

KHOA KỸ THUẬT VÀ CÔNG NGHỆ
BỘ MÔN CÔNG NGHỆ THÔNG TIN



THỰC TẬP ĐỒ ÁN CHUYÊN NGÀNH
HỌC KỲ I, NĂM HỌC 2024 - 2025
**XÂY DỰNG WEBSITE
NGHE NHẠC TRỰC TUYẾN
THEO THỜI GIAN THỰC
BẰNG NEXTJS**

Giáo viên hướng dẫn:
TS. Nguyễn Bảo Ân

Sinh viên thực hiện:
Họ tên: Đinh Tấn Mãi
MSSV: 110121063
Lớp: DA21TTB

Trà Vinh, tháng..... năm.....

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

Trà Vinh, ngày tháng năm

Giáo viên hướng dẫn

(Ký tên và ghi rõ họ tên)

NHẬN XÉT CỦA THÀNH VIÊN HỘI ĐỒNG

Trà Vinh, ngày tháng năm
Thành viên hội đồng
(Ký tên và ghi rõ họ tên)

LỜI CẢM ƠN

Trước hết, em xin gửi lời cảm ơn chân thành đến quý thầy cô, các bạn bè đã luôn quan tâm, giúp đỡ em trong suốt quá trình học tập và thực hiện đề tài này.

Đề tài này là một thử thách lớn đối với em, đòi hỏi em phải có kiến thức chuyên môn vững vàng và kỹ năng lập trình thành thạo. Tuy nhiên, nhờ sự hướng dẫn tận tình của thầy, sự động viên của gia đình và bạn bè, em đã có thể hoàn thành đề tài đúng tiến độ và đạt được kết quả tốt.

Em xin chân thành cảm ơn thầy đã luôn dành thời gian quý báu để hướng dẫn, giải đáp thắc mắc của em trong suốt quá trình thực hiện đề tài. Thầy đã giúp em hiểu rõ hơn về đề tài, từ đó có thể xây dựng được đề tài đáp ứng được nhu cầu của người dùng.

Em cũng xin gửi lời cảm ơn đến các bạn bè trong lớp đã luôn giúp đỡ và động viên em trong suốt quá trình học tập và thực hiện đề tài. Sự giúp đỡ của các bạn đã giúp em có thêm động lực để hoàn thành đề tài một cách tốt nhất.

Đề tài này tuy đã hoàn thành nhưng vẫn còn nhiều thiếu sót. Em rất mong nhận được ý kiến đóng góp của quý thầy cô, các bạn để đề tài được hoàn thiện hơn.

Một lần nữa, em xin gửi lời cảm ơn chân thành đến quý thầy cô, các bạn bè đã luôn quan tâm, giúp đỡ em trong suốt quá trình học tập và thực hiện đề tài.

Em xin chân thành cảm ơn!

Trân trọng,

MỤC LỤC

CHƯƠNG 1 TỔNG QUAN.....	10
CHƯƠNG 2 NGHIÊN CỨU LÝ THUYẾT	11
1.1 Tìm hiểu RESTful API.....	11
2.1.1 Tổng quan về RESTful API	11
2.1.2 Tài nguyên (Resources)	12
2.1.3 Phương thức HTTP (HTTP Methods)	12
2.1.4 Trạng thái (Status).....	12
2.1.5 Ưu điểm của RESTful API	13
2.1.6 Nhược điểm của RESTful API	14
2.2 Xác thực người dùng	14
2.2.1 Xác thực người dùng là gì ?	14
2.2.2 Phương pháp xác thực phổ biến.....	15
2.2.3 Sự cần thiết của xác thực người dùng	16
2.3 Tìm hiểu Next.js	17
2.3.1 Tổng quan về Next.js	17
2.3.2 Kiến trúc hoạt động.....	17
2.3.3 Ưu điểm.....	18
2.3.4 Nhược điểm.....	19
2.4 Tìm hiểu NodeJS	20
2.4.1 Giới thiệu tổng quan NodeJS	20
2.4.2 Kiến trúc hoạt động.....	20
2.4.3 Ưu điểm.....	21
2.4.4 Nhược điểm.....	21
CHƯƠNG 3 HIỆN THỰC HÓA NGHIÊN CỨU	22
3.1 Đặc tả yêu cầu hệ thống	22
3.2 Mô hình chức năng	23
3.3 Sơ đồ use-case	24
3.4 Thiết kế dữ liệu.....	25
3.4.1 Lược đồ cơ sở dữ liệu	25
3.4.2 Chi tiết các thực thể	27
3.5 Thiết kế, kiến trúc ứng dụng	34
3.5.1 Kiến trúc toàn cảnh	34
3.5.2 Thiết kế API	35

3.5.3	Thiết kế Giao diện.....	40
3.6	Triển khai	42
CHƯƠNG 4 KẾT QUẢ NGHIÊN CỨU		45
4.1	Kiểm thử API với Postman	45
4.1.1	API Xác thực người dùng	45
4.1.2	API liên quan đến bài hát.....	47
4.1.3	API liên quan đến danh sách phát.....	48
4.1.4	API liên quan đến người dùng	50
4.1.5	API liên quan đến vai trò và quyền hạn	51
4.2	Giao diện người dùng.....	53
4.2.1	Giao diện trang đăng nhập	53
4.2.2	Giao diện trang đăng ký	54
4.2.3	Giao diện trang chủ	54
4.2.4	Giao diện trang tìm kiếm	55
4.2.5	Giao diện trang nghệ sĩ	56
4.2.6	Giao diện trang bài hát.....	56
4.2.7	Giao diện trang danh sách phát	57
4.2.8	Giao diện trang phòng nghe nhạc	58
4.2.9	Giao diện trang tạo bài hát (Dành cho nghệ sĩ)	58
4.2.10	Giao diện trang tạo phòng nghe nhạc	59
4.2.11	Giao diện tạo danh sách phát	60
4.2.12	Giao diện phát nhạc	60
CHƯƠNG 5 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN		62
5.1	Kết luận:	62
5.2	Hướng phát triển:	62
DANH MỤC TÀI LIỆU THAM KHẢO		63

DANH MỤC HÌNH ẢNH

Hình 1. Sơ đồ hoạt động của RESTful API	11
Hình 2. Next.js	17
Hình 3. Cấu trúc tập tin File-based Routing	18
Hình 4. Các tính năng cơ bản của nodejs	20
Hình 5. Mô hình chức năng của hệ thống	23
Hình 6. Sơ đồ use-case	24
Hình 7. Lược đồ cơ sở dữ liệu	25
Hình 8. Sơ đồ kiến trúc toàn cảnh của hệ thống	34
Hình 9. Cấu trúc thư mục phía API	36
Hình 10. Tập tin index.ts	37
Hình 11. Tập tin routes/index.ts	38
Hình 12. Tập tin schema/genre.schema.ts	39
Hình 13. controllers/auth.controller.ts	39
Hình 14. Tập tin models/Account.ts	40
Hình 15. Hi-fi prototypes của trang chủ	41
Hình 16. Hi-fi prototypes của trang phòng nghe nhạc	42
Hình 17. Kiểm thử API đăng nhập	45
Hình 18. Kiểm thử API đăng ký	46
Hình 19. Kiểm thử API làm mới token	46
Hình 20. Kiểm thử API đăng xuất	46
Hình 21. Kiểm thử API tạo bài hát	47
Hình 22. Kiểm thử API lấy thông tin bài hát	47
Hình 23. Kiểm thử API cập nhật bài hát	48
Hình 24. Kiểm thử API yêu thích bài hát	48
Hình 25. Kiểm thử API bỏ yêu thích bài hát	48
Hình 26. Kiểm thử API lấy toàn bộ danh sách phát	49
Hình 27. Kiểm thử API tạo danh sách phát	49
Hình 28. Kiểm thử API cập nhật danh sách phát	49
Hình 29. Kiểm thử API xóa danh sách phát	50
Hình 30. Kiểm thử API lấy tất cả người dùng	50
Hình 31. Kiểm thử API lấy danh sách các nghệ sĩ	51
Hình 32. Kiểm thử API cập nhật người dùng	51
Hình 33. Kiểm thử API lấy tất cả các vai trò và quyền	52
Hình 34. Kiểm thử API tạo mới vai trò với các quyền	52
Hình 35. Kiểm thử API cập nhật vai trò	52
Hình 36. Kiểm thử API xóa vai trò	53
Hình 37. Giao diện trang đăng nhập Vibely	53
Hình 38. Giao diện trang đăng ký Vibely	54
Hình 39. Giao diện trang chủ Vibely	54
Hình 40. Giao diện trang tìm kiếm Vibely	55
Hình 41. Giao diện trang nghệ sĩ Vibely	56
Hình 42. Giao diện trang bài hát Vibely	56
Hình 43. Giao diện trang danh sách phát	57
Hình 44. Giao diện trang phòng nghe nhạc Vibely	58
Hình 45. Giao diện trang tạo bài hát Vibely	58
Hình 46. Giao diện trang tạo phòng nghe nhạc trực tuyến Vibely	59

Hình 47. Giao diện tạo danh sách phát Vibely.....	60
Hình 48. Giao diện phát nhạc Vibely.....	60

BẢNG BIỂU

Bảng 1. Bảng danh sách các thực thể.....	26
Bảng 2. Chi tiết thực thể “ user ”: người dùng	27
Bảng 3. Chi tiết thực thể “ account ”: tài khoản	28
Bảng 4. Chi tiết thực thể “ roles ”: vai trò	28
Bảng 5. Chi tiết thực thể “ permissions ”: quyền	29
Bảng 6. Chi tiết thực thể “ songs ”: bài hát	29
Bảng 7. Chi tiết thực thể “ playlist ”: danh sách phát	30
Bảng 8. Chi tiết thực thể “ genre ”: thể loại	31
Bảng 9. Chi tiết thực thể “ moods ”: tâm trạng	31
Bảng 10. Chi tiết thực thể “ room ”: phòng	32
Bảng 11. Chi tiết thực thể “ room-members ”: thành viên trong phòng	32
Bảng 12. Chi tiết thực thể “ room- song ”: bài hát trong phòng	33
Bảng 13 . Chi tiết thực thể “ room- chat ”: tin nhắn trong phòng	33
Bảng 14. Chi tiết thực thể “ room_current_playing ”: bài hát đang phát	34

TÓM TẮC ĐỒ ÁN CHUYÊN NGÀNH

Dự án "*Xây dựng website nghe nhạc trực tuyến theo thời gian thực bằng NextJs*" tập trung vào việc tạo ra một nền tảng cho phép người dùng nghe nhạc trực tuyến với tính năng đồng bộ thời gian thực, mang lại trải nghiệm tương tác giữa các người dùng. Vấn đề đặt ra là làm thế nào để thiết kế một hệ thống có khả năng xử lý đồng bộ nhiều yêu cầu cùng lúc, đảm bảo hiệu suất cao, bảo mật, và khả năng mở rộng.

Hướng tiếp cận đề tài:

- Nghiên cứu lý thuyết: Tìm hiểu các công nghệ cốt lõi như RESTful API, xác thực người dùng (JWT, Refresh Token), giao tiếp thời gian thực bằng WebSocket, và cách hoạt động của Next.js, Express.js, Sequelize, cùng các công cụ hỗ trợ Socket.io.

- Nghiên cứu thực nghiệm: Xây dựng từng thành phần hệ thống dựa trên lý thuyết, gồm API với Express.js, giao tiếp thời gian thực bằng WebSocket, và frontend bằng Next.js. Thực nghiệm các phương pháp tối ưu hiệu suất (SSR, SSG), triển khai xác thực bằng JWT, và kiểm tra toàn diện trước khi đưa vào hoạt động.

Kết quả mong đợi của đề tài:

- Website nghe nhạc trực tuyến được xây dựng với giao diện hiện đại, dễ sử dụng, có khả năng tương thích trên nhiều thiết bị.

- Tính năng đồng bộ thời gian thực hoạt động ổn định, cho phép nhiều người dùng cùng nghe một bài nhạc tại cùng thời điểm.

- Hệ thống xác thực an toàn, hỗ trợ đăng nhập và đăng ký, quản lý phiên hiệu quả.

- Hiệu suất hệ thống được tối ưu, tốc độ tải nhanh, và khả năng mở rộng phù hợp với lượng người dùng lớn.

MỞ ĐẦU

Lý do chọn đề tài:

Với sự phát triển mạnh mẽ của công nghệ web và nhu cầu ngày càng cao về các dịch vụ nghe nhạc trực tuyến, việc xây dựng một nền tảng nghe nhạc theo thời gian thực mang lại trải nghiệm mượt mà và đồng bộ cho người dùng là rất cần thiết. Dự án này được chọn để nghiên cứu và phát triển một website nghe nhạc trực tuyến, ứng dụng các công nghệ hiện đại như Next.js và WebSocket, nhằm tối ưu hóa hiệu suất và trải nghiệm người dùng.

Mục đích nghiên cứu:

Mục đích của dự án là xây dựng một hệ thống website nghe nhạc trực tuyến có khả năng đồng bộ bài hát giữa các người dùng theo thời gian thực. Dự án sẽ nghiên cứu cách kết hợp các công nghệ hiện đại như Next.js cho frontend, Node.js và Express.js cho backend, cùng với WebSocket để đảm bảo tính năng đồng bộ trong thời gian thực, giúp người dùng có trải nghiệm nghe nhạc mượt mà và liên tục.

Đối tượng vi nghiên cứu:

Các công nghệ phát triển web:

- Next.js: Framework cho React.js, hỗ trợ SSR, SSG và tối ưu hóa hiệu suất.
- WebSocket: Cho phép truyền tải dữ liệu thời gian thực giữa client và server.
- Node.js: Môi trường runtime giúp xử lý backend không đồng bộ.
- Express.js: Framework cho Node.js để xây dựng các API RESTful.

Các phương pháp xác thực người dùng:

- JWT (JSON Web Token): Đảm bảo bảo mật và xác thực người dùng trong API.
- OAuth 2.0: Xác thực thông qua bên thứ ba (Google, Facebook,...).

Phạm vi nghiên cứu:

Tìm hiểu và áp dụng các công nghệ frontend và backend để phát triển website nghe nhạc trực tuyến. Phạm vi nghiên cứu cũng giới hạn trong việc xây dựng một hệ thống với tính năng đồng bộ nhạc thời gian thực và xác thực người dùng.

CHƯƠNG 1 TỔNG QUAN

Dự án "Xây dựng website nghe nhạc trực tuyến theo thời gian thực bằng Next.js" tập trung vào việc phát triển một nền tảng web cho phép người dùng nghe nhạc trực tuyến với tính năng đồng bộ bài hát giữa các người dùng trong thời gian thực. Với sự phát triển nhanh chóng của công nghệ web, việc tạo ra một hệ thống nghe nhạc mượt mà và liền mạch, không bị gián đoạn, đã trở thành nhu cầu quan trọng đối với người dùng hiện đại.

Các vấn đề chính mà dự án hướng đến giải quyết bao gồm:

- Đồng bộ hóa bài hát: Đảm bảo mọi người dùng nghe cùng một bài hát, ở cùng một thời điểm, bất kể vị trí hay thiết bị của họ.
- Tối ưu hóa hiệu suất: Sử dụng các công nghệ như Next.js để cải thiện tốc độ tải trang và đảm bảo trải nghiệm người dùng mượt mà.
- Bảo mật và xác thực người dùng: Áp dụng các phương pháp xác thực như JWT hoặc OAuth 2.0 để bảo vệ tài khoản người dùng và dữ liệu cá nhân.

Với sự kết hợp của các công nghệ hiện đại như Next.js, Node.js, Express.js, và WebSocket, dự án không chỉ đảm bảo tính năng nghe nhạc theo thời gian thực mà còn tối ưu hóa SEO, cải thiện hiệu suất và bảo mật cho người dùng. Mục tiêu cuối cùng là xây dựng một nền tảng nghe nhạc trực tuyến tiện ích, thân thiện và hiệu quả.

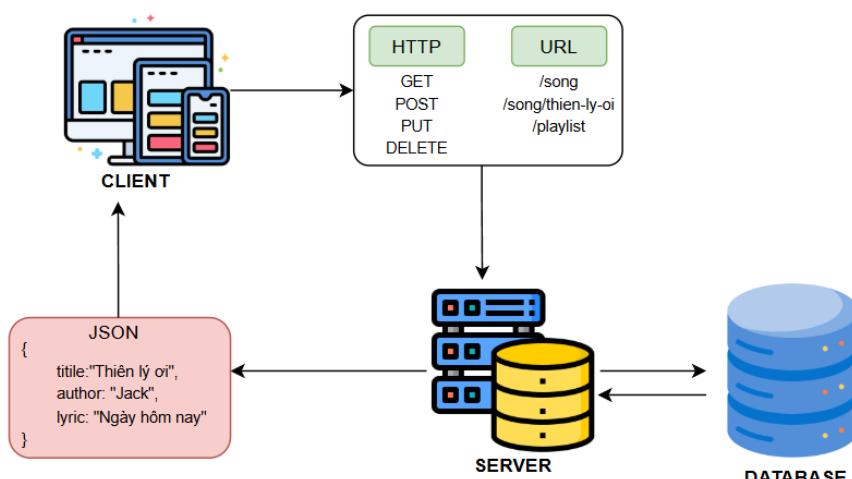
CHƯƠNG 2 NGHIÊN CỨU LÝ THUYẾT

1.1 Tìm hiểu RESTful API

2.1.1 Tổng quan về RESTful API

RESTful API (Representational State Transfer Application Programming Interface) [1] là một kiểu thiết kế giao diện lập trình ứng dụng dựa trên nguyên tắc REST (Representational State Transfer) – một phong cách kiến trúc phần mềm được giới thiệu lần đầu bởi Roy Fielding vào năm 2000. RESTful API sử dụng giao thức HTTP để trao đổi dữ liệu giữa client (frontend) và server (backend), giúp các ứng dụng có thể giao tiếp với nhau một cách hiệu quả, nhanh chóng và dễ dàng mở rộng.

RESTful API (Representational State Transfer) là một kiến trúc tiêu chuẩn được sử dụng để thiết kế các giao diện lập trình ứng dụng (API) cho phép các ứng dụng web giao tiếp với nhau một cách hiệu quả và đơn giản. RESTful API tập trung vào việc quản lý các tài nguyên (resources) trên hệ thống, bao gồm các tệp, ảnh, dữ liệu động, ...



Hình 1. Sơ đồ hoạt động của RESTful API

RESTful API (Representational State Transfer Application Programming Interface) là một kiểu thiết kế API phổ biến trong việc xây dựng các ứng dụng web. Nó cho phép các ứng dụng (web, mobile, desktop,...) giao tiếp với nhau thông qua giao thức HTTP.

RESTful API hoạt động dựa trên khái niệm **resource** (tài nguyên), trong đó mỗi tài nguyên được xác định bằng một URL duy nhất. Nó sử dụng các phương thức HTTP như GET, POST, PUT, DELETE để thao tác với dữ liệu. [1]

2.1.2 Tài nguyên (Resources)

Mọi thứ được truy xuất hoặc thao tác trên server đều được xem là một tài nguyên. Ví dụ: một bài viết, một người dùng, một sản phẩm, ...

Mỗi tài nguyên được xác định bởi một URI (Uniform Resource Identifier) duy nhất. URI này thường được xây dựng theo cấu trúc: <http://hostname/resource>.

Ví dụ:

<http://api.example.com/songs> (lấy tất cả bài hát)

<http://api.example.com/song/123> (bài hát có ID là 123)

2.1.3 Phương thức HTTP (HTTP Methods)

RESTful API sử dụng các phương thức HTTP để xác định hành động cần thực hiện trên tài nguyên:

GET: Lấy hoặc truy vấn dữ liệu của một hoặc nhiều tài nguyên từ server mà không làm thay đổi trạng thái dữ liệu trên server.

POST: Tạo mới một tài nguyên trên server. Phương thức này gửi dữ liệu từ client đến server để thêm tài nguyên mới.

PUT: Cập nhật toàn bộ dữ liệu của một tài nguyên hiện có trên server. Client cần gửi toàn bộ dữ liệu của tài nguyên, không chỉ các trường cần thay đổi.

DELETE: Xóa tài nguyên cụ thể trên server.

Các phương thức HTTP như GET, POST, PUT, DELETE trong RESTful API không chỉ xác định hành động mà còn giúp API tuân thủ các chuẩn giao tiếp, đảm bảo tính rõ ràng và dễ sử dụng. Việc áp dụng đúng các phương thức này giúp server dễ dàng xử lý các yêu cầu từ client và duy trì hệ thống nhất quán.

2.1.4 Trạng thái (Status)

RESTful API sử dụng các mã trạng thái HTTP để thông báo kết quả của các yêu cầu từ client. Các mã trạng thái này giúp client hiểu được tình trạng của yêu cầu và hướng xử lý tiếp theo, mỗi mã trạng thái mang một ý nghĩa cụ thể về kết quả của yêu cầu, có thể là thành công, lỗi phía người dùng, hoặc lỗi hệ thống. Dưới đây là các mã trạng thái HTTP phổ biến mà một RESTful API có thể trả về:

- 200 (OK): Yêu cầu đã được xử lý thành công và server đã trả về kết quả.

- 201 (Created): Tài nguyên đã được tạo thành công trên server.
- 400 (Bad Request): Yêu cầu của client không hợp lệ hoặc thiếu thông tin cần thiết để xử lý.
- 401 (Unauthorized): Client không có quyền truy cập tài nguyên hoặc yêu cầu này yêu cầu phải xác thực.
- 403 (Forbidden): Client đã được xác thực nhưng không có quyền truy cập tài nguyên yêu cầu.
- 404 (Not Found): Tài nguyên yêu cầu không tồn tại hoặc không thể tìm thấy trên server.

Mã trạng thái HTTP giúp xác định rõ ràng kết quả của một yêu cầu và giúp client hiểu được tình trạng của tài nguyên hoặc hành động yêu cầu. Các mã trạng thái này không chỉ phản ánh tình trạng thành công hay lỗi, mà còn giúp client xử lý thông báo lỗi một cách hiệu quả hơn thông qua các phản hồi chi tiết.

2.1.5 Ưu điểm của RESTful API

RESTful API mang lại nhiều ưu điểm, giúp cho việc phát triển và triển khai các ứng dụng trở nên dễ dàng và hiệu quả. Dưới đây là một số ưu điểm chính của RESTful API:

Đơn giản và dễ dùng

Mô hình giao tiếp đơn giản: RESTful API sử dụng HTTP làm giao thức truyền thông, dễ hiểu và đơn giản để triển khai. Các phương thức như GET, POST, PUT, DELETE đều được xác định rõ ràng, giúp việc sử dụng và bảo trì API trở nên dễ dàng hơn.

Khả năng mở rộng

RESTful API dễ mở rộng nhờ vào nguyên lý đơn giản và chuẩn HTTP. Nó hỗ trợ nhiều loại ứng dụng khác nhau, từ web đến mobile, và không gặp vấn đề tương thích giữa các nền tảng hay ngôn ngữ lập trình.

Tính linh hoạt của ứng dụng

RESTful API không phụ thuộc vào trạng thái trước đó, giúp server dễ bảo trì và mở rộng. Nó có thể được sử dụng cho nhiều loại ứng dụng mà không bị hạn chế bởi hệ điều hành hay framework.

Tính tương thích và mở

RESTful API có thể tích hợp và giao tiếp với nhiều hệ thống khác nhau, như cơ sở dữ liệu, dịch vụ web khác, hoặc các hệ thống bên ngoài. Điều này giúp mở rộng khả năng kết nối và tích hợp trong một môi trường đa dạng. RESTful API được sử dụng rộng rãi, vì vậy cộng đồng phát triển luôn cung cấp tài liệu, công cụ, và hỗ trợ kỹ thuật để giải quyết các vấn đề liên quan đến API.

2.1.6 Nhược điểm của RESTful API

Không hỗ trợ trạng thái (Stateless)

Vì RESTful API không lưu trữ trạng thái giữa các yêu cầu (stateless), mỗi yêu cầu phải chứa toàn bộ thông tin cần thiết để server xử lý. Điều này có thể dẫn đến việc yêu cầu nhiều dữ liệu trong mỗi lần gọi API, làm tăng băng thông và giảm hiệu suất, đặc biệt với các ứng dụng cần duy trì trạng thái người dùng hoặc phiên làm việc.

Quá tải khi có quá nhiều yêu cầu

Khi hệ thống yêu cầu nhiều tài nguyên và giao dịch, RESTful API có thể gặp khó khăn trong việc xử lý hàng loạt yêu cầu. Việc phải thực hiện nhiều yêu cầu HTTP liên tiếp có thể gây tải cao cho server và giảm hiệu suất tổng thể của hệ thống.

2.2 Xác thực người dùng

2.2.1 Xác thực người dùng là gì ?

Xác thực (Authentication) là một bước quan trọng trong bảo mật RESTful API, đảm bảo chỉ những người dùng hợp lệ hoặc ứng dụng được cấp quyền mới có thể truy cập tài nguyên. Xác thực giúp bảo vệ dữ liệu nhạy cảm, ngăn chặn truy cập trái phép, và kiểm soát quyền truy cập dựa trên vai trò hoặc đặc quyền của người dùng.

Một hệ thống xác thực hiệu quả không chỉ bảo vệ tài nguyên mà còn đảm bảo tính dễ sử dụng và khả năng mở rộng cho ứng dụng. Tùy thuộc vào loại ứng dụng và yêu cầu bảo mật, có nhiều phương pháp xác thực khác nhau, từ các cách đơn giản như Basic Authentication, đến các phương pháp hiện đại như JWT hoặc OAuth 2.0. Những phương pháp này được thiết kế để đáp ứng các mức độ bảo mật và khả năng tích hợp khác nhau.

2.2.2 Phương pháp xác thực phổ biến

Bảo mật thông tin là một yếu tố vô cùng quan trọng. Để đảm bảo rằng chỉ những người dùng được ủy quyền mới có thể truy cập vào dữ liệu và chức năng của một ứng dụng, phải sử dụng các phương thức xác thực phù hợp. RESTful API, với tính chất mở và thường xuyên được truy cập từ nhiều nguồn khác nhau, càng cần phải có các cơ chế xác thực mạnh mẽ.

Basic Authentication: Đây là phương pháp đơn giản nhất, nhưng cũng là phương pháp ít an toàn nhất. Khi sử dụng Basic Authentication, username và password của người dùng sẽ được mã hóa bằng Base64 trước khi gửi trong header của request. Tuy nhiên, Base64 chỉ là một dạng mã hóa đơn giản, dễ dàng bị giải mã. Do đó, mật khẩu vẫn có thể bị lộ trong quá trình truyền đi.

Token: là một giải pháp hiện đại hơn, trong đó người dùng đăng nhập để nhận một token (mã). Token này được gửi kèm theo các yêu cầu API để xác thực. Token có thể hết hạn, giúp tăng cường bảo mật, nhưng việc quản lý token trên cả client và server yêu cầu nhiều công sức hơn.

OAuth: là một giao thức phổ biến, đặc biệt phù hợp với các ứng dụng lớn hoặc cần tích hợp bên thứ ba như Google hay Facebook. Người dùng xác thực qua bên thứ ba và nhận được một token để truy cập API. Tuy nhiên, việc cấu hình và triển khai OAuth 2.0 khá phức tạp.

JSON Web Token (JWT): Đây cũng là một phương pháp phổ biến. JWT là một đoạn mã chứa thông tin người dùng, được “ký” bằng khóa bí mật để đảm bảo tính toàn vẹn. JWT gọn nhẹ, dễ triển khai và không cần lưu trữ trên server, nhưng cần cẩn thận khi xử lý dữ liệu nhạy cảm trong payload. [3]

Refresh Token: Là một phần quan trọng trong cơ chế xác thực dựa trên Token-Based Authentication, đặc biệt khi sử dụng JWT (JSON Web Token) hoặc

các giao thức như OAuth 2.0. Mục tiêu chính của Refresh Token là cải thiện bảo mật và trải nghiệm người dùng bằng cách giảm nguy cơ lộ Access Token trong khi vẫn cho phép người dùng duy trì phiên đăng nhập lâu dài. [3]

2.2.3 Sự cần thiết của xác thực người dùng

Xác thực người dùng là một thành phần thiết yếu trong bất kỳ hệ thống hoặc ứng dụng nào, đặc biệt là trong các API RESTful. Nó đảm bảo rằng chỉ những người dùng hợp lệ mới có quyền truy cập vào tài nguyên hoặc dịch vụ.

Bảo vệ tài nguyên và dữ liệu

Xác thực giúp ngăn chặn các truy cập trái phép vào hệ thống, đảm bảo rằng chỉ người dùng đã được cấp quyền mới có thể truy cập dữ liệu hoặc tài nguyên nhạy cảm. Điều này rất quan trọng với các ứng dụng web và di động, nơi dữ liệu cá nhân (như thông tin tài khoản, giao dịch tài chính) hoặc dữ liệu nhạy cảm (hồ sơ bệnh án, tài liệu nội bộ) cần được bảo mật.

Quản lý quyền truy cập:

Xác thực không chỉ đảm bảo người dùng là hợp lệ, mà còn hỗ trợ kiểm soát quyền truy cập. Người dùng bình thường có thể chỉ xem thông tin, nhưng quản trị viên có thể chỉnh sửa hoặc xóa dữ liệu. Điều này giúp hệ thống hoạt động linh hoạt và an toàn hơn, hạn chế sai sót hoặc lạm dụng từ người dùng.

Bảo vệ chống tấn công:

Không xác thực hoặc xác thực kém có thể dẫn đến nhiều loại tấn công bảo mật như:

- Truy cập trái phép (Unauthorized Access): Kẻ tấn công có thể lợi dụng lỗ hổng để xâm nhập hệ thống.
- Đánh cắp dữ liệu (Data Breach): Dữ liệu nhạy cảm bị lộ ra ngoài.
- Tấn công từ chối dịch vụ (DDoS): Kẻ tấn công có thể gửi yêu cầu giả mạo đến hệ thống mà không bị kiểm soát.

Duy trì đăng nhập:

Xác thực hỗ trợ duy trì các phiên làm việc của người dùng, cho phép họ tiếp tục sử dụng dịch vụ mà không cần đăng nhập lại thường xuyên. Tuy nhiên, hệ thống

cũng cần xác thực lại khi có hành động quan trọng (như thay đổi mật khẩu hoặc giao dịch lớn) để đảm bảo an toàn.

2.3 Tìm hiểu Next.js

2.3.1 Tổng quan về Next.js

Next.js là một framework React mã nguồn mở được phát triển bởi Vercel, ra đời nhằm cung cấp giải pháp toàn diện cho việc xây dựng các ứng dụng web hiện đại. Với việc kết hợp các chiến lược render tiên tiến như Server-Side Rendering (SSR), Static Site Generation (SSG) và Client-Side Rendering (CSR), Next.js giúp các nhà phát triển tạo ra các ứng dụng web với hiệu năng vượt trội, khả năng tối ưu hóa SEO và trải nghiệm người dùng xuất sắc. [2]



Hình 2. Logo Next.js

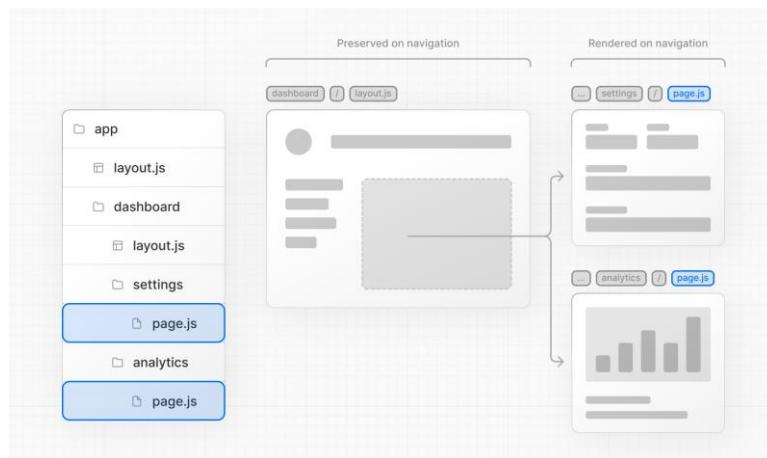
Framework này được thiết kế để đơn giản hóa quá trình phát triển ứng dụng web bằng cách cung cấp các tính năng mạnh mẽ mà không yêu cầu quá nhiều cấu hình phức tạp. Nhờ tích hợp sẵn những công cụ hữu ích như routing dựa trên file, hỗ trợ API nội bộ, và khả năng mở rộng linh hoạt, Next.js trở thành một trong những lựa chọn hàng đầu cho các dự án web quy mô nhỏ đến lớn.

2.3.2 Kiến trúc hoạt động

Next.js sử dụng một kiến trúc linh hoạt, kết hợp giữa React, Node.js, và các chiến lược render như SSR, SSG và Client-Side Rendering (CSR).

File-based Routing:

Cấu trúc routing trong Next.js dựa trên hệ thống file. Mỗi file trong thư mục /app tương ứng với một route của ứng dụng. Điều này giúp đơn giản hóa việc tạo và quản lý các route.



Hình 3. Cấu trúc tập tin File-based Routing

Rendering linh hoạt:

- **Server-Side Rendering (SSR):** Nội dung trang được render trên server với dữ liệu động và gửi về client.
- **Static Site Generation (SSG):** Các trang tĩnh được tạo trong quá trình build và phục vụ trực tiếp dưới dạng file tĩnh.
- **Client-Side Rendering (CSR):** Một số thành phần hoặc trang có thể được render trên client sau khi tải.

API Routes:

Next.js tích hợp backend nhẹ bằng cách sử dụng các file trong thư mục /api, cho phép xử lý các request API mà không cần một server backend riêng biệt.

Hỗ trợ Pre-fetching:

Các liên kết giữa các trang được pre-fetch tự động, giúp cải thiện tốc độ tải trang và trải nghiệm người dùng.

2.3.3 Ưu điểm

Hiệu suất cao:

Next.js tận dụng sự kết hợp giữa Server-Side Rendering (SSR) và Static Site Generation (SSG) để tối ưu hóa tốc độ tải trang. SSR cho phép tải dữ liệu động từ server trước khi gửi về client, trong khi SSG tạo ra các trang tĩnh được phục vụ ngay lập tức từ CDN. Cách tiếp cận này không chỉ cải thiện tốc độ mà còn đảm bảo trải nghiệm người dùng mượt mà hơn, đặc biệt trên các thiết bị di động hoặc mạng có băng thông thấp.

Tối ưu hóa SEO:

Khả năng render phía server giúp các trang web được hiển thị với đầy đủ nội dung ngay từ đầu, điều này rất quan trọng đối với các công cụ tìm kiếm như Google trong việc lập chỉ mục và xếp hạng trang web. Ngoài ra, các tính năng như dynamic routing và hỗ trợ meta tags nâng cao khả năng tối ưu hóa SEO cho từng trang hoặc sản phẩm cụ thể.

Dễ dàng tích hợp:

Next.js hỗ trợ tích hợp với các thư viện và công cụ như TypeScript, Tailwind CSS, Redux, GraphQL,...

Khả năng mở rộng:

Next.js phù hợp với các ứng dụng nhỏ lẫn các hệ thống lớn nhờ tính linh hoạt trong cách tổ chức và triển khai.

2.3.4 Nhược điểm

Phụ thuộc nhiều vào Node.js:

Vì Next.js hoạt động trên môi trường Node.js, hiệu suất của ứng dụng phụ thuộc vào khả năng cấu hình và tối ưu hóa Node.js.

Tăng kích thước bundle:

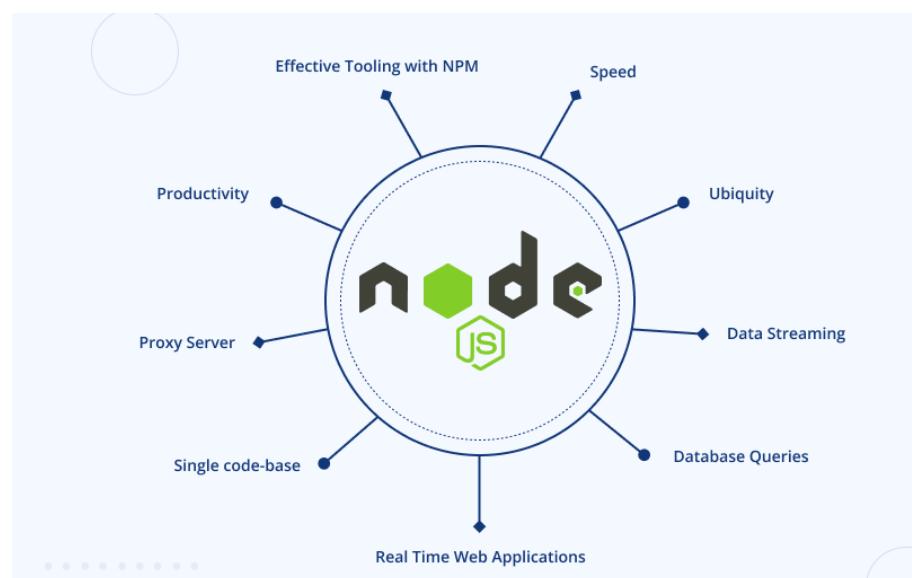
Khi ứng dụng mở rộng hoặc sử dụng nhiều thư viện, kích thước bundle có thể tăng, ảnh hưởng đến thời gian tải trang.

Chi phí hosting cao hơn:

Nếu ứng dụng yêu cầu SSR nhiều, chi phí hosting có thể cao hơn so với ứng dụng chỉ sử dụng SSG hoặc CSR.

2.4 Tìm hiểu NodeJS

2.4.1 Giới thiệu tổng quan NodeJS



Hình 4. Các tính năng cơ bản của nodejs

Node.js là một nền tảng xây dựng ứng dụng mạng có hiệu suất cao dựa trên JavaScript runtime của Chrome V8. Nó cho phép lập trình viên sử dụng JavaScript để viết mã cho cả phía server và client, đồng thời cung cấp khả năng xử lý không đồng bộ và lập trình sự kiện, tối ưu cho ứng dụng web thời gian thực. Nó là một nền tảng JavaScript thuần túy điều này có nghĩa là có thể sử dụng tất cả các kiến thức JavaScript để phát triển ứng dụng web. Node.js có thể xử lý nhiều yêu cầu cùng lúc một cách hiệu quả.

2.4.2 Kiến trúc hoạt động

Node.js sử dụng mô hình **Event-driven** (hướng sự kiện) và **Non-blocking I/O** (I/O không chặn), cho phép xử lý nhiều yêu cầu cùng lúc mà không cần phải chờ từng yêu cầu hoàn tất. Điều này khác biệt so với các mô hình đồng bộ truyền thống:

Single-threaded: Node.js sử dụng một luồng duy nhất để xử lý yêu cầu thay vì tạo nhiều luồng.

Event Loop: Là trái tim của Node.js, giúp xử lý các yêu cầu theo cơ chế bất đồng bộ. Khi nhận được một yêu cầu, Node.js chuyển tác vụ nặng sang các luồng phụ hoặc hệ thống bên ngoài để xử lý, và sau đó tiếp tục xử lý các yêu cầu khác.

Non-blocking I/O: Cho phép các tác vụ như đọc/ghi file hoặc truy vấn cơ sở dữ liệu thực hiện song song, giúp tiết kiệm thời gian và tài nguyên.

2.4.3 Ưu điểm

Hiệu suất cao:

Nhờ mô hình I/O không đồng bộ, Node.js rất phù hợp cho các ứng dụng cần xử lý nhiều kết nối đồng thời, như các ứng dụng thời gian thực hoặc các API, làm tăng hiệu suất của các ứng dụng.

Dễ dàng mở rộng:

Node.js hỗ trợ xây dựng các ứng dụng có khả năng mở rộng cao thông qua mô hình sự kiện và bất đồng bộ của mình. Điều này cho phép xử lý hàng ngàn kết nối đồng thời mà không làm giảm hiệu suất.

Cộng đồng lớn:

Node.js có một cộng đồng phát triển mạnh mẽ, với hàng triệu package trên npm, hỗ trợ các nhà phát triển giải quyết hầu hết các vấn đề thông qua thư viện hoặc module có sẵn.

Hỗ trợ xây dựng ứng dụng thời gian thực:

Node.js là lựa chọn hàng đầu để xây dựng các ứng dụng real-time như chat, game, hay các hệ thống theo dõi trực tuyến, nhờ khả năng xử lý đồng thời và hỗ trợ WebSocket.

2.4.4 Nhược điểm

Chạy đơn luồng: Vì chạy trên một luồng chính, Node.js không phù hợp với các tác vụ nặng về CPU như xử lý dữ liệu lớn hoặc tính toán phức tạp. Điều này có thể làm nghẽn và giảm hiệu năng tổng thể.

Xử lý đồng thời: Node.js có thể xử lý nhiều yêu cầu cùng lúc, nhưng vì nó chỉ chạy trên một luồng chính, nên không thể tận dụng hết sức mạnh của máy tính có nhiều nhân CPU. Để làm được điều đó, cần thiết lập thêm cách chia công việc cho các luồng phụ, nhưng điều này đòi hỏi cấu hình phức tạp hơn.

CHƯƠNG 3 HIỆN THỰC HÓA NGHIÊN CỨU

3.1 Đặc tả yêu cầu hệ thống

Đề tài "Xây dựng website nghe nhạc trực tuyến theo thời gian thực bằng Next.js" tập trung vào việc giải quyết bài toán đồng bộ hóa phát nhạc giữa nhiều người dùng trong thời gian thực trên nền tảng web. Đảm bảo hệ thống hoạt động ổn định, bảo mật, và dễ sử dụng trên mọi thiết bị. Cung cấp trải nghiệm nghe nhạc đồng bộ, tối ưu và hiện đại cho người dùng. Các khía cạnh của bài toán bao gồm:

Đồng bộ hóa phát nhạc theo thời gian thực:

- Đảm bảo nhiều người dùng có thể nghe cùng một bài hát, cùng thời điểm bất kể thiết bị hay vị trí địa lý.
- Khi một người dùng thực hiện hành động như phát, tạm dừng hoặc chuyển bài, tất cả người dùng khác cũng được cập nhật trạng thái ngay lập tức.

Quản lý người dùng và danh sách phát cá nhân:

- Người dùng có thể đăng nhập, tạo danh sách phát cá nhân và chia sẻ danh sách này với bạn bè.
- Tính năng phân quyền: chỉ chủ sở hữu hoặc người được mời mới có thể quản lý danh sách phát.

Tối ưu hóa trải nghiệm người dùng:

- Sử dụng giao diện mượt mà, tốc độ tải nhanh và thân thiện với người dùng.
- Thiết kế giao diện hỗ trợ tốt cho cả thiết bị di động và máy tính để bàn.

Bảo mật dữ liệu và xác thực:

- Bảo vệ tài khoản người dùng bằng các cơ chế xác thực hiện đại như JWT.
- Đảm bảo dữ liệu cá nhân, danh sách phát và lịch sử nghe nhạc được bảo mật.

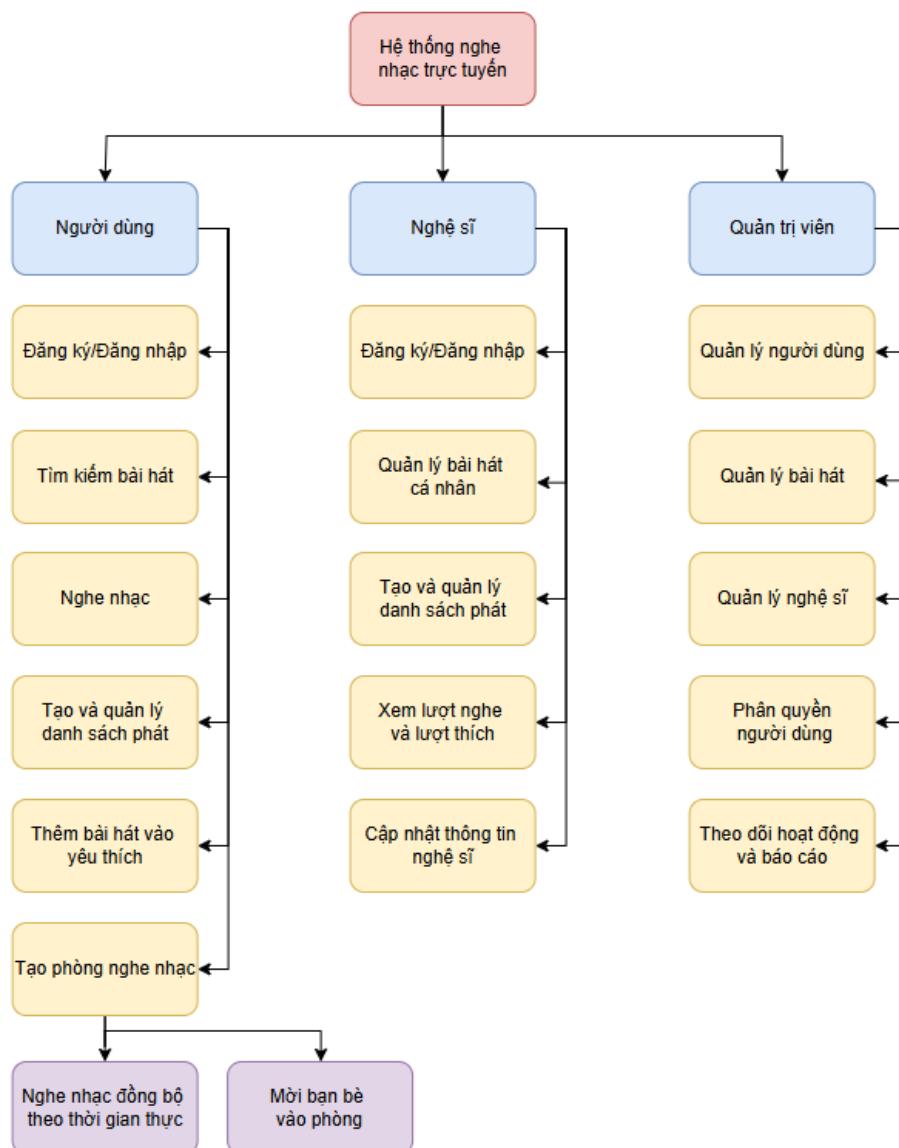
Quản trị:

- Cấp quyền hoặc thu hồi quyền (quản trị viên, nghệ sĩ, người dùng thông thường).
- Quản lý thông tin nghệ sĩ.
- Theo dõi lượt truy cập, lượt nghe nhạc, số lượng người dùng đăng ký và hoạt động trên hệ thống.

- Xem báo cáo tổng quan về các nội dung phổ biến, nghệ sĩ được yêu thích và các xu hướng âm nhạc.

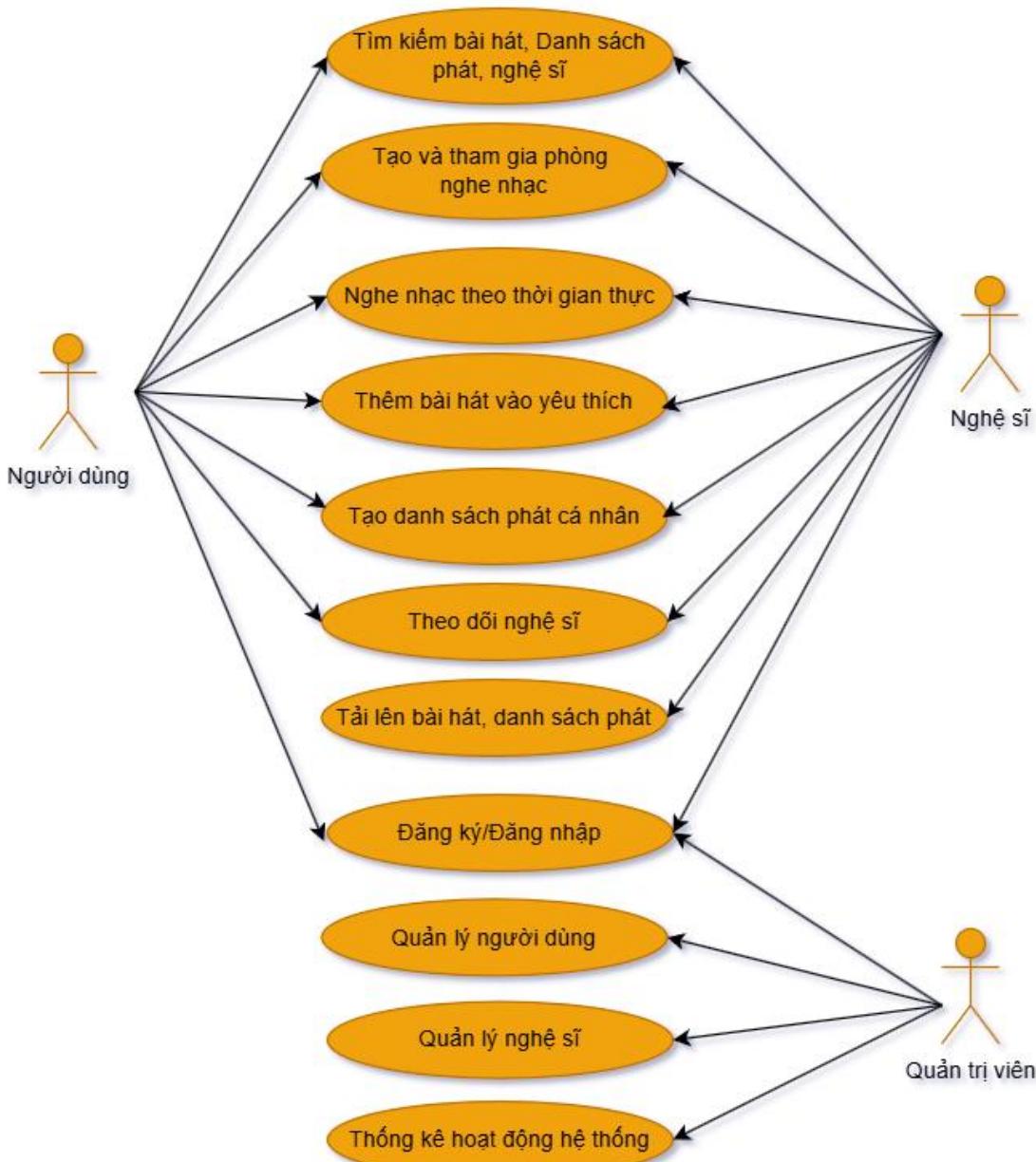
3.2 Mô hình chức năng

Sơ đồ chức năng hệ thống minh họa các vai trò và quyền hạn chính: Người dùng, Nghệ sĩ, và Quản trị viên. Người dùng có thể thực hiện các hành động như tìm kiếm, nghe nhạc, quản lý danh sách phát, tạo phòng nghe nhạc đồng bộ theo thời gian thực và tùy chỉnh hồ sơ cá nhân. Nghệ sĩ được phép quản lý bài hát của riêng mình, cập nhật thông tin cá nhân và theo dõi lượt thích, lượt nghe. Quản trị viên đảm nhiệm việc quản lý toàn bộ hệ thống, bao gồm người dùng, bài hát, nghệ sĩ và phân quyền, đảm bảo tính bảo mật và hiệu suất thông qua giám sát các hoạt động và phân tích báo cáo chi tiết.



Hình 5. Mô hình chức năng của hệ thống

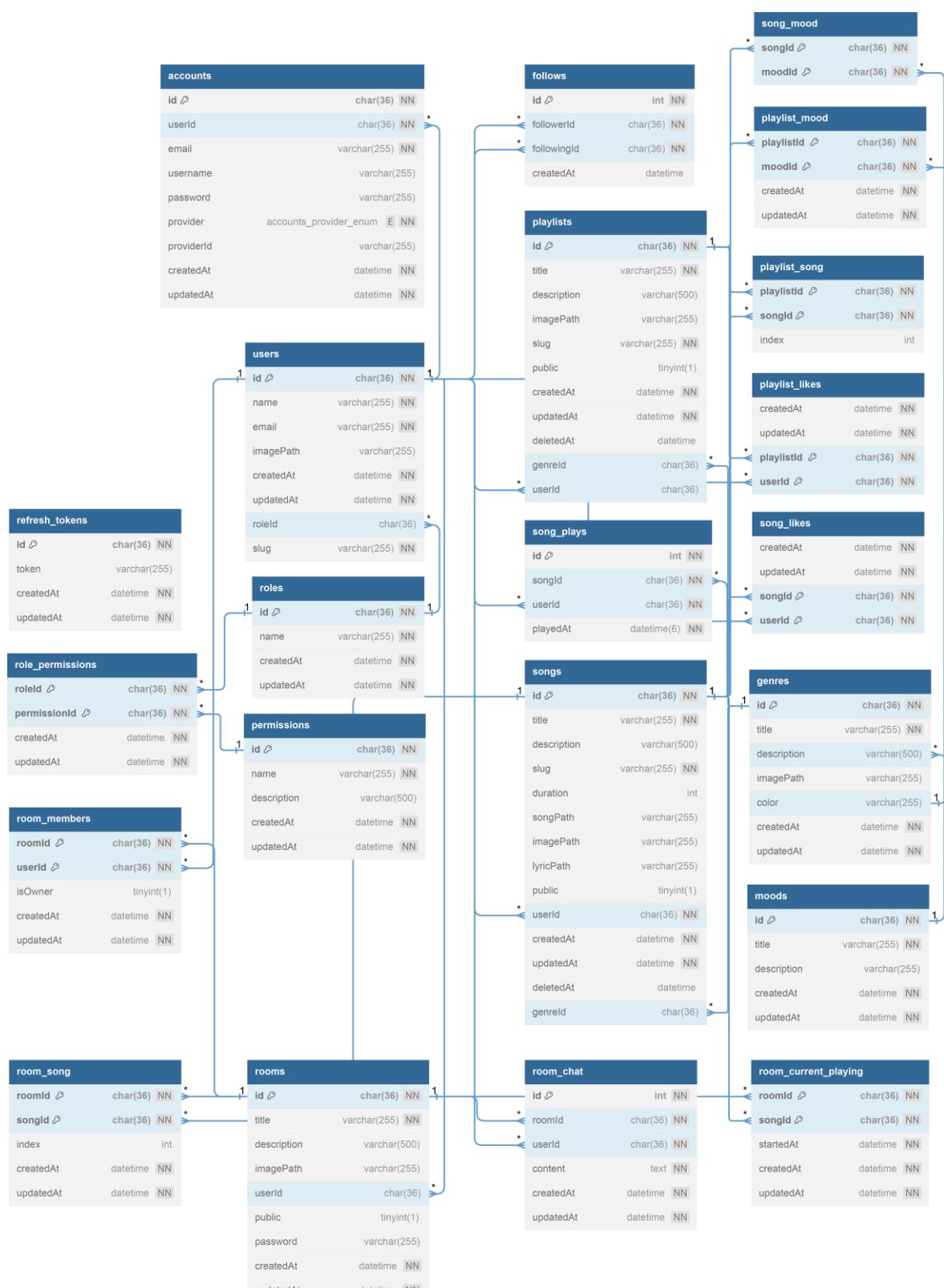
3.3 Sơ đồ use-case



Hình 6. Sơ đồ use-case

3.4 Thiết kế dữ liệu

3.4.1 Lược đồ cơ sở dữ liệu



Hình 7. Lược đồ cơ sở dữ liệu

Bảng 1. Bảng danh sách các thực thể

STT	Tên thực thể	Diễn giải
1	users	Thông tin người dùng
2	accounts	Tài khoản
3	roles	Vai trò
4	permissions	Quyền
5	role_permissions	Quyền của vai trò
6	songs	Bài hát
7	playlists	Danh sách phát
8	genres	Thể loại
9	moods	Tâm trạng
10	playlist_song	Các bài hát của danh sách phát
11	playlist_likes	Yêu thích danh sách phát
12	song_likes	Yêu thích bài hát
13	song_plays	Phát bài hát
14	follows	Theo dõi người dùng
15	song_mood	Tâm trạng của bài hát
16	playlist_mood	Tâm trạng của danh sách phát
17	room	Phòng
18	room_members	Danh sách thành viên của phòng
19	room_song	Danh sách bài hát của phòng

20	room_chat	Tin nhắn trong phòng
21	room_current_playing	Bài hát đang phát trong phòng
22	refresh_token	Làm mới mã

3.4.2 Chi tiết các thực thể

Một số bảng chính trong dự án bao gồm:

* Bảng user

- Tên thực thể: user
- Mô tả: Lưu trữ thông tin người dùng
- Chi tiết thực thể:

Bảng 2. Chi tiết thực thể “user”: người dùng

STT	Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc toàn vẹn
1	id	id	char	Khóa chính
2	name	Họ tên	varchar	
3	email	Email	varchar	
4	slug	Đường dẫn	varchar	
5	roleId	Id quyền	char	Khóa ngoại
6	imagePath	Ảnh đại diện	varchar	
7	createdAt	Ngày tạo	datetime	
8	updateAt	Ngày cập nhật	datetime	

* Bảng accounts

- Tên thực thể: accounts
- Mô tả: Lưu trữ thông tin tài khoản
- Chi tiết thực thể:

Bảng 3. Chi tiết thực thể “*account*”: tài khoản

STT	Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc toàn vẹn
1	id	id	char	Khóa chính
2	userId	Id người dùng	char	Khóa ngoại
3	email	Email	varchar	
4	username	Tên đăng nhập	varchar	
5	password	Mật khẩu	varchar	
6	provide	Tên nhà cung cấp	varchar	
7	provideId	Id nhà cung cấp	varchar	

* Bảng roles

- Tên thực thể: **roles**
- Mô tả: Lưu trữ thông tin của vai trò
- Chi tiết thực thể:

Bảng 4. Chi tiết thực thể “*roles*”: vai trò

STT	Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc toàn vẹn
1	Id	id	char	Khóa chính
2	name	Tên vai trò	varchar	
3	createdAt	Ngày tạo	datetime	
4	updateAt	Ngày cập nhật	datetime	

* Bảng permissions

- Tên thực thể: **permissions**
- Mô tả: Lưu trữ thông tin của quyền
- Chi tiết thực thể:

Bảng 5. Chi tiết thực thể “**permissions**”: quyền

STT	Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc toàn vẹn
1	Id	id	char	Khóa chính
2	Name	Tên vai trò	varchar	
3	createdAt	Ngày tạo	datetime	
4	updatedAt	Ngày cập nhật	datetime	

* **Bảng songs**

- Tên thực thể: **songs**
- Mô tả: Lưu trữ thông tin bài hát
- Chi tiết thực thể:

Bảng 6. Chi tiết thực thể “**songs**”: bài hát

STT	Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc toàn vẹn
1	id	id	char	Khóa chính
2	title	Tiêu đề	char	
3	description	Mô tả	varchar	
4	slug	Đường dẫn	varchar	
5	duration	Thời lượng bài hát	varchar	
6	songPath	Đường dẫn bài hát	varchar	
7	imagePath	Đường dẫn hình ảnh	varchar	
8	lyricPath	Đường dẫn lời bài hát	varchar	
9	public	Công khai	boolean	
10	userId	Tác giả	char	Khóa ngoại

11	genreId	Thể loại	char	
12	createdAt	Ngày tạo	datetime	
13	updateAt	Ngày cập nhật	datetime	
14	deletedAt	Ngày xóa	datetime	

* **Bảng playlist**

- Tên thực thể: **playlists**
- Mô tả: Lưu trữ thông tin danh sách phát
- Chi tiết thực thể:

Bảng 7. Chi tiết thực thể “**playlist**”: danh sách phát

STT	Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc toàn vẹn
1	id	id	char	Khóa chính
2	title	Tiêu đề	char	
3	description	Mô tả	varchar	
4	imagePath	Đường dẫn hình ảnh	varchar	
5	slug	Đường dẫn	varchar	
6	public	Công khai	varchar	
7	genreId	Thể loại	char	Khóa ngoại
8	userId	Tác giả	char	Khóa ngoại
9	deletedAt	Ngày xóa	datetime	
10	createdAt	Ngày tạo	datetime	
11	updateAt	Ngày cập nhật	datetime	

* **Bảng genre**

- Tên thực thể: **genre**
- Mô tả: Lưu trữ thông tin thể loại bài hát
- Chi tiết thực thể:

Bảng 8. Chi tiết thực thể “genre”: thể loại

STT	Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc toàn vẹn
1	id	id	char	Khóa chính
2	title	Tiêu đề	char	
3	description	Mô tả	varchar	
4	imagePath	Đường dẫn hình ảnh	varchar	
5	color	Màu sắc	varchar	

* **Bảng moods**

- Tên thực thể: **moods**
- Mô tả: Lưu trữ thông tin tâm trạng bài hát
- Chi tiết thực thể:

Bảng 9. Chi tiết thực thể “moods”: tâm trạng

STT	Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc toàn vẹn
1	id	id	char	Khóa chính
2	title	Tiêu đề	char	
3	description	Mô tả	varchar	

* **Bảng room**

- Tên thực thể: **room**
- Mô tả: Lưu trữ thông tin phòng nghe nhạc

- Chi tiết thực thể:

Bảng 10. Chi tiết thực thể “room”: phòng

STT	Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc toàn vẹn
1	id	id	char	Khóa chính
2	title	Tiêu đề	varchar	
3	description	Mô tả	varchar	
4	imagePath	Đường dẫn ảnh nền	varchar	
5	userId	Id tác giả	char	Khóa ngoại
6	public	Công khai	boolean	
7	password	Mật khẩu	varchar	

* **Bảng room-members**

- Tên thực thể: **room-members**
- Mô tả: Lưu trữ danh sách thành viên trong phòng
- Chi tiết thực thể:

Bảng 11. Chi tiết thực thể “room-members”: thành viên trong phòng

STT	Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc toàn vẹn
1	userId	Id thành viên	char	Khóa ngoại
2	roomId	Id phòng	char	Khóa ngoại
3	createdAt	Ngày tạo	datetime	
4	updateAt	Ngày cập nhật	datetime	

* **Bảng room-song**

- Tên thực thể: **room- song**

- Mô tả: Lưu trữ danh sách bài hát trong phòng

- Chi tiết thực thể:

Bảng 12. Chi tiết thực thể “room- song”: bài hát trong phòng

STT	Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc toàn vẹn
1	songId	Id bài hát	char	Khóa ngoại
2	roomId	Id phòng	char	Khóa ngoại
3	index	Thứ tự phát	interger	
4	createdAt	Ngày tạo	datetime	
5	updateAt	Ngày cập nhật	datetime	

* Bảng room-chat

- Tên thực thể: **room- chat**

- Mô tả: Lưu trữ danh sách các tin nhắn trong phòng

- Chi tiết thực thể:

Bảng 13 . Chi tiết thực thể “room- chat ”: tin nhắn trong phòng

STT	Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc toàn vẹn
1	Id	Id	char	Khóa chính
2	roomId	Id phòng	char	Khóa ngoại
3	userId	Id người dùng	char	Khóa ngoại
4	content	Nội dung	varchar	
5	createdAt	Ngày tạo	datetime	
6	updateAt	Ngày cập nhật	datetime	

* Bảng room_current_playing

- Tên thực thể: **room_current_playing**

- Mô tả: Lưu trữ thông tin bài hát đang phát

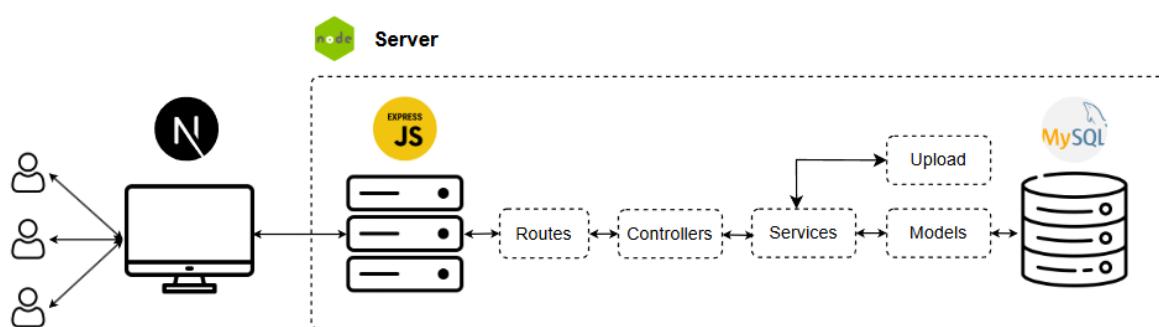
- Chi tiết thực thể:

Bảng 14. Chi tiết thực thể “room_current_playing”: bài hát đang phát trong phòng

STT	Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc toàn vẹn
1	roomId	Id phòng	char	Khóa ngoại
2	songId	Id bài hát	char	Khóa ngoại
3	startedAt	Thời gian bắt đầu phát	datetime	
4	createdAt	Ngày tạo	datetime	
5	updateAt	Ngày cập nhật	datetime	

3.5 Thiết kế, kiến trúc ứng dụng

3.5.1 Kiến trúc toàn cảnh



Hình 8. Sơ đồ kiến trúc toàn cảnh của hệ thống

Kiến trúc toàn cảnh của hệ được thiết kế theo mô hình phân tầng, bao gồm các thành phần chính:

- **Frontend:** Đây là phần giao diện người dùng, được xây dựng bằng **Next.js**. Frontend sẽ đảm nhiệm việc tương tác với người dùng, cung cấp các tính năng như tìm kiếm bài hát, nghe nhạc, tạo và quản lý danh sách phát, và phòng nghe nhạc

đồng bộ. Các yêu cầu từ người dùng sẽ được gửi đến server thông qua các API RESTful và kết nối WebSocket cho việc đồng bộ nhạc theo thời gian thực.

- **Backend:** Backend sử dụng **Node.js** với **Express.js** để xử lý các yêu cầu từ client. Các API RESTful sẽ thực hiện việc truy xuất dữ liệu, quản lý người dùng, bài hát, nghệ sĩ, và các tính năng khác của hệ thống. Ngoài ra, backend cũng sử dụng **Socket.io** để quản lý kết nối và đồng bộ nhạc theo thời gian thực giữa các người dùng trong cùng một phòng nghe nhạc.

- **Database:** Cơ sở dữ liệu được quản lý bằng **MySQL** và sử dụng **Sequelize** để kết nối và thao tác với dữ liệu. Các bảng dữ liệu sẽ chứa thông tin về người dùng, bài hát, nghệ sĩ, danh sách phát, và các phòng nghe nhạc.

- **Authentication & Authorization:** Hệ thống xác thực và phân quyền người dùng được triển khai thông qua các phương thức JWT (JSON Web Tokens) và OAuth 2.0, đảm bảo an toàn khi đăng nhập và sử dụng các tính năng của hệ thống.

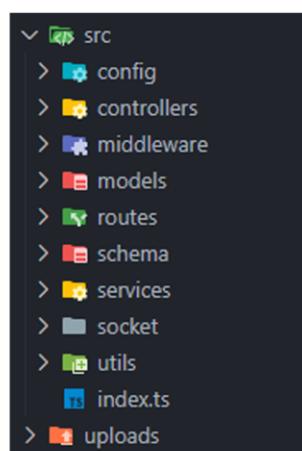
- **File Storage:** Các bài hát và tài nguyên media sẽ được lưu trữ trong thư mục **upload** trên server hoặc sử dụng dịch vụ lưu trữ đám mây để đảm bảo tính ổn định và mở rộng của hệ thống.

- **Real-time:** **Socket.io** sẽ quản lý việc đồng bộ nhạc trong thời gian thực, cho phép người dùng trong cùng một phòng nghe nhạc có thể cùng nghe một bài hát ở cùng thời điểm, bao gồm các hành động như play, pause, seek.

Kiến trúc giúp hệ thống hoạt động mượt mà, đáp ứng được các yêu cầu về hiệu suất, khả năng mở rộng và tính năng thời gian thực, đồng thời bảo mật và bảo vệ quyền lợi người dùng.

3.5.2 Thiết kế API

Trong quá trình thiết kế API cho dự án, mục tiêu chính để tạo ra một giao diện người dùng và hệ thống có thể giao tiếp một cách hiệu quả, an toàn và dễ bảo trì. Với TypeScript, có thể tận dụng tính năng kiểu tĩnh (static typing) để đảm bảo tính đúng đắn trong quá trình phát triển API, giảm thiểu lỗi và cải thiện hiệu suất làm việc. Để thực hiện điều này, cần xây dựng các API RESTful rõ ràng và dễ sử dụng cho các tính năng của hệ thống, bao gồm các chức năng như đăng ký người dùng, đăng nhập, tìm kiếm bài hát, tạo playlist, chia sẻ bài hát,... Cấu trúc API được sử dụng như sau:



- src:** Chứa toàn bộ mã nguồn
- config:** Lưu trữ các cấu hình toàn cục và kết nối dịch vụ.
- controllers:** Chứa các tệp xử lý logic và yêu cầu HTTP từ client.
- middleware:** Xử lý các tác vụ giữa việc nhận yêu cầu và gửi phản hồi từ server (xác thực, kiểm tra quyền truy cập).
- models:** Định nghĩa cấu trúc dữ liệu và các phương thức tương tác với DB.
- routes:** Định nghĩa các tuyến đường API và ánh xạ chúng tới các controller.
- schema:** Chứa các tệp dùng Zod để xác thực và kiểm tra tính hợp lệ của dữ liệu đầu vào trước khi xử lý.
- services:** Chứa các tệp thực thi logic nghiệp vụ và các chức năng hỗ trợ.
- socket:** Xử lý các kết nối WebSocket, lắng nghe và phản hồi các sự kiện, bao gồm các chức năng như phát nhạc, gửi tin nhắn và tham gia phòng.
- utils:** Chứa các hàm tiện ích và công cụ dùng chung trong toàn bộ ứng dụng.
- index.ts:** Nơi khởi tạo và cấu hình cho ứng dụng.
- uploads:** Chứa các file tĩnh.

Hình 9. Cấu trúc thư mục phía API

Tập tin index.ts: Chịu trách nhiệm xử lý giao tiếp thời gian thực giữa client và server trong hệ thống nghe nhạc trực tuyến. Nó khởi tạo một máy chủ Socket.IO và thiết lập middleware để xác thực người dùng thông qua token JWT. Mỗi khi client kết nối, token sẽ được kiểm tra và giải mã để xác định người dùng. Sau khi xác thực thành công, server lắng nghe và xử lý các sự kiện như tham gia phòng, gửi tin nhắn, và phát bài hát. Bằng cách sử dụng EventEmitter, các sự kiện phát bài hát được đồng bộ để đảm bảo mọi người trong cùng phòng nghe cùng một bài. Việc tách biệt các handler như chat, phòng, và bài hát giúp tổ chức mã nguồn rõ ràng và dễ bảo trì, đồng thời tối ưu hóa trải nghiệm người dùng với khả năng kết nối nhanh và đồng bộ dữ liệu hiệu quả.



```
1 const PORT = process.env.PORT || 8000;
2 const url = process.env.URL_FRONTEND || "http://localhost:3000";
3
4 const app = express();
5 const server = http.createServer(app);
6 const io = new Server(server, { ...socketConfig }); // Tạo socket server
7
8 // Đăng ký các folder chứa file tĩnh
9 app.use("/audio", express.static(path.join(__dirname, "../uploads/audio")));
10 app.use("/image", express.static(path.join(__dirname, "../uploads/images")));
11 app.use("/lyric", express.static(path.join(__dirname, "../uploads/lyrics")));
12
13 app.use(
14   cors({
15     origin: url,
16     credentials: true,
17   })
18 );
19
20 app.use(compression());
21 app.use(bodyParser.json());
22 app.use(bodyParser.urlencoded({ extended: true }));
23
24 app.use(express.static("public"));
25
26 app.use(globalAuthorize); // Middleware xử lý token
27 app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(swaggerDocument));
28 app.use("/api/", router());
29 app.get("/", (req, res) => {
30   res.send("Hello");
31 });
32
33 // Middleware xử lý lỗi
34 app.use(errorMiddleware);
35
36 // Middleware xử lý form data
37 app.use(express.urlencoded({ extended: true }));
38
39 socketHandler(io);
40
41 server.listen(PORT, () => console.log(`✓ Server is running on port: ${PORT}`));
42
```

Hình 10. Tập tin index.ts

Thư mục routes: các tệp trong thư mục routes được sử dụng để định nghĩa các điểm cuối (endpoints) của API. Mỗi tệp tương ứng với một tài nguyên và một chức năng cụ thể, như người dùng, nghệ sĩ, bài hát, hoặc danh sách phát. Các tệp sẽ xử lý các yêu cầu HTTP như GET, POST, PUT, DELETE và chuyển chúng đến các controller để xử lý logic nghiệp vụ. Các routes được kết nối vào ứng dụng Express thông qua một tệp gốc (ví dụ: index.ts), nơi tất cả các routes được gắn vào URL gốc của API, giúp tổ chức các điểm cuối API theo cấu trúc rõ ràng, dễ bảo trì và mở rộng. Việc tách riêng các routes theo chức năng cũng tăng cường khả năng quản lý và phát triển hệ thống.

```
● ● ●
1 import express from "express";
2 const router = express.Router();
3
4 import ArtistRouter from "./artists.routes";
5 import AuthRouter from "./auth.routes";
6 import GenreRouter from "./genre.routes";
7 import MoodRouter from "./mood.routes";
8 import PermissionsRouter from "./permissions.routes";
9 import PlaylistRouter from "./playlist.routes";
10 import RoleRouter from "./role.routes";
11 import RoomRouter from "./room.routes";
12 import RoomChatRouter from "./room_chat.routes";
13 import SongRouter from "./song.routes";
14 import SongPlayRouter from "./song_play.routes";
15 import UploadRouter from "./upload.routes";
16 import UserRouter from "./user.routes";
17
18 export default (): express.Router => {
19   router.use("/auth", AuthRouter);
20   router.use("/user", UserRouter);
21   router.use("/song", SongRouter);
22   router.use("/genre", GenreRouter);
23   router.use("/playlist", PlaylistRouter);
24   router.use("/role", RoleRouter);
25   router.use("/permissions", PermissionsRouter);
26   router.use("/artist", ArtistRouter);
27   router.use("/song_play", SongPlayRouter);
28   router.use("/mood", MoodRouter);
29   router.use("/room", RoomRouter);
30   router.use("/room-chat", RoomChatRouter);
31   router.use("/upload", UploadRouter);
32   return router;
33 }
```

Hình 11. Tập tin routes/index.ts

Thư mục schema: được tổ chức các tệp Zod để thực hiện việc xác thực dữ liệu (validation) trước khi nhận dữ liệu từ client. Zod là một thư viện để tạo các schema xác thực trong JavaScript/TypeScript, giúp đảm bảo rằng dữ liệu đầu vào đáp ứng các quy tắc định trước. Các tệp trong thư mục schema có thể được phân loại theo các tài nguyên và chức năng trong hệ thống, ví dụ như user.schema.ts, song.schema.ts, playlist.schema.ts. Mỗi tệp sẽ chứa một schema Zod để kiểm tra tính hợp lệ của dữ liệu nhập vào. Cấu trúc này giúp đảm bảo rằng dữ liệu đầu vào từ client sẽ được xác thực một cách chính xác, tránh các lỗi không mong muốn và giảm thiểu nguy cơ bảo mật.

```
1 import { object, string, TypeOf } from "zod";
2 import { SIZE } from "../utils/contants";
3
4 const payload = {
5   body: object({
6     title: string({
7       required_error: "Title is required",
8     }).max(SIZE.TITLE),
9     description: string().max(SIZE.DESCRIPTION).optional(),
10    color: string().optional(),
11  }),
12};
13
14 const payloadUpdate = {
15   body: object({
16     title: string().max(SIZE.TITLE).optional(),
17     description: string().max(SIZE.DESCRIPTION).optional(),
18     color: string().optional(),
19   }),
20};
21
22 const params = {
23   params: object({
24     id: string({
25       required_error: "Id is required",
26     })
27       .min(1)
28       .max(SIZE.UUID, `Id must be less than ${SIZE.UUID} characters`),
29   }),
30};
31
32 export const getGenreSchema = object({ ...params });
33 export const createGenreSchema = object({ ...payload });
34 export const updateGenreSchema = object({ ...payloadUpdate, ...params });
35 export const deleteGenreSchema = object({ ...params });
36
37 export type GetGenreInput = TypeOf<typeof getGenreSchema>;
38 export type CreateGenreInput = TypeOf<typeof createGenreSchema>;
39 export type UpdateGenreInput = TypeOf<typeof updateGenreSchema>;
40 export type DeleteGenreInput = TypeOf<typeof deleteGenreSchema>;
41
```

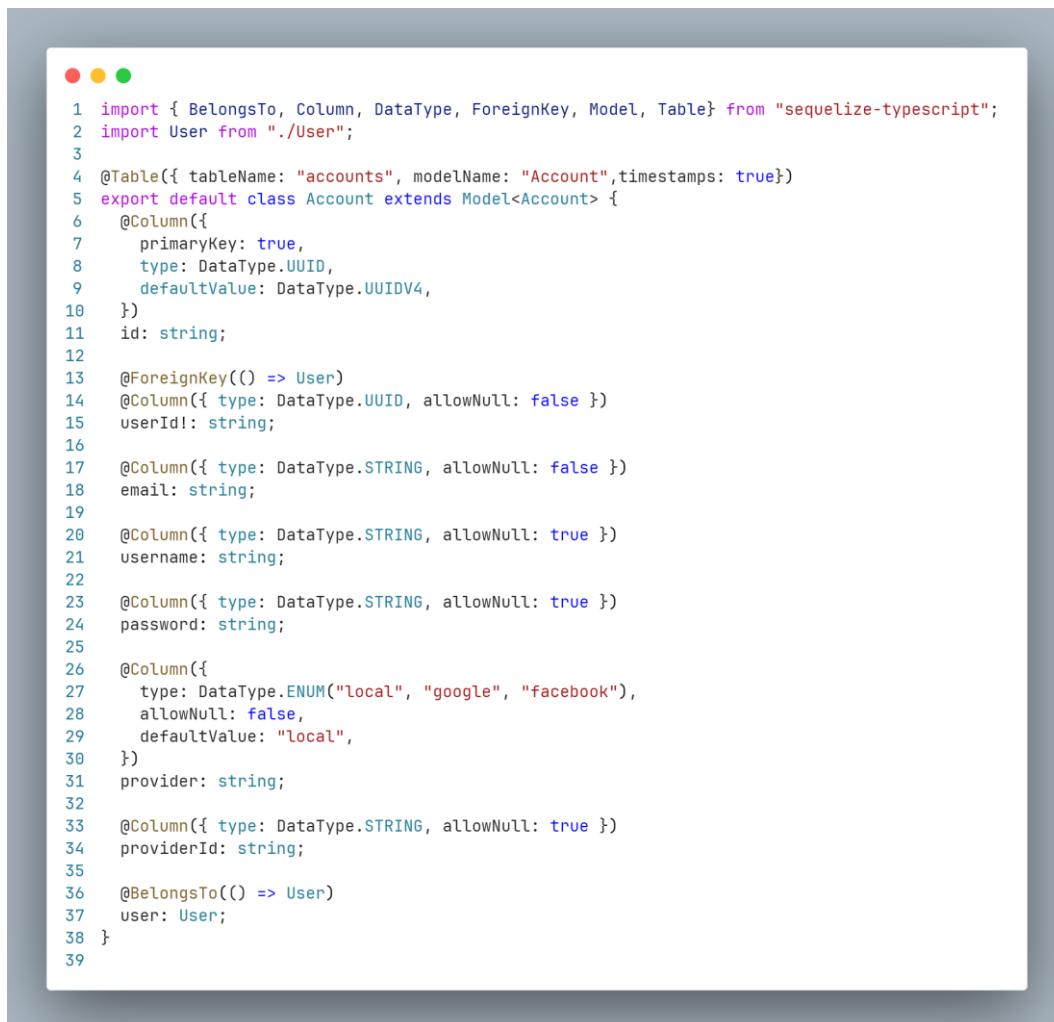
Hình 12. Tập tin schema/genre.schema.ts

Thư mục controllers: được sử dụng để chứa các tệp xử lý logic và xử lý các yêu cầu từ client. Các controller sẽ nhận các yêu cầu từ các route, thực hiện các thao tác cần thiết (như xử lý dữ liệu, gọi các service, tương tác với cơ sở dữ liệu) và trả lại phản hồi cho client. Thư mục controllers chứa các tệp xử lý các yêu cầu liên quan đến các chức năng như quản lý người dùng, bài hát, danh sách phát, phòng nghe nhạc và các tương tác trong ứng dụng. Mỗi controller có các phương thức tương ứng với các yêu cầu HTTP, như get, post, put, và delete, giúp xử lý các thao tác như tạo mới, cập nhật, xóa và lấy thông tin.

```
1 import { OAuth2Client } from "google-auth-library";
2 const client = new OAuth2Client(process.env.GG_CLIENT_ID);
3
4 export async function login(req: Request, res: Response, next: NextFunction) {...}
5 export async function register(req: Request, res: Response, next: NextFunction) {...}
6 export async function validate(req: Request, res: Response, next: NextFunction) {...}
7 export async function refreshToken(req: Request, res: Response, next: NextFunction) {...}
8 export async function loginGoogle(req: Request, res: Response, next: NextFunction) {...}
9 export async function forgotPassword(req: Request, res: Response, next: NextFunction) {...}
10 export async function changePassword(req: Request, res: Response, next: NextFunction) {...}
11 export async function resetPassword(req: Request, res: Response, next: NextFunction) {...}
12
```

Hình 13. controllers/auth.controller.ts

Thư mục models: Được sử dụng để định nghĩa các mô hình dữ liệu và tương tác với cơ sở dữ liệu. Các mô hình này đóng vai trò quan trọng trong việc quản lý và lưu trữ dữ liệu cho ứng dụng. Mỗi mô hình tương ứng với một bảng trong cơ sở dữ liệu. Trong dự án models chứa các tệp mô hình dữ liệu cho các đối tượng như người dùng, bài hát, danh sách phát, phòng nghe nhạc,... Những mô hình này sẽ định nghĩa cấu trúc của dữ liệu, các mối quan hệ giữa chúng, và các phương thức để tương tác với cơ sở dữ liệu.



```
● ● ●
1 import { BelongsTo, Column, DataType, ForeignKey, Model, Table} from "sequelize-typescript";
2 import User from "./User";
3
4 @Table({ tableName: "accounts", modelName: "Account", timestamps: true})
5 export default class Account extends Model<Account> {
6   @Column({
7     primaryKey: true,
8     type: DataType.UUID,
9     defaultValue: DataType.UUIDV4,
10   })
11   id: string;
12
13   @ForeignKey(() => User)
14   @Column({ type: DataType.UUID, allowNull: false })
15   userId!: string;
16
17   @Column({ type: DataType.STRING, allowNull: false })
18   email: string;
19
20   @Column({ type: DataType.STRING, allowNull: true })
21   username: string;
22
23   @Column({ type: DataType.STRING, allowNull: true })
24   password: string;
25
26   @Column({
27     type: DataType.ENUM("local", "google", "facebook"),
28     allowNull: false,
29     defaultValue: "local",
30   })
31   provider: string;
32
33   @Column({ type: DataType.STRING, allowNull: true })
34   providerId: string;
35
36   @BelongsTo(() => User)
37   user: User;
38 }
39
```

Hình 14. Tập tin models/Account.ts

3.5.3 Thiết kế Giao diện

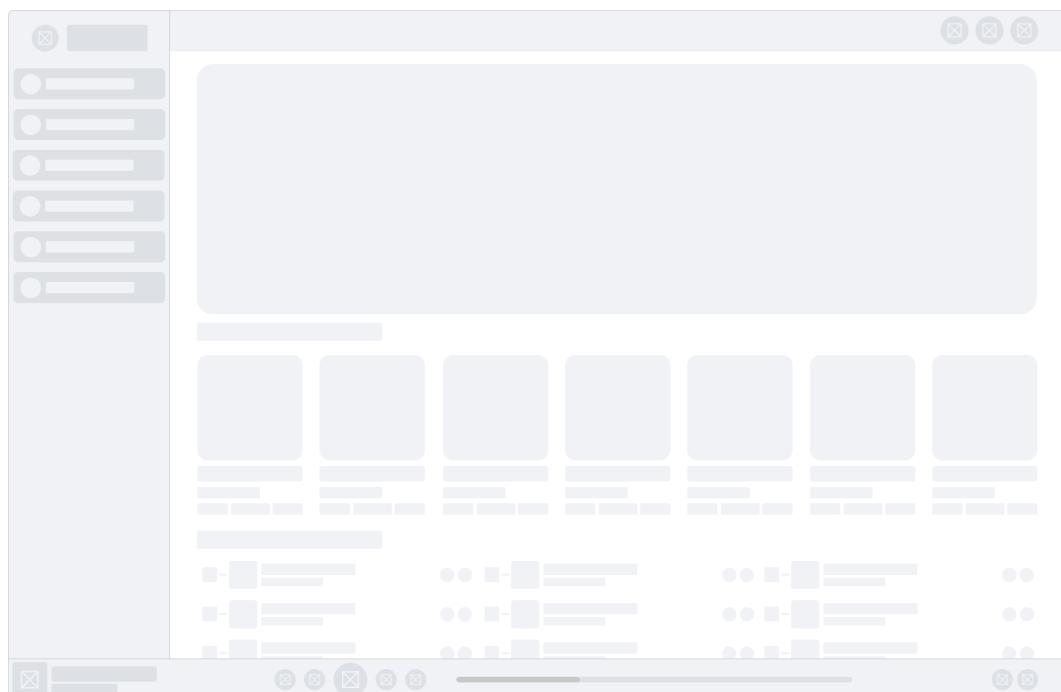
Giao diện của hệ thống được thiết kế với thanh điều hướng chính nằm ở bên trái của màn hình, giúp người dùng dễ dàng truy cập các tính năng như trang chủ, danh sách phát, tìm kiếm bài hát, và quản lý tài khoản.

Phần nội dung chính nằm ở phía bên phải, hiển thị thông tin động theo từng lựa chọn của người dùng. Khu vực này bao gồm danh sách bài hát, thông tin nghệ sĩ, chi tiết danh sách phát.

Giao diện trang chủ:

Hiển thị các danh mục nổi bật như bài hát phổ biến, danh sách phát được đề xuất, hoặc nghệ sĩ được yêu thích. Các thẻ bài hát hoặc danh sách phát được trình bày dưới dạng lưới hoặc danh sách có hình ảnh bìa, tên bài hát hoặc danh sách phát, và tên nghệ sĩ, tạo cảm giác sinh động và dễ duyệt nội dung.

Phía dưới cùng của trang là trình phát nhạc với các nút điều khiển cơ bản như phát/dừng, bỏ qua bài hát, và thanh tiến trình để theo dõi trạng thái phát nhạc hiện tại. Trình phát này luôn hiển thị để người dùng điều chỉnh nhạc mà không cần rời khỏi trang.



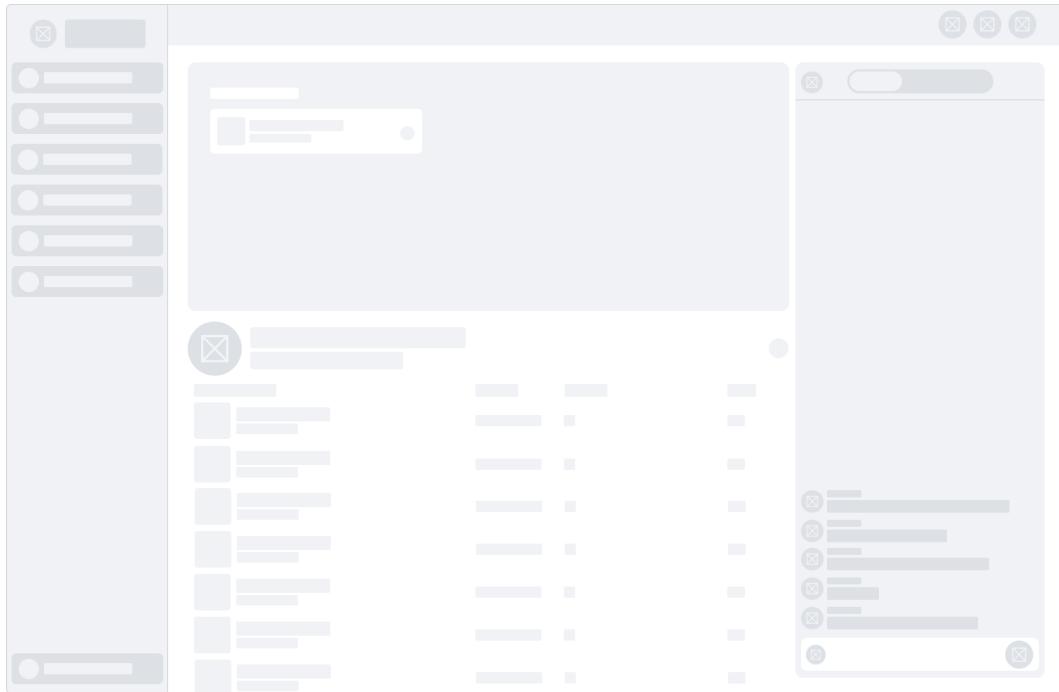
Hình 15. Low-fi prototypes của trang chủ

Giao diện phòng nghe nhạc:

Phần trung tâm của giao diện hiển thị thông tin về bài hát đang phát, bao gồm tên bài hát, nghệ sĩ, và hình ảnh bìa. Bên dưới là các nút điều khiển phát nhạc như phát/dừng, chuyển bài, và điều chỉnh âm lượng.

Phía bên phải hoặc dưới cùng, tùy thuộc vào thiết bị, là khung chat thời gian thực, nơi người dùng trong cùng phòng có thể giao tiếp với nhau. Tin nhắn hiển thị theo thứ tự thời gian, cùng với tên và hình đại diện của người gửi, tạo cảm giác kết nối gần gũi giữa các thành viên. Khung nhập tin nhắn luôn sẵn sàng để người dùng dễ dàng gửi bình luận hoặc phản hồi. Ngoài ra, giao diện cũng cung cấp danh sách

những người dùng đang có mặt trong phòng, cho phép người tham gia biết ai đang cùng nghe nhạc.



Hình 16. Low-fidelity prototypes của trang phòng nghe nhạc

3.6 Triển khai

Để triển khai ứng dụng với Docker lên VPS giúp đơn giản hóa quá trình quản lý môi trường mà còn giúp tăng cường khả năng mở rộng và tính linh hoạt của hệ thống. Sử dụng Docker cho cả frontend và backend giúp dễ dàng tạo ra các container độc lập cho từng phần của ứng dụng, bảo đảm rằng mỗi dịch vụ chạy trong một môi trường sạch và không bị ảnh hưởng bởi các dịch vụ khác. Điều này giúp cải thiện hiệu suất, dễ dàng mở rộng các dịch vụ khi cần và giảm thiểu lỗi liên quan đến cấu hình môi trường.

Docker Compose cho phép định nghĩa và chạy các ứng dụng đa dịch vụ chỉ bằng một lệnh, giảm thiểu việc phải cấu hình riêng lẻ cho từng dịch vụ. Điều này đặc biệt hữu ích trong các dự án có nhiều phần mềm hoặc microservices. Ngoài ra, có thể dễ dàng cập nhật, kiểm tra và duy trì các container mà không ảnh hưởng đến toàn bộ hệ thống, đảm bảo khả năng hoạt động liên tục của ứng dụng trong môi trường sản xuất. Các cấu hình được triển khai như sau:

- Backend: Tạo một Dockerfile cho backend. Trong Dockerfile, sẽ cài đặt các phụ thuộc của ứng dụng backend và cấu hình để ứng dụng có thể chạy trong môi trường container.

```
FROM node:16
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
EXPOSE 8000
CMD ["npm", "start"]
```

- **Frontend:** Dockerfile này sẽ giúp build và chạy ứng dụng Next.js trong container.

```
FROM node:16
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
RUN npm run build
EXPOSE 3000
CMD ["npm", "start"]
```

- **Docker Compose:** Sử dụng Docker Compose để định nghĩa các dịch vụ (services) cho frontend và backend. Trong file docker-compose.yml, sẽ cấu hình hai dịch vụ này, cùng với các cổng và các môi trường cần thiết.

```
version: "3.7"
services:
  frontend:
    build:
      context: ./frontend
    ports:
      - "3000:3000"
    depends_on:
      - backend
  backend:
    build:
      context: ./backend
    ports:
      - "8000:8000"
    environment:
      - DB_NAME=name
      - DB_PASS=123
```

- **Reverse Proxy (Nginx):** Để truy cập ứng dụng qua HTTP, phải sử dụng Nginx làm reverse proxy. Nginx sẽ chuyển tiếp các yêu cầu HTTP từ cổng 80 (hoặc 443 cho HTTPS) tới cổng 3000 (frontend) và cổng 8000 (backend).

```
server {
  listen 80;
  server_name vibely.vn;

  location / {
    proxy_pass http://frontend:3000;
    proxy_set_header Host $host;
```

```
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}

location /api {
    proxy_pass http://backend:5000;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}
}
```

Việc triển khai backend và frontend trên Docker giúp giảm thiểu các vấn đề liên quan đến môi trường, bảo mật, và dễ dàng mở rộng ứng dụng khi cần thiết. Docker cũng cho phép quản lý và triển khai ứng dụng một cách tự động và hiệu quả.

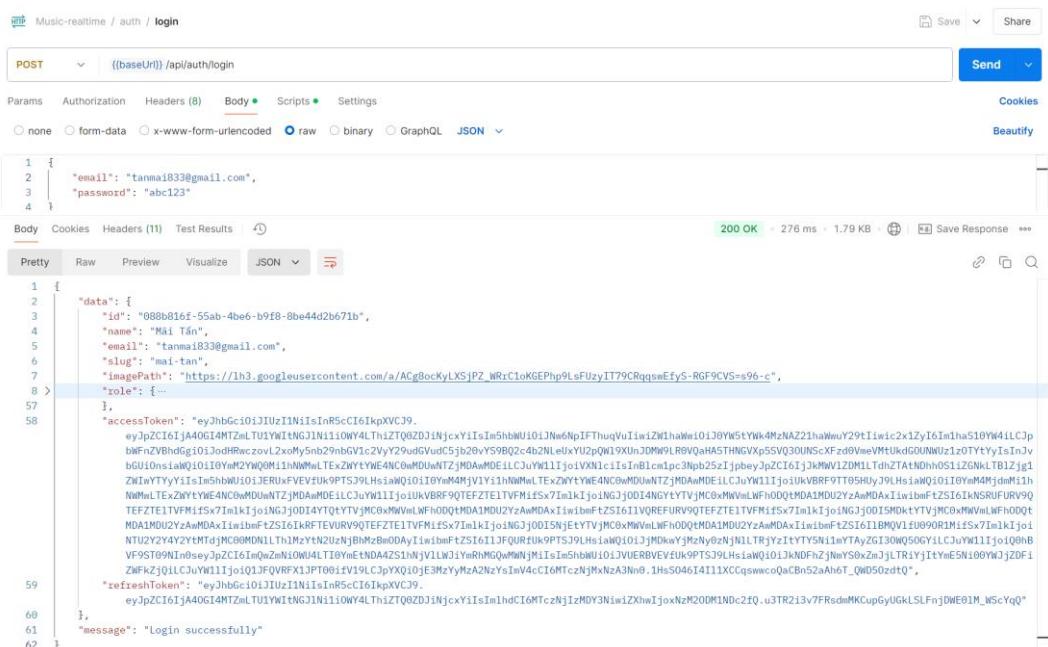
CHƯƠNG 4 KẾT QUẢ NGHIÊN CỨU

4.1 Kiểm thử API với Postman

Hệ thống API trong ứng dụng được thiết kế để cung cấp các chức năng quan trọng như xác thực người dùng, quản lý bài hát, danh sách phát, và phân quyền. Các API này cho phép giao tiếp hiệu quả giữa frontend và backend, đảm bảo trải nghiệm người dùng liền mạch và bảo mật. Trong phần này, tôi tập trung giới thiệu một số API tiêu biểu nhằm minh họa cách hệ thống hoạt động. Những API này không chỉ đảm bảo hiệu năng mà còn tối ưu hóa khả năng mở rộng và bảo trì hệ thống.

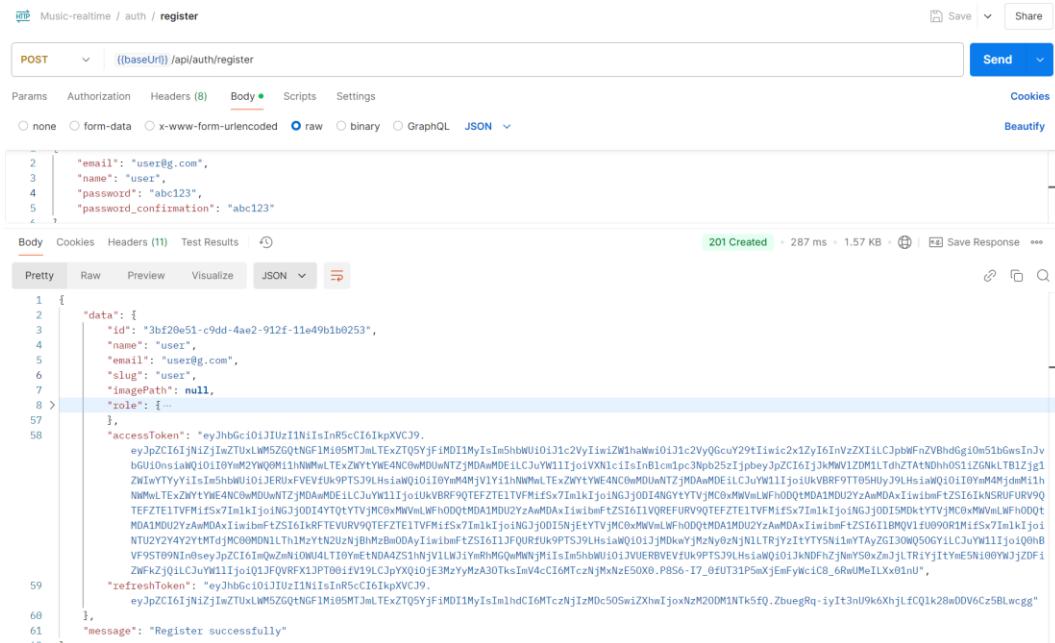
4.1.1 API Xác thực người dùng

Sử dụng các API để đăng ký, đăng nhập và quản lý phiên làm việc thông qua token. Cụ thể, API đăng ký cho phép người dùng mới tạo tài khoản với thông tin hợp lệ, trong khi API đăng nhập xác thực thông tin và trả về token truy cập. Các API làm mới token (refresh token) giúp gia hạn phiên mà không cần đăng nhập lại. Tất cả đều tuân thủ các tiêu chuẩn bảo mật để ngăn ngừa rủi ro truy cập trái phép.

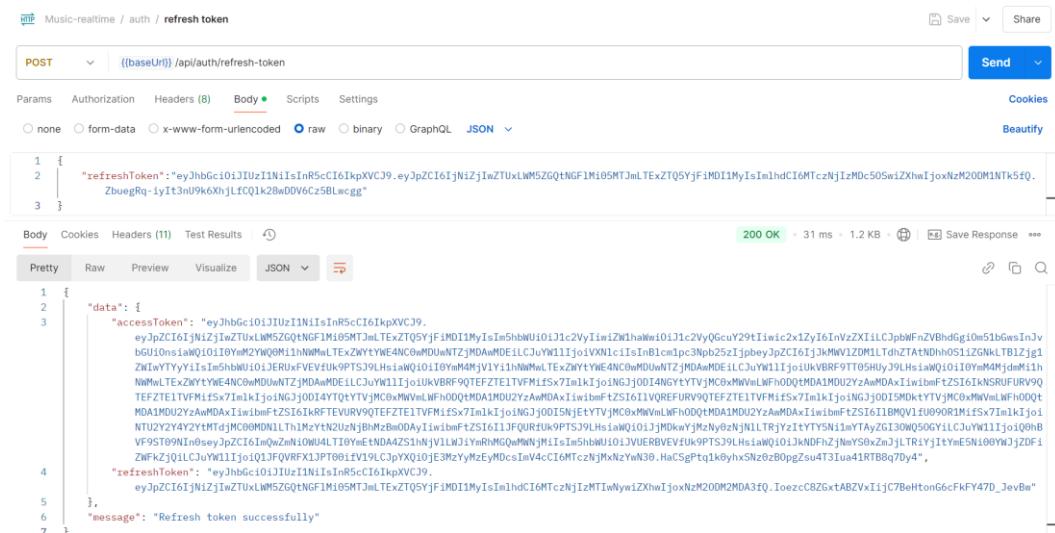


Hình 17. Kiểm thử API đăng nhập

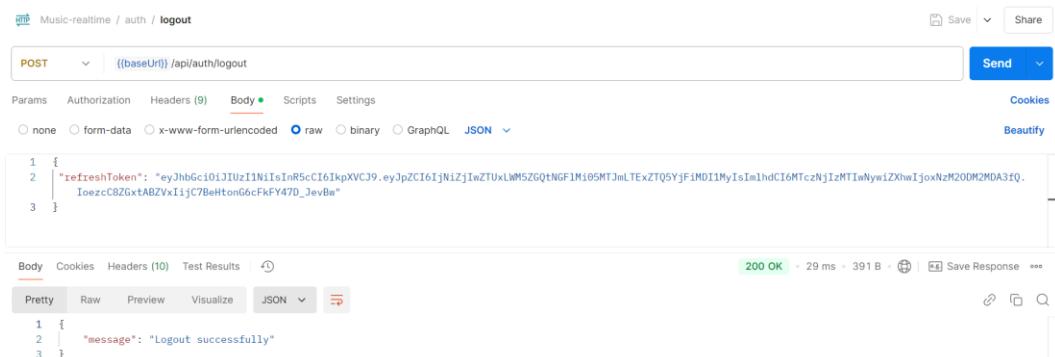
Xây dựng website nghe nhạc trực tuyến theo thời gian thực bằng NextJs



Hình 18. Kiểm thử API đăng ký



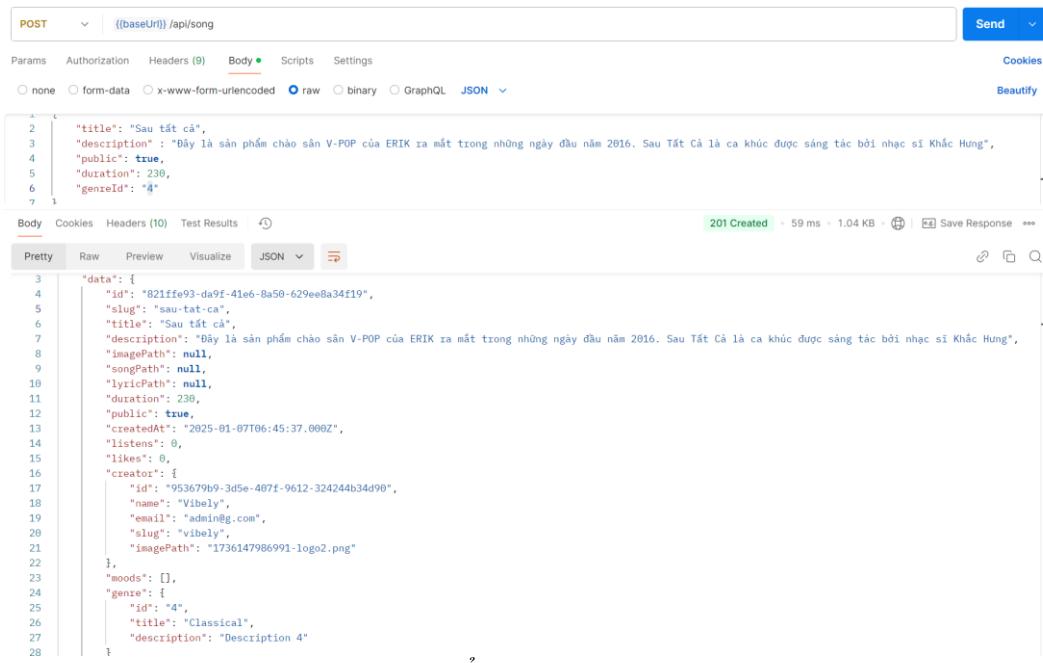
Hình 19. Kiểm thử API làm mới token



Hình 20. Kiểm thử API đăng xuất

4.1.2 API liên quan đến bài hát

Cho phép người dùng truy vấn danh sách bài hát, tìm kiếm bài hát theo tên hoặc nghệ sĩ, và truy cập thông tin chi tiết về từng bài hát như tên, thời lượng, và thể loại. Ngoài ra, API hỗ trợ thêm bài hát vào danh sách phát cá nhân, yêu thích hoặc tạo các phòng nghe nhạc theo thời gian thực. Các hành động như phát nhạc, và điều khiển bài hát cũng được thực hiện thông qua các API này để mang lại trải nghiệm nghe nhạc liền mạch và cá nhân hóa cho người dùng.



```

POST {{baseUrl}}/api/song
{
  "title": "Sau tất cả",
  "description": "Đây là sản phẩm chào sân V-POP của ERIK ra mắt trong những ngày đầu năm 2016. Sau Tất Cả là ca khúc được sáng tác bởi nhạc sĩ Khắc Hưng",
  "public": true,
  "duration": 230,
  "genreId": "4"
}
  
```

Body Cookies Headers (10) Test Results

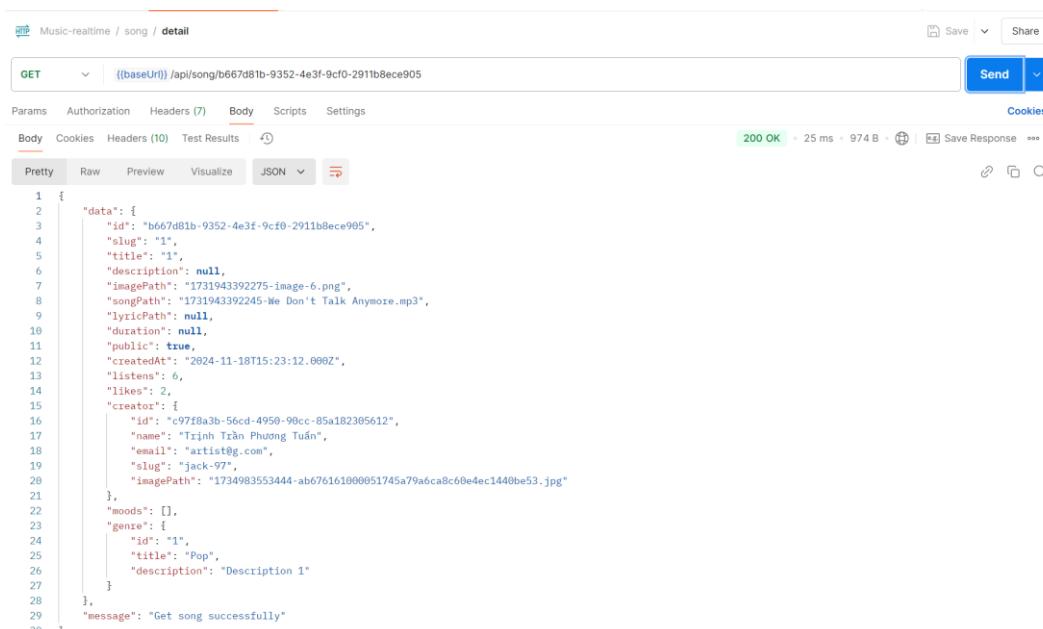
Pretty Raw Preview Visualize JSON

201 Created - 59 ms - 1.04 KB | Save Response

```

{
  "data": {
    "id": "821ffe93-da9f-41e6-8a50-629ee8a34f19",
    "slug": "sau-tat-ca",
    "title": "Sau tất cả",
    "description": "Đây là sản phẩm chào sân V-POP của ERIK ra mắt trong những ngày đầu năm 2016. Sau Tất Cả là ca khúc được sáng tác bởi nhạc sĩ Khắc Hưng",
    "imagePath": null,
    "songPath": null,
    "lyricPath": null,
    "duration": 230,
    "public": true,
    "createdAt": "2025-01-07T06:45:37.000Z",
    "listens": 0,
    "likes": 0,
    "creator": {
      "id": "953679b9-3d5e-407f-9612-324244b34d90",
      "name": "Vibely",
      "email": "admin@vibely.com",
      "slug": "vibely",
      "imagePath": "1736147986991-logo2.png"
    },
    "moods": [],
    "genre": {
      "id": "4",
      "title": "Classical",
      "description": "Description 4"
    }
  }
}
  
```

Hình 21. Kiểm thử API tạo bài hát



```

GET {{baseUrl}}/api/song/b667d81b-9352-4e3f-9cf0-2911b8ece905
  
```

Params Authorization Headers (7) Body Scripts Settings

Body Cookies Headers (10) Test Results

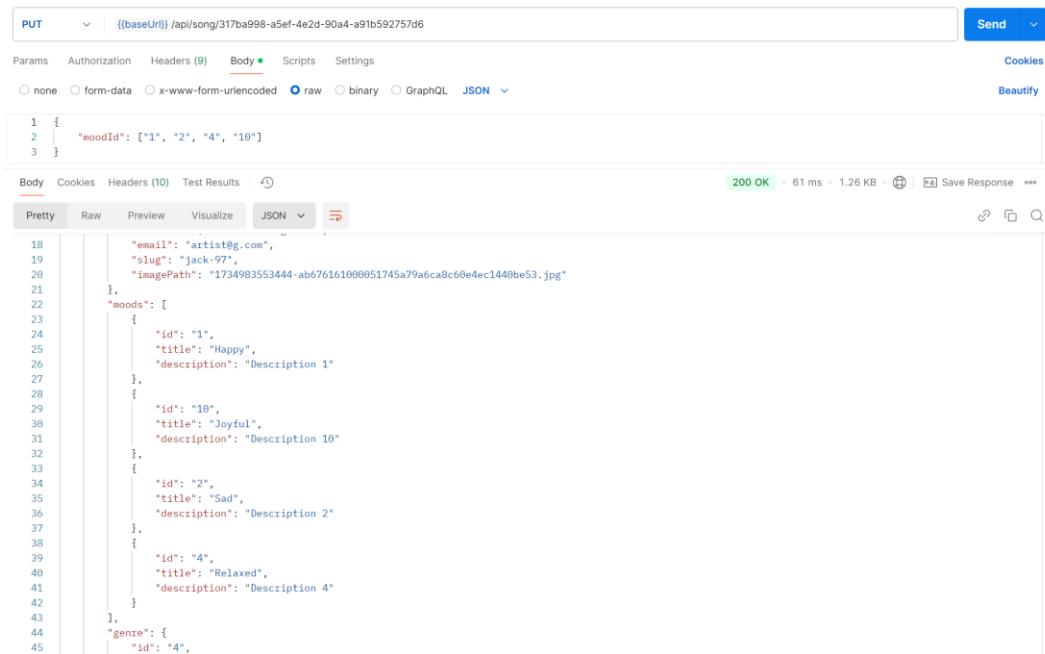
200 OK - 25 ms - 974 B | Save Response

```

{
  "data": {
    "id": "b667d81b-9352-4e3f-9cf0-2911b8ece995",
    "slug": "1",
    "title": "1",
    "description": null,
    "imagePath": "1731943392275-image-6.png",
    "songPath": "1731943392245-We Don't Talk Anymore.mp3",
    "lyricPath": null,
    "duration": null,
    "public": true,
    "createdAt": "2024-11-18T15:23:12.000Z",
    "listens": 6,
    "likes": 2,
    "creator": {
      "id": "c97f8a3b-56cd-4950-90cc-85a182305612",
      "name": "Trịnh Trần Phương Tuấn",
      "email": "artiststg.com",
      "slug": "jack-97",
      "imagePath": "1734983553444-ab676161000051745a79a6ca8c60e4ec1440be53.jpg"
    },
    "moods": [],
    "genre": {
      "id": "1",
      "title": "Pop",
      "description": "Description 1"
    }
  },
  "message": "Get song successfully"
}
  
```

Hình 22. Kiểm thử API lấy thông tin bài hát

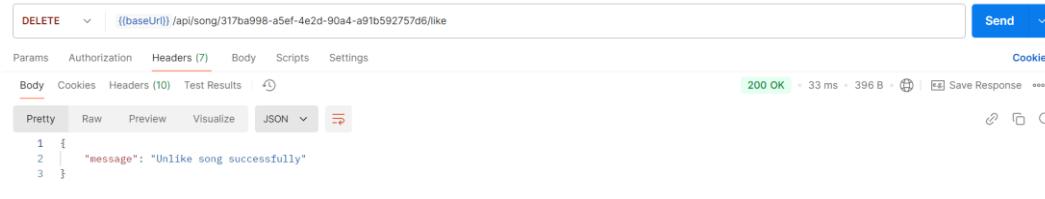
Xây dựng website nghe nhạc trực tuyến theo thời gian thực bằng NextJs



Hình 23. Kiểm thử API cập nhật bài hát



Hình 24. Kiểm thử API yêu thích bài hát

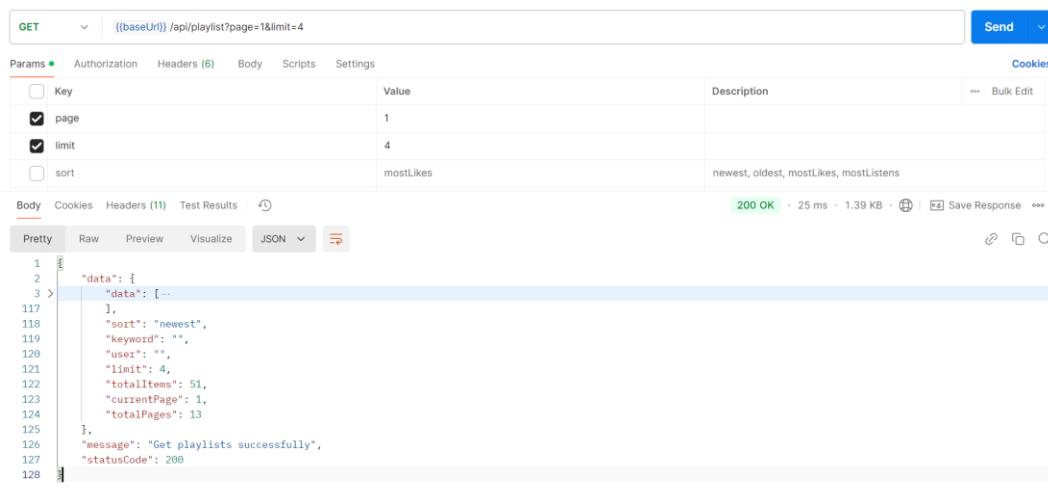


Hình 25. Kiểm thử API bỏ yêu thích bài hát

4.1.3 API liên quan đến danh sách phát

Cho phép người dùng tạo, quản lý và tương tác với danh sách phát cá nhân. Các chức năng chính bao gồm tạo danh sách phát mới, thêm hoặc xóa bài hát khỏi danh sách, chỉnh sửa thông tin như tên hoặc mô tả, và xóa danh sách phát. Người dùng có thể truy vấn danh sách các playlist đã tạo hoặc được gợi ý, xem thông tin chi tiết của một danh sách phát, và phát nhạc trực tiếp từ đó. API cũng hỗ trợ tính năng theo dõi danh sách phát của người dùng khác để nâng cao trải nghiệm cộng đồng âm nhạc.

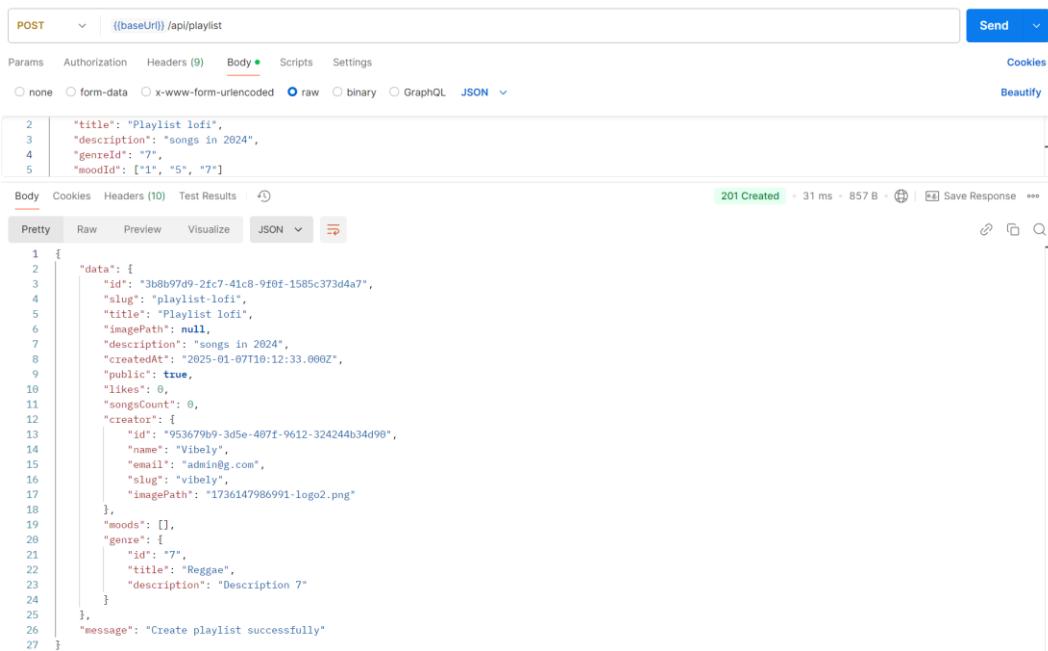
Xây dựng website nghe nhạc trực tuyến theo thời gian thực bằng NextJs



```
GET {{baseUrl}} /api/playlist?page=1&limit=4

Params Authorization Headers (6) Body Scripts Settings Cookies
Key page Value 1 Description
Key limit Value 4 Description
Key sort Value mostLikes Description newest, oldest, mostLikes, mostListens
Body Cookies Headers (11) Test Results
Pretty Raw Preview Visualize JSON ⚡
1
2   "data": [
3     "data": [...]
4   ],
5   "sort": "newest",
6   "keyword": "",
7   "user": "",
8   "limit": 4,
9   "totalItems": 51,
10  "currentPage": 1,
11  "totalPages": 13
12 },
13 "message": "Get playlists successfully",
14 "statusCode": 200
15
16
17
18
19
20
21
22
23
24
25
26
27
28
```

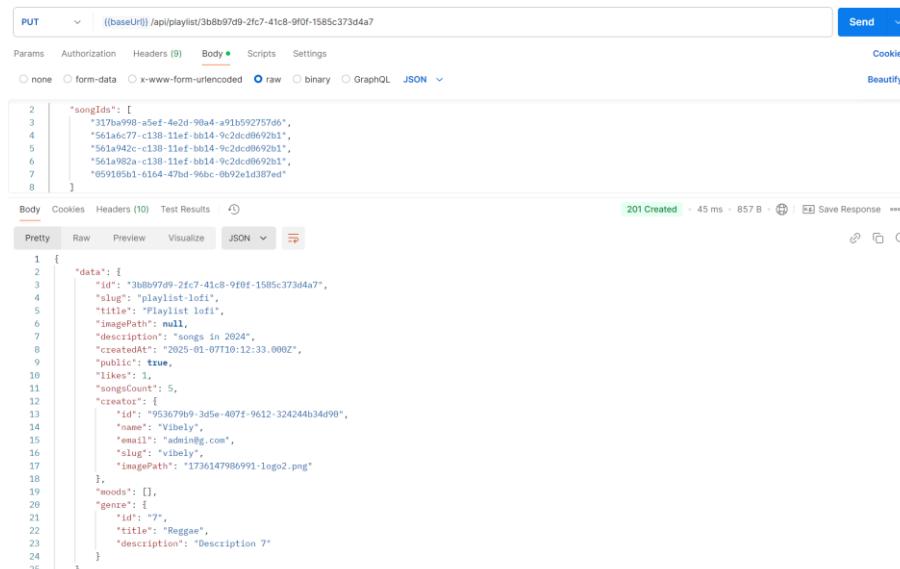
Hình 26. Kiểm thử API lấy toàn bộ danh sách phát



```
POST {{baseUrl}} /api/playlist

Params Authorization Headers (9) Body Scripts Settings Cookies
Body x-www-form-urlencoded raw binary GraphQL JSON
None form-data
Body Cookies Headers (10) Test Results
Pretty Raw Preview Visualize JSON ⚡
1
2   "data": {
3     "id": "3b8b97d9-2fc7-41c8-9f0f-1585c373d4a7",
4     "slug": "playlist-lofi",
5     "title": "Playlist lofi",
6     "imagePath": null,
7     "description": "songs in 2024",
8     "createdAt": "2025-01-07T10:12:33.000Z",
9     "public": true,
10    "likes": 0,
11    "songsCount": 0,
12    "creator": {
13      "id": "953679b9-3d5e-407f-9612-324244b34d90",
14      "name": "Vibely",
15      "email": "admin@vibely.com",
16      "slug": "vibely",
17      "imagePath": "1736147986991-logo2.png"
18    },
19    "moods": [],
20    "genre": [
21      {
22        "id": "7",
23        "title": "Reggae",
24        "description": "Description 7"
25      }
26    ],
27  },
28  "message": "Create playlist successfully"
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
```

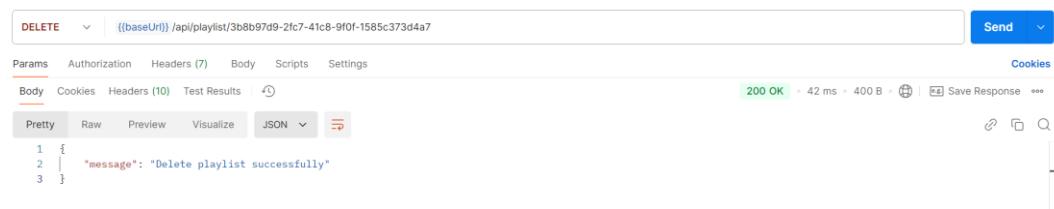
Hình 27. Kiểm thử API tạo danh sách phát



```
PUT {{baseUrl}} /api/playlist/3b8b97d9-2fc7-41c8-9f0f-1585c373d4a7

Params Authorization Headers (9) Body Scripts Settings Cookies
Body x-www-form-urlencoded raw binary GraphQL JSON
None form-data
Body Cookies Headers (10) Test Results
Pretty Raw Preview Visualize JSON ⚡
1
2   "data": [
3     "songIds": [
4       "317ba998-a5ef-4e4d-98a4-a91592757d6",
5       "561a6c77-c138-11ef-bb14-9c2dc06962b1",
6       "561a942c-c138-11ef-bb14-9c2dc06962b1",
7       "561a982a-c138-11ef-bb14-9c2dc06962b1",
8       "0591b5b1-6164-47bd-9bdc-092e1d387ed"
9     ]
10  ],
11  "data": {
12    "id": "3b8b97d9-2fc7-41c8-9f0f-1585c373d4a7",
13    "slug": "playlist-lofi",
14    "title": "Playlist lofi",
15    "imagePath": null,
16    "description": "songs in 2024",
17    "createdAt": "2025-01-07T10:12:33.000Z",
18    "public": true,
19    "likes": 1,
20    "songsCount": 5,
21    "creator": {
22      "id": "953679b9-3d5e-407f-9612-324244b34d90",
23      "name": "Vibely",
24      "email": "admin@vibely.com",
25      "slug": "vibely",
26      "imagePath": "1736147986991-logo2.png"
27    },
28    "moods": [],
29    "genre": [
30      {
31        "id": "7",
32        "title": "Reggae",
33        "description": "Description 7"
34      }
35    ],
36  },
37  "message": "Update playlist successfully"
38
39
40
41
42
43
44
45
46
47
48
49
49
```

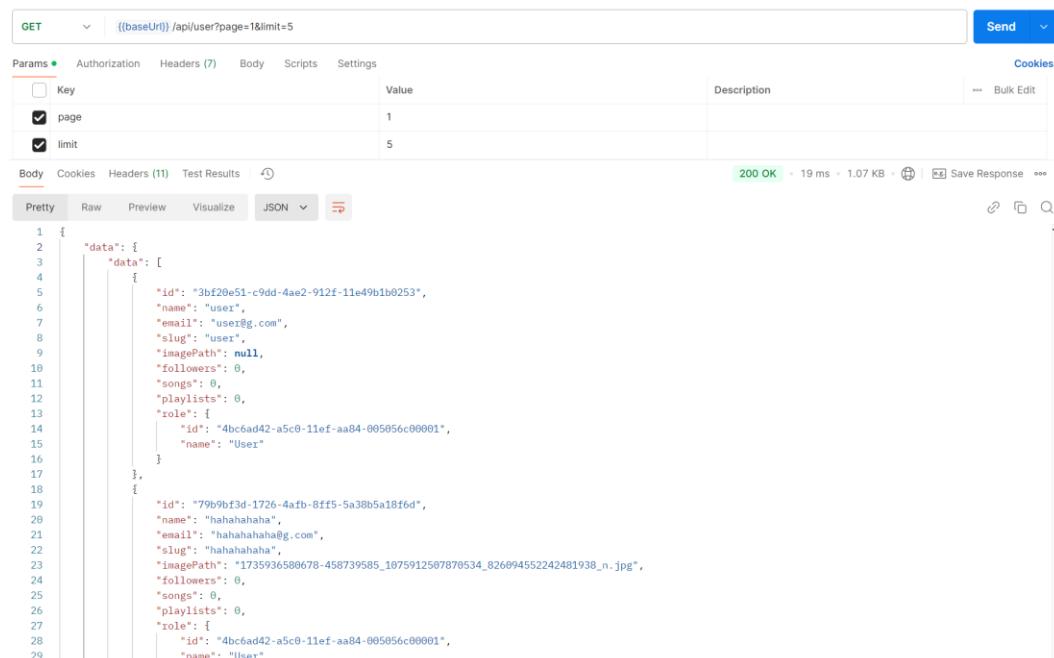
Hình 28. Kiểm thử API cập nhật danh sách phát



Hình 29. Kiểm thử API xóa danh sách phát

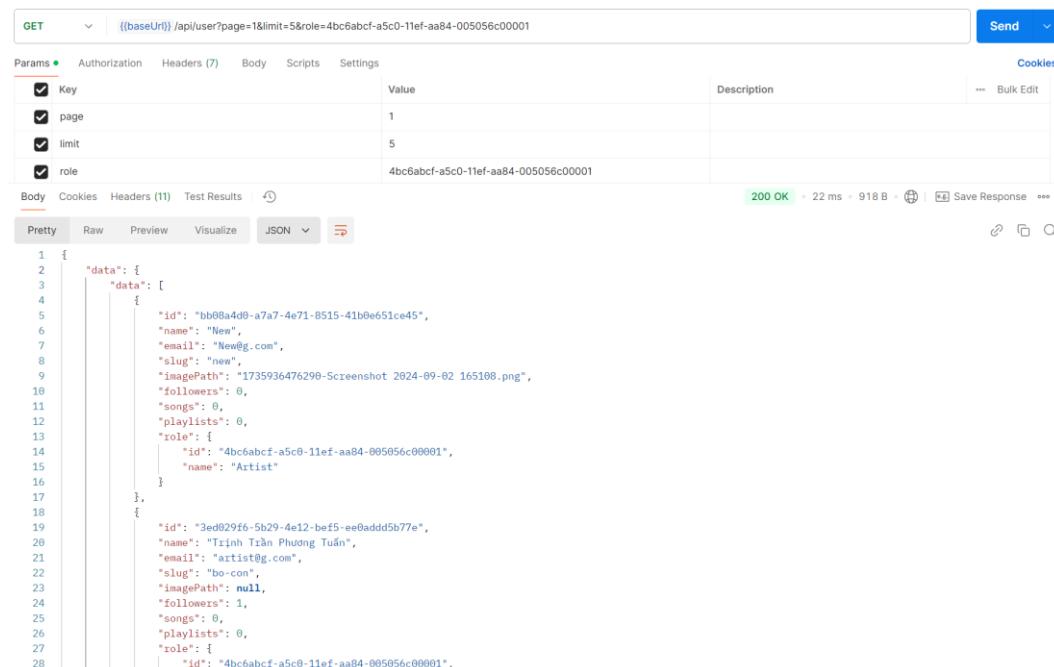
4.1.4 API liên quan đến người dùng

Cho phép thực hiện các chức năng quan trọng. Người dùng có thể cập nhật thông tin cá nhân như tên, email, ảnh đại diện, và mật khẩu. Ngoài ra, API hỗ trợ quản lý danh sách bài hát yêu thích, danh sách phát cá nhân, và theo dõi các nghệ sĩ hoặc người dùng khác. Các API quản trị viên cũng cho phép quản lý quyền truy cập và kiểm soát tài khoản người dùng để đảm bảo an toàn và hiệu suất cho toàn bộ hệ thống.



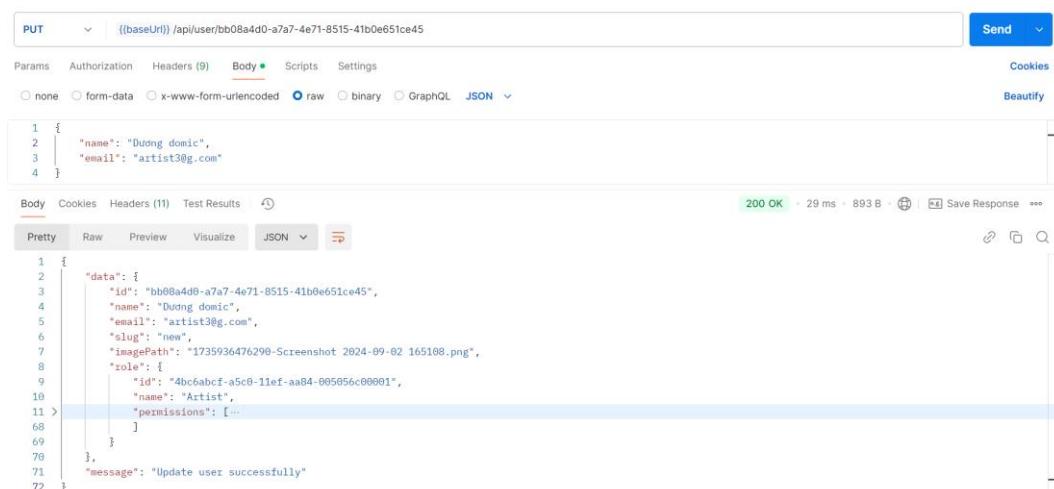
Hình 30. Kiểm thử API lấy tất cả người dùng

Xây dựng website nghe nhạc trực tuyến theo thời gian thực bằng NextJs



```
1 {
  "data": [
    {
      "id": "bb08a4d0-a7a7-4e71-8515-41b0e651ce45",
      "name": "New",
      "email": "New@eg.com",
      "slug": "new",
      "imagePath": "1735936476290-Screenshot 2024-09-02 165108.png",
      "followers": 0,
      "songs": 0,
      "playlists": 0,
      "role": {
        "id": "4bc6abcf-a5c0-11ef-aa84-005056c00001",
        "name": "Artist"
      }
    },
    {
      "id": "3ed029f6-5b29-4e12-bef5-ee0add5b77e",
      "name": "Trịnh Trần Phương Tuấn",
      "email": "artiststg.com",
      "slug": "bo-con",
      "imagePath": null,
      "followers": 1,
      "songs": 0,
      "playlists": 0,
      "role": {
        "id": "4bc6abcf-a5c0-11ef-aa84-005056c00001"
      }
    }
  ]
}
```

Hình 31. Kiểm thử API lấy danh sách các nghệ sĩ



```
1 {
  "name": "Dương domic",
  "email": "artist3@g.com"
}
```

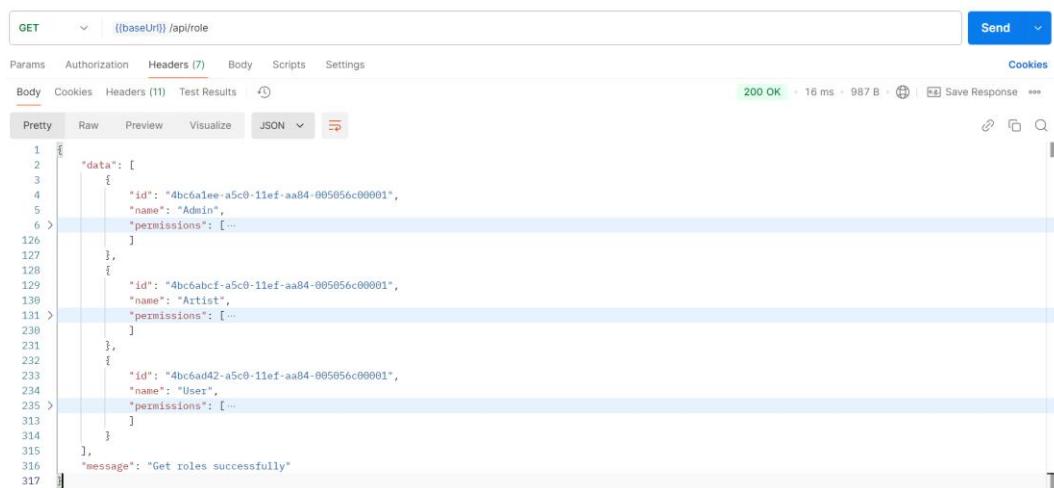
```
1 {
  "data": {
    "id": "bb08a4d0-a7a7-4e71-8515-41b0e651ce45",
    "name": "Dương domic",
    "email": "artist3@g.com",
    "slug": "new",
    "imagePath": "1735936476290-Screenshot 2024-09-02 165108.png",
    "followers": 0,
    "songs": 0,
    "playlists": 0,
    "role": {
      "id": "4bc6abcf-a5c0-11ef-aa84-005056c00001",
      "name": "Artist",
      "permissions": [...]
    }
  },
  "message": "Update user successfully"
}
```

Hình 32. Kiểm thử API cập nhật người dùng

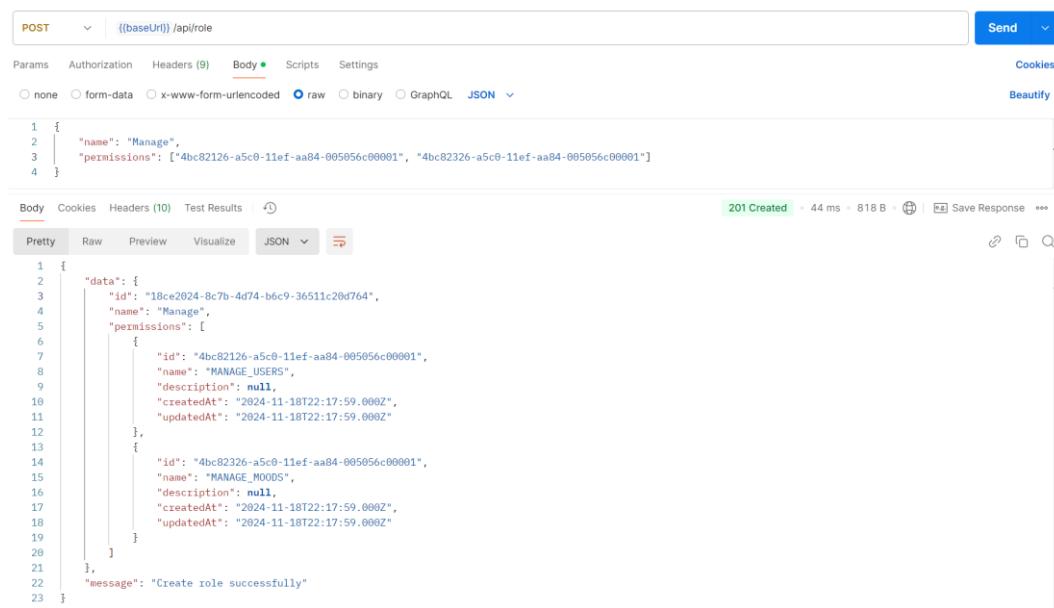
4.1.5 API liên quan đến vai trò và quyền hạn

Các API liên quan đến vai trò và quyền trong hệ thống đảm bảo rằng người dùng chỉ có thể thực hiện các hành động phù hợp với vai trò của họ. Chức năng này bao gồm việc định nghĩa và quản lý các quyền truy cập cho từng loại người dùng như quản trị viên, nghệ sĩ, và người nghe. API cho phép gán vai trò cho tài khoản, kiểm tra quyền hạn khi truy cập tài nguyên, và hạn chế các thao tác nhạy cảm như quản lý bài hát hoặc danh sách phát. Điều này giúp duy trì bảo mật, ngăn chặn truy cập trái phép và tạo nền tảng cho việc mở rộng các tính năng.

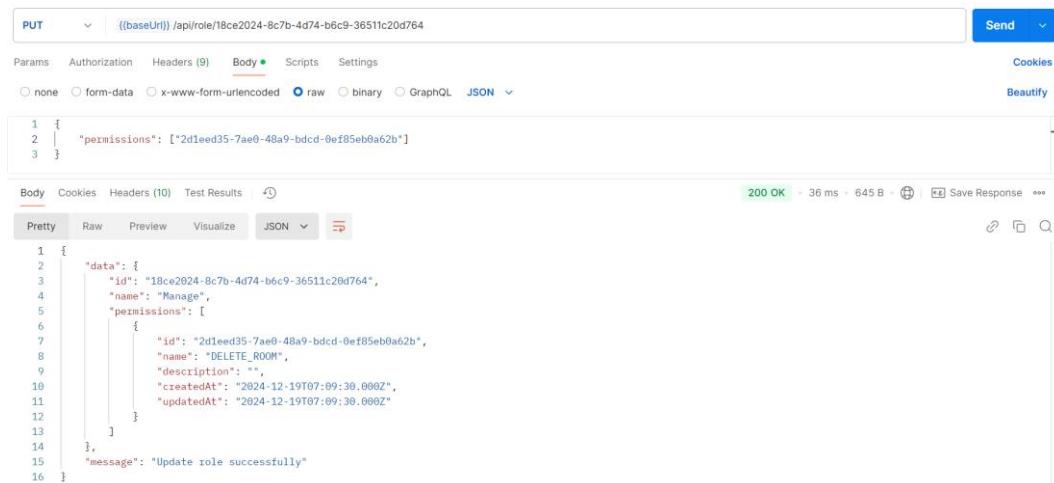
Xây dựng website nghe nhạc trực tuyến theo thời gian thực bằng NextJs



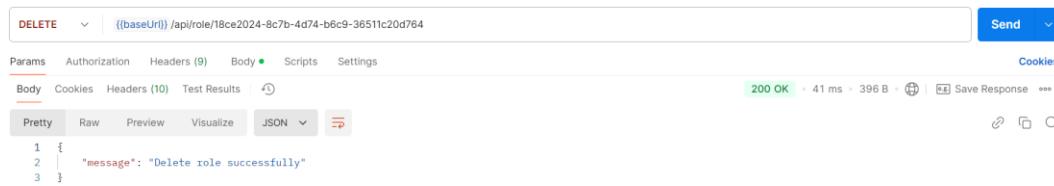
Hình 33. Kiểm thử API lấy tất cả các vai trò và quyền



Hình 34. Kiểm thử API tạo mới vai trò với các quyền



Hình 35. Kiểm thử API cập nhật vai trò

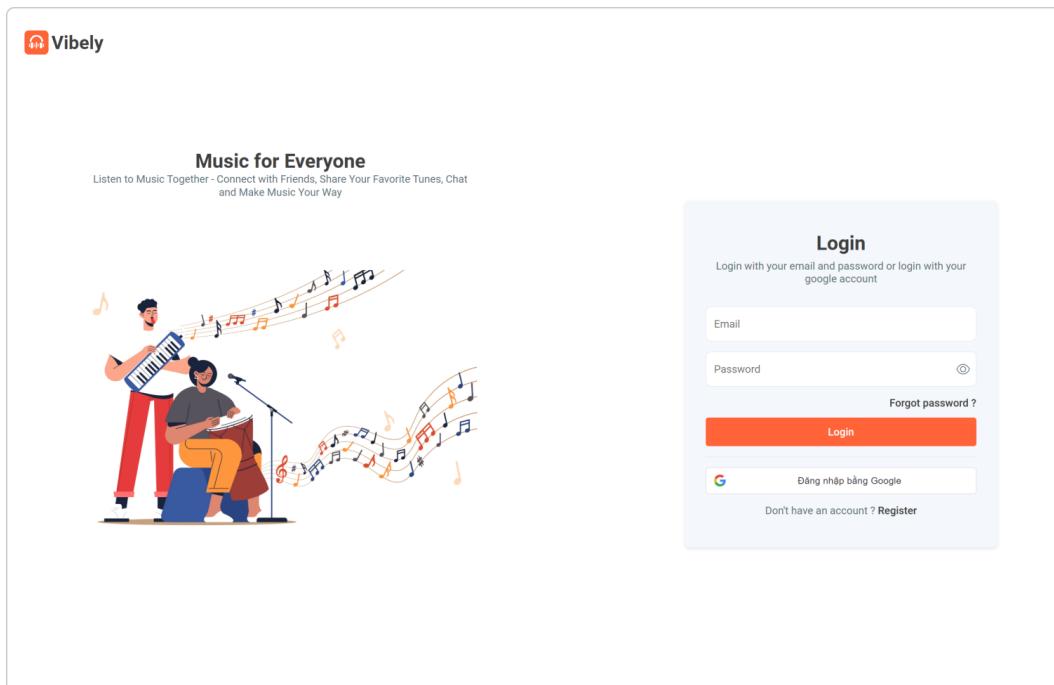


```
DELETE ((baseUrl)) /api/role/18ce2024-8c7b-4d74-b6c9-36511c20d764
200 OK · 41 ms · 396 B · Save Response · Cookies
Params Authorization Headers (9) Body Scripts Settings
Body Cookies Headers (10) Test Results ⚡
Pretty Raw Preview Visualize JSON ⚡
1 {
2   "message": "Delete role successfully"
3 }
```

Hình 36. Kiểm thử API xóa vai trò

4.2 Giao diện người dùng

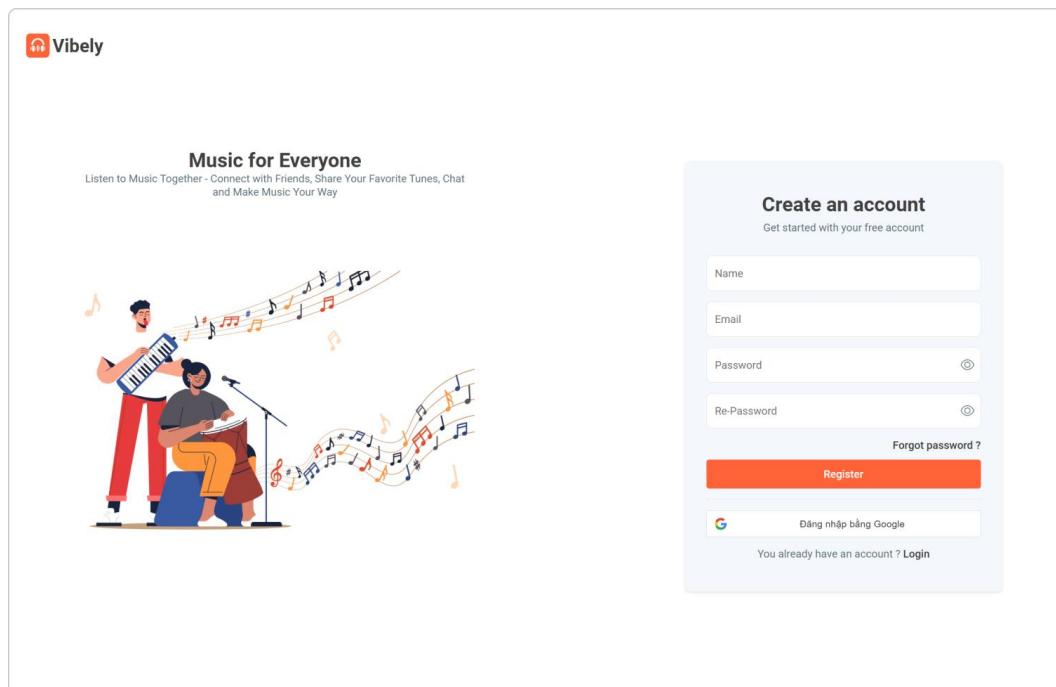
4.2.1 Giao diện trang đăng nhập



Hình 37. Giao diện trang đăng nhập Vibely

Trang đăng nhập người dùng có thể đăng nhập bằng cách sử dụng email và mật khẩu đã đăng ký hoặc lựa chọn đăng nhập bằng tài khoản Google để tăng cường tính tiện lợi. Giao diện đi kèm với thông báo lỗi rõ ràng khi thông tin không chính xác, đảm bảo người dùng dễ dàng thao tác và truy cập vào các tính năng của ứng dụng.

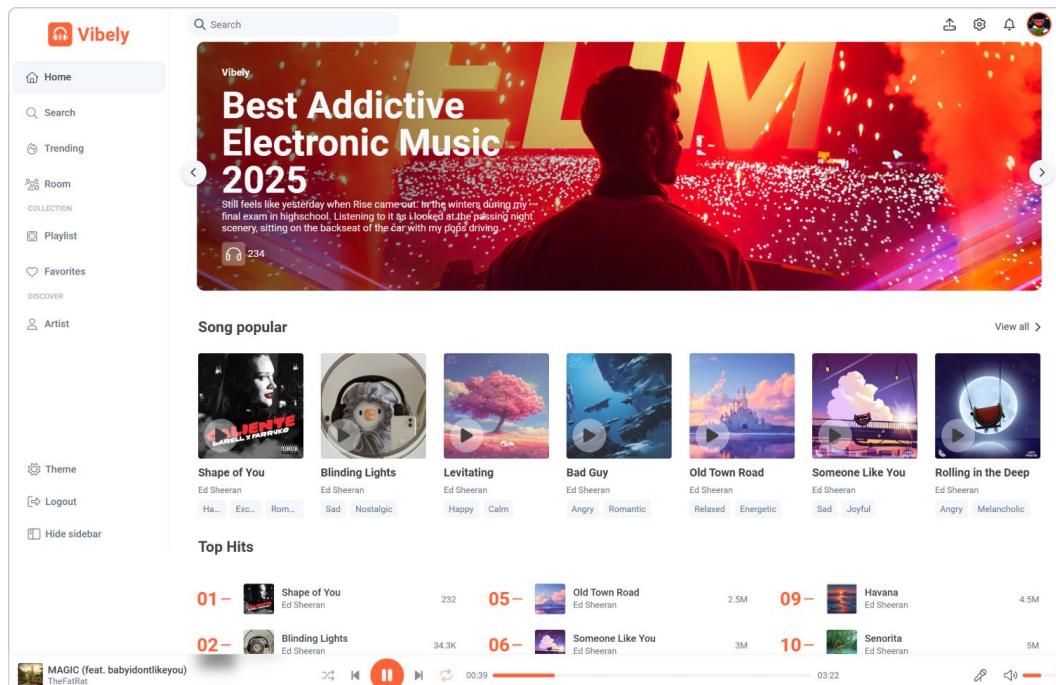
4.2.2 Giao diện trang đăng ký



Hình 38. Giao diện trang đăng ký Vibely

Người dùng có thể đăng ký bằng cách cung cấp các thông tin cơ bản bao gồm tên, email và mật khẩu. Để tăng cường tính tiện lợi, hệ thống cũng hỗ trợ đăng ký nhanh bằng tài khoản Google, giúp loại bỏ thao tác nhập liệu thủ công và đảm bảo thông tin xác thực an toàn.

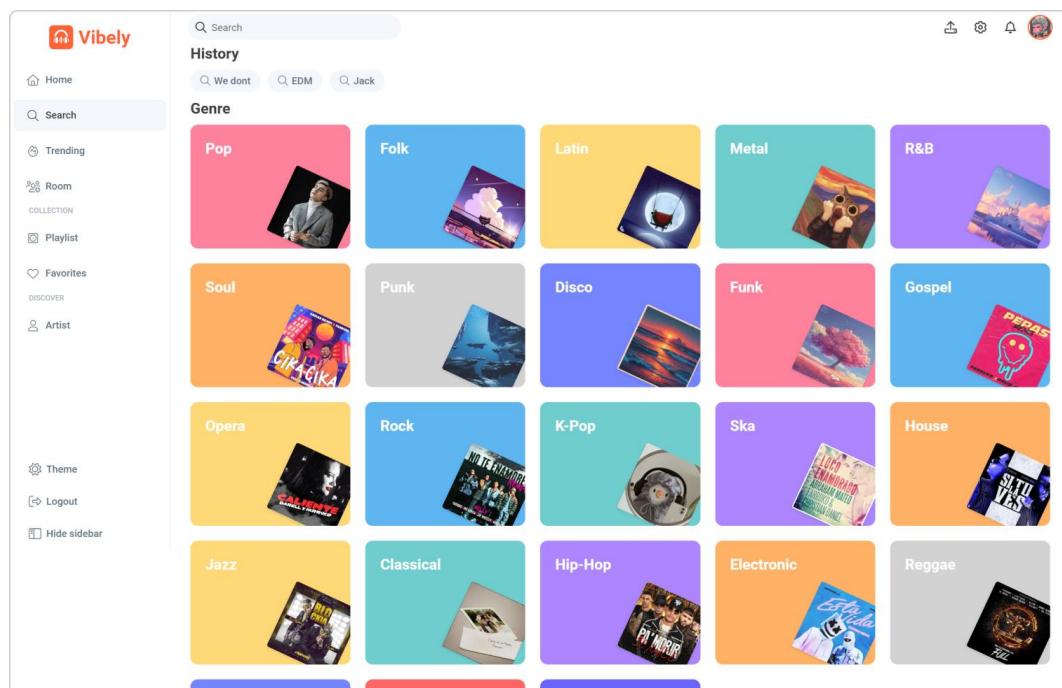
4.2.3 Giao diện trang chủ



Hình 39. Giao diện trang chủ Vibely

Phía trên cùng trang chủ là một banner động hiển thị danh sách các phòng nghe nhạc nổi bật, cho phép người dùng nhanh chóng tham gia và nghe nhạc theo thời gian thực cùng bạn bè. Bên dưới banner, người dùng có thể khám phá các bài hát phổ biến, được trình bày dưới dạng danh sách hoặc lưới với ảnh bìa và thông tin tóm tắt. Tiếp theo là khu vực giới thiệu các bài hát theo thứ hạng, dựa trên số lượt nghe hoặc yêu thích.

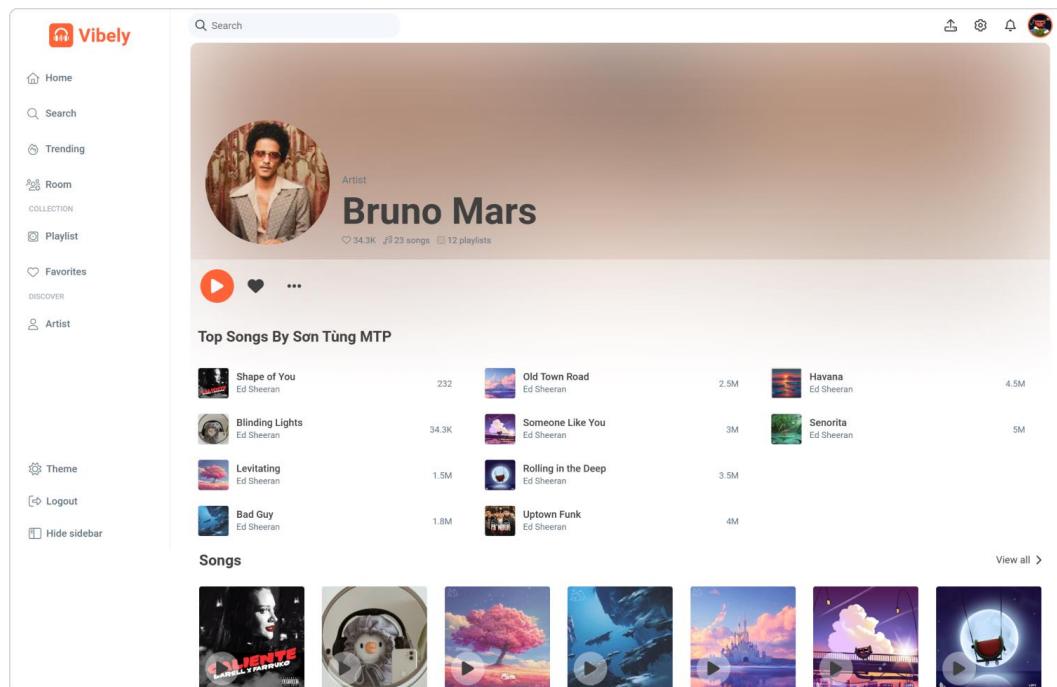
4.2.4 Giao diện trang tìm kiếm



Hình 40. Giao diện trang tìm kiếm Vibely

Trang tìm kiếm được thiết kế trên cùng hiển thị lịch sử tìm kiếm gần đây, cho phép người dùng dễ dàng truy cập lại các bài hát, nghệ sĩ, hoặc danh sách phát đã tìm trước đó. Ngay bên dưới, giao diện cung cấp danh mục các thể loại bài hát được trình bày trực quan, giúp người dùng khám phá nội dung theo sở thích.

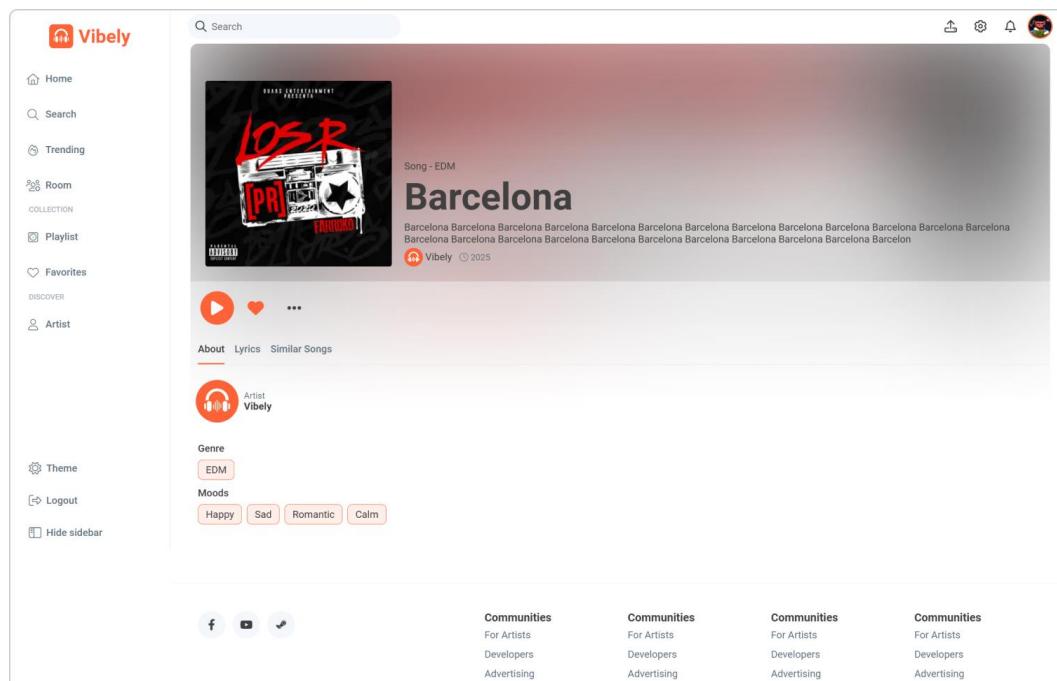
4.2.5 Giao diện trang nghệ sĩ



Hình 41. Giao diện trang nghệ sĩ Vibely

Trang nghệ sĩ phần đầu trang hiển thị ảnh đại diện cùng với tên nghệ sĩ và một số thông tin cơ bản như thể loại âm nhạc, số lượng người theo dõi, mô tả ngắn gọn. Bên dưới, danh sách các bài hát gần đây được phát hành bởi nghệ sĩ, cho phép người dùng nghe trực tiếp. Trang cũng cung cấp danh sách phát mà nghệ sĩ đã tạo hoặc có liên quan, giúp người dùng dễ dàng khám phá thêm các tác phẩm nổi bật.

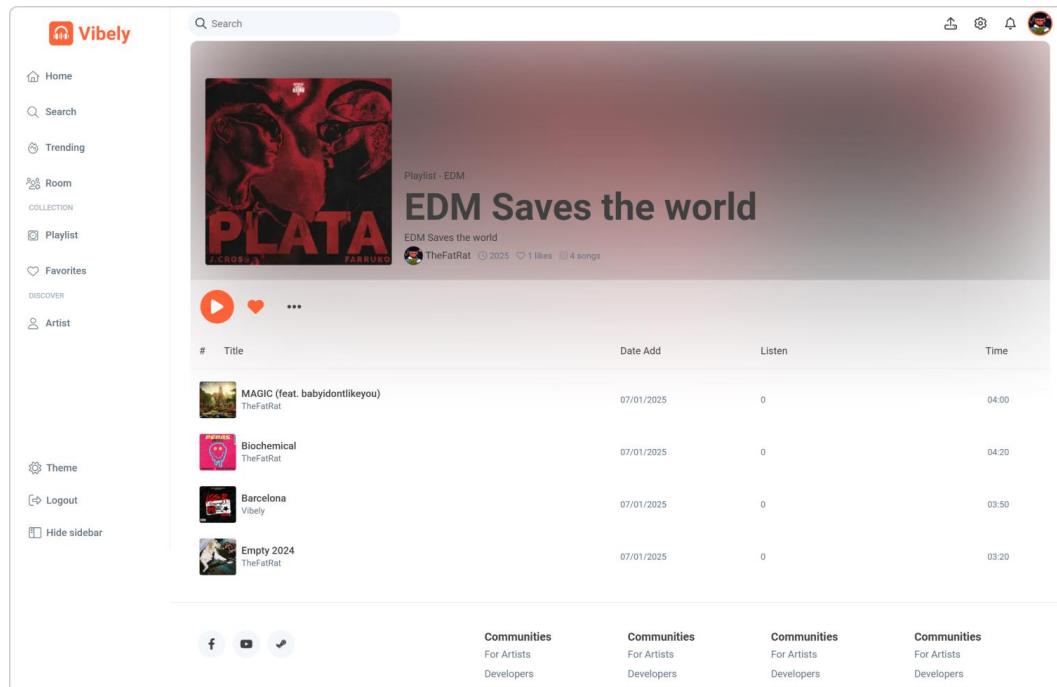
4.2.6 Giao diện trang bài hát



Hình 42. Giao diện trang bài hát Vibely

Trang bài hát cung cấp thông tin chi tiết và các tính năng tương tác cho từng bài hát. Phần đầu trang hiển thị ảnh đại diện lớn cùng với tiêu đề và mô tả ngắn gọn để giới thiệu nội dung. Ngay bên dưới là các nút chức năng như phát bài hát, thêm vào danh sách phát, chỉnh sửa thông tin (đối với người có quyền). Phần tiếp theo hiển thị thông tin về tác giả, lời bài hát giúp người dùng dễ dàng theo dõi khi nghe nhạc, và mục gợi ý bài hát liên quan để tiếp tục khám phá.

4.2.7 Giao diện trang danh sách phát

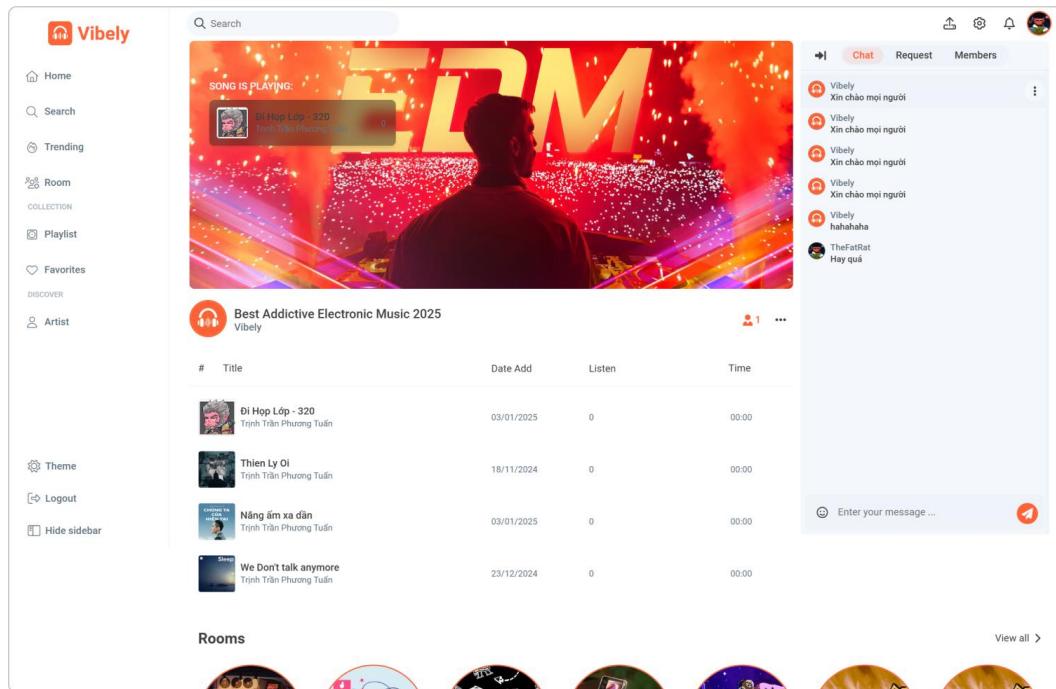


Hình 43. Giao diện trang danh sách phát

Trang danh sách phát với phần đầu trang chứa ảnh đại diện, tiêu đề, và mô tả ngắn gọn để giới thiệu danh sách phát. Ngay bên dưới là các nút chức năng như phát toàn bộ bài hát, chỉnh sửa thông tin (nếu người dùng có quyền), chia sẻ danh sách. Danh sách các bài hát được hiển thị theo thứ tự với tùy chọn phát riêng lẻ hoặc xóa bài hát. Cuối trang là gợi ý các danh sách phát liên quan, giúp người dùng khám phá thêm nội dung phù hợp với sở thích.

Giao diện trang

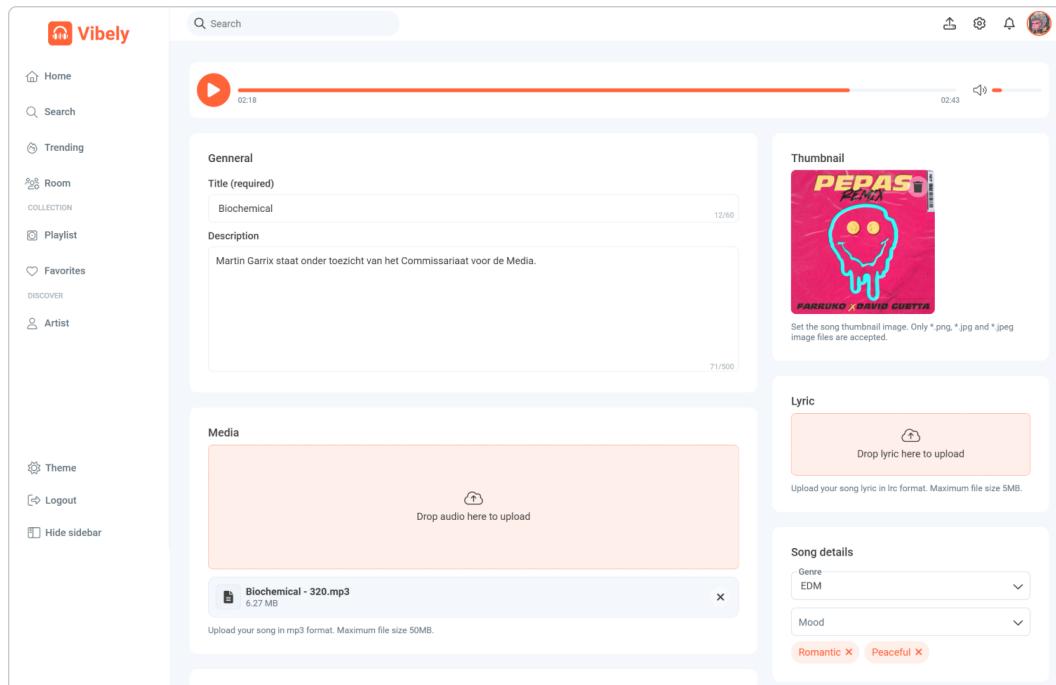
4.2.8 Giao diện trang phòng nghe nhạc



Hình 44. Giao diện trang phòng nghe nhạc Vibely

Trang phòng nghe phía trái hiển thị banner và thông tin bài hát đang phát, bao gồm ảnh bìa, tên bài hát, và nghệ sĩ. Phía dưới là thông tin chi tiết về phòng như tên phòng, người tạo, và mô tả, cùng danh sách các bài hát sắp phát tiếp theo. Phía phải là khung chat cho phép thành viên trao đổi tin nhắn trong thời gian thực và danh sách các thành viên đang tham gia. Ở cuối trang là các phòng gợi ý.

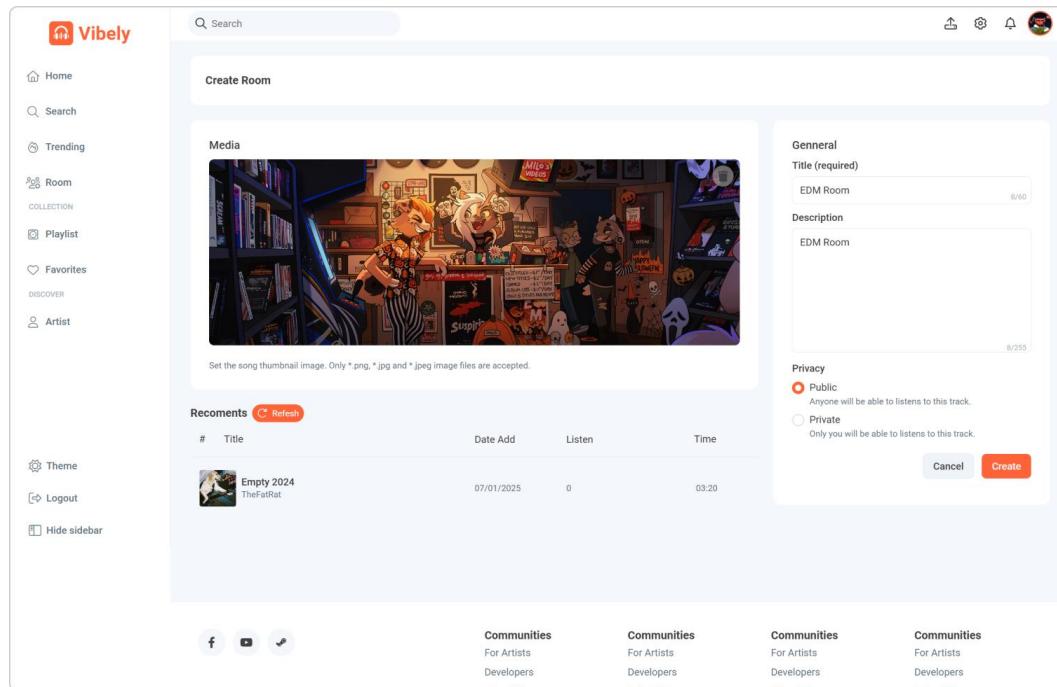
4.2.9 Giao diện trang tạo bài hát (Dành cho nghệ sĩ)



Hình 45. Giao diện trang tạo bài hát Vibely

Trang tạo bài hát với phần trên cùng cung cấp tính năng nghe thử bài hát để kiểm tra trước khi lưu. Phía dưới là các trường nhập liệu chi tiết gồm tên bài hát, mô tả, ảnh bìa, và tập tin lời bài hát để tải lên. Người dùng cũng có thể đính kèm tập tin âm thanh và lựa chọn các thẻ loại cùng tâm trạng phù hợp cho bài hát.

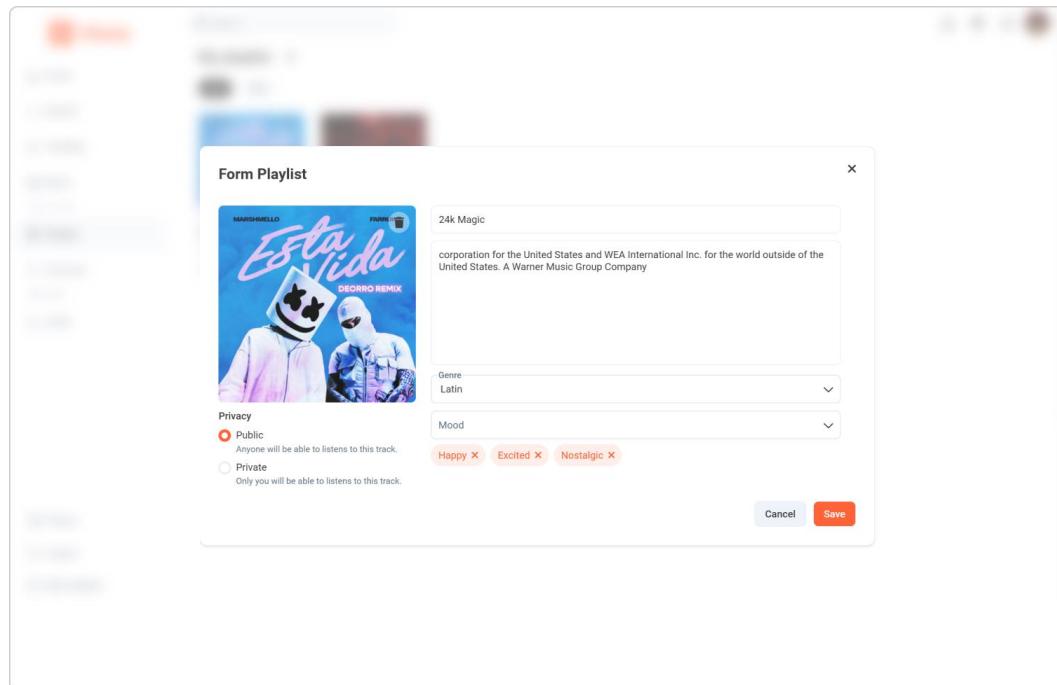
4.2.10 Giao diện trang tạo phòng nghe nhạc



Hình 46. Giao diện trang tạo phòng nghe nhạc trực tuyến Vibely

Trang tạo phòng bên trái hiển thị banner phòng và phần gợi ý các bài hát để thêm vào danh sách phát trong phòng. Bên phải là các trường nhập liệu bao gồm tiêu đề phòng, mô tả, và tùy chọn thiết lập phòng ở chế độ công khai hoặc riêng tư. Thiết kế tập trung vào sự đơn giản và thuận tiện, cho phép người dùng tạo phòng nhanh chóng và bắt đầu trải nghiệm nghe nhạc thời gian thực cùng bạn bè hoặc cộng đồng.

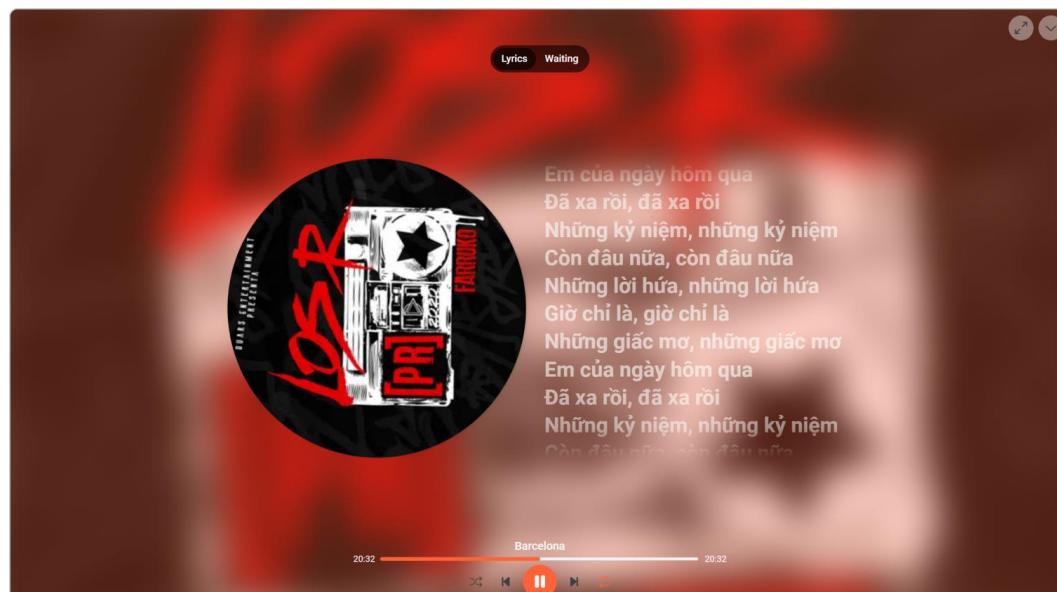
4.2.11 Giao diện tạo danh sách phát



Hình 47. Giao diện tạo danh sách phát Vibely

Giao diện tạo danh sách phát được thiết kế với bên trái bao gồm khu vực để tải lên ảnh đại diện cho danh sách phát và tùy chọn thiết lập chế độ công khai hoặc riêng tư. Bên phải là các trường nhập liệu như tiêu đề, mô tả, cùng với lựa chọn thể loại và tâm trạng phù hợp với danh sách phát.

4.2.12 Giao diện phát nhạc



Hình 48. Giao diện phát nhạc Vibely

Giao diện phát nhạc của Vibely được thiết kế với phần chính giữa hiển thị ảnh bìa bài hát lớn để thu hút sự chú ý. Bên dưới ảnh bìa, lời bài hát sẽ hiển thị theo thời gian phát nhạc, giúp người dùng theo dõi nội dung đồng bộ với nhịp điệu bài

hát. Phía dưới cùng là thanh phát nhạc bao gồm thanh tiến trình, các nút điều khiển phát/tạm dừng, chuyển bài, lặp lại, và thêm vào danh sách phát.

CHƯƠNG 5 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1 Kết luận:

Xây dựng nền tảng vững chắc: Dự án đã xây dựng thành công một website nghe nhạc trực tuyến với nhiều tính năng cơ bản, bao gồm:

- **Nghe nhạc theo thời gian thực:** Cho phép nhiều người dùng đồng thời nghe cùng một bài hát, đồng bộ hóa trạng thái âm thanh.
- **Tạo phòng nghe nhạc:** Người dùng có thể tạo phòng và tham gia vào các phòng nghe nhạc, chia sẻ trải nghiệm với bạn bè hoặc cộng đồng.
- **Quản lý bài hát và danh sách phát:** Cho phép người dùng dễ dàng tạo và quản lý danh sách phát, cũng như chia sẻ bài hát yêu thích.
- **Tính năng chat trong phòng:** Người dùng có thể trò chuyện với nhau khi đang nghe nhạc cùng, tạo ra không gian tương tác cao.

API và bảo mật:

- **Xác thực người dùng:** Hệ thống xác thực người dùng thông qua email và password, cũng như các phương thức đăng nhập bằng Google.
- **Quản lý quyền và phân quyền:** Dự án hỗ trợ phân quyền cho người dùng và nghệ sĩ, giúp bảo vệ tài nguyên và đảm bảo quyền lợi của từng người.
- **API hiệu quả:** Các API được xây dựng với Next.js, giúp dễ dàng lấy và gửi dữ liệu từ frontend đến backend.

5.2 Hướng phát triển:

- Tối ưu hóa hiệu suất và cải thiện khả năng mở rộng của hệ thống.
- Nghiên cứu thêm các tính năng nâng cao như trò chơi, thi đấu âm nhạc hoặc tạo sự kiện trực tuyến để thu hút người dùng.
- Tích hợp AI để gợi ý bài hát hoặc danh sách phát cho người dùng.
- Triển khai các công cụ phân tích dữ liệu để thu thập phản hồi từ người dùng, qua đó cải thiện khả năng gợi ý nhạc, danh sách phát và cải thiện tính năng tìm kiếm.

DANH MỤC TÀI LIỆU THAM KHẢO

- [1] nextjs, 12 2024. [Trực tuyến]. Available: <https://nextjs.org/docs>.
- [2] jwt, 2024 12. [Trực tuyến]. Available: <https://jwt.io/introduction>.
- [3] AWS, “What is an API (Application Programming Interface)?,” 12 2024. [Trực tuyến]. Available: Available: <https://aws.amazon.com/what-is/api>.
- [4] 200lab, “Nextjs là gì,” 12 2024. [Trực tuyến]. Available: <https://200lab.io/blog/nextjs-la-gi>.
- [5] socket, “What Socket.IO is,” 12 2024. [Trực tuyến]. Available: <https://socket.io/docs/v4/>.

