào - nhtuanhao@gmail.com

3121410301 - Huỳnh Thanh Lộc - huynhthanhloc913@gmail.com

Giảng viên: Từ Lãng Phiêu

Ngày 23 tháng 5 năm 2025

Lời cam đoan

Chúng em, Nhóm 8, xác nhận rằng đây là công trình của riêng nhóm và các hình ảnh, bảng biểu, phương trình, đoạn mã, tác phẩm nghệ thuật và hình minh họa trong báo cáo này là nguyên bản và không được lấy từ công trình của bất kỳ người nào khác, ngoại trừ những trường hợp mà các công trình của người khác đã được trích dẫn, ghi nhận và tham khảo rõ ràng. Chúng em hiểu rằng việc không tuân thủ điều này sẽ bị coi là hành vi đạo văn. Đạo văn là một hình thức vi phạm học thuật và sẽ bị xử lý theo quy định.

Chúng em đồng ý cho phép chia sẻ bản sao báo cáo này với sinh viên các khóa sau như một tài liệu tham khảo.

Chúng em đồng ý cho phép thư viện sử dụng báo cáo này rộng rãi hơn cho các thành viên của trường và công chúng quan tâm đến giảng dạy, học tập và nghiên cứu.

Nhóm 8
Ngày 23 tháng 5 năm 2025

Mục lục

Danh sách hình vẽ

Danh sách bảng

Danh sách từ viết tắt

PTPMMNM	Phát triển phần mềm mã nguồn mở (tên môn học)
MVC	Mô hình Model/View/Controller
C/S/R	Cấu trúc (pattern) Controller/Service/Repository

Chương 1

Lời mở đầu

1.1 Bối cảnh

Trong bối cảnh sự phát triển mạnh mẽ của âm nhạc trực tuyến và nhu cầu giải trí đa dạng của người dùng, các nền tảng phát nhạc trực tuyến như Spotify đã trở thành một phần không thể thiếu trong cuộc sống số. Với khả năng truy cập hàng triệu bài hát, podcast và video âm nhạc mọi lúc mọi nơi, những ứng dụng này mang lại trải nghiệm âm nhạc liền mạch và cá nhân hóa.

Dự án này tập trung vào việc phát triển một phần mềm mô phỏng Spotify (Spotify Clone) nhằm khám phá các công nghệ web hiện đại và xây dựng một ứng dụng phát nhạc trực tuyến cơ bản. Dự án sẽ đi sâu vào việc thiết kế giao diện người dùng trực quan, xây dựng các chức năng phát nhạc và video, quản lý thư viện cá nhân của người dùng, cũng như cung cấp một trang quản trị hệ thống. Các công nghệ chủ yếu được sử dụng trong dự án bao gồm ngôn ngữ lập trình Python, framework backend Django, thư viện frontend React, hệ quản trị cơ sở dữ liệu PostgreSQL, hạ tầng đám mây AWS để host ứng dụng, dịch vụ lưu trữ S3, và giao thức gRPC với Protobuf cho giao tiếp microservice.

1.2 Phát biểu bài toán

Bài toán đặt ra là xây dựng một ứng dụng web Spotify Clone có khả năng cung cấp các chức năng phát nhạc và video âm nhạc cơ bản, cho phép người dùng tạo và quản lý thư viện nhạc cá nhân (album, bài hát yêu thích), đồng thời cung cấp một giao diện quản trị cho phép quản lý hệ thống. Ứng dụng cần đảm bảo khả năng hoạt động ổn định và có khả năng mở rộng thông qua kiến trúc microservice và triển khai trên môi trường đám mây.

Các khía cạnh cụ thể cần được giải quyết bao gồm:

- Thiết kế giao diện người dùng (frontend) thân thiện và dễ sử dụng cho các chức năng phát nhạc, video, quản lý thư viện và chat bằng React.
- Xây dựng hệ thống backend bằng Python và Django theo kiến trúc MVC kết hợp Controller/Service/Repository và microservice để xử lý các yêu cầu từ frontend, quản lý dữ liệu âm nhạc, video và người dùng.
- Thiết kế và xây dựng cơ sở dữ liệu PostgreSQL để lưu trữ thông tin về bài hát, video, album, người dùng và các dữ liệu liên quan khác.
- Hiện thực hóa các chức năng phát nhạc, phát video, tải video (tùy chọn), tạo và quản lý thư viện cá nhân, và trang quản trị.

- Triển khai ứng dụng trên hạ tầng AWS và sử dụng S3 cho lưu trữ nội dung.
- Sử dụng gRPC và Protobuf cho giao tiếp hiệu quả giữa các microservice.
- Tích hợp chức năng chat trực tiếp vào giao diện web.

1.3 Mục tiêu và nhiệm vụ

Mục tiêu: Mục tiêu chính của dự án là xây dựng thành công một phần mềm Spotify Clone cơ bản, có khả năng trình diễn các chức năng phát nhạc, video âm nhạc, quản lý thư viện người dùng và cung cấp giao diện quản trị trên nền tảng web, sử dụng các công nghệ Python, Django (theo kiến trúc MVC và microservice), React, PostgreSQL và triển khai trên AWS.

Nhiệm vụ: Để đạt được mục tiêu trên, nhóm chúng em sẽ thực hiện các nhiệm vụ cụ thể sau:

- Nghiên cứu và phân tích yêu cầu của đề bài, tìm hiểu về kiến trúc và các tính năng của Spotify.
- Lựa chọn và làm quen với các công nghệ phát triển (Python, Django, React, PostgreSQL, AWS, Docker, gRPC, Protobuf).
- Thiết kế cơ sở dữ liệu PostgreSQL.
- Thiết kế kiến trúc tổng thể của ứng dụng theo mô hình MVC và microservice, bao gồm việc container hóa các service bằng Docker.
- Thiết kế giao diện người dùng (wireframe và mockup) bằng React.
- Phát triển các microservice backend (API) bằng Python và Django theo kiến trúc Controller/Service/Repository cho việc phát nhạc, video, quản lý người dùng và thư viện.
- Phát triển giao diện người dùng (frontend) bằng React tương tác với các microservice backend.
- Hiện thực hóa trang quản trị hệ thống.
- Triển khai và quản lý ứng dụng trên hạ tầng AWS, sử dụng S3 cho lưu trữ nội dung.
- Thiết lập giao tiếp giữa các microservice bằng gRPC và Protobuf.
- Nghiên cứu và tích hợp chức năng chat vào ứng dụng.
- Kiểm thử và gỡ lỗi phần mềm.
- Viết báo cáo chi tiết về quá trình phát triển và kết quả của dự án.
- Chuẩn bị tài liệu hướng dẫn cài đặt và sử dụng (README.md).

1.4 Phương pháp tiếp cận

Dự án này sẽ được triển khai theo phương pháp phát triển phần mềm Waterfall cho các giai đoạn chính, kết hợp với mô hình MVC (Model-View-Controller) trong kiến trúc Django backend và kiến trúc microservice với việc sử dụng Docker để container hóa các dịch vụ. Chúng em sẽ tập trung vào việc phân tách ứng dụng thành các dịch vụ nhỏ độc lập, giao tiếp với nhau thông qua giao thức gRPC và Protobuf, nhằm đảm bảo tính linh hoạt, khả năng mở rộng và dễ bảo trì của hệ thống.

1.4.1 Công nghệ sử dụng

Các công nghệ chính được sử dụng trong dự án bao gồm:

- **Ngôn ngữ lập trình Backend:** Python 3
- **Framework Backend:** Django (theo kiến trúc MVC và microservice)
- **Ngôn ngữ lập trình Frontend:** JavaScript
- **Framework Frontend:** React
- **Hệ quản trị cơ sở dữ liệu:** PostgreSQL
- **Hạ tầng đám mây:** Amazon Web Services (AWS)
- **Lưu trữ nội dung:** Amazon S3
- **Container hóa:** Docker
- **Giao tiếp Microservice:** gRPC
- **Serialization:** Protobuf
- **Công cụ quản lý phiên bản:** Git
- **Nền tảng cộng tác phát triển:** GitHub

1.4.2 Quy trình phát triển

Quy trình phát triển dự kiến sẽ bao gồm các giai đoạn chính theo mô hình Waterfall:

1. Phân tích yêu cầu và lập kế hoạch chi tiết.
2. Thiết kế cơ sở dữ liệu PostgreSQL và kiến trúc microservice.
3. Thiết kế giao diện người dùng (UI/UX) bằng React.
4. Phát triển các microservice backend (API) bằng Django theo kiến trúc Controller/Service/Repository.
5. Phát triển frontend bằng React và tích hợp với các microservice.
6. Triển khai và container hóa các microservice bằng Docker, triển khai trên AWS và cấu hình S3.
7. Thiết lập giao tiếp gRPC giữa các microservice.
8. Kiểm thử và gỡ lỗi toàn bộ hệ thống.
9. Viết tài liệu và báo cáo chi tiết.
10. Đóng gói và chuẩn bị tài liệu hướng dẫn cài đặt và sử dụng (README.md).

Chương 2

Các công nghệ sử dụng và Quy trình phát triển

2.1 Công nghệ sử dụng

2.1.1 Python

Python là một ngôn ngữ lập trình cấp cao, thông dịch, đa mục đích. Nó nổi tiếng với cú pháp rõ ràng, dễ đọc và một hệ sinh thái thư viện phong phú, mạnh mẽ. Trong dự án này, Python được sử dụng chủ yếu cho việc phát triển backend. *Tìm hiểu thêm tại:* https://www.python.org/

2.1.2 Django

Django là một framework web cấp cao của Python, tuân theo kiến trúc Model-Template-View (MTV). Django cung cấp nhiều công cụ và tính năng sẵn có, giúp việc phát triển các ứng dụng web phức tạp trở nên nhanh chóng và hiệu quả. Chúng em sử dụng Django để xây dựng các API backend và quản lý logic nghiệp vụ của ứng dụng. *Tìm hiểu thêm tại:* https://www.djangoproject.com/

2.1.3 React

React là một thư viện JavaScript mã nguồn mở để xây dựng giao diện người dùng (UI) hoặc các thành phần UI. Nó được duy trì bởi Facebook và một cộng đồng lớn các nhà phát triển. React nổi bật với khả năng xây dựng các ứng dụng web đơn trang (SPA) hiệu suất cao và trải nghiệm người dùng mượt mà. Chúng em sử dụng React để phát triển frontend của ứng dụng Spotify Clone. *Tìm hiểu thêm tại:* https://react.dev/

2.1.4 PostgreSQL

PostgreSQL là một hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) mã nguồn mở mạnh mẽ và tuân thủ tiêu chuẩn SQL. Nó nổi tiếng với tính ổn định, độ tin cậy và nhiều tính năng nâng cao. Chúng em sử dụng PostgreSQL để lưu trữ tất cả dữ liệu liên quan đến ứng dụng, bao gồm thông tin người dùng, bài hát, album và các dữ liệu khác. *Tìm hiểu thêm tại:* https://www.postgresql.org/

2.1.5 Amazon Web Services (AWS)

Amazon Web Services (AWS) là một nền tảng dịch vụ đám mây toàn diện và được sử dụng rộng rãi trên toàn thế giới. Nó cung cấp nhiều dịch vụ khác nhau, bao gồm điện toán, lưu trữ, cơ sở dữ liệu và nhiều hơn nữa. Chúng em sử dụng AWS để host ứng dụng và tận dụng các dịch vụ của nó để đảm bảo tính khả dụng và khả năng mở rộng. *Tìm hiểu thêm tại:* https://aws.amazon.com/

2.1.6 Amazon S3 (Simple Storage Service)

Amazon S3 là một dịch vụ lưu trữ đối tượng có khả năng mở rộng cao, an toàn và hiệu suất cao của AWS. Chúng em sử dụng S3 để lưu trữ các tệp âm nhạc, video và các nội dung tĩnh khác của ứng dụng. *Tìm hiểu thêm tại:* https://aws.amazon.com/s3/

2.1.7 Docker

Docker là một nền tảng cho phép đóng gói và chạy ứng dụng trong các container. Container là các gói phần mềm độc lập, chứa tất cả các thư viện, dependencies và cấu hình cần thiết để chạy ứng dụng. Chúng em sử dụng Docker để container hóa các microservice của ứng dụng, giúp việc triển khai và quản lý trở nên dễ dàng và nhất quán trên các môi trường khác nhau. *Tìm hiểu thêm tại:* https://www.docker.com/

2.1.8 gRPC

gRPC là một framework RPC (Remote Procedure Call) mã nguồn mở, hiệu suất cao, được phát triển bởi Google. Nó sử dụng Protocol Buffers (Protobuf) làm ngôn ngữ định nghĩa interface, cho phép định nghĩa các dịch vụ và cấu trúc dữ liệu một cách hiệu quả. Chúng em sử dụng gRPC để xây dựng các giao tiếp giữa các microservice trong kiến trúc của ứng dụng. *Tìm hiểu thêm tại:* https://grpc.io/

2.1.9 Protocol Buffers (Protobuf)

Protocol Buffers (Protobuf) là một cơ chế serialization dữ liệu ngôn ngữ trung lập, nền tảng trung lập, có thể mở rộng. Nó được sử dụng với gRPC để định nghĩa cấu trúc dữ liệu (message) và các dịch vụ, đảm bảo việc truyền dữ liệu giữa các microservice diễn ra một cách nhanh chóng và hiệu quả. *Tìm hiểu thêm tại:* https://protobuf.dev/

2.2 Quy trình phát triển

2.2.1 Mô hình Waterfall

Mô hình Waterfall là một mô hình phát triển phần mềm tuyến tính, tuần tự. Trong mô hình này, các giai đoạn phát triển (phân tích yêu cầu, thiết kế, hiện thực, kiểm thử, triển khai, bảo trì) được thực hiện theo một trình tự nhất định, với mỗi giai đoạn chỉ bắt đầu khi giai đoạn trước đó đã hoàn thành. Chúng em áp dụng mô hình Waterfall cho các giai đoạn chính của dự án để đảm bảo một quy trình có cấu trúc và dễ quản lý. *Tìm hiểu thêm tại:* https://en.wikipedia.org/wiki/Waterfall_model

2.2.2 Mô hình MVC (Model-View-Controller)

MVC là một mẫu thiết kế phần mềm phổ biến được sử dụng để tổ chức mã nguồn trong các ứng dụng web. Nó chia ứng dụng thành ba thành phần chính:

- **Model:** Quản lý dữ liệu của ứng dụng.
- **View:** Hiển thị dữ liệu cho người dùng.
- **Controller:** Xử lý các tương tác của người dùng và cập nhật Model và View.

Chúng em áp dụng mô hình MVC trong framework Django để xây dựng backend của ứng dụng một cách có cấu trúc và dễ bảo trì. *Tìm hiểu thêm tại:* https://en.wikipedia.org/wiki/Model-view-controller

2.2.3 Kiến trúc Microservice

Kiến trúc Microservice là một phương pháp tiếp cận để xây dựng một ứng dụng như một tập hợp các dịch vụ nhỏ, độc lập, giao tiếp với nhau thông qua mạng. Mỗi microservice thường tập trung vào một chức năng cụ thể của ứng dụng và có thể được phát triển, triển khai và mở rộng một cách độc lập. Chúng em sử dụng kiến trúc microservice để tăng tính linh hoạt, khả năng mở rộng và khả năng chịu lỗi của ứng dụng Spotify Clone. *Tìm hiểu thêm tại:* https://martinfowler.com/articles/microservices.html

2.2.4 Container hóa với Docker

Container hóa là quá trình đóng gói một ứng dụng và tất cả các dependencies của nó (thư viện, runtime, công cụ hệ thống, code) vào một container duy nhất. Docker là một nền tảng hàng đầu cho việc container hóa. Việc sử dụng Docker giúp đảm bảo rằng ứng dụng có thể chạy một cách nhất quán trên bất kỳ môi trường nào hỗ trợ Docker, từ máy phát triển đến môi trường production trên AWS. *Tìm hiểu thêm tại:* https://www.docker.com/what-is-a-container/

2.2.5 Controller/Service/Repository Pattern

Đây là một pattern thường được sử dụng trong các ứng dụng backend phức tạp để tách biệt các tầng trách nhiệm khác nhau:

- **Controller:** Xử lý các request từ bên ngoài (frontend) và điều hướng logic nghiệp vụ.
- **Service:** Chứa logic nghiệp vụ cốt lõi của ứng dụng.
- **Repository:** Chịu trách nhiệm cho việc truy cập và thao tác với dữ liệu từ cơ sở dữ liệu.

Chúng em áp dụng pattern này trong các microservice Django để tổ chức code một cách rõ ràng và dễ quản lý.

2.2.6 gRPC và Protobuf trong giao tiếp Microservice

gRPC là framework RPC và Protobuf là ngôn ngữ serialization được sử dụng để định nghĩa và trao đổi dữ liệu giữa các microservice. Việc sử dụng gRPC và Protobuf giúp cho giao tiếp giữa các dịch vụ trở nên nhanh chóng, hiệu quả và có cấu trúc rõ ràng nhờ vào việc định nghĩa contract (interface) một cách chặt chẽ. *Tìm hiểu thêm về gRPC:* https://grpc.io/docs/what-is-grpc/core-concepts/ *Tìm hiểu thêm về Protobuf:* https://protobuf.dev/getting-started/overview/

Chương 3

Cấu trúc dự án và các tính năng

3.1 Cấu trúc mã nguồn và mô hình ứng dụng

Dự án Spotify Clone được xây dựng theo kiến trúc microservice, với mã nguồn được tổ chức trong thư mục gốc. Môi trường phát triển được thiết lập bằng Dev Containers và công cụ quản lý gói `uv`. File `serverclontify.sql` được sử dụng để mount vào PostgreSQL, đóng vai trò như một bản backup cơ sở dữ liệu. Thư mục `baseSetup` chứa các file mẫu để tạo nhanh các service mới.

3.1.1 Cấu trúc thư mục gốc

Cấu trúc thư mục gốc của dự án bao gồm:

```
./
+-- appServices/
|   +-- auth/
|   +-- common/
|   +-- music/
|   +-- storage/
|   +-- users/
+-- baseSetup/
+-- database/
+-- .devcontainer/
+-- example.env
+-- images/
+-- requirements.txt
+-- serverclontify.sql
+-- test/
+-- uv.lock
+-- pyproject.toml
```

Trong đó:

- `appServices/`: Chứa các microservice của ứng dụng.
- `baseSetup/`: Chứa các file cấu hình mẫu để tạo service mới.
- `database/`: Chứa các script liên quan đến cơ sở dữ liệu.

- `.devcontainer/`: Chứa cấu hình cho môi trường phát triển Dev Containers.
- `.env`: File cho các biến môi trường.
- `images/`: Chứa các hình ảnh sử dụng trong tài liệu hoặc ứng dụng.
- `requirements.txt`: Liệt kê các dependency chung của dự án.
- `serverclontify.sql`: File SQL backup cho cơ sở dữ liệu PostgreSQL.
- `test/`: Chứa các test case của dự án.
- `uv.lock`: File lock cho công cụ quản lý gói `uv`.
- `pyproject.toml`: File cấu hình cho các công cụ build và quản lý dự án Python.

3.1.2 Cấu trúc một Microservice

Mỗi microservice trong thư mục `appServices` được xây dựng dựa trên framework Django và tuân theo mô hình MVC kết hợp pattern Controller/Service/Repository. Cấu trúc thư mục điển hình của một service (ví dụ: `auth`) như sau:

```
appServices/*/
+-- app/
|   +-- __init__.py
|   +-- controllers/
|   |   +-- ...
|   +-- entities/
|   |   +-- ...
|   +-- grpc/
|   |   +-- protos/
|   |       +-- *Service.proto
|   |       +-- *Service_pb2.py
|   |       +-- *Service_pb2_grpc.py
|   +-- migrations/
|   |   +-- ...
|   +-- models.py
|   +-- repositories/
|   |   +-- ...
|   +-- serializers/
|   |   +-- ...
|   +-- services/
|   |   +-- ...
|   +-- urls.py
+-- common/
+-- config/
|   +-- __init__.py
|   +-- asgi.py
|   +-- settings.py
|   +-- urls.py
|   +-- wsgi.py
+-- Dockerfile
```

```
+-- manage.py
+-- requirements.txt
```

Trong đó, thư mục app/ chứa các thành phần MVC và pattern C/S/R, thư mục grpc/protos/ chứa các file định nghĩa gRPC và các file Python được build từ chúng.

File Dockerfile của mỗi service chịu trách nhiệm build image Docker cho service đó. Quá trình build bao gồm cài đặt các dependency từ requirements.txt và build các file Python từ các file .proto.

3.1.3 Các Microservice hiện tại

Hiện tại, dự án bao gồm các microservice sau:

- **auth service**: Quản lý việc đăng ký và đăng nhập người dùng, bao gồm 2 bảng account và roles. Khi đăng ký, service này gọi gRPC sang users service để tạo profile người dùng.
- **users service**: Quản lý thông tin người dùng thông qua bảng profile. Trong tương lai, service này sẽ có thêm bảng favorite để quản lý quan hệ nhiều-nhiều giữa người dùng và bài hát.
- **music service**: Quản lý thông tin về nhạc, thể loại và album với 6 bảng: albums, albumSong, genres, genreSong, songs, songSubArtist. Service này tương tác với users service qua gRPC để kiểm tra thông tin người dùng trước khi thêm nhạc hoặc sub-artist, và tương tác với storage service qua gRPC để lấy thông tin về phiên upload trên S3 (lưu ID phiên storage thay vì link trực tiếp).
- **storage service**: Quản lý các phiên upload thông qua bảng storageData.
- **common/**: Chứa các module hoặc code dùng chung giữa các service.

3.2 Xây dựng Database cho Project

Dự án sử dụng PostgreSQL làm hệ quản trị cơ sở dữ liệu. Mỗi microservice quản lý các bảng dữ liệu riêng biệt để đảm bảo tính độc lập và phân tách mối quan tâm. Dưới đây là mô tả các bảng dữ liệu chính trong từng microservice:

3.2.1 Database của auth service

Bảng accounts: Lưu trữ thông tin tài khoản người dùng.

- id (UUID, primary key): ID duy nhất của tài khoản.
- roleId (UUID): ID của vai trò người dùng, tham chiếu đến bảng roles.
- username (VARCHAR(50), unique): Tên đăng nhập duy nhất của người dùng.
- email (EmailField, unique): Địa chỉ email duy nhất của người dùng.
- password (TEXT): Mật khẩu đã được hash của người dùng.

Bảng roles: Lưu trữ thông tin về các vai trò người dùng (ví dụ: admin, user).

- id (UUID, primary key): ID duy nhất của vai trò.
- name (VARCHAR(50), unique): Tên duy nhất của vai trò.
- description (TEXT, nullable): Mô tả chi tiết về vai trò.

3.2.2 Database của music service

Bảng `albums`: Lưu trữ thông tin về các album nhạc.

- `id` (UUID, primary key): ID duy nhất của album.
- `name` (VARCHAR(255)): Tên của album.
- `description` (TEXT, nullable): Mô tả chi tiết về album.
- `storageImageId` (UUID, nullable): ID tham chiếu đến ảnh bìa của album trong bảng `storageData` của `storage service`.
- `artistId` (UUID): ID của nghệ sĩ tạo album.

Bảng `albumSong`: Bảng trung gian thể hiện mối quan hệ nhiều-nhiều giữa album và bài hát.

- `albumId` (UUID): ID của album, tham chiếu đến bảng `albums`.
- `songId` (UUID): ID của bài hát, tham chiếu đến bảng `songs`.
- `order` (PositiveIntegerField, nullable): Thứ tự của bài hát trong album.

Bảng `genres`: Lưu trữ thông tin về các thể loại nhạc.

- `id` (UUID, primary key): ID duy nhất của thể loại.
- `name` (VARCHAR(100), unique): Tên duy nhất của thể loại.
- `description` (TEXT, nullable): Mô tả chi tiết về thể loại.

Bảng `genreSong`: Bảng trung gian thể hiện mối quan hệ nhiều-nhiều giữa thể loại và bài hát.

- `songID` (UUID): ID của bài hát, tham chiếu đến bảng `songs`.
- `genreID` (UUID): ID của thể loại, tham chiếu đến bảng `genres`.

Bảng `songs`: Lưu trữ thông tin về các bài hát.

- `id` (UUID, primary key): ID duy nhất của bài hát.
- `title` (VARCHAR(255)): Tiêu đề của bài hát.
- `description` (VARCHAR, nullable): Mô tả ngắn về bài hát.
- `artistId` (UUID): ID của nghệ sĩ thể hiện bài hát.
- `storageId` (UUID, unique): ID tham chiếu đến file âm thanh của bài hát trong bảng `storageData` của `storage service`.
- `storageImageId` (UUID, nullable): ID tham chiếu đến ảnh bìa của bài hát trong bảng `storageData` của `storage service`.
- `duration` (PositiveIntegerField, nullable): Thời lượng của bài hát (tính bằng giây).
- `songType` (VARCHAR(255)): Loại bài hát (ví dụ: audio, video).

Bảng `songSubArtist`: Bảng trung gian thể hiện mối quan hệ nhiều-nhiều giữa bài hát và các nghệ sĩ phụ (sub-artist).

- `songID` (UUID): ID của bài hát, tham chiếu đến bảng `songs`.
- `subArtistID` (UUID): ID của nghệ sĩ phụ.

3.2.3 Database của storage service

Bảng `storageData`: Lưu trữ thông tin về các file đã upload.

- `id` (UUID, primary key): ID duy nhất của bản ghi storage.
- `userId` (UUID): ID của người dùng đã tải file lên.
- `fileName` (VARCHAR(255), unique): Tên file gốc.
- `fileType` (VARCHAR(255)): Loại file (ví dụ: audio/mpeg, video/mp4, image/jpeg).
- `fileSize` (IntegerField): Kích thước file (tính bằng byte).
- `fileUrl` (URLField, nullable): URL truy cập đến file trên S3.
- `description` (TEXT, nullable): Mô tả về file.

3.2.4 Database của users service

Bảng `profiles`: Lưu trữ thông tin hồ sơ người dùng.

- `id` (UUID, primary key): ID duy nhất của profile.
- `accountID` (UUID, unique): ID tham chiếu đến tài khoản người dùng trong bảng `accounts` của `auth service`.
- `fullName` (TEXT(150)): Tên đầy đủ của người dùng.
- `avatarUrl` (URLField, nullable): URL đến ảnh đại diện của người dùng.
- `bio` (TEXT(500), nullable): Tiểu sử ngắn của người dùng.
- `dateOfBirth` (DateField, nullable): Ngày sinh của người dùng.
- `phoneNumber` (CharField(20), nullable): Số điện thoại của người dùng.

3.3 Các tính năng được xây dựng

Dưới đây là danh sách các tính năng cơ bản đã được xây dựng trong dự án:

3.3.1 Chức năng đăng nhập

Chức năng đăng nhập cho phép người dùng truy cập vào hệ thống bằng tài khoản cá nhân. Người dùng cần nhập đúng tên đăng nhập và mật khẩu đã đăng ký trước đó. Hệ thống sẽ xác thực thông tin và chuyển hướng người dùng đến trang chính nếu thành công.

3.3.2 Chức năng đăng ký

Chức năng đăng ký cho phép người dùng tạo tài khoản mới để sử dụng hệ thống. Người dùng cần cung cấp thông tin như tên đăng nhập, mật khẩu và các thông tin cá nhân cần thiết. Sau khi đăng ký thành công, người dùng có thể sử dụng tài khoản để đăng nhập và trải nghiệm dịch vụ.

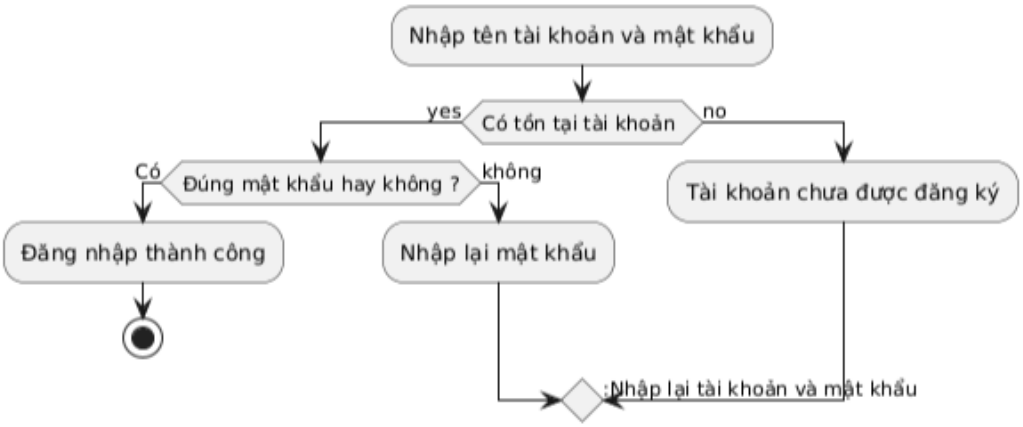

Figure 3.1: Flowchart chức năng đăng nhập

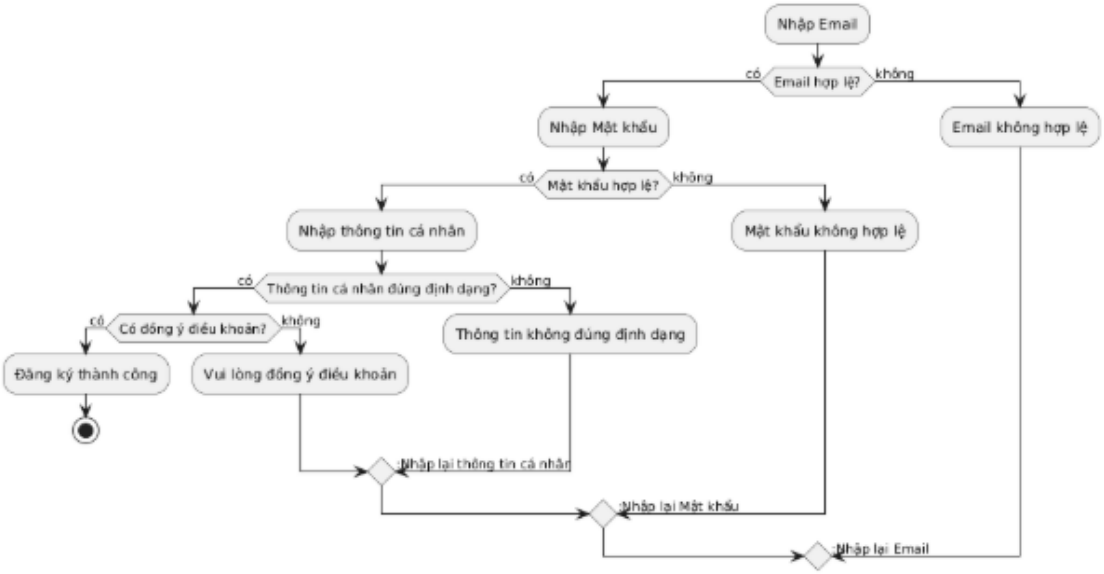

Figure 3.2: Flowchart chức năng đăng ký

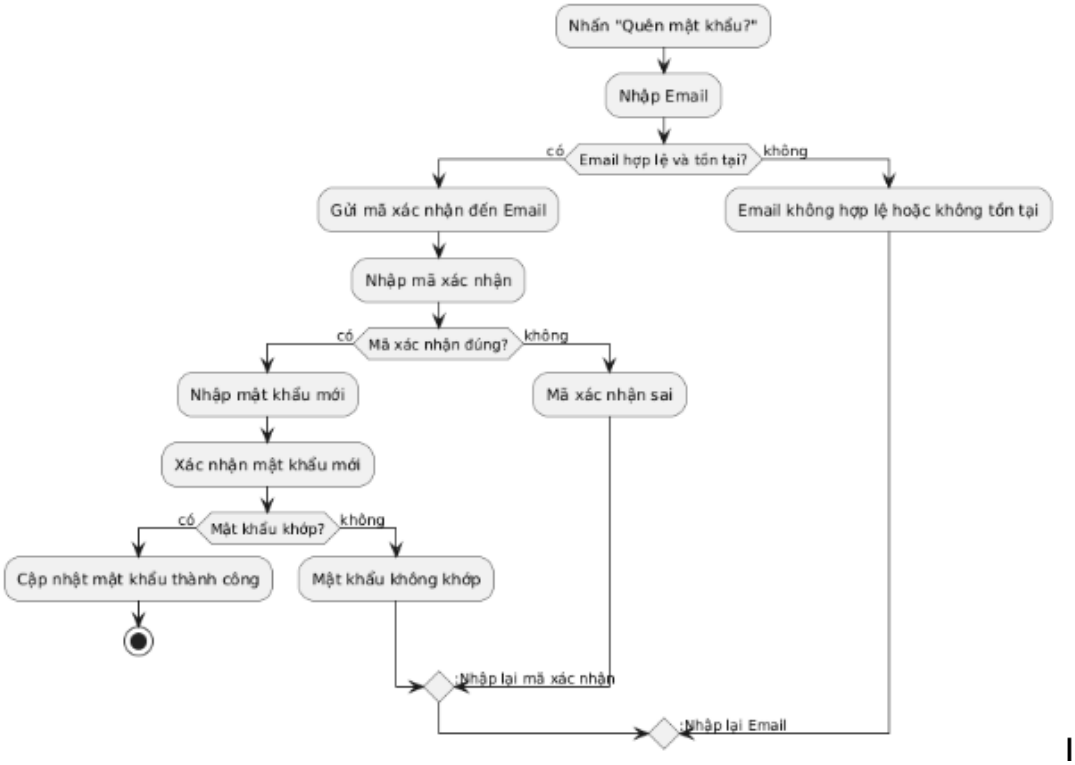

Figure 3.3: Flowchart chức năng quên mật khẩu

3.3.3 Chức năng quên mật khẩu

Chức năng quên mật khẩu cho phép người dùng khôi phục mật khẩu bằng cách nhập email đã đăng ký và nhận liên kết hoặc mã xác thực để thiết lập mật khẩu mới, đảm bảo bảo mật tài khoản và hỗ trợ truy cập lại hệ thống một cách nhanh chóng.

3.3.4 Màn hình trang chủ

Trang chủ hiển thị danh sách nhạc nổi bật, playlist cá nhân hóa và các đề xuất dựa trên thói quen nghe nhạc của người dùng, giúp trải nghiệm âm nhạc trở nên phong phú, trực quan và sinh động như giao diện của Spotify.

3.3.5 Màn hình hồ sơ

Trang hồ sơ cá nhân hiển thị ảnh đại diện, tên người dùng và thay đổi mật khẩu giúp người dùng quản lý thông tin của mình dễ hơn.

3.3.6 Màn hình nghệ sĩ

Trang nghệ sĩ hiển thị ảnh đại diện, tiểu sử, số người theo dõi, danh sách album, bài hát nổi bật và các sự kiện liên quan, giúp người dùng khám phá âm nhạc.

3.3.7 Màn hình Album

Trang album hiển thị ảnh bìa, tên album, tên nghệ sĩ, ngày phát hành và danh sách bài hát trong album, cho phép người dùng nghe từng bài hoặc phát toàn bộ, mang đến trải nghiệm nghe nhạc trọn vẹn và trực quan như trên Spotify.

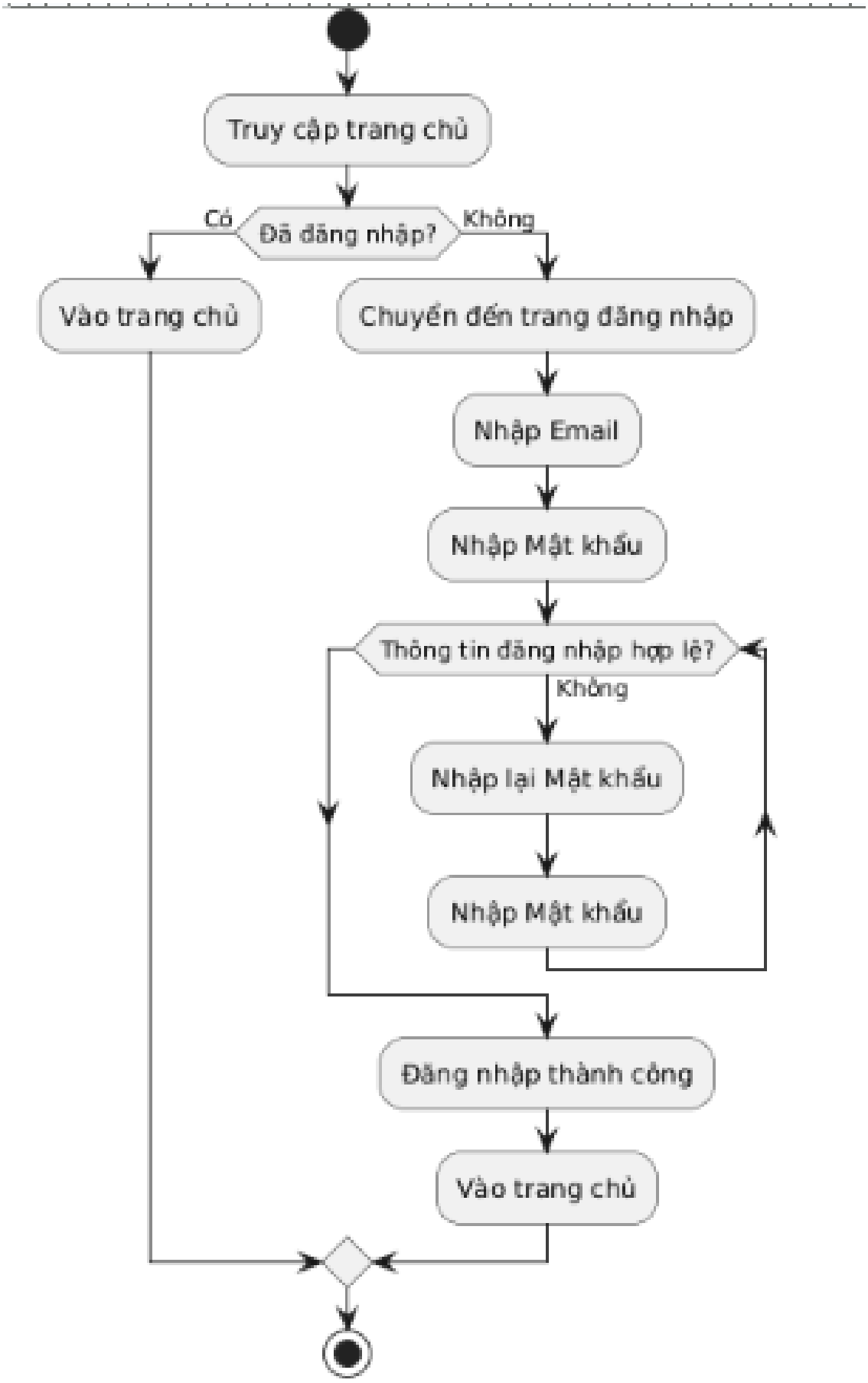

Figure 3.4: Flowchart màn hình trang chủ

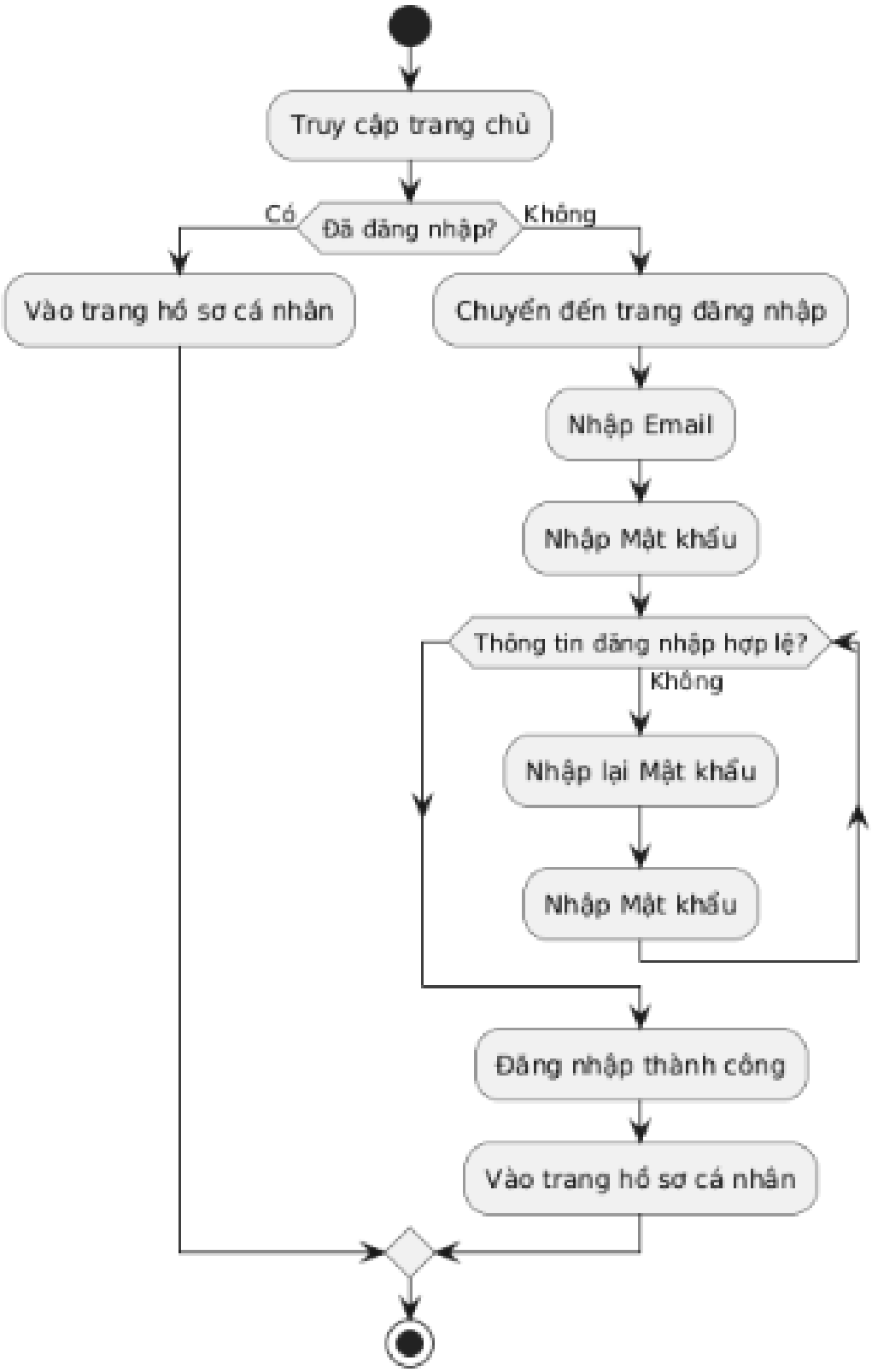

Figure 3.5: Flowchart hồ sơ

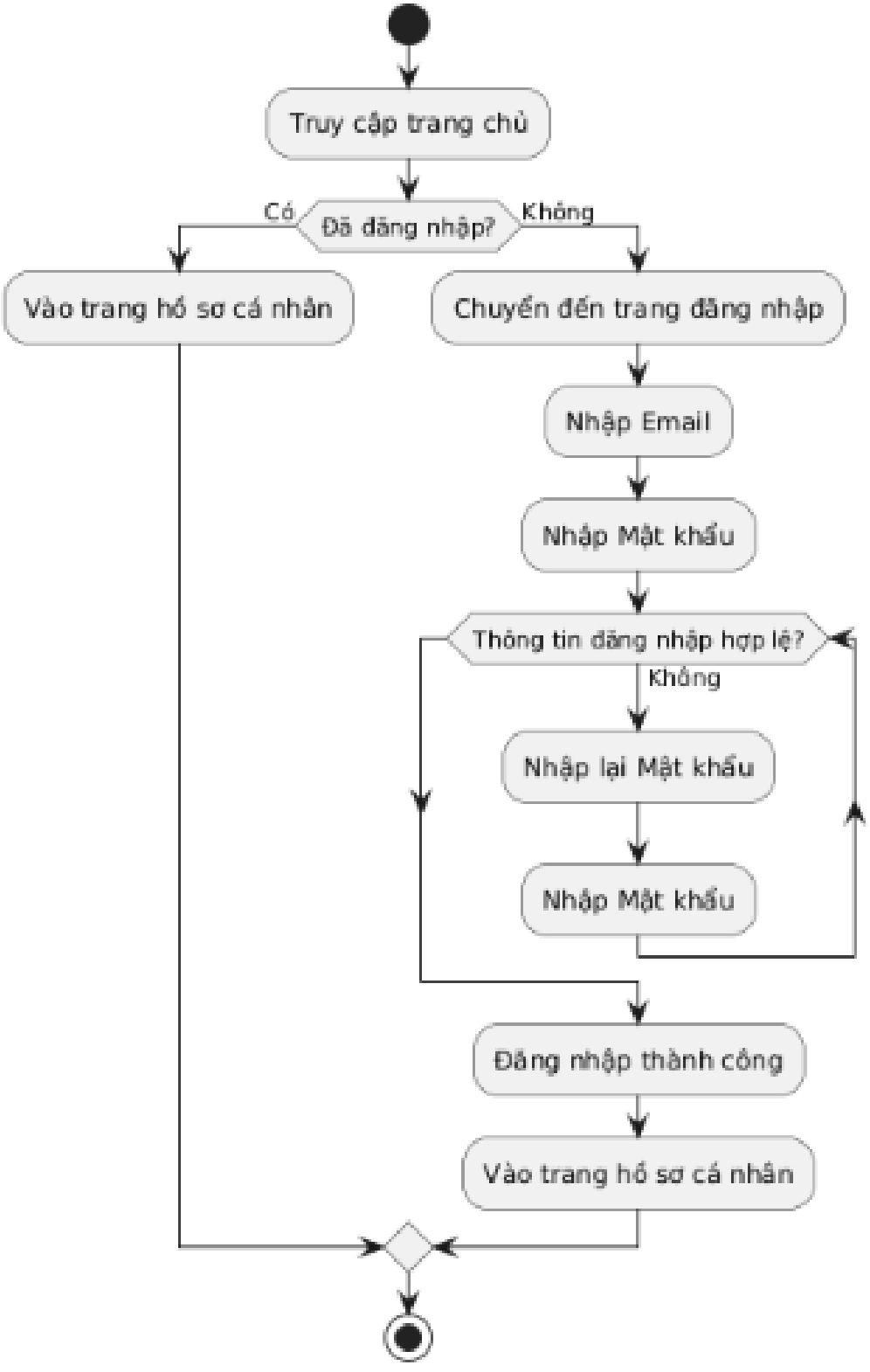

Figure 3.6: Flowchart màn hình nghệ sĩ

Figure 3.7: Flowchart màn hình album

3.3.8 Chức năng tìm kiếm

Trang tìm kiếm nhạc cho phép người dùng nhập từ khóa để tìm bài hát, nghệ sĩ, album, đồng thời hiển thị kết quả theo từng danh mục rõ ràng, giúp việc khám phá âm nhạc trở nên nhanh chóng và tiện lợi như trên Spotify.

3.3.9 Màn hình bài hát yêu thích

Chức năng tạo playlist cho phép người dùng tạo danh sách phát riêng, chọn tên, sau đó thêm nhạc từ các bài hát yêu thích hoặc tìm kiếm, giúp cá nhân hóa trải nghiệm nghe nhạc giống như trên Spotify.

3.3.10 Admin

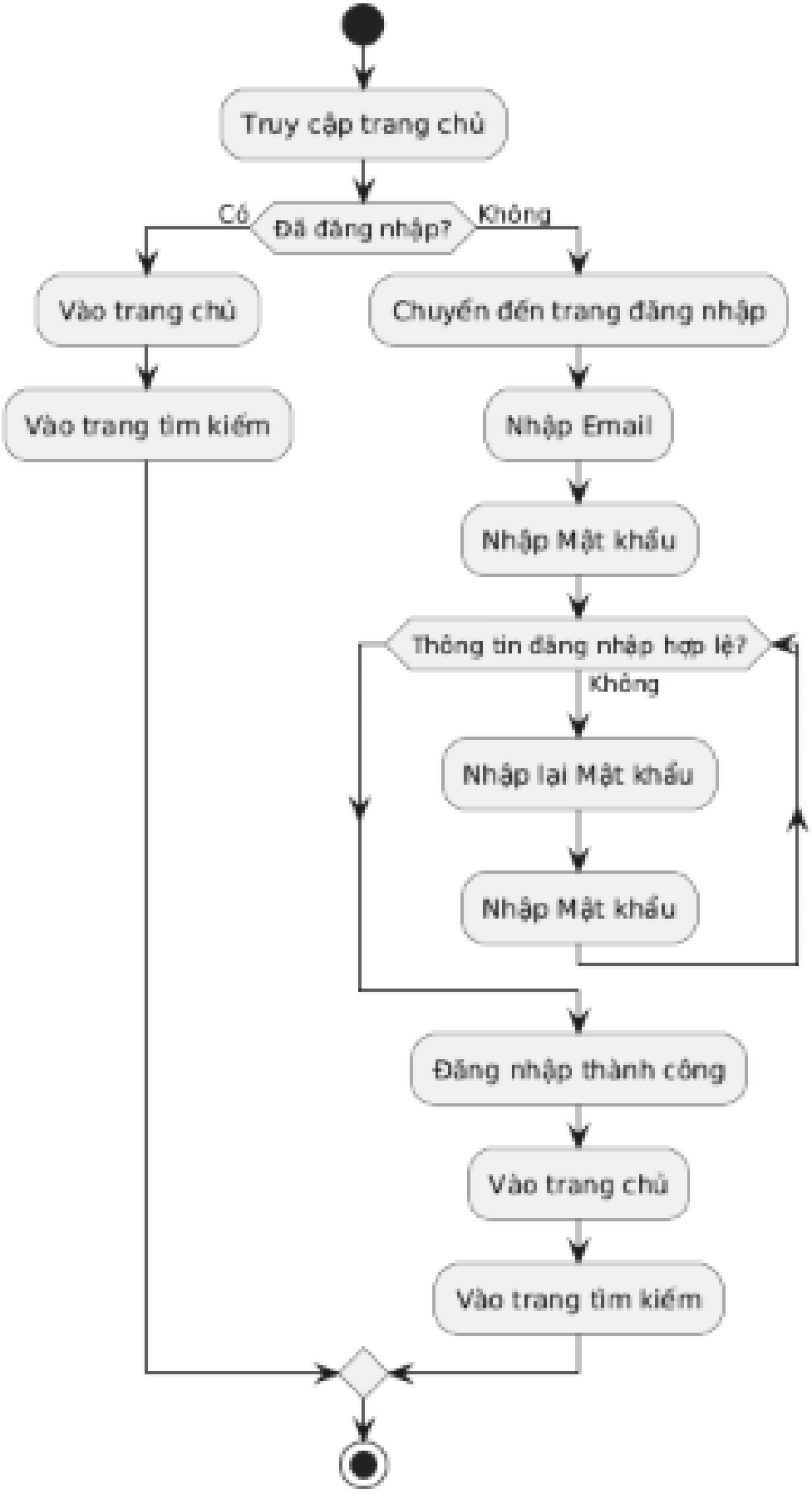

Figure 3.8: Flowchart tìm kiếm

Figure 3.9: Flowchart màn hình bài hát yêu thích

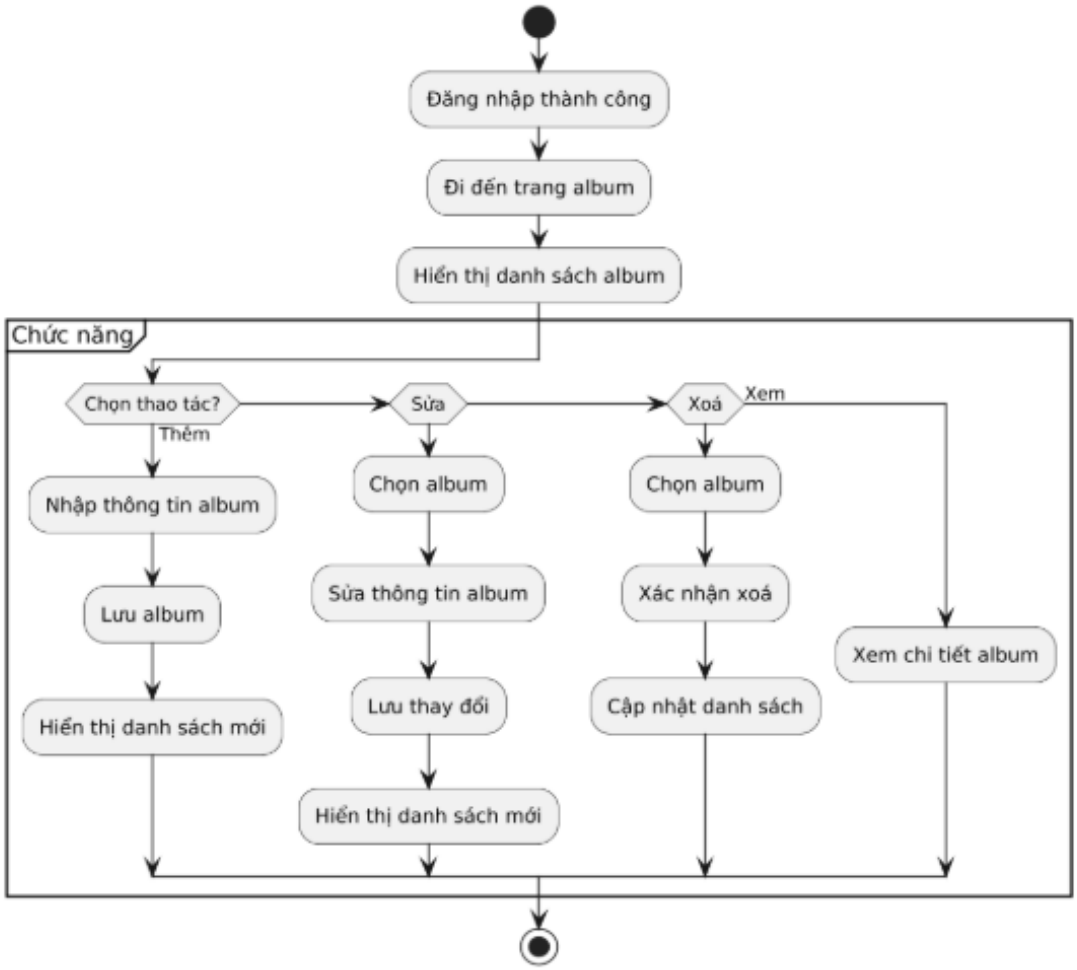

Figure 3.10: Flowchart chức năng quản lý Album của Admin

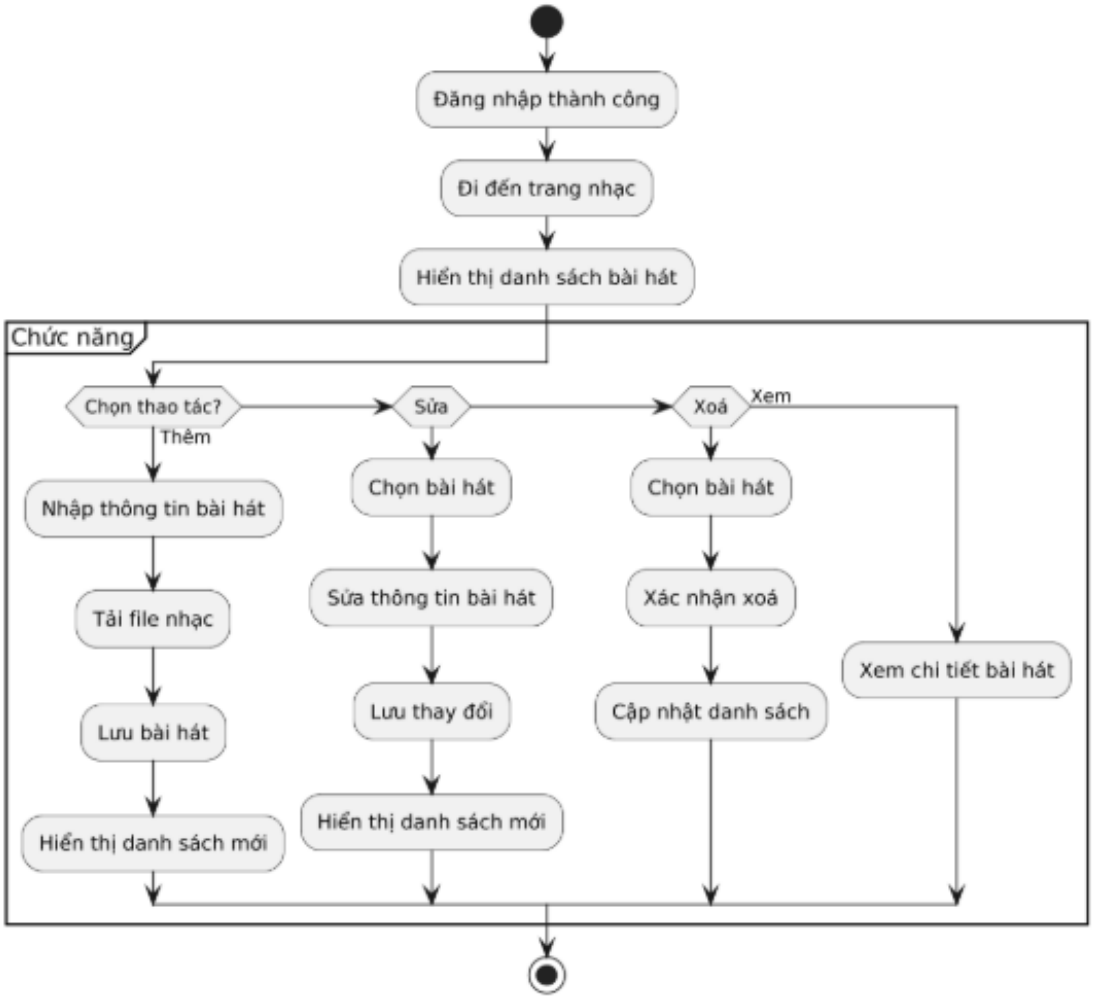

Figure 3.11: Flowchart chức năng quản lý Music của Admin

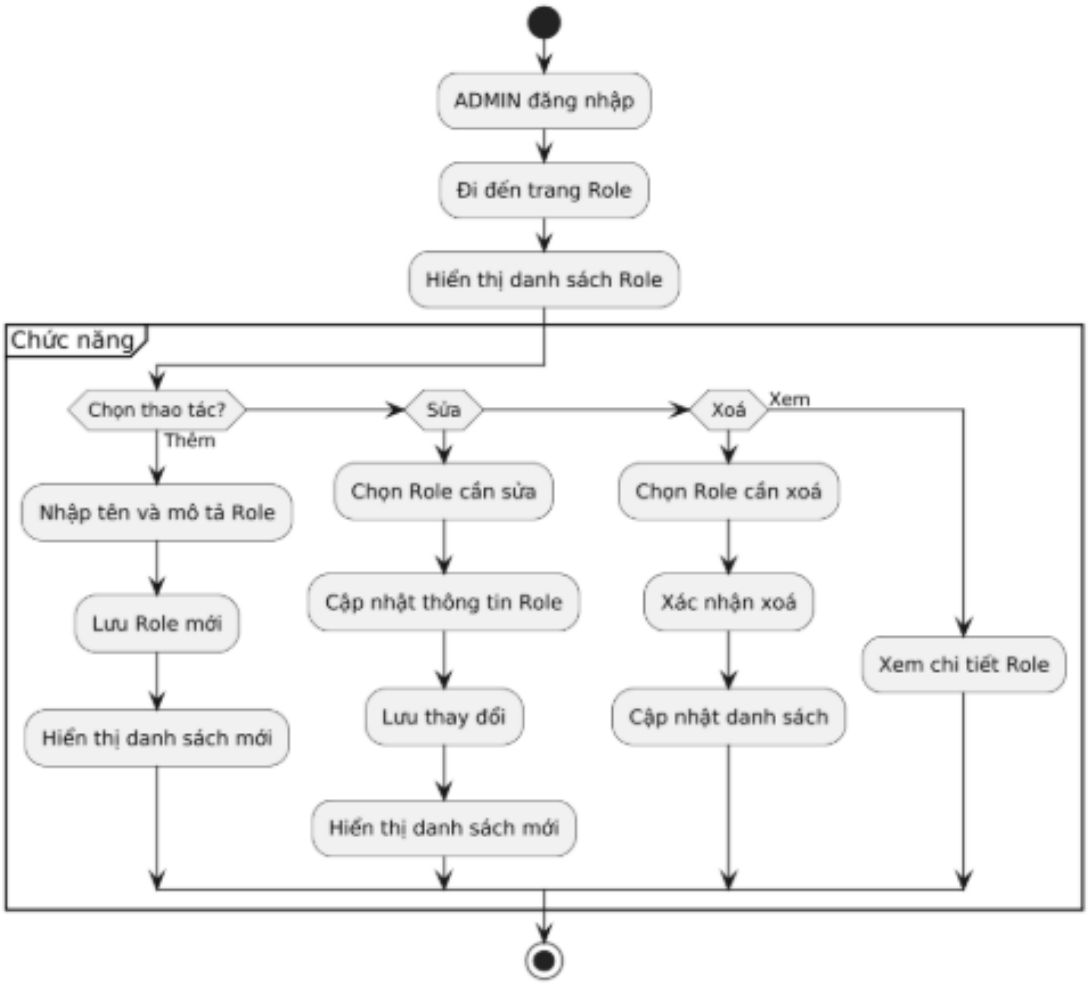

Figure 3.12: Flowchart chức năng quản lý Role của Admin

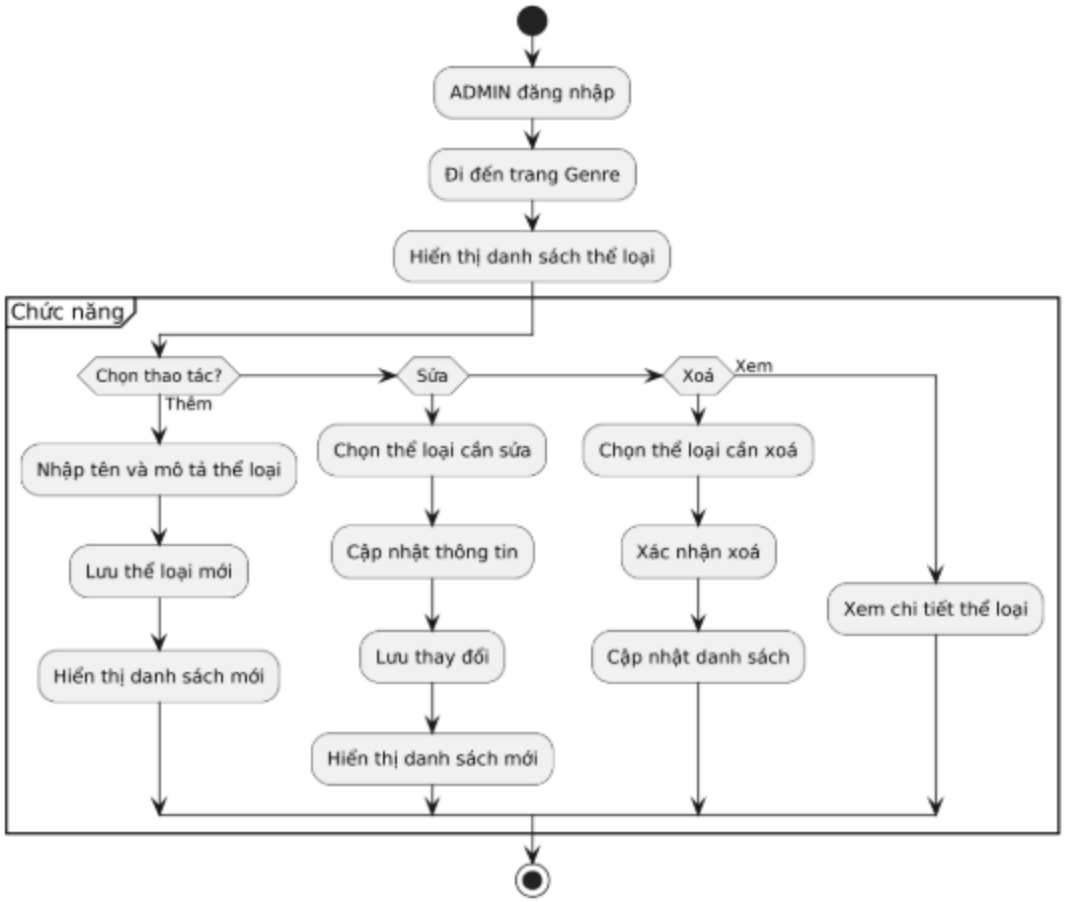

Figure 3.13: Flowchart chức năng quản lý Genre của Admin

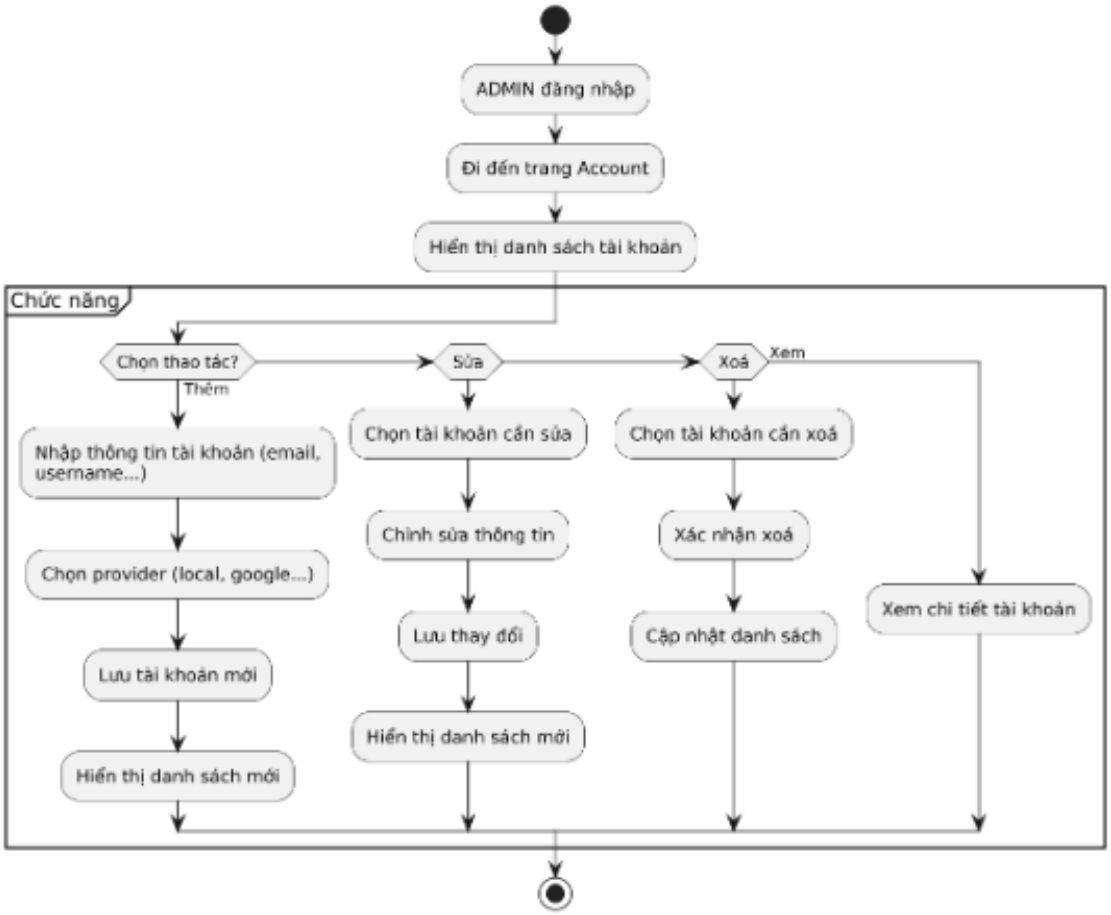

Figure 3.14: Flowchart chức năng quản lý Account của Admin

Chương 4

Hiện thực

4.1 Giao diện người dùng

4.1.1 Trang chủ

[Chỗ để dán hình ảnh trang chủ]

4.1.2 Giao diện phát nhạc

[Chỗ để dán hình ảnh giao diện phát nhạc]

4.1.3 Giao diện phát video âm nhạc

[Chỗ để dán hình ảnh giao diện phát video âm nhạc]

4.1.4 Giao diện tạo album

[Chỗ để dán hình ảnh giao diện tạo album]

4.1.5 Giao diện quản lý bài hát yêu thích

[Chỗ để dán hình ảnh giao diện quản lý bài hát yêu thích]

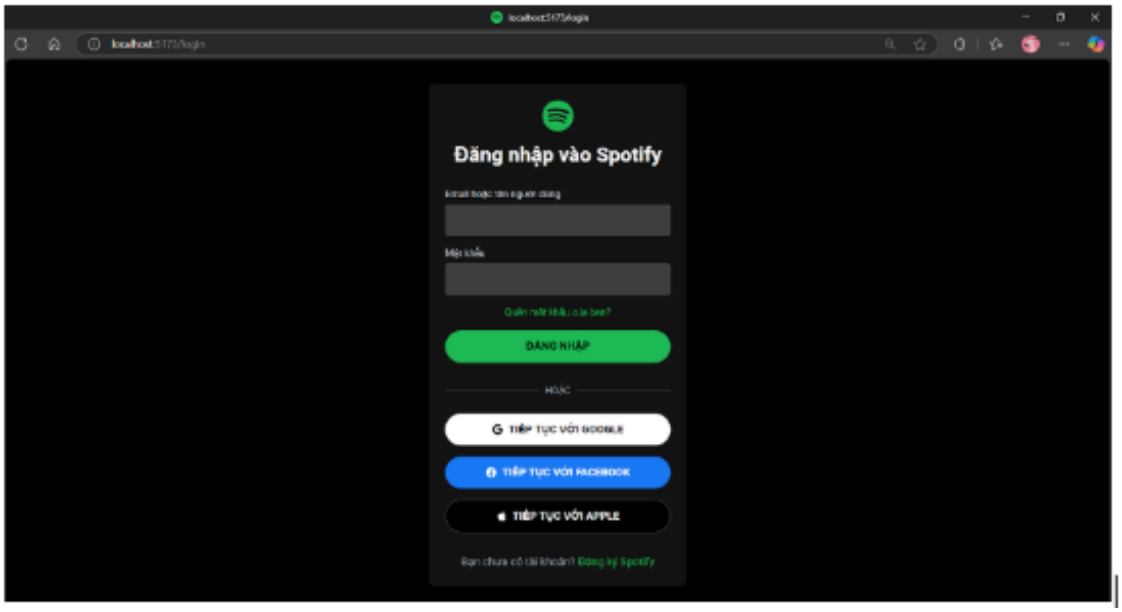

Figure 4.1: Giao diện đăng nhập

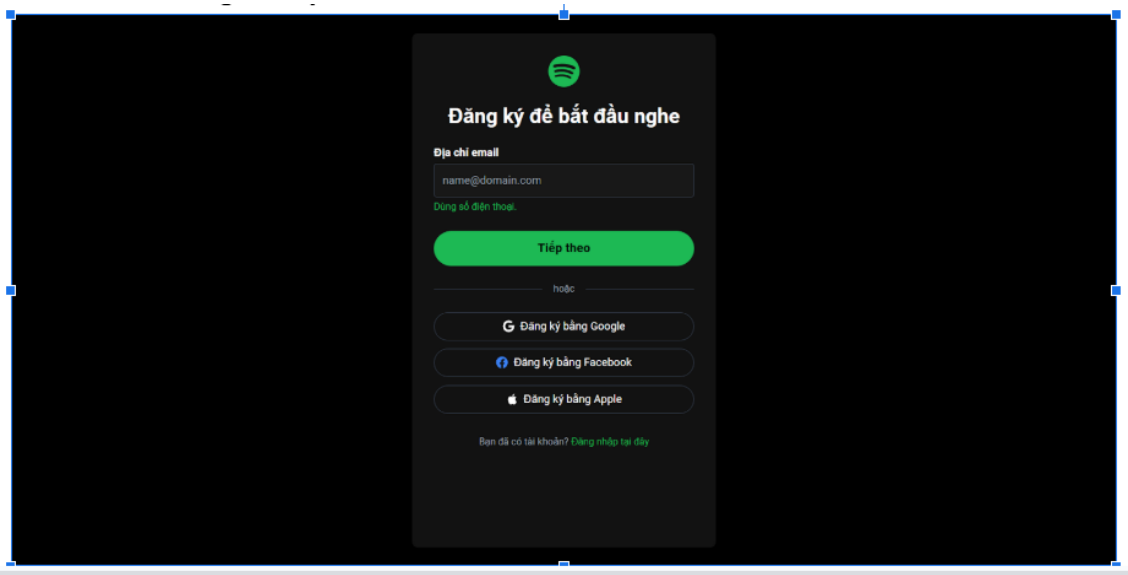

Figure 4.2: Giao diện đăng ký

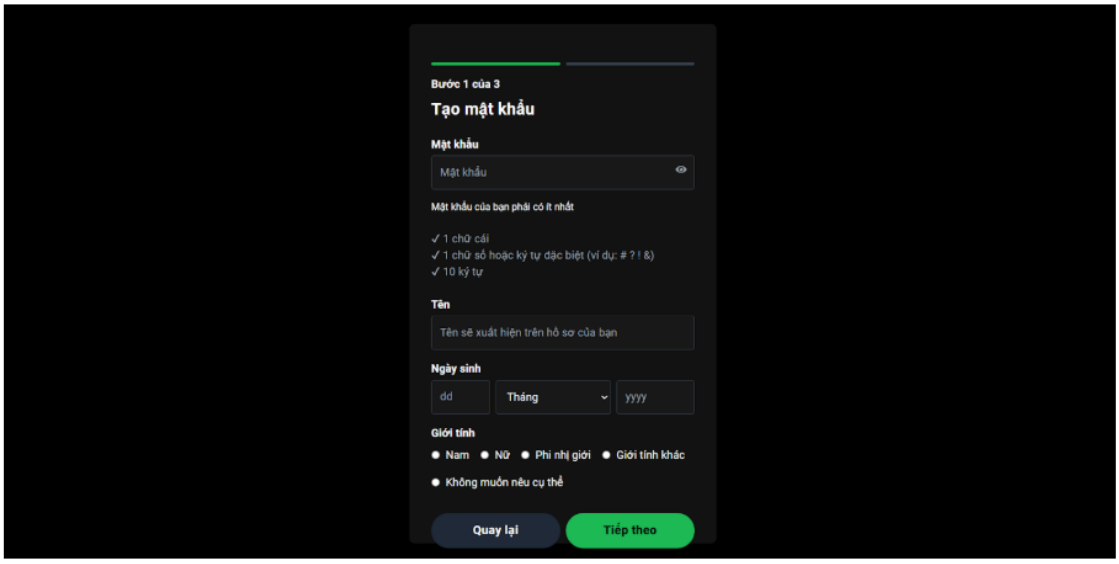

Figure 4.3: Giao diện đăng ký

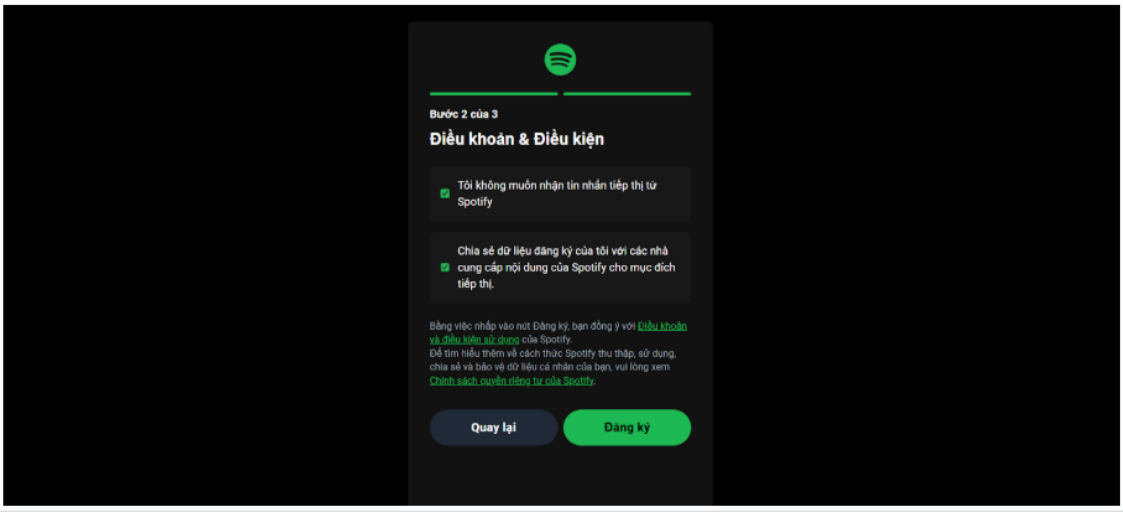

Figure 4.4: Giao diện đăng ký

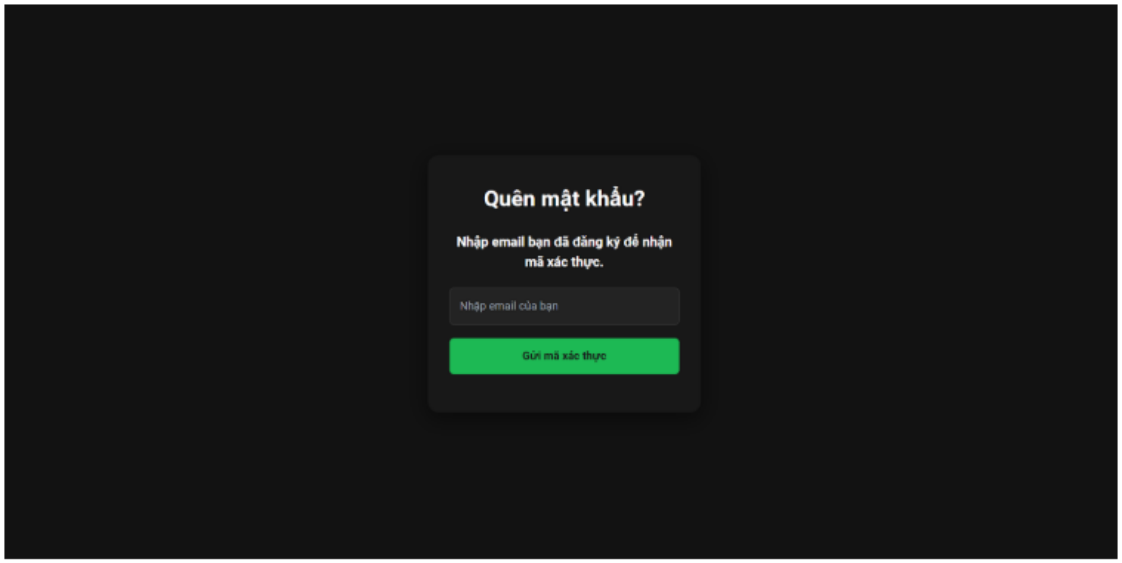

Figure 4.5: Giao diện quên mật khẩu

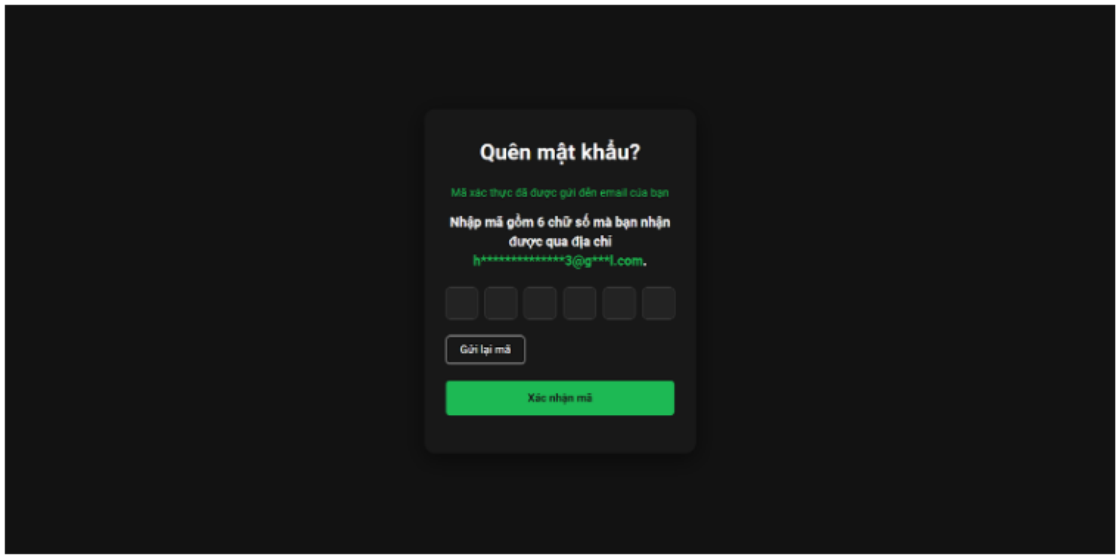

Figure 4.6: Giao diện quên mật khẩu

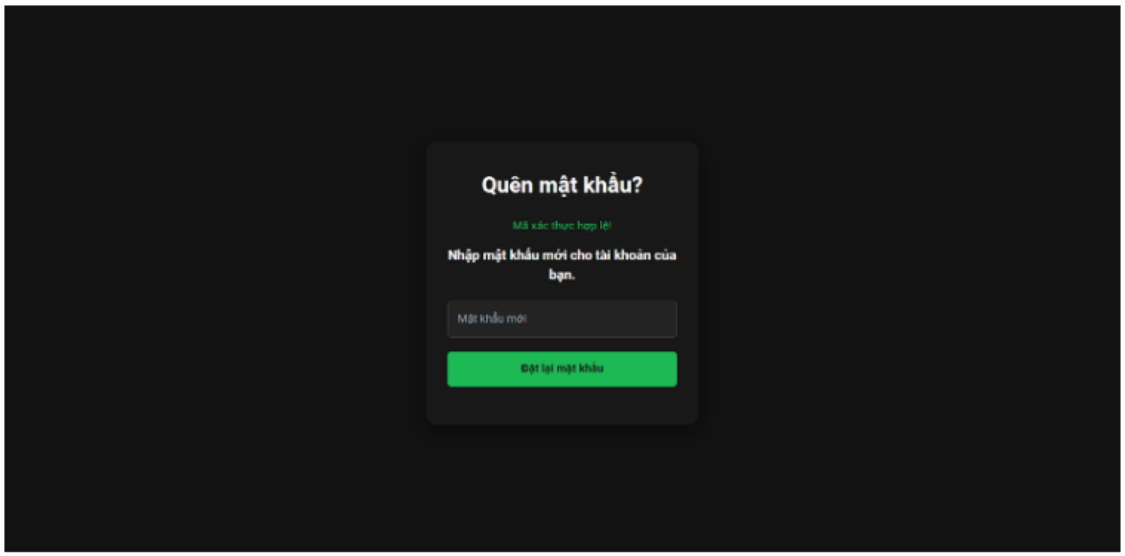

Figure 4.7: Giao diện quên mật khẩu

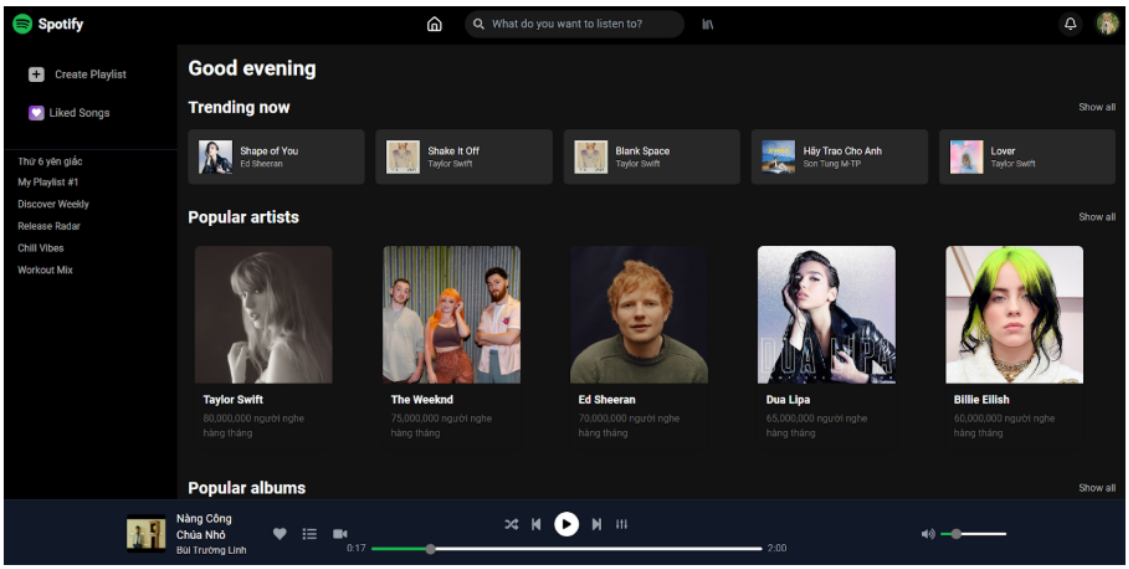

Figure 4.8: Giao diện trang chủ

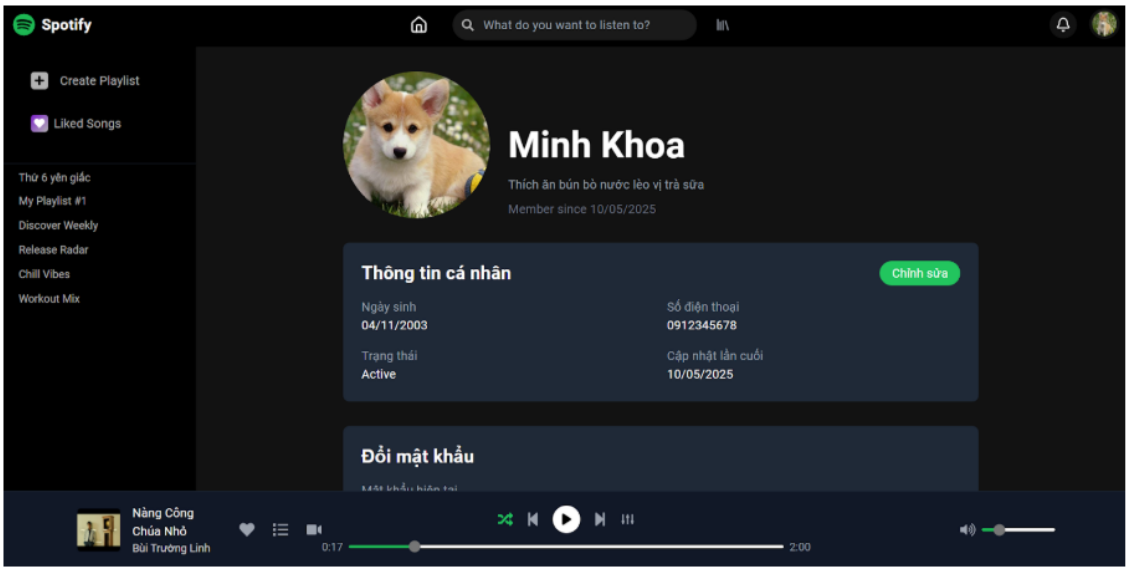

Figure 4.9: Giao diện hồ sơ

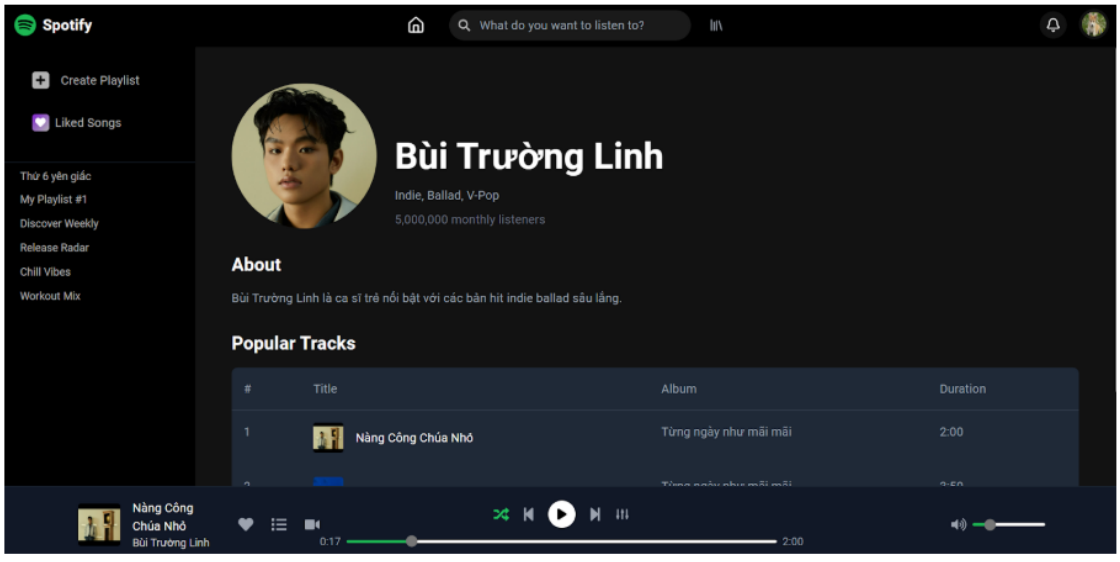

Figure 4.10: Giao diện nghệ sĩ

Figure 4.11: Giao diện xem Album

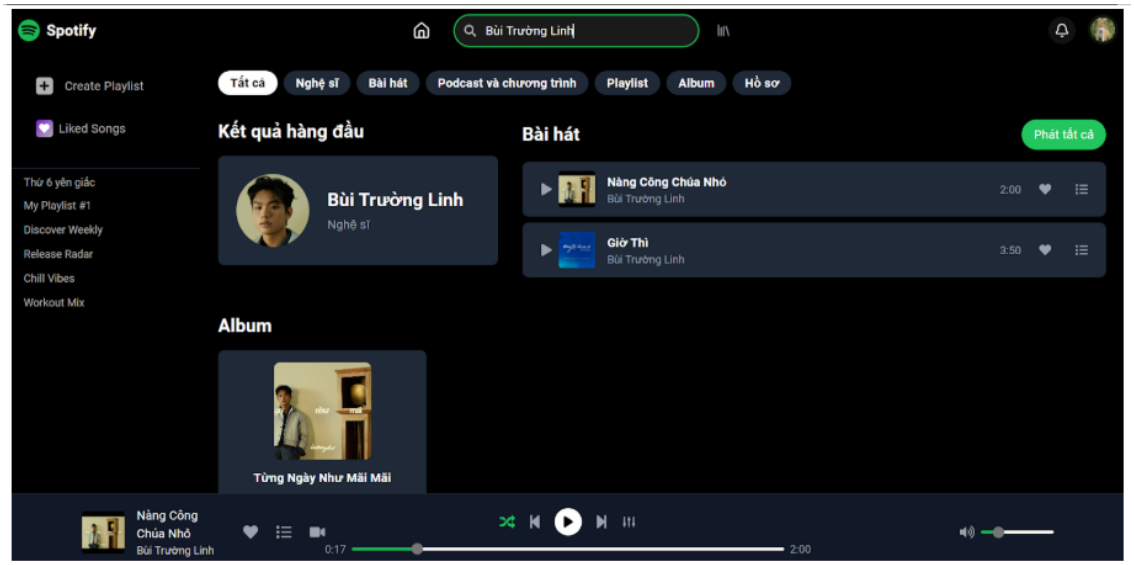

Figure 4.12: Giao diện tìm kiếm

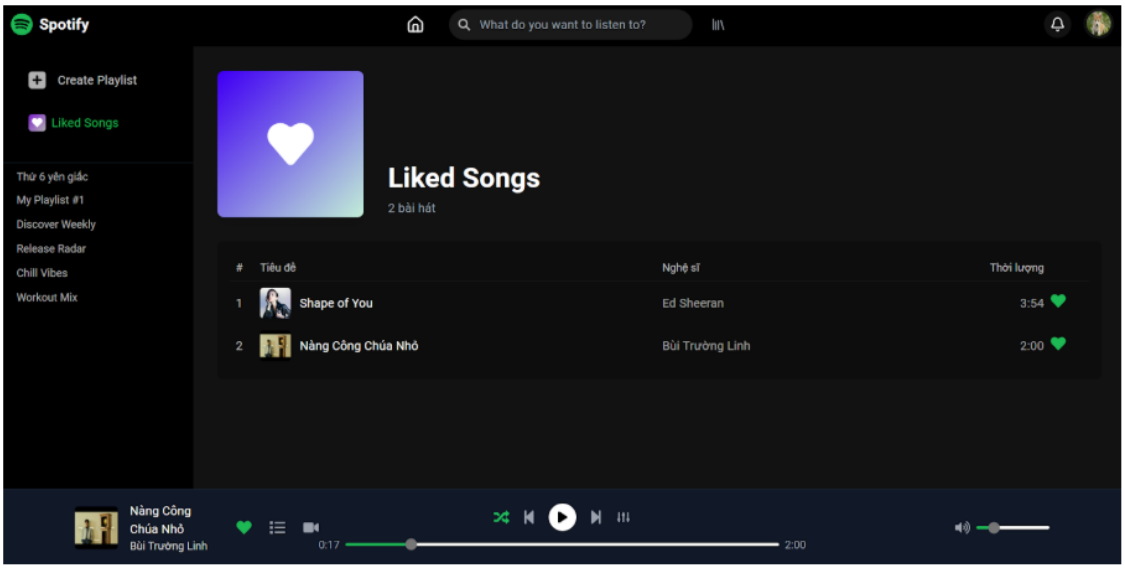

Figure 4.13: Giao diện yêu thích

Chương 5

Cách thức cài đặt và Môi trường chạy ứng dụng

5.1 Cài đặt môi trường phát triển (với Dev Containers)

Để có một môi trường phát triển nhất quán và dễ dàng thiết lập, dự án này khuyến khích sử dụng Dev Containers. Với Dev Containers, môi trường phát triển đã được cấu hình sẵn trong các container Docker, giúp bạn không cần lo lắng về việc cài đặt các dependency phức tạp trên máy cục bộ.

Yêu cầu:

- **Visual Studio Code (VS Code):** Hãy đảm bảo bạn đã cài đặt VS Code trên máy tính. Bạn có thể tải xuống tại https://code.visualstudio.com/.
- **Extension Dev Containers:** Trong VS Code, hãy cài đặt extension có tên **Dev Containers** từ Microsoft. Bạn có thể tìm kiếm và cài đặt nó trong mục Extensions của VS Code.
- **Docker:** Docker phải được cài đặt trên hệ thống của bạn. Bạn có thể tải xuống và cài đặt Docker Desktop cho Windows và macOS từ https://www.docker.com/products/docker-desktop/ hoặc Docker Engine cho Linux thông qua trình quản lý gói của hệ thống.

Hướng dẫn sử dụng Dev Containers:

1. **Clone mã nguồn:** Mở terminal hoặc command prompt và clone repository mã nguồn của dự án, đảm bảo clone cả các submodule:

   ```
   git clone --recursive https://github.com/bhbghghbgb/ptpmmnm-clontify
   cd ptpmmnm-clontify
   ```

2. **Mở trong VS Code:** Mở thư mục `ptpmmnm-clontify` trong VS Code.
3. **Mở bằng Dev Container:** VS Code sẽ tự động phát hiện file cấu hình Dev Container (trong thư mục `.devcontainer` của cả `clontify_server` và `fe-spotify`). Bạn sẽ nhận được một thông báo gợi ý mở thư mục trong Dev Container. Nhấn vào nút **"Reopen in Container"** hoặc sử dụng lệnh **"Dev Containers: Reopen in Container"** từ Command Palette (Ctrl+Shift+P hoặc Cmd+Shift+P).

4. **Phát triển tách biệt (tùy chọn):** Bạn cũng có thể mở trực tiếp thư mục `clontify_server` hoặc `fe-spotify` trong VS Code và thực hiện tương tự để phát triển backend và frontend một cách độc lập trong các container riêng biệt.

Sau khi VS Code mở trong Dev Container, bạn sẽ có một môi trường phát triển đã được cấu hình sẵn với các công cụ và dependencies cần thiết cho backend (Python) hoặc frontend (Node.js, tùy thuộc vào submodule bạn mở).

5.2 Cài đặt và chạy ứng dụng

5.2.1 Cấu hình các biến môi trường

Các thông tin cấu hình quan trọng của ứng dụng được quản lý thông qua các biến môi trường. Bạn sẽ tìm thấy một file mẫu `env.example` trong thư mục gốc của repository.

Các bước cấu hình:

1. **Sao chép file mẫu:** Sao chép file `env.example` và đổi tên thành `.env`.
2. **Chỉnh sửa file `.env`:** Mở file `.env` và điền các giá trị cần thiết. Dưới đây là giải thích cho một số cấu hình quan trọng mà bạn cần thiết lập:
   - `JWT_SECRET_KEY=your_jwt_secret_key`: **Bắt buộc.** Đây là khóa bí mật được sử dụng để ký và xác minh JSON Web Tokens (JWT) cho việc xác thực người dùng. Hãy thay `your_jwt_secret_key` bằng một chuỗi bí mật, phức tạp và duy nhất.
   - `EMAIL_HOST_USER=your_email@example.com`: **Bắt buộc nếu sử dụng tính năng gửi email.** Địa chỉ email của bạn sẽ được sử dụng làm tài khoản gửi email (ví dụ: cho việc khôi phục mật khẩu). Thay `your_email@example.com` bằng địa chỉ email thực của bạn.
   - `EMAIL_HOST_PASSWORD=your_email_password`: **Bắt buộc nếu sử dụng tính năng gửi email.** Mật khẩu của tài khoản email bạn đã cung cấp ở trên. Thay `your_email_password` bằng mật khẩu email thực của bạn.
   - `AWS_ACCESS_KEY_ID=your_aws_access_key_id`: **Bắt buộc nếu triển khai trên AWS hoặc sử dụng S3.** Khóa truy cập AWS của bạn.
   - `AWS_SECRET_ACCESS_KEY=your_aws_secret_access_key`: **Bắt buộc nếu triển khai trên AWS hoặc sử dụng S3.** Khóa bí mật AWS của bạn.
   - `AWS_STORAGE_BUCKET_NAME=your_bucket_name`: **Bắt buộc nếu sử dụng S3.** Tên của S3 bucket bạn đã tạo để lưu trữ dữ liệu. Thay `your_bucket_name` bằng tên bucket thực tế của bạn.
   - `AWS_S3_CUSTOM_DOMAIN=your_custom_domain`: **Tùy chọn.** Nếu bạn đã cấu hình một custom domain cho S3 bucket của mình, hãy điền domain đó vào đây. Nếu không, bạn có thể để trống.

   Các biến môi trường khác thường đã có giá trị mặc định hợp lý, nhưng bạn vẫn nên xem qua để tùy chỉnh nếu cần.

5.2.2 Chạy ứng dụng cục bộ (Local) bằng Docker Compose

Yêu cầu: Đảm bảo Docker Desktop (trên Windows/macOS) hoặc Docker Engine và Docker Compose (trên Linux) đã được cài đặt và đang chạy.

Hướng dẫn:

1. **Clone mã nguồn (nếu chưa):** Thực hiện lệnh clone recursive như đã hướng dẫn ở phần cài đặt Dev Containers.
2. **Chuyển đến thư mục gốc:** Mở terminal hoặc command prompt và chuyển đến thư mục gốc của repository `ptpmmnm-clontify`.
3. **Chạy Docker Compose:** Chạy lệnh sau để khởi động toàn bộ ứng dụng (bao gồm cả backend và frontend):

   ```
   docker-compose up -d
   ```

   Hoặc, để chạy riêng backend hoặc frontend, bạn có thể chuyển đến thư mục `clontify_server` hoặc `fe-spotify` và chạy lệnh `docker-compose up -d` tại đó.
4. **Truy cập ứng dụng:**
   - **Frontend:** Được chạy tại `http://localhost:5173` (có thể cấu hình lại trong file `fe-spotify/docker-compose.yml`), port mở của server trong container là 5173.
   - **Backend:** Được chạy tại `http://localhost:8080-8083` (có thể cấu hình lại trong file `clontify_server/docker-compose.yml`), port mở của server trong container là từ 8080 đến 8083 tùy service.

5.2.3 Triển khai trên AWS

Việc triển khai ứng dụng lên AWS đòi hỏi bạn phải có tài khoản AWS và kiến thức cơ bản về các dịch vụ của AWS. Dưới đây là một hướng dẫn tổng quan cho người mới bắt đầu:

Các bước cơ bản:

1. **Tạo tài khoản AWS:** Nếu bạn chưa có, hãy tạo một tài khoản AWS tại https://aws.amazon.com/.
2. **Cấu hình AWS CLI:** Cài đặt và cấu hình AWS Command Line Interface (CLI) trên máy tính của bạn để tương tác với các dịch vụ AWS thông qua dòng lệnh. Tham khảo hướng dẫn tại https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html.
3. **Tạo S3 Bucket:** Sử dụng AWS Management Console hoặc AWS CLI để tạo một S3 bucket trong region bạn muốn (ví dụ: `ap-southeast-2` như cấu hình). Hãy nhớ tên bucket này và điền vào biến môi trường `AWS_STORAGE_BUCKET_NAME`.
4. **Tạo ECR Repository (cho Docker Images):** Sử dụng AWS Management Console hoặc AWS CLI để tạo các repository trong Amazon Elastic Container Registry (ECR) cho Docker images của backend và frontend.
5. **Build và Push Docker Images:**
   - Chuyển đến thư mục `clontify_server` và build image backend: `docker build -t <tên_ecr_repository_backend>:latest .` Sau đó, login vào ECR và push image lên.

- Tương tự, chuyển đến thư mục `fe-spotify` và build image frontend: `docker build -t <tên_ecr_repository_frontend>:latest .` Sau đó, login vào ECR và push image lên.

6. **Chọn dịch vụ triển khai:** AWS cung cấp nhiều dịch vụ để chạy container, phổ biến nhất là Amazon ECS (Elastic Container Service) hoặc Amazon EKS (Elastic Kubernetes Service). Đối với người mới bắt đầu, ECS có thể đơn giản hơn.
7. **Cấu hình và triển khai ECS (ví dụ):**
   - Tạo một Cluster ECS.
   - Định nghĩa Task Definition cho backend và frontend, chỉ định image từ ECR, tài nguyên (CPU, memory), và các biến môi trường (bao gồm cả AWS keys và bucket name).
   - Tạo Service ECS để chạy các task trên cluster.
   - Cấu hình Network (ví dụ: Security Groups, Load Balancer) để ứng dụng có thể truy cập được từ bên ngoài.
8. **Quản lý S3 Permissions:** Đảm bảo rằng các ECS tasks (hoặc các IAM roles liên quan) có quyền đọc và ghi vào S3 bucket của bạn.

Chương 6

Nhiệm vụ và vai trò của từng thành viên trong nhóm

Tên thành viên	Công việc	Tiến độ hoàn thành
Nguyễn Huỳnh Tuấn Hào	- Cấu hình backend - Service backend: auth, music, chatbot, storage - Cấu hình docker - Deploy AWS	100%
Huỳnh Thanh Lộc	- Frontend - Frontend + đăng ký đăng nhập - Frontend + backend quên mật khẩu - Frontend CRUD song - Frontend phát nhạc - Frontend chỉnh sửa profile user	100%
Huỳnh Gia Bảo	- Frontend admin - Backend role + favorites - Frontend user album, favorites - Viết báo cáo	100%

Table 6.1: Phân công công việc của các thành viên trong nhóm

Tài liệu tham khảo

Amazon Web Services, Inc. (n.d.*a*), 'Amazon s3 - cloud object storage - aws'. (accessed May 12, 2025).
URL: *https://aws.amazon.com/s3/*

Amazon Web Services, Inc. (n.d.*b*), 'Amazon web services (aws) - cloud computing services'. (accessed May 12, 2025).
URL: *https://aws.amazon.com/*

Banks, A. and Porcello, E. (2020), *Learning React: Modern Patterns for Developing React Apps*, 2nd edn, O'Reilly Media. ISBN: 9781492051725.

Beazley, D. and Jones, B. K. (2013), *Python Cookbook*, 3rd edn, O'Reilly Media. ISBN: 9781449340377.

Culkin, J. and Zazon, M. (2021), *AWS Cookbook: Recipes for Success on AWS*, O'Reilly Media. ISBN: 9781492092605.

Django Software Foundation (n.d.), 'The web framework for perfectionists with deadlines'. (accessed May 12, 2025).
URL: *https://www.djangoproject.com/*

Docker, Inc. (n.d.), 'Docker: Accelerated, containerized application development'. (accessed May 12, 2025).
URL: *https://www.docker.com/*

Fowler, M. (2002), *Patterns of Enterprise Application Architecture*, Addison-Wesley. ISBN: 9780321127426.

Fowler, M. (n.d.), 'Microservices'. (accessed May 12, 2025).
URL: *https://martinfowler.com/articles/microservices.html*

Gilmore, W. J. (2024), *Protobuf: The Definitive Guide*, O'Reilly Media. ISBN: 9781098130125.

Indrasiri, K. and Kuruppu, D. (2020), *gRPC: Up and Running*, O'Reilly Media. ISBN: 9781492058335.

Meta (n.d.), 'A javascript library for building user interfaces'. (accessed May 12, 2025).
URL: *https://react.dev/*

Microsoft (n.d.), 'Developing inside a container'. (accessed May 12, 2025).
URL: *https://code.visualstudio.com/docs/devcontainers/containers*

Newman, S. (2021), *Building Microservices: Designing Fine-Grained Systems*, 2nd edn, O'Reilly Media. ISBN: 9781492034025.

Nishimura, H. (2022), *AWS for Non-Engineers*, O'Reilly Media. ISBN: 9781098113647.

Nunez, C. (2023), *Dev Containers: Simplify Your Development Workflow with Containers in VS Code*, Packt Publishing. ISBN: 9781805125669.

Obe, R. O. and Hsu, L. S. (2017), *PostgreSQL: Up and Running*, 3rd edn, O'Reilly Media. ISBN: 9781491963418.

PostgreSQL Global Development Group (n.d.), 'The world's most advanced open source relational database'. (accessed May 12, 2025).
URL: *https://www.postgresql.org/*

Poulton, N. (2023), *Docker Deep Dive*, 5th edn, Leanpub. ISBN: 9781838436875.

Protocol Buffers (n.d.*a*), 'Overview'. (accessed May 12, 2025).
URL: *https://protobuf.dev/getting-started/overview/*

Protocol Buffers (n.d.*b*), 'Protocol buffers'. (accessed May 12, 2025).
URL: *https://protobuf.dev/*

Python Software Foundation (n.d.), 'Welcome to python.org'. (accessed May 12, 2025).
URL: *https://www.python.org/*

Rice, L. (2020), *Container Security: Fundamental Technology Concepts that Protect Containerized Applications*, O'Reilly Media. ISBN: 9781492056706.

Richardson, C. (2018), *Microservices Patterns: With examples in Java*, Manning Publications. ISBN: 9781617294549.

Sommerville, I. (2015), *Software Engineering*, 10th edn, Pearson. ISBN: 9780133943030.

The gRPC Authors (n.d.*a*), 'Core concepts'. (accessed May 12, 2025).
URL: *https://grpc.io/docs/what-is-grpc/core-concepts/#use-cases*

The gRPC Authors (n.d.*b*), 'grpc'. (accessed May 12, 2025).
URL: *https://grpc.io/*

Vincent, W. S. (2022), *Django for APIs: Build web APIs with Python & Django*, WelcomeToCode. ISBN: 9781735467221.

Wikipedia (n.d.*a*), 'Model–view–controller'. (accessed May 12, 2025).
URL: *https://en.wikipedia.org/wiki/Model–view–controller*

Wikipedia (n.d.*b*), 'Operating system–level virtualization'. (accessed May 12, 2025) - Liên quan đến container hóa.
URL: *https://en.wikipedia.org/wiki/OS-level_virtualization*

Wikipedia (n.d.*c*), 'Waterfall model'. (accessed May 12, 2025).
URL: *https://en.wikipedia.org/wiki/Waterfall_model*