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

THỰC TẬP ĐỒ ÁN CHUYÊN NGÀNH
HỌC KỲ I, NĂM HỌC 2023 – 2024

NGHIÊN CỨU NODEJS VÀ XÂY DỰNG WEBSITE TRÌNH CHIẾU LỜI BÀI HÁT

Giáo viên hướng dẫn:
TS. Nguyễn Bảo Ân

Sinh viên thực hiện:
Họ tên: Cao Ka Ka
MSSV: 110120138
Lớp: DA20TTB

Trà Vinh, tháng 01 năm 2024

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

THỰC TẬP ĐỒ ÁN CHUYÊN NGÀNH
HỌC KỲ I, NĂM HỌC 2023 – 2024

NGHIÊN CỨU NODEJS VÀ XÂY DỰNG WEBSITE TRÌNH CHIẾU LỜI BÀI HÁT

Giáo viên hướng dẫn:
TS. Nguyễn Bảo Ân

Sinh viên thực hiện:
Họ tên: Cao Ka Ka
MSSV: 110120138
Lớp: DA20TTB

Trà Vinh, tháng 01 năm 2024

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 đến toàn thể quý thầy cô, giảng viên Trường Đại học Trà Vinh, đặc biệt là các thầy cô ở Khoa Kỹ thuật & Công nghệ, bộ môn Công nghệ thông tin, đã tạo điều kiện tốt nhất để em hoàn thành trọn vẹn bài báo cáo này.

Tiếp theo, em xin tỏ lòng biết ơn sâu sắc đến thầy Nguyễn Bảo Ân – Giảng viên Khoa Kỹ thuật & Công nghệ, Trường Đại học Trà Vinh, trong quá trình hướng dẫn đồ án đã vô cùng tâm huyết trong việc truyền đạt kiến thức đến em nói riêng và sinh viên Khoa Kỹ thuật & Công nghệ nói chung.

Trong bài báo cáo, do lượng kiến thức và kinh ngiệm còn khiêm tốn, vẫn còn một số sai sót nhỏ không đáng kể. Do đó, em kính mong quý thầy cô thông cảm, góp ý để em có thể tiếp thu và cải thiện cho những nghiên cứu trong tương lai.

Sau tất cả, kính chúc các thầy cô luôn dồi dào sức khoẻ.

Em xin chân thành cảm ơn!

Trà Vinh, ngày ….. tháng …… năm ……

Sinh viên thực hiện

Cao Ka Ka

MỤC LỤC

DANH MỤC HÌNH ẢNH – BẢNG BIỂU

TÓM TẮT ĐỒ ÁN CHUYÊN NGÀNH

Trong bài báo cáo này, chúng ta sẽ nghiên cứu công nghệ NodeJS để phát triển một trang web trình chiếu lời bài hát theo cơ chế Restful API.

Các công nghệ được hướng đến sử dụng trong bài báo cáo này:

- Cơ sở lý thuyết: Restful API, xác thực người dùng (User Authentication).

- Ngôn ngữ Back-end: NodeJS, thư viện ExpressJS.

- Cơ sở dữ liệu: MongoDB.

- Front-end: Framework ReactJS, thư viện Ant Design.

- Nền tảng lưu trữ và xác thực trên đám mây: Firebase.

Báo cáo đã xây dựng thành công được trang web trình chiếu lời bài hát cùng các chức năng đã đặt ra, quản trị hệ thống, quản trị người dùng,… Chi tiết sẽ trình bày ở phần sau.

MỞ ĐẦU

Lý do chọn đề tài:

Hiện nay, Restful API đã trở nên cực kỳ phổ biến trong các ứng dụng web. Việc hiểu rõ và nắm vững các kiến thức đó là điều hết sức cần thiết khi chúng ta đang sống trong một thế giới công nghệ luôn thay đổi từng ngày. NodeJS cùng với thư viện ExpressJS là một sự lựa chọn tuyệt vời để bắt đầu tìm hiểu về Restful API.

Cùng với đó, nhận thấy nhu cầu về một trang web đáp ứng việc trình chiếu lời bài hát đang ngày một nhiều hơn. Không chỉ cần phải đáp ứng được các chức năng cần thiết, mà phải có ngôn ngữ thiết kế phù hợp, bắt mắt, thân thiện, dể dàng trong việc trải nghiệm và sử dụng.

Mục đích nghiên cứu:

Hiểu rõ về NodeJS, Restful API và ứng dụng chúng để xây dựng website trình chiếu lời bài hát.

Đối tượng nghiên cứu:

- Các kiến thức liên quan như Restful API, xác thực người dùng.
- NodeJS và thư viện ExpressJS.
- Cơ sở dữ liệu MongoDB.
- Framework ReactJS.

Phạm vi nghiên cứu:

- Tìm hiểu và nắm vững các cơ sở lý thuyết về cơ chế Restful API, xác thực người dùng.
- Tìm hiểu NodeJS để xây dựng trang web trình chiếu lời bài hát.

CHƯƠNG 1: TỔNG QUAN

Restful API đang ngày càng phổ biến và có mặt hầu hết trong các ứng dụng web hiện đại ngày nay. Việc áp dụng cơ chế Restful API làm cho việc phát triển một ứng dụng web trở nên dễ dàng hơn. Cùng với đó, sự ra đời của nhiều công nghệ mới đem đến nhiều sự lựa chọn hơn trong việc phát triển ứng dụng. Mà trong đó, NodeJS là cái tên đáng được quan tâm nhất.

Trong bài báo cáo này, chúng ta sẽ cùng tìm hiểu từng bước xây dựng một trang web có thể trình chiếu lời bài hát, thông qua việc vận dụng kiến thức về Restful API và sự nhanh chóng, mạnh mẽ từ NodeJS

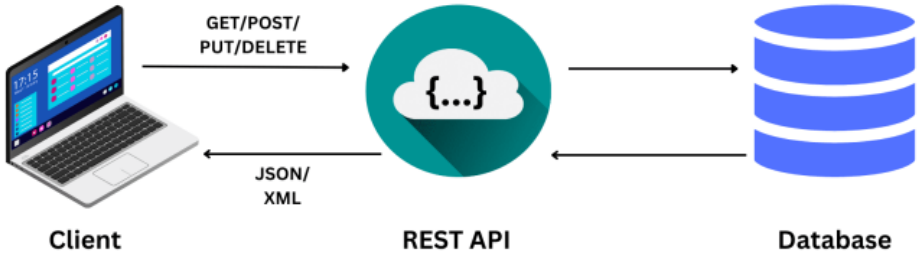
CHƯƠNG 2: NGHIÊN CỨU LÝ THUYẾT

2.1 Tìm hiểu về Restful API

2.1.1 Tổng quan về Restful API

RESTful API (Representational State Transfer API) là một kiến trúc thiết kế cho việc xây dựng các dịch vụ web linh hoạt, đơn giản và dễ dàng tích hợp. Được đặt ra bởi Roy Fielding trong luận văn tiến sĩ của mình vào năm 2000, RESTful API dựa trên các nguyên tắc cơ bản của REST, một mô hình truyền thông không trạng thái và có khả năng mở rộng.

RESTful API sử dụng các URL dễ đọc và dễ hiểu, và sử dụng các định dạng dữ liệu như JSON hoặc XML để trao đổi thông tin giữa máy chủ và máy khách. RESTful API có tính khả di động cao, cho phép các ứng dụng khác nhau có thể truy cập và sử dụng các tài nguyên một cách dễ dàng.



Hình 1. Mô hình trang web sử dụng Restful API

2.1.2 Các thành phần chính trong Restful API

API (Application Programming Interface) là một tập các quy tắc và cơ chế mà theo đó, một ứng dụng hay một thành phần sẽ tương tác với một ứng dụng hay thành phần khác. API có thể trả về dữ liệu mà bạn cần cho ứng dụng của mình ở những kiểu dữ liệu phổ biến như JSON hay XML.

REST (REpresentational State Transfer) là một dạng chuyển đổi cấu trúc dữ liệu, một kiểu kiến trúc để viết API. Nó sử dụng phương thức HTTP đơn giản để tạo cho giao tiếp giữa các máy. Vì vậy, thay vì sử dụng một URL cho việc xử lý một số thông tin người dùng, REST gửi một yêu cầu HTTP như GET, POST, DELETE,… đến một URL để xử lý dữ liệu.

2.1.3 Các phương thức sử dụng trong Restful API

[GET]: Sử dụng để đọc dữ liệu từ một tài nguyên. Một yêu cầu GET sẽ trả về thông tin của tài nguyên được yêu cầu.

[POST]: Được sử dụng để tạo mới một tài nguyên. Thông thường, dữ liệu gửi kèm theo yêu cầu POST được sử dụng để tạo mới tài nguyên trên máy chủ.

[PUT]: Sử dụng để cập nhật một tài nguyên hoặc tạo mới nếu nó không tồn tại. Yêu cầu PUT thường đi kèm với dữ liệu hoặc thông tin cần cập nhật.

[DELETE]: Dùng để xóa một tài nguyên. Yêu cầu DELETE sẽ gửi thông điệp đến máy chủ để yêu cầu xóa tài nguyên được xác định.

[PATCH]: Sử dụng để thực hiện cập nhật một phần nhỏ của tài nguyên. Nó giúp giảm lượng dữ liệu gửi đi so với việc sử dụng PUT.

[OPTIONS]: Yêu cầu thông tin về các phương thức được hỗ trợ trên một tài nguyên cụ thể hoặc toàn bộ ứng dụng.

[HEAD]: Tương tự như GET, nhưng máy chủ chỉ trả về tiêu đề của yêu cầu, không có nội dung thực tế. Thường được sử dụng để kiểm tra trạng thái của một tài nguyên mà không tải toàn bộ nội dung.

2.1.4 Các trạng thái của Restful API

- 200 (OK): Thực hiện thành công.
- 201 (Created): Trả về khi một tài nguyên vừa được tạo thành công.
- 204 (No Content): Trả về khi tài nguyên xoá thành công.
- 304 (Not Modified): Client có thể sử dụng dữ liệu cache.
- 400 (Bad Request): Yêu cầu không hợp lệ.
- 401 (Unauthorized): Yêu cầu cần được xác thực.
- 403 (Forbidden): Bị từ chối không cho phép.
- 404 (Not Found): Không tìm thấy tài nguyên từ URI.
- 405 (Method Not Allowed): Phương thức này không được phép truy cập với người dùng hiện tại.
- 410 (Gone – Resource): Không còn tồn tại, không còn hỗ trợ.
- 415 (Unsupported Media Type): Không hỗ trợ loại tài nguyên này.
- 422 (Unprocessable Entity): Dữ liệu không được xác thực.
- 429 (Too Many Requests): Yêu cầu bị từ chối do bị giới hạn.

2.1.5 Ưu điểm của Restful API

Restful API là một công nghệ quan trọng trong phát triển ứng dụng web hiện nay. Việc sử dụng Restful API giúp cho việc tương tác giữa các ứng dụng trở nên dễ dàng và thuận tiện hơn. Sau đây là một số ưu điểm của Restful API trong việc phát triển website:

Đơn giản hóa việc phát triển

Restful API sử dụng các phương thức HTTP tiêu chuẩn, giúp cho việc phát triển ứng dụng trở nên đơn giản hơn. Nó cũng giúp cho các lập trình viên có thể tập trung vào phát triển các chức năng chính của ứng dụng thay vì phải quản lý việc tương tác giữa các thành phần khác nhau của ứng dụng.

Tính mở rộng

Restful API cho phép các ứng dụng có thể kết nối với nhau và trao đổi dữ liệu một cách dễ dàng, từ đó giúp cho việc mở rộng ứng dụng trở nên thuận tiện hơn. Các ứng dụng cũng có thể thêm các tính năng mới bằng cách tương tác với các API của ứng dụng khác.

Tính độc lập của ứng dụng

Restful API cho phép các ứng dụng hoạt động độc lập với nhau, từ đó giúp cho việc bảo trì và nâng cấp ứng dụng trở nên thuận tiện hơn.

Khả năng tương thích

Restful API có khả năng tương thích với các ứng dụng khác nhau, từ đó giúp cho việc tích hợp các ứng dụng với nhau trở nên dễ dàng hơn. Các ứng dụng có thể sử dụng cùng một API để tương tác với các dữ liệu khác nhau mà không cần phải tạo ra các API riêng biệt.

2.1.6 Nhược điểm của Restful API

- Chi phí: Việc phát triển và triển khai API đôi khi rất tốn kém và đòi hỏi bảo trì cũng như hỗ trợ cao từ các nhà phát triển.

- Vấn đề bảo mật: Việc sử dụng API sẽ thêm một tầng khác trong kiến trúc, khiến hệ thống dễ bị tấn công và do đó vấn đề rủi ro bảo mật thường xảy ra trong API. Ngoài ra, việc chỉ có thể chạy trên giao thức HTTP cũng khiến các API dễ dàng bị tấn công.

2.2 Tìm hiểu về xác thực người dùng (Auth)

2.2.1 Tổng quan về xác thực người dùng

Xác thực người dùng là quá trình xác định xem một người dùng có quyền truy cập vào một hệ thống, ứng dụng hoặc dịch vụ cụ thể không. Đây là một phần quan trọng trong bảo mật thông tin và đảm bảo rằng chỉ những người dùng được ủy quyền mới có thể truy cập vào các tài nguyên hay chức năng nhất định.

Quá trình xác thực người dùng đóng vai trò quan trọng trong bảo mật hệ thống và dữ liệu, đặc biệt là khi xử lý thông tin quan trọng hoặc cá nhân. Các phương pháp xác thực thường được kết hợp để tăng cường tính an toàn và ngăn chặn các mối đe dọa bảo mật.

2.2.2 Các cách xác thực người dùng phổ biến

Khi sử dụng API, quá trình xác thực là quan trọng để đảm bảo rằng chỉ các bên được ủy quyền mới có thể truy cập vào tài nguyên hay dịch vụ. Một số các xác thực phổ biến có thể kể đến như:

Xác thực API Key:

Mỗi bên sử dụng API đều có một khóa xác thực duy nhất. API key này thường được truyền trong tiêu đề hoặc các yêu cầu HTTP để xác định danh tính người dùng. Điều này dễ triển khai và hiệu quả cho việc xác thực.

OAuth (Authorization Code):

Sử dụng OAuth để cung cấp quyền truy cập an toàn và phân quyền. Người dùng đăng nhập thông qua một dịch vụ tư cách người dùng, và sau đó, dựa vào mã ủy quyền, API sẽ cấp cho họ một token để thực hiện các yêu cầu.

Bearer Token (Token Access):

Khi đăng nhập thành công, người dùng nhận được một token, thường là JWT, và sau đó sử dụng token này để xác thực trong các yêu cầu tiếp theo. Token thường được truyền trong tiêu đề "Authorization".

Basic Authentication:

Sử dụng tên người dùng và mật khẩu, kết hợp và mã hóa dưới dạng chuỗi Base64. Chuỗi này sau đó được truyền trong tiêu đề "Authorization". Mặc dù phổ biến, nhưng không nên sử dụng nếu không có HTTPS vì thông tin đăng nhập có thể bị hiển thị trong gói tin mạng.

Token Refresh:

Khi token hết hạn, một token mới có thể được yêu cầu bằng cách sử dụng token làm phương tiện để đổi lấy một token mới. Điều này giúp giảm rủi ro bảo mật liên quan đến việc giữ token trên máy khách.

Xác thực mã nhúng (Client Credentials):

Đối với các trường hợp nơi một ứng dụng đăng nhập mà không phải là một người dùng cụ thể, xác thực mã nhúng được sử dụng. Ứng dụng sẽ cung cấp các thông tin xác thực (client ID và client secret) để nhận token.

Xác thực Biometics:

Sử dụng các đặc điểm sinh học như vân tay, quét khuôn mặt hoặc nhận dạng giọng nói để xác thực người dùng.

Xác thực với SSO (Single Sign-On):

Sử dụng mô hình đăng nhập một lần để xác thực người dùng một lần duy nhất và chia sẻ thông tin xác thực qua nhiều ứng dụng.

2.2.3 Sự cần thiết của việc xác thực người dùng trong ứng dụng Restful API

Việc xác thực người dùng với ứng dụng RESTful API là một yếu tố cực kỳ quan trọng và cần thiết để đảm bảo tính bảo mật và kiểm soát quyền truy cập trong môi trường truyền thông qua API:

Bảo vệ dữ liệu nhạy cảm:

RESTful API thường cung cấp và quản lý dữ liệu nhạy cảm hoặc thông tin cá nhân của người dùng. Xác thực giúp đảm bảo rằng chỉ những người dùng được ủy quyền mới có thể truy cập và thao tác với những dữ liệu này.

Kiểm soát quyền truy cập:

Xác thực là cách để kiểm soát quyền truy cập của người dùng đến các tài nguyên và chức năng cụ thể của ứng dụng. Nó giúp định rõ những gì một người dùng được phép làm và những gì không.

Phòng ngừa tấn công:

Nếu một API không có xác thực, nó mở cửa cho rủi ro tấn công từ phía người dùng không ủy quyền hoặc từ bên ngoài. Xác thực giúp ngăn chặn việc truy cập trái phép và bảo vệ ứng dụng khỏi các loại tấn công như thử nghiệm mật khẩu, tấn công theo dõi và đánh cắp thông tin.

Ngăn chặn lạm dụng:

Sử dụng xác thực để ngăn chặn việc lạm dụng tài nguyên của máy chủ và giữ cho hoạt động của ứng dụng trong phạm vi kiểm soát.

2.3 Tìm hiểu về NodeJS

2.3.1 Tổng quan về NodeJS

Node.js là một môi trường thực thi mã nguồn mở xây dựng trên JavaScript engine của Chrome, giúp chạy mã JavaScript trên máy chủ. Với kiến trúc không đồng bộ và sự kiện lớn, Node.js tối ưu hóa việc xử lý đa nhiệm và làm cho các ứng dụng real-time trở nên dễ dàng. Nền tảng này giới thiệu sự linh hoạt trong phát triển với khả năng chia sẻ mã giữa máy chủ và trình duyệt, đồng thời tích hợp npm, hệ thống quản lý gói, để quản lý dependencies một cách hiệu quả. Với cộng đồng phát triển lớn, Node.js đã trở thành một công nghệ quan trọng, hỗ trợ xây dựng ứng dụng hiệu suất cao và real-time.

Node.js không chỉ hỗ trợ nhiều nền tảng mà còn giúp đơn giản hóa quy trình phát triển và triển khai ứng dụng. Sự xuất hiện của các framework nổi tiếng như Express.js và công nghệ như Socket.io cung cấp những công cụ mạnh mẽ để phát triển ứng dụng web và real-time. Được đánh giá cao với tính linh hoạt và hiệu suất, Node.js đang ngày càng trở thành lựa chọn hàng đầu cho việc xây dựng các dự án phức tạp và ứng dụng đa nhiệm trên nền tảng web.

2.3.2 Lịch sử hình thành của NodeJS

Node.js ban đầu được viết bởi Ryan Dahl vào năm 2009, khoảng 13 năm sau khi giới thiệu môi trường JavaScript phía máy chủ đầu tiên, LiveWire Pro Web của Netscape. Phiên bản ban đầu chỉ hỗ trợ Linux và Mac OS X. Quá trình phát triển và bảo trì được Ryan Dahl dẫn đầu và sau đó được Joyent tài trợ.

Ryan Dahl đã chỉ trích khả năng giới hạn của Apache HTTP Server trong việc xử lý nhiều kết nối đồng thời (10,000+), cũng như mô hình lập trình tuần tự chiếm ưu thế, trong đó ứng dụng có thể chặn toàn bộ quy trình hoặc tạo nên nhiều ngăn xếp thực thi cho các kết nối đồng thời. Dahl trình diễn dự án này tại hội nghị European JSConf đầu tiên vào ngày 8 tháng 11 năm 2009. Node.js kết hợp giữa Google's V8 JavaScript engine, một vòng lặp sự kiện và một API I/O cấp thấp.

Tháng 1 năm 2010, một trình quản lý gói dành cho môi trường Node.js được giới thiệu với tên là npm. Trình quản lý gói này cho phép các lập trình viên xuất bản và chia sẻ gói Node.js, cũng như mã nguồn đi kèm, và được thiết kế để đơn giản hóa quá trình cài đặt, cập nhật và gỡ cài đặt các gói.

Tháng 6 năm 2011, Microsoft và Joyent triển khai một phiên bản Windows native của Node.js. Phiên bản đầu tiên hỗ trợ Windows được phát hành vào tháng 7 năm 2011.

Tháng 1 năm 2012, Ryan Dahl nhường quyền quản lý dự án cho tác giả npm Isaac Schlueter. Tháng 1 năm 2014, Schlueter thông báo rằng Timothy J. Fontaine sẽ lãnh đạo dự án.

Tháng 12 năm 2014, Fedor Indutny tạo ra io.js, một nhánh của Node.js do không hài lòng với quản lý của Joyent, với ý định tạo ra một cộng đồng mở rộng với một ủy ban kỹ thuật độc lập. Node.js Foundation được thành lập để hòa giải Node.js và io.js dưới một bức màn duy nhất và được công bố vào tháng 2 năm 2015. Sáp nhập được thực hiện vào tháng 9 năm 2015, mang theo Node.js v0.12 và io.js v3.3 để tạo ra Node v4.0.

Năm 2019, JS Foundation và Node.js Foundation hợp nhất để tạo ra OpenJS Foundation.

Ngày 6 tháng 9 năm 2023, Node.js 20.6.0 được phát hành với sự thêm mới của hỗ trợ tích hợp cho tệp chứa biến môi trường .env.

2.3.3 Ưu điểm của NodeJS

Khả năng đồng thời (concurrency):

Node.js sử dụng kiến trúc không đồng bộ và sự kiện lớn, cho phép xử lý hàng nghìn kết nối đồng thời mà không cần tạo ra các luồng mới. Điều này giúp cải thiện khả năng mở rộng và hiệu suất của ứng dụng. Node.js thích hợp cho việc xây dựng ứng dụng real-time như chat, trò chơi trực tuyến, cập nhật thời gian thực và ứng dụng streaming.

Hiệu suất cao:

Sử dụng Google's V8 JavaScript engine, Node.js cung cấp hiệu suất cao bằng cách biên dịch mã JavaScript thành mã máy ngay lúc chạy. Điều này giúp giảm độ trễ và tăng tốc độ xử lý.

Chia sẻ mã giữa máy chủ và trình duyệt:

JavaScript có thể được sử dụng cả ở phía máy chủ và trình duyệt, giúp đơn giản hóa việc chia sẻ mã giữa cả hai môi trường và tăng cường tính tái sử dụng mã nguồn.

Hệ sinh thái mô-đun nguồn mở (open source):

Node.js sử dụng npm (Node Package Manager), một hệ thống mô-đun mạnh mẽ, giúp quản lý dependencies và chia sẻ mã nguồn dễ dàng. Cộng đồng mô-đun của npm rất lớn và đa dạng.

Nhẹ và linh hoạt:

Node.js là một môi trường nhẹ, giúp giảm tải cho hệ thống và tối ưu hóa việc triển khai ứng dụng. Nó cũng linh hoạt và có thể sử dụng cho nhiều mục đích khác nhau.

Phát triển nhanh:

Việc sử dụng JavaScript cho cả phía máy chủ và trình duyệt giúp giảm thời gian phát triển, vì các nhà phát triển có thể sử dụng cùng một ngôn ngữ và môi trường.

Sự hỗ trợ tốt từ cộng đồng:

Node.js có một cộng đồng phát triển lớn, đam mê và năng động. Sự hỗ trợ từ cộng đồng làm cho việc giải quyết vấn đề và tìm giải pháp trở nên dễ dàng. Node.js thường xuyên nhận được cập nhật và cải tiến từ cộng đồng, giúp đảm bảo rằng nó luôn duy trì được sự hiện đại và an toàn.

2.3.4 Nhược điểm của NodeJS

Chạy đơn luồng và chờ I/O:

Node.js hoạt động trong mô hình chạy đơn luồng, điều này có nghĩa là nó chỉ xử lý một công việc tại một thời điểm. Khi phải thực hiện các công việc chờ đợi như đọc dữ liệu từ cơ sở dữ liệu, nó có thể tạo ra độ trễ cho các công việc khác.

Nhóm Callback và lập trình bất đồng bộ:

Khi phải thực hiện nhiều công việc không đồng bộ, mã nguồn có thể trở nên rối bời với việc sử dụng nhiều hàm callback. Đôi khi, điều này được mô tả như một tình trạng "callback hell", làm cho việc hiểu và duy trì mã nguồn trở nên khó khăn.

Xử lý công việc tính toán nặng kém:

Node.js không phải là sự lựa chọn tốt nếu ứng dụng của bạn đòi hỏi nhiều tính toán nặng, chẳng hạn như xử lý hình ảnh lớn hay tính toán phức tạp. Do cách Node.js xử lý đơn luồng, một công việc lâu dài có thể tạo ra độ trễ cho toàn bộ ứng dụng.

Quản lý Dependency khó khăn:

Node.js sử dụng một công cụ quản lý gói gọi là npm, nhưng với sự phổ biến của nó, quản lý các phụ thuộc có thể trở nên phức tạp. Sự phụ thuộc giữa các gói có thể tạo ra khó khăn khi cần duy trì và cập nhật ứng dụng.

Thay đổi liên tục trong API:

Có thể xuất hiện những thay đổi không dự kiến trong API của Node.js khi chuyển sang các phiên bản mới. Điều này có thể làm cho mã nguồn không tương thích và yêu cầu thêm công việc để cập nhật mã nguồn.

Vấn đề bảo mật:

Với khả năng thực hiện các hoạt động không an toàn, quản lý bảo mật có thể trở nên khó khăn. Nếu triển khai và cấu hình không an toàn, có thể tăng rủi ro về bảo mật cho ứng dụng.

2.4 Tìm hiểu về MongoDB

2.4.1 Tổng quan về MongoDB

MongoDB là một hệ quản trị cơ sở dữ liệu phi quan hệ, mã nguồn mở, được thiết kế để đáp ứng nhu cầu của các ứng dụng hiện đại với dữ liệu đa dạng và phức tạp. Điểm độc đáo của MongoDB là khả năng lưu trữ dữ liệu dưới dạng BSON (Binary JSON), một định dạng nhị phân tương tự JSON, mang lại tính linh hoạt trong việc định hình cấu trúc dữ liệu mà không yêu cầu sự ràng buộc cứng nhắc của một schema truyền thống.

MongoDB hỗ trợ mô hình mở rộng ngang, cho phép dễ dàng mở rộng cơ sở dữ liệu khi cần thiết, giúp ứng phó với lưu lượng dữ liệu tăng cao. Điều này thích hợp cho các ứng dụng yêu cầu tính mở rộng và khả năng xử lý dữ liệu lớn. MongoDB cũng nổi bật với khả năng tìm kiếm phức tạp và các chỉ mục hiệu quả, giúp cải thiện hiệu suất truy vấn.

Ngoài ra, MongoDB cung cấp các công cụ và thư viện hỗ trợ cho nhiều ngôn ngữ lập trình, giúp dễ dàng tích hợp với các ứng dụng được phát triển bằng nhiều ngôn ngữ khác nhau. Cộng đồng lớn và sự hỗ trợ từ MongoDB, Inc. làm cho MongoDB trở thành một lựa chọn phổ biến trong cộng đồng phần mềm mã nguồn mở và doanh nghiệp, đặc biệt là trong lĩnh vực của các ứng dụng web, phân tích dữ liệu, và các dự án yêu cầu tính linh hoạt và mở rộng.

2.4.2 Lịch sử hình thành của MongoDB

Công ty phần mềm 10gen của Mỹ bắt đầu phát triển MongoDB vào năm 2007 như một thành phần của sản phẩm nền tảng dưới dạng dịch vụ đã được lên kế hoạch. Năm 2009, công ty chuyển sang mô hình phát triển nguồn mở và bắt đầu cung cấp hỗ trợ thương mại và các dịch vụ khác. Năm 2013, 10gen đổi tên thành MongoDB Inc.

Vào ngày 20 tháng 10 năm 2017, MongoDB trở thành công ty giao dịch công khai, được niêm yết trên NASDAQ với tên MDB với giá IPO là 24 USD một cổ phiếu.

Vào ngày 8 tháng 11 năm 2018 với bản phát hành ổn định 4.0.4, giấy phép của phần mềm đã thay đổi từ AGPL 3.0 thành SSPL.

Vào ngày 30 tháng 10 năm 2019, MongoDB đã hợp tác với Alibaba Cloud để cung cấp cho khách hàng của Alibaba Cloud giải pháp MongoDB dưới dạng dịch vụ. Khách hàng có thể sử dụng dịch vụ được quản lý từ các trung tâm dữ liệu toàn cầu của Alibaba.

2.4.3 Ưu điểm của MongoDB

Schema linh hoạt:

MongoDB không yêu cầu một schema cố định trước khi lưu trữ dữ liệu. Điều này cho phép dữ liệu có cấu trúc linh hoạt và có thể thay đổi theo thời gian mà không yêu cầu sự can thiệp vào cấu trúc cơ sở dữ liệu.

Dữ liệu dạng json-like (bson):

MongoDB lưu trữ dữ liệu dưới dạng BSON (Binary JSON), một định dạng nhị phân tương tự JSON. Điều này làm cho việc làm việc với dữ liệu linh hoạt và thuận tiện, đồng thời hỗ trợ các loại dữ liệu phong phú.

Khả năng mở rộng ngang (Horizontal scalability):

MongoDB hỗ trợ mô hình mở rộng ngang, giúp dễ dàng mở rộng cơ sở dữ liệu bằng cách thêm máy chủ vào hệ thống. Điều này làm tăng khả năng chịu tải và hiệu suất của hệ thống khi có thêm dữ liệu.

Tìm kiếm phức tạp và indexing hiệu quả:

MongoDB cung cấp khả năng tìm kiếm phức tạp thông qua các truy vấn và indexing. Các index giúp tăng tốc độ truy vấn, giảm độ phức tạp của chúng và cải thiện hiệu suất tìm kiếm.

Hỗ trợ cho dữ liệu lớn và tài liệu phức tạp:

MongoDB được thiết kế để xử lý dữ liệu lớn và tài liệu phức tạp. Dữ liệu có thể được tổ chức trong các tài liệu phong phú với các trường lồng nhau và mảng, cung cấp sự linh hoạt trong cách dữ liệu được biểu diễn.

Tích hợp tốt với ngôn ngữ lập trình:

MongoDB có các driver hỗ trợ cho nhiều ngôn ngữ lập trình, giúp dễ dàng tích hợp với các ứng dụng được phát triển bằng nhiều ngôn ngữ khác nhau.

Cộng đồng lớn và tài trợ chính thức:

MongoDB có một cộng đồng lớn, với nhiều tài nguyên hỗ trợ và thông tin. Nó cũng được tài trợ chính thức bởi MongoDB, Inc., đảm bảo tính ổn định và sự phát triển liên tục.

Hỗ trợ ACID và các chiến lược sao lưu:

MongoDB hỗ trợ các thuộc tính ACID (Atomicity, Consistency, Isolation, Durability) giúp đảm bảo tính nhất quán và an toàn của giao dịch. MongoDB cũng cung cấp các chiến lược sao lưu để bảo vệ dữ liệu khỏi mất mát.

2.4.4 Nhược điểm của MongoDB

Khả năng tăng tải nhờ mở rộng ngang:

Mặc dù MongoDB hỗ trợ mở rộng ngang, nhưng quá trình này có thể gặp khó khăn khi dữ liệu tăng lên đột ngột và cần sự mở rộng. Có thể yêu cầu quản lý phân phối dữ liệu và xử lý transaction một cách cẩn thận để tránh sự phức tạp.

Sử dụng lượng bộ nhớ đáng kể:

MongoDB cần sử dụng lượng bộ nhớ đáng kể để hoạt động hiệu quả. Điều này có thể tạo ra vấn đề nếu ứng dụng của bạn đang chạy trên các máy chủ có tài nguyên hạn chế.

Chưa hỗ trợ Transactions nguồn gốc (Native transactions):

Trước phiên bản MongoDB 4.0, hệ thống này không hỗ trợ transactions nguồn gốc (native transactions) qua nhiều document, điều này có thể tạo ra khó khăn đối với các ứng dụng yêu cầu giao dịch phức tạp.

Thiếu công cụ quản lý và thống kê:

MongoDB thiếu một số công cụ quản lý và thống kê tích hợp mạnh mẽ so với một số hệ quản trị cơ sở dữ liệu quan hệ. Điều này có thể làm cho quá trình giám sát và tối ưu hóa hiệu suất trở nên khó khăn hơn.

Khó khăn khi thực hiện những truy vấn phức tạp:

Mặc dù MongoDB mạnh mẽ khi thực hiện các truy vấn cơ bản, nhưng nếu bạn cần thực hiện các truy vấn phức tạp và lớn, nó có thể gặp khó khăn và yêu cầu việc quản lý chỉ mục một cách cẩn thận.

Vấn đề bảo mật:

Có thể gặp phải vấn đề về bảo mật nếu MongoDB không được cấu hình và triển khai đúng cách. Cần phải thiết lập các biện pháp bảo mật để đảm bảo an toàn cho dữ liệu.

2.5 Tìm hiểu về ReactJS

2.5.1 Tổng quan về ReactJS

ReactJS là một thư viện JavaScript mã nguồn mở được thiết kế bởi Facebook để tạo ra những ứng dụng web hấp dẫn, nhanh và hiệu quả với mã hóa tối thiểu. Mục đích cốt lõi của ReactJS không chỉ khiến cho trang web phải thật mượt mà còn phải nhanh, khả năng mở rộng cao và đơn giản.

Sức mạnh của ReactJS xuất phát từ việc tập trung vào các thành phần riêng lẻ. Chính vì vậy, thay vì làm việc trên toàn bộ ứng dụng web, ReactJS cho phép một lập trình viên có thể chia nhỏ giao diện người dùng phức tạp thành các thành phần đơn giản hơn, giúp việc quản lý dễ dàng hơn.

2.5.2 Lịch sử hình thành ReactJS

ReactJS được tạo ra bởi Jordan Walke, một kỹ sư phần mềm tại Facebook. Người bị ảnh hưởng bởi XHP (Một nền tảng thành phần HTML cho PHP). React lần đầu tiên được triển khai cho ứng dụng Newsfeed của Facebook năm 2011, sau

đó được triển khai cho Instagram năm 2012. Nó được mở mã nguồn (open-sourced) tại JSConf US tháng 5 năm 2013.

React Native (Phiên bản của React dành cho mobile) , cho phép phát triển Android, iOS và UWP gốc với React, đã được công bố tại React Conf của Facebook vào tháng 2 năm 2015 và mã nguồn mở vào tháng 3 năm 2015.

Vào ngày 18 tháng 4 năm 2017, Facebook đã công bố React Fiber, một bộ thuật toán nội bộ mới để kết xuất, trái ngược với thuật toán kết xuất cũ của React, Stack. React Fiber đã trở thành nền tảng của mọi cải tiến trong tương lai và phát triển tính năng của thư viện React. Cú pháp lập trình thực tế với React không thay đổi; chỉ có cách mà cú pháp được thực thi đã thay đổi. Hệ thống kết xuất cũ của React, Stack, được phát triển vào thời điểm mà hệ thống không tập trung vào thay đổi động. Ví dụ, Stack chậm vẽ hoạt ảnh phức tạp khi cố gắng hoàn thành tất cả hoạt ảnh đó trong một đoạn. Sợi chia hoạt ảnh thành các phân đoạn có thể trải rộng trên nhiều khung hình. Tương tự như vậy, cấu trúc của một trang có thể được chia thành các phân đoạn có thể được duy trì và cập nhật riêng biệt. Các hàm JavaScript và đối tượng DOM ảo được gọi là "sợi" và mỗi hàm có thể được vận hành và cập nhật riêng biệt, cho phép hiển thị trên màn hình mượt mà hơn.

Vào ngày 26 tháng 9 năm 2017, React 16.0 đã được phát hành ra công chúng.

Vào ngày 10 tháng 8 năm 2020, nhóm React đã công bố ứng cử viên phát hành đầu tiên cho React v17.0, đáng chú ý là bản phát hành chính đầu tiên không có thay đổi lớn đối với API dành cho nhà phát triển React.

Vào ngày 29 tháng 3 năm 2022, React 18 đã được phát hành, giới thiệu trình kết xuất đồng thời mới, tạo nhóm tự động và hỗ trợ kết xuất phía máy chủ với Suspense.

2.5.3 Ưu điểm của ReactJS

ReactJS có những ưu điểm như sau:

- Vì ReactJS sử dụng DOM ảo để cache cấu trúc dữ liệu trong bộ nhớ và chỉ những thay đổi cuối cùng mới được cập nhật vào trong DOM trình duyệt. Điều này làm cho ứng dụng trở nên nhanh hơn.

- Chúng ta có thể tạo các component theo từng chức năng mà chúng ta muốn bằng cách sử dụng tính năng react component. Các component này có thể tái sử dụng khi cần thiết, đồng thời việc tạo các component theo từng chức năng cũng giúp cho việc bảo trì sau này trở nên dễ dàng hơn.

- ReactJS là một opensource, vì vậy cũng rất dễ cho tiếp cận với những người mới bắt đầu tìm hiểu.

- Trong những năm gần đây, ReactJS đang trở nên phổ biến hơn và được duy trì bởi Facebook và Instagram. Ngoài ra nó cũng được sử dụng bởi các công ty nổi tiếng như Apple, Netflix, …

- Facebook vẫn đang duy trì, phát triển, và cho ra những thay đổi mới.

- ReactJS có thể được sử dụng để xây dựng giao diện người dùng cho cả các ứng dụng dành cho máy tính và các ứng dụng di động.

- Dễ dàng cho việc tìm và sửa lỗi, vì hầu hết các mã nguồn đều được thực hiện bằng JavaScript chứ không phải bằng HTML.

2.5.4 Nhược điểm của ReactJS

- Vì hầu hết code được viết dưới dạng JSX, tức là HTML và CSS là một phần của JavaScript, nó không giống như những framework khác vẫn tách biệt giữa HTML và CSS nên những người mới làm quen với ReactJS sẽ hơi lúng túng và dễ nhầm lẫn giữa JSX và HTML.

- Một nhược điểm nữa của ReactJS đó là dung lượng các file khi ở giai đoạn phát triển khá lớn.

CHƯƠNG 3: HIỆN THỰC HÓA NGHIÊN CỨU

3.1 Mô tả bài toán

Khách hàng cần xây dựng trang web trình chiếu lời bài hát Reflow với các chức năng: đăng nhập, đăng ký, xem danh sách các bài hát, trình chiếu lời bài hát, thay đổi thứ tự các đoạn trong bài hát, lưu danh sách những bài hát yêu thích,…

Bài hát có các thông tin như mã bài hát, tên bài hát, ca sĩ thể hiện, thể loại, ảnh bìa,… Mỗi bài hát gồm nhiều đoạn nhỏ, mỗi đoạn có một tên gọi riêng (Verse, Chorus, Bridge, Rap,…) cùng với lời bài hát thuộc đoạn đó.

Một bài hát thuộc vào một thể loại, thông tin thể loại gồm có mã thể loại, tên thể loại, ảnh. Một bài hát do một ca sĩ trình bày, thông tin ca sĩ gồm mã ca sĩ, tên ca sĩ, ảnh ca sĩ.

Người dùng khi đăng nhập vào hệ thống có thể thêm các bài hát vào danh sách yêu thích để tiện theo dõi hoặc xoá chúng khỏi danh sách yêu thích nếu muốn thay đổi. Người dùng cũng có thể tuỳ chỉnh được các thông tin cá nhân của họ. Thông tin của người dùng gồm có mã người dùng, tên người dùng, email, mật khẩu, phân quyền,…

Ngoài ra, người dùng được phân theo hai loại, người dùng miễn phí và người dùng Premium. Một số chức năng chỉ người dùng Premium mới được sử dụng như: thay đổi nền trình chiếu theo các nền đặc biệt, thay đổi thứ tự các đoạn của bài hát.

Người quản trị có toàn quyền trên hệ thống, có thể thực hiện các chức năng như thống kế, quản lý người dùng, quản lý bài hát, quản lý thể loại, quản lý ca sĩ,…

3.2 Đặc tả ứng dụng

Hình 2. Sơ đồ Use-case của Reflow

*** Mô tả các Actor:**

STT	Tên Actor	Ý nghĩa
1	Khách	Người dùng thông thường (chưa đăng nhập)
2	Người dùng	Người dùng đã đăng nhập
3	Admin	Người quản trị toàn bộ hệ thống

*** Mô tả các Use-case:**

STT	Tên use-case	Ý nghĩa
1	Xem danh sách bài hát	Người dùng có thể xem danh sách các bài hát có trên trang web và chọn xem thông tin của chúng.
2	Trình chiếu bài hát	Người dùng có thể trình chiếu lời bài hát lên màn hình, cùng các thao tác điều khiển.
3	Quản lý danh sách yêu thích	Người dùng có thể thêm một bài hát bất kỳ vào danh sách yêu thích để tiện theo dõi, và cũng có thể xoá chúng ra khỏi danh sách.
4	Tuỳ chỉnh thông tin cá nhân	Người dùng có thể điều chỉnh các thông tin cá nhân như tên, mật khẩu, ảnh đại diện,…
5	Đăng nhập/Đăng ký	Người dùng có thể đăng nhập vào tài khoản của mình hoặc đăng ký tài khoản mới.
6	Thống kê	Người quản trị có thể xem thống kê tổng quan của trang web như số lượt truy cập, số người dùng đã đăng ký, doanh thu,…
7	Quản lý người dùng	Người quản trị có thể quản lý các thông tin của người dùng trên trang web, thêm, sửa, xoá nếu cần thiết.
8	Quản lý thể loại	Người quản trị có thể quản lý các thông tin về thể loại.
9	Quản lý bài hát	Người quản trị có thể quản lý các thông tin về bài hát, thêm, sửa, xoá bài hát, quản lý các đoạn trong một bài hát.
10	Quản lý ca sĩ	Người quản trị có thể quản lý các thông tin của ca sĩ.

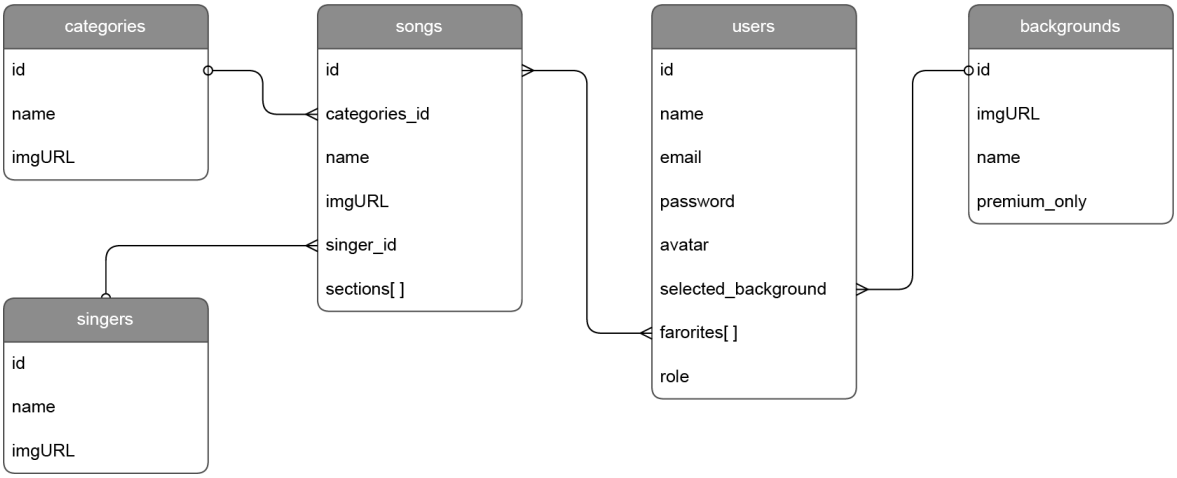
Ứng dụng được xác định có 3 vai trò người dùng gồm: Khách truy cập (chưa thực hiện đăng nhập), người dùng đã đăng nhập và người quản trị (Admin). Các người dùng sẽ có các đặc quyền tương ứng:

- Vai trò khách: Được sử dụng một số chức năng như xem các thông tin trên trang web như thể loại, bài hát, ca sĩ,…Được trình chiếu bài hát với một số chức năng bị giới hạn.

- Vai trò người dùng: Được thực hiện các chức năng mà vai trò khách có thể thực hiện, và thêm các chức năng cá nhân như danh sách yêu thích, nâng cấp tài khoản, tuỳ chỉnh thông tin tài khoản,…Ngoài ra, người dùng còn được chia làm hai loại là người dùng Free và người dùng Premium. Người dùng Free sẽ bị giới hạn một số chức năng mà người dùng Premium có thể sử dụng.

- Vai trò người quản trị: Là loại đặc quyền cao nhất, có tất cả các chức năng của hai loại người dùng trên. Ngoài ra, người quản trị có thể quản trị toàn bộ hệ thống với các chức năng như quản lý người dùng, quản lý bài hát, quản lý thể loại, quản lý ca sĩ, thống kê,…

3.3 Mô hình dữ liệu



Hình 3. Sơ đồ cơ sở dữ liệu của Reflow

*** Mô tả bảng categories:**

Lưu dữ liệu thông tin của các thể loại.

STT	Tên thuộc tính	Kiểu dữ liệu	Ý nghĩa
1	id	ObjectId (MongoDB)	Mã thể loại

2	name	String	Tên thể loại
3	imgURL	String	Đường dẫn hình ảnh của thể loại

*** Mô tả bảng singers:**

Lưu dữ liệu thông tin của các ca sĩ.

STT	Tên thuộc tính	Kiểu dữ liệu	Ý nghĩa
1	id	ObjectId (MongoDB)	Mã ca sĩ
2	name	String	Tên ca sĩ
3	imgURL	String	Đường dẫn hình ảnh của ca sĩ

*** Mô tả bảng backgrounds:**

Lưu dữ liệu các hình nền của trang trình chiếu.

STT	Tên thuộc tính	Kiểu dữ liệu	Ý nghĩa
1	id	ObjectId (MongoDB)	Mã hình nền
2	name	String	Tên hình nền
3	imgURL	String	Đường dẫn hình ảnh của hình nền
4	premium_only	Boolean	Hình nền có dành cho người dùng Premium hay không

*** Mô tả bảng users:**

Lưu dữ liệu thông tin của các người dùng.

STT	Tên thuộc tính	Kiểu dữ liệu	Ý nghĩa
1	id	ObjectId (MongoDB)	Mã người dùng
2	name	String	Tên người dùng
3	email	String	Email của người dùng
4	password	String	Lưu lại thông tin mật khẩu (đã hash) của người dùng
5	role	Enum [“Free”, “Premium”, “Admin”]	Phân quyền của người dùng
6	avatar	String	Đường dẫn đến ảnh đại diện của người dùng
7	favorites	Array	Danh sách các bài hát mà người dùng yêu thích
8	selected_background	ObjectId (MongoDB)	Mã hình nền mà người dùng đã chọn làm hình nền trình chiếu

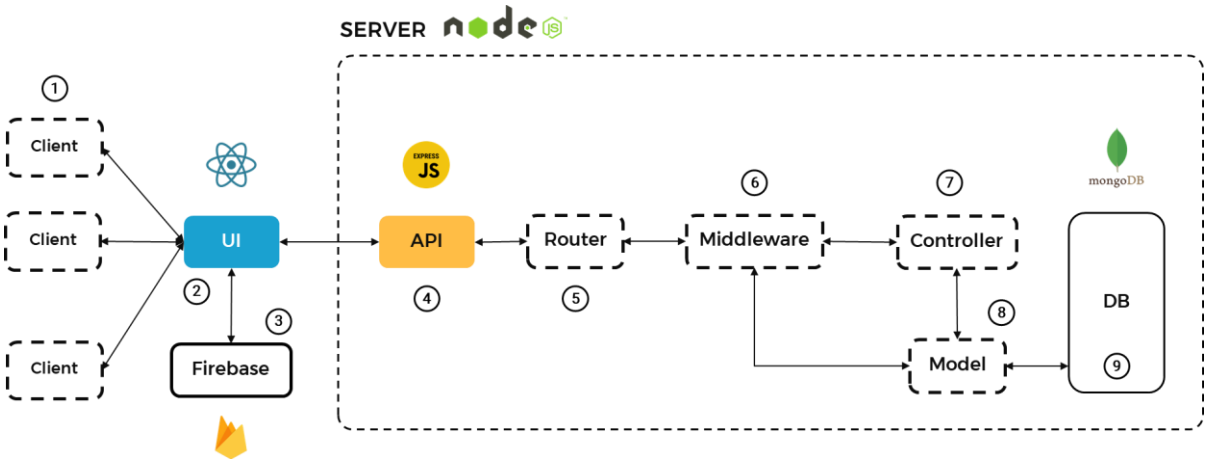
*** Mô tả bảng songs:**

Lưu dữ liệu thông tin của các bài hát.

STT	Tên thuộc tính	Kiểu dữ liệu	Ý nghĩa
1	id	ObjectId (MongoDB)	Mã bài hát
2	name	String	Tên bài hát
3	categoryId	ObjectId (MongoDB)	Mã thể loại của bài hát
4	singerId	ObjectId	Mã ca sĩ của bài hát

		(MongoDB)	
5	imgURL	String	Đường dẫn hình ảnh của bài hát
6	sections	Array	Chứa danh sách các đoạn trong bài hát, bao gồm lời bài hát và tên của từng đoạn

3.4 Kiến trúc hệ thống



Hình 4. Kiến trúc hệ thống của Reflow

Trong đó:

(1): Các người dùng, trình duyệt.

(2): Giao diện người dùng (ReactJS).

(3): Dịch vụ lưu trữ đám mây (Firebase), chứa các file tĩnh, ít thay đổi như hình ảnh, âm thanh, video,…

(4): Các API từ phía Server (ExpressJS).

(5): Các định tuyến, dùng để truy xuất đúng các API theo đường dẫn.

(6): Middleware, chứa các hàm dùng để xác thực người dùng, xác thực yêu cầu được gửi đến API.

(7): Controller, chứa các hàm xử lý theo từng yêu cầu khác nhau.

(8): Model, vùng giao tiếp trung gian giữa Database và Server, tránh việc truy xuất trực tiếp dữ liệu từ Database.

(9): Database (MongoDB), nơi lưu trữ toàn bộ dữ liệu của trang web.

Luồng xử lý của hệ thống:

Khi người dùng truy cập vào trang web (1), giao diện người dùng (2) sẽ yêu cầu Firebase (3) trả về các file tĩnh và hiển thị ra bên ngoài. Nếu cần thiết, giao diện người dùng có thể thực hiện việc gọi đến các API (4), yêu cầu này sau đó được định tuyến bởi Router (5), thông qua bước xác thực bằng Middleware (6). Nếu xác thực thành công, yêu cầu sẽ tiếp tục được đưa đến Controller (7) tương ứng để xử lý. Tại đây, Model (8) sẽ thực hiện kết nối với Database (9) để đọc và ghi dữ liệu. Dữ liệu sau đó được trả về giao diện người dùng hiển thị ra bên ngoài.

3.5 Xây dựng Back-end

3.5.1 Định nghĩa các Model

* Định nghĩa Model Category:

```
const mongoose = require('mongoose');

const categorySchema = new mongoose.Schema({
  name: { type: String, required: true },
  image: { type: String },
});

const Category = mongoose.model('Category', categorySchema);

module.exports = Category;
```

* Định nghĩa Model Singer:

```
const singerSchema = new mongoose.Schema({
  name: { type: String, required: true },
  image: { type: String },
});

const Singer = mongoose.model('Singer', singerSchema);

module.exports = Singer;
```

* Định nghĩa Model Background:

```
const backgroundSchema = new mongoose.Schema({
  name: { type: String, required: true },
  image: { type: String },
  premium_only: { type: Boolean },
});

const Background = mongoose.model('Background', backgroundSchema);

module.exports = Background;
```

* Định nghĩa Model User:

```
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true },
  username: { type: String, required: true },
  password: { type: String, required: true },
  avatar: { type: String, required: true },
  selected_backround: { type: mongoose.Schema.Types.ObjectId, ref:
'Background'},
  role: { type: String, enum: ['Free', 'Premium', 'Admin'], required:
true },
  favorites: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Song' }],
});

const User = mongoose.model('User', userSchema);

module.exports = User;
```

* Định nghĩa Model Song:

```
const mongoose = require('mongoose');

const sectionSchema = new mongoose.Schema({
  name: { type: String, required: true },
  lyrics: { type: String, required: true },
  order: { type: Number, required: true },
});

const songSchema = new mongoose.Schema({
  title: { type: String, required: true },
  sections: [sectionSchema],
  categoryId: { type: mongoose.Schema.Types.ObjectId, ref: 'Category'},
  singerId: { type: mongoose.Schema.Types.ObjectId, ref: 'Singer' },
  image: { type: String },
});

const Song = mongoose.model('Song', songSchema);

module.exports = Song;
```

3.5.2 Định nghĩa các Controller

Controller chứa các hàm xử lý riêng cho từng công việc, lấy ví dụ với Song Controller (Định nghĩa các hàm thêm, sửa, xoá các bài hát, đoạn bài hát).

Controller lấy ra tất cả bài hát:

```
const getAllSongs = async (req, res) => {
    try {
        const songs = await Song.find()
            .populate('categoryId', 'name')
            .populate('singerId', 'name');

        res.json(songs);
```

```
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: 'Internal Server Error' });
    }
};
```

Controller lấy ra bài hát theo ID bài hát:

```
const getSongById = async (req, res) => {
    try {
        const { id } = req.params;

        if (!id.match(/^[0-9a-fA-F]{24}$/)) {
            return res.status(400).json({ message: 'Invalid song ID
format' });
        }

        const song = await Song.findById(id)
            .populate('categoryId', 'name')
            .populate('singerId', 'name');

        if (!song) {
            return res.status(404).json({ message: 'Song not found' });
        }

        res.json(song);
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: 'Internal Server Error' });
    }
};
```

Controller tạo bài hát mới:

```
const createSong = async (req, res) => {
    const { title, sections, categoryId, singerId, image } = req.body;

    try {
        const newSong = new Song({
            title,
            sections,
            categoryId,
            singerId,
            image,
        });

        await newSong.save();

        res.status(201).json({ message: 'Song created successfully',
song: newSong });
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: 'Internal Server Error' });
    }
};
```

Controller xoá bài hát theo danh sách ID:

```
const deleteSongsByIds = async (req, res) => {
    try {
        const { songIds } = req.body;

        if (!Array.isArray(songIds) || songIds.some(id =>
!id.match(/^[0-9a-fA-F]{24}$/))) {
            return res.status(400).json({ message: 'Invalid song IDs
format' });
        }

        const songs = await Song.find({ _id: { $in: songIds } });

        if (songs.length !== songIds.length) {
            return res.status(404).json({ message: 'One or more songs
not found' });
        }

        await Song.deleteMany({ _id: { $in: songIds } });

        res.json({ message: 'Songs deleted successfully' });
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: 'Internal Server Error' });
    }
};
```

Controller cập nhật bài hát theo ID:

```
const updateSongById = async (req, res) => {
    try {
        const { id } = req.params;

        if (!id.match(/^[0-9a-fA-F]{24}$/)) {
            return res.status(400).json({ message: 'Invalid song ID
format' });
        }

        const song = await Song.findById(id);

        if (!song) {
            return res.status(404).json({ message: 'Song not found' });
        }

        if (req.body.title) {
            song.title = req.body.title;
        }

        if (req.body.sections) {
            song.sections = req.body.sections;
        }

        if (req.body.categoryId) {
            song.categoryId = req.body.categoryId;
        }
```

```
        if (req.body.singerId) {
            song.singerId = req.body.singerId;
        }

        if (req.body.image) {
            song.image = req.body.image;
        }

        await song.save();

        res.json(song);
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: 'Internal Server Error' });
    }
};
```

Các Controller khác định nghĩa tương tự.

3.5.3 Định nghĩa Middleware

Middleware là nơi để kiểm tra xác thực của người dùng, kiểm tra xem người dùng gửi yêu cầu API có phân quyền hợp lệ hay không, là thành phần quan trọng trong việc bảo mật hệ thống sử dụng Restful API.

Middleware kiểm tra người dùng đã đăng nhập chưa:

```
const authenticate = (req, res, next) => {
    const token = req.cookies.token;

    if (!token) {
        return res.status(401).json({ message: 'Unauthorized - No token
provided' });
    }

    try {
        const secretKey = "REFLOW";

        const decoded = jwt.verify(token, secretKey);

        req.user = decoded.user;

        next();
    } catch (err) {
        console.error(err);
        return res.status(401).json({ message: 'Unauthorized - Invalid
token' });
    }
};
```

Middleware kiểm tra người dùng có phải là Admin hay không:

```
const isAdmin = (req, res, next) => {

    if (!req.user || req.user.role !== 'Admin') {
```

```
        return res.status(403).json({ message: 'Forbidden - Admin access
required' });
    }

    next();
};
```

3.5.4 Định nghĩa Router

Router chính của trang web, khi API khớp với đường dẫn nào, app sẽ tự động chuyển sang route tương ứng để xử lý.

```
app.use('/api/auth', require('./routes/auth'));
app.use('/api/songs', require('./routes/song'));
app.use('/api/users', require('./routes/user'));
app.use('/api/categorys', require('./routes/category'));
app.use('/api/singers', require('./routes/singer'));
```

Cấu trúc định tuyến trong routes/auth (chứa các API xử lý xác thực người dùng, đăng nhập, đăng ký,…):

```
// [POST] /api/auth/register
router.post('/register', authController.register);

// [POST] /api/auth/login
router.post('/login', authController.login);

// [GET] /api/auth/logout
router.get('/logout', authController.logout);
```

Cấu trúc định tuyến trong routes/song (chứa các API xử lý việc thêm, sửa, xoá bài hát, đoạn bài hát,…):

```
router.get('/', songController.getAllSongs);

router.post('/', authMiddleware.authenticate, authMiddleware.isAdmin,
songController.createSong);

router.delete('/multiple', authMiddleware.authenticate,
authMiddleware.isAdmin, songController.deleteSongsByIds);

router.delete('/:id/sections/multiple', authMiddleware.authenticate,
authMiddleware.isAdmin, songController.deleteSectionsByIds);

router.get('/:songId/sections/:sectionId',
songController.getSectionById);

router.put('/:songId/sections/:sectionId', authMiddleware.authenticate,
authMiddleware.isAdmin, songController.updateSectionById);

router.get('/:id', songController.getSongById);

router.put('/:id', authMiddleware.authenticate, authMiddleware.isAdmin,
songController.updateSongById);
```

```
router.post('/addSection/:id', authMiddleware.authenticate,
authMiddleware.isAdmin, songController.addSectionToSong);
```

Cấu trúc định tuyến trong routes/user (chứa các API xử lý việc thêm, sửa, xoá người dùng):

```
router.get('/', authMiddleware.authenticate, authMiddleware.isAdmin,
userController.getAllUsers);

router.get('/info', authMiddleware.authenticate,
userController.getCurrentUser);

router.get('/favorites', authMiddleware.authenticate,
userController.getFavoriteSongs);

router.post('/favorites', authMiddleware.authenticate,
userController.addFavoriteSong);

router.delete('/favorites/multiple', authMiddleware.authenticate,
userController.removeFavoriteSongsByIds);

router.delete('/favorites/:id', authMiddleware.authenticate,
userController.removeFavoriteSongById);

router.get('/:id', authMiddleware.authenticate,
userController.getUserById);

router.put('/:id', authMiddleware.authenticate,
userController.updateUserById);

router.delete('/multiple', authMiddleware.authenticate,
authMiddleware.isAdmin, userController.deleteUsersByIds);

router.delete('/:id', authMiddleware.authenticate,
authMiddleware.isAdmin, userController.deleteUserById);
```

Cấu trúc định tuyến trong routes/category (chứa các API xử lý việc thêm, sửa, xoá thể loại):

```
router.get('/', categoryController.getAllCategories);

router.post('/', authMiddleware.authenticate, authMiddleware.isAdmin,
categoryController.createCategory);

router.get('/:id', categoryController.getCategoryById);

router.put('/:id', authMiddleware.authenticate, authMiddleware.isAdmin,
categoryController.updateCategoryById);

router.delete('/:id', authMiddleware.authenticate,
authMiddleware.isAdmin, categoryController.deleteCategoryById);
```

Cấu trúc định tuyến trong routes/singer (chứa các API xử lý việc thêm, sửa, xoá ca sĩ):

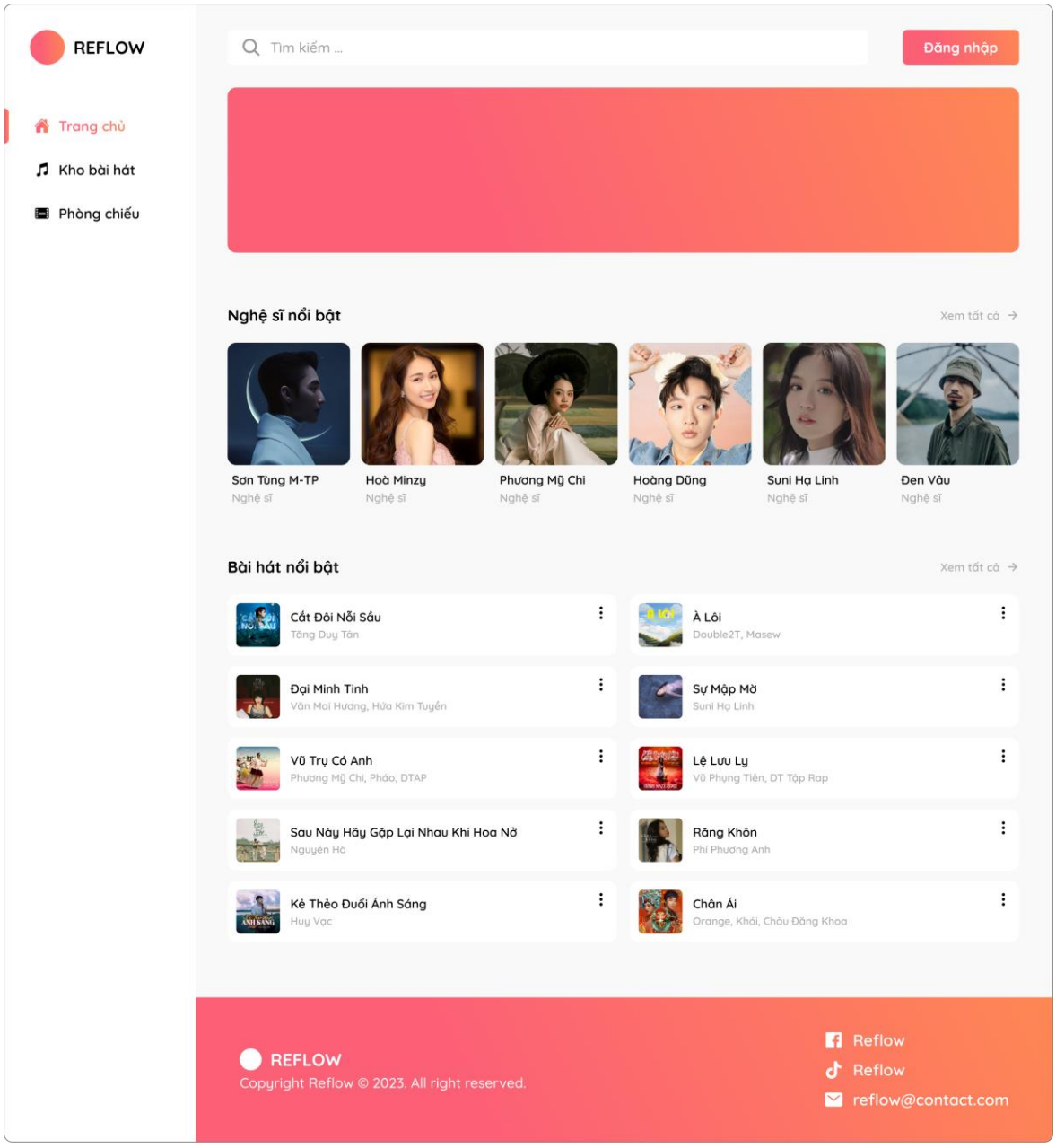
```
router.get('/', singerController.getAllSingers);

// [POST] /api/singers
router.post('/', authMiddleware.authenticate, authMiddleware.isAdmin,
singerController.createSinger);

// [GET] /api/singers/:id
router.get('/:id', singerController.getSingerById);

// [PUT] /api/singers/:id
router.put('/:id', authMiddleware.authenticate, authMiddleware.isAdmin,
singerController.updateSingerById);

// [DELETE] /api/singers/:id
router.delete('/:id', authMiddleware.authenticate,
authMiddleware.isAdmin, singerController.deleteSingerById);
```

3.6 Thiết kế giao diện

3.6.1 Giao diện Trang chủ

Trang web được thiết kế với một thanh điều hướng bên trái, và phần nội dung bên phải. Phía trên là thanh tìm kiếm và các nút chức năng.

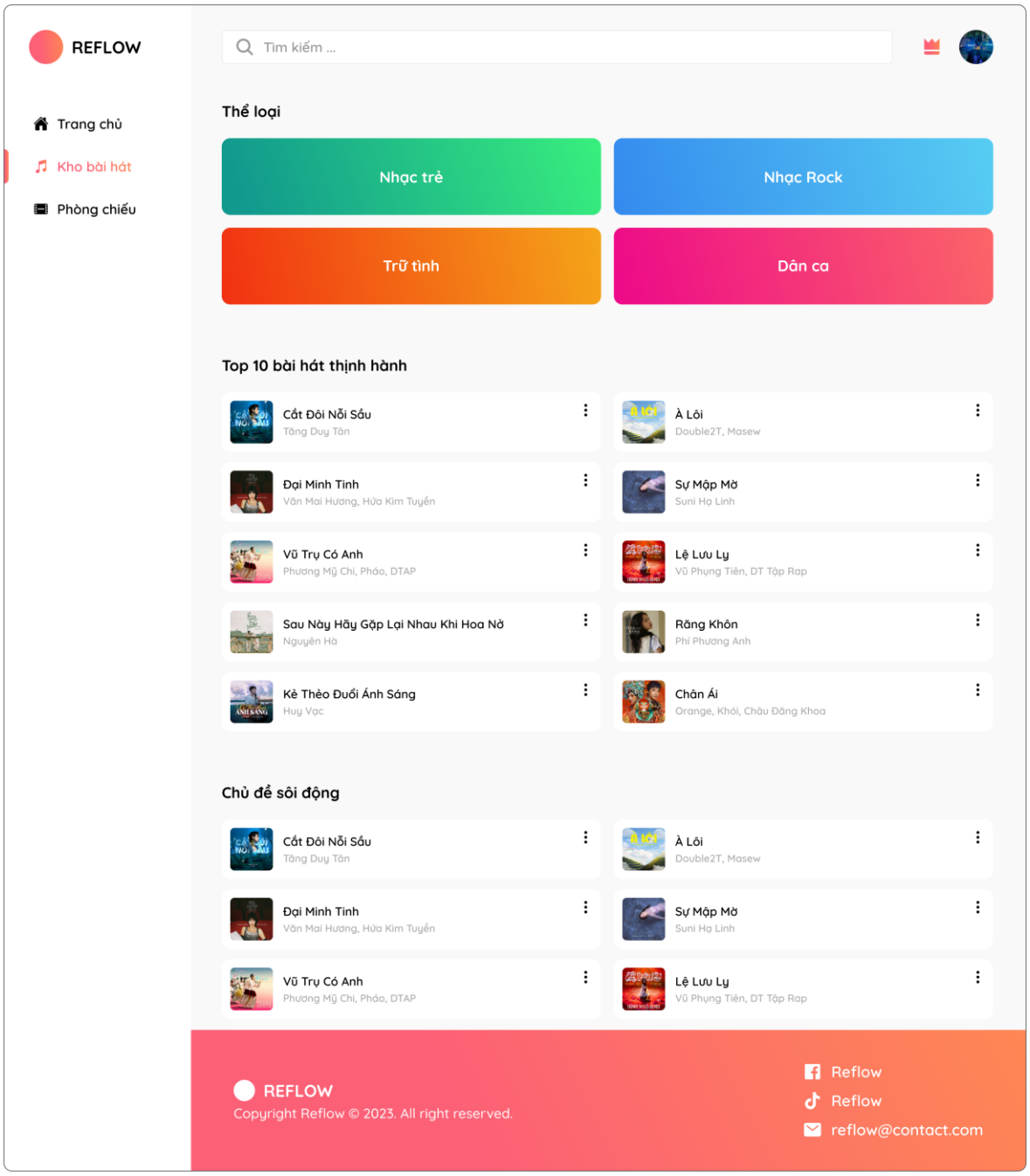
Trang chủ hiển thị các thông tin nổi bật của trang web như Banner quản cáo, các ca sĩ nổi bật, các bài hát nổi bật được đề xuất,…



Hình 5. Thiết kế giao diện Trang chủ

3.6.2 Giao diện trang "Kho bài hát"

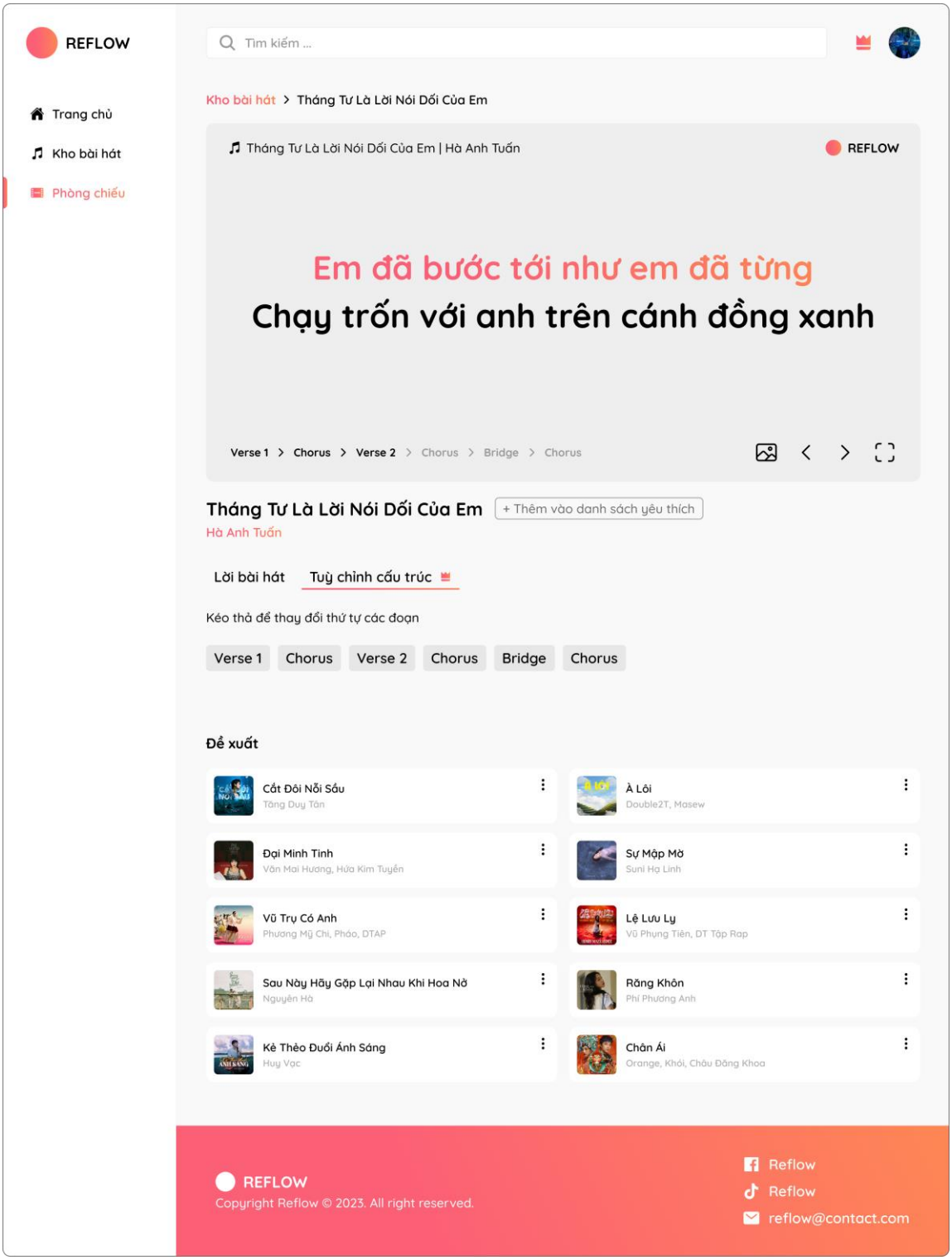
Kho bài hát chứa toàn bộ bài hát của trang web, được phân theo các thể loại như nhạc trẻ, nhạc rock, nhạc trữ tình, dân ca,…



Hình 6. Thiết kế giao diện Kho bài hát

3.6.3 Giao diện trang "Phòng chiếu"

Giao diện Phòng chiếu trình chiếu lời bài hát của một bài hát cụ thể. Người dùng có thể theo dõi trên màn hình, phóng to màn hình, di chuyển sang câu kế tiếp hoặc trước đó, thay đổi hình nền của khung chiếu,…Ngoài ra, đối với người dùng Premium, có thể thay đổi thứ tự các đoạn hiển thị của bài hát.



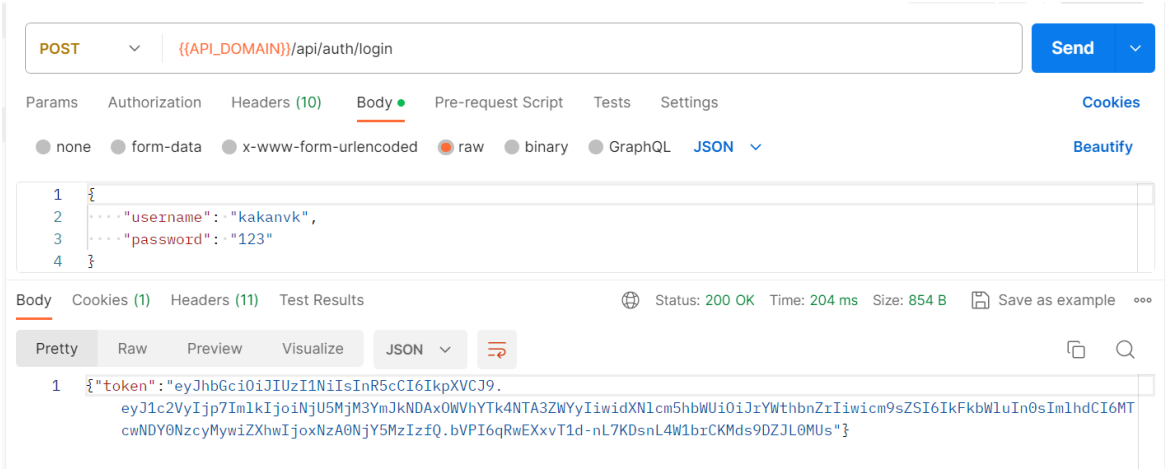
Hình 7. Thiết kế giao diện Phòng chiếu

CHƯƠNG 4: KẾT QUẢ NGHIÊN CỨU

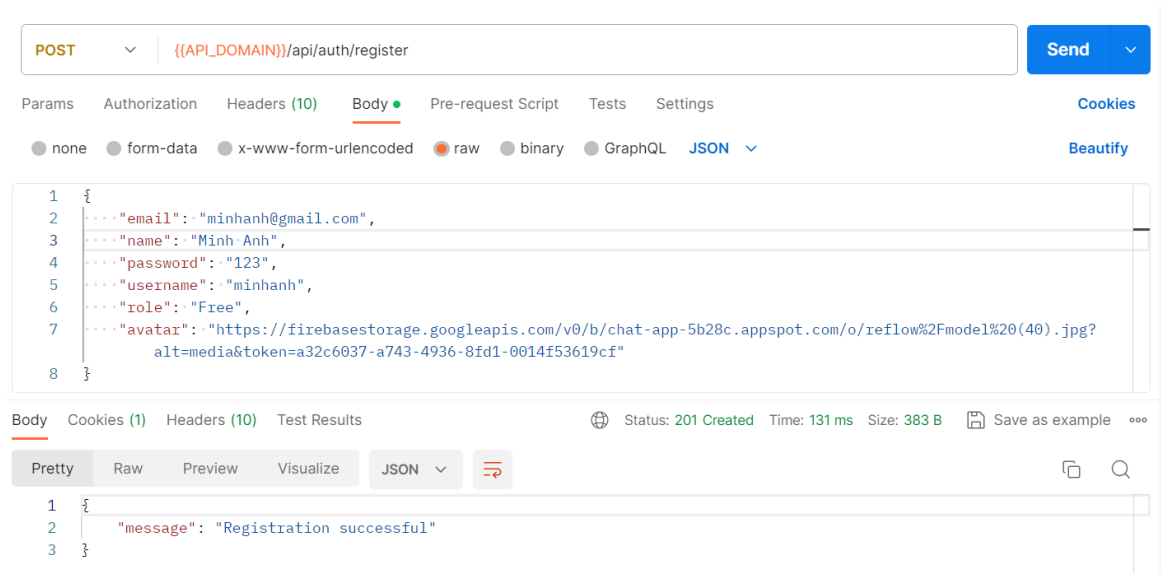
4.1 Thử nghiệm các API với Postman

4.1.1 Các API xác thực

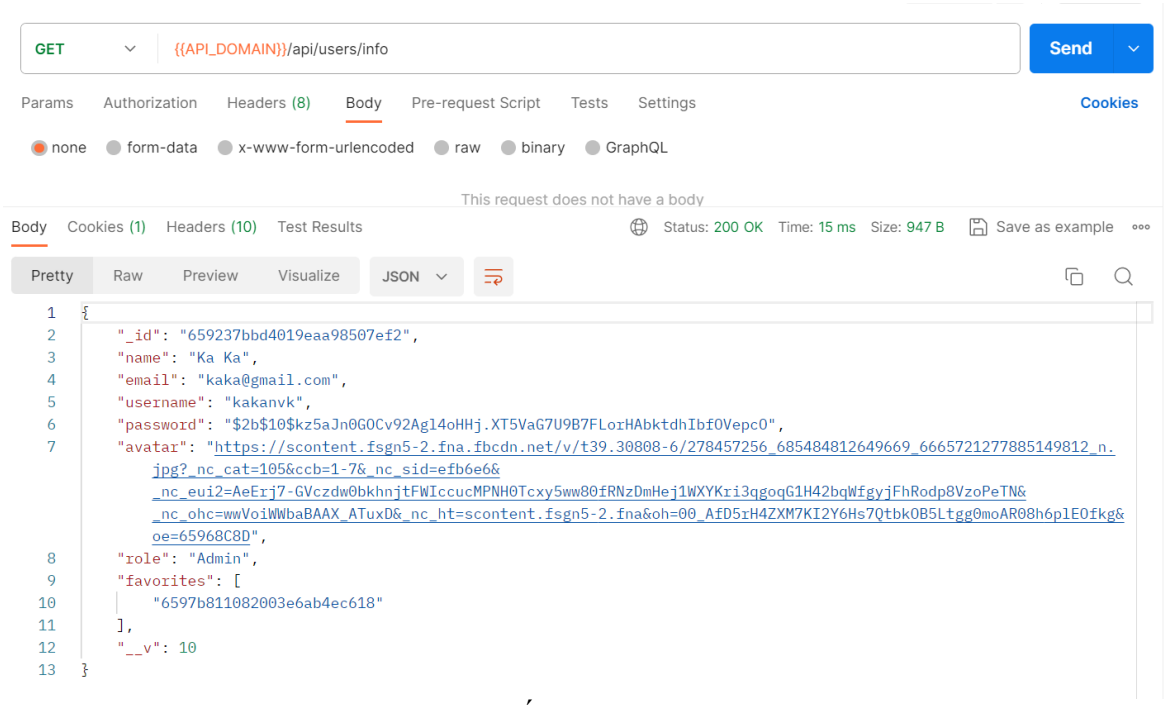
API đăng nhập với tài khoản và mật khẩu, dữ liệu trả về là một chuỗi Token mã hoá thông tin của người dùng. Dữ liệu này được tự động lưu vào Cookies, mỗi khi thực hiện các yêu cầu khác, Token sẽ tự động được gửi kèm với yêu cầu, giúp quá trình xác thực được dễ dàng.



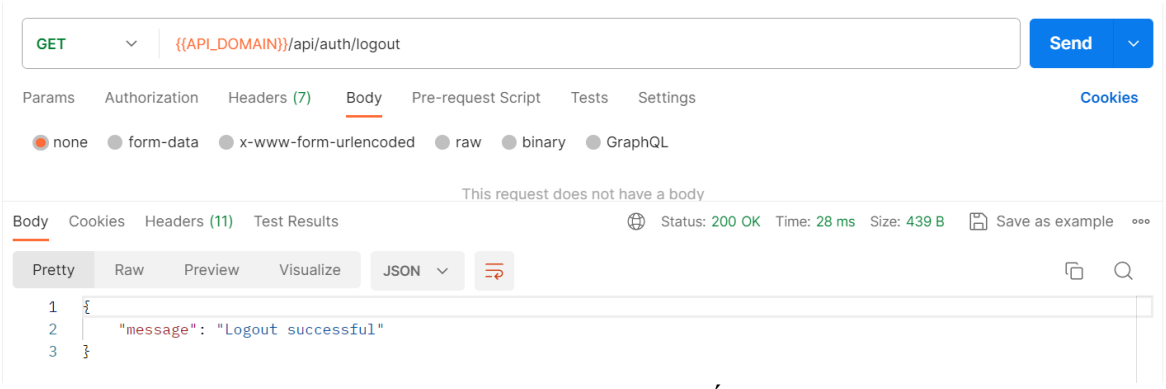
Hình 8. Thử nghiệm API đăng nhập với Postman



Hình 9. Thử nghiệm API đăng ký tài khoản mới với Postman

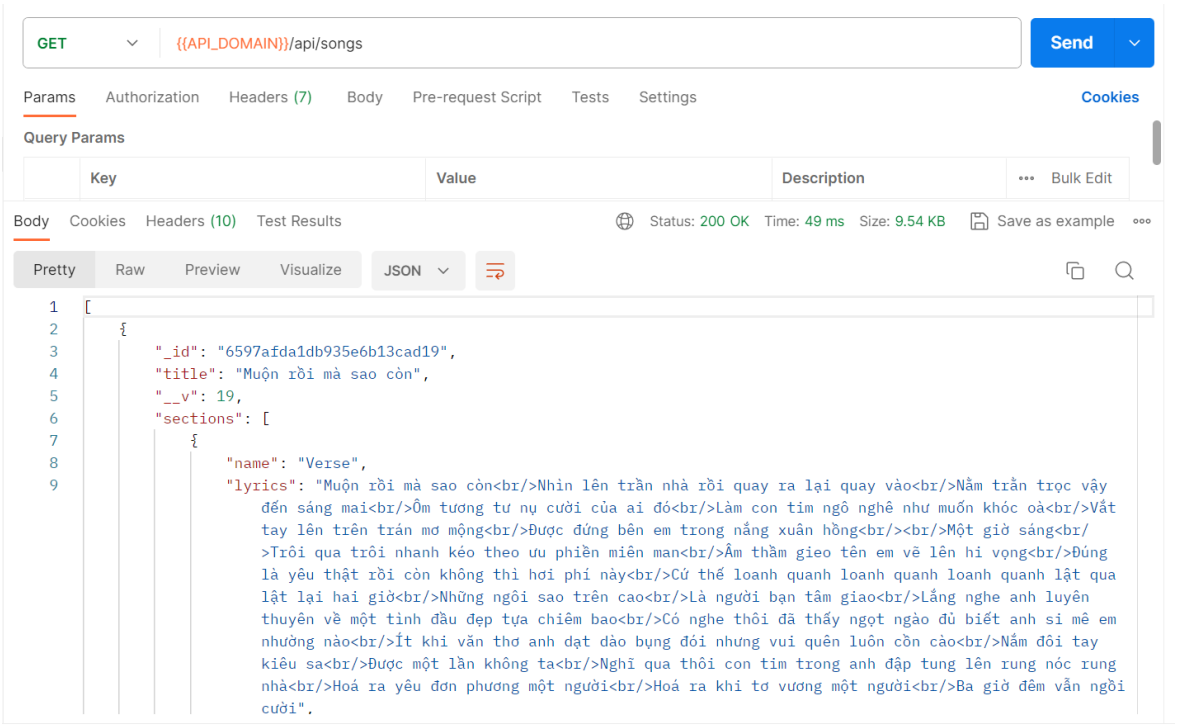


Hình 10. Thử nghiệm API lấy ra người dùng hiện tại với Postman

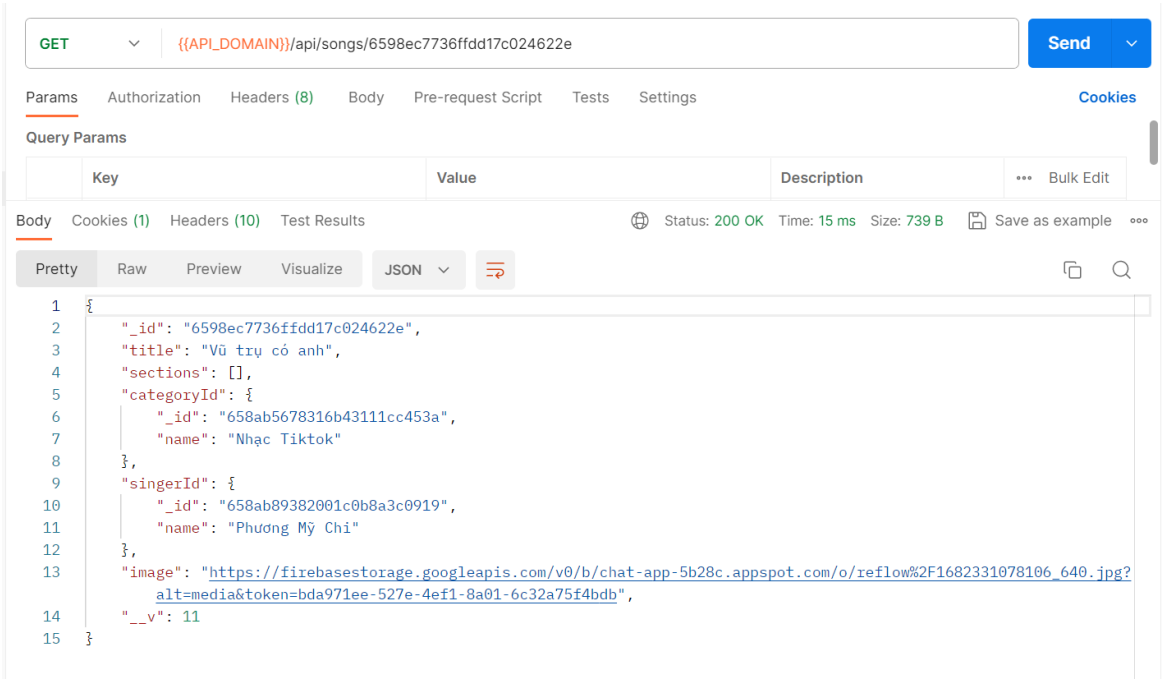


Hình 11. Thử nghiệm API đăng xuất với Postman

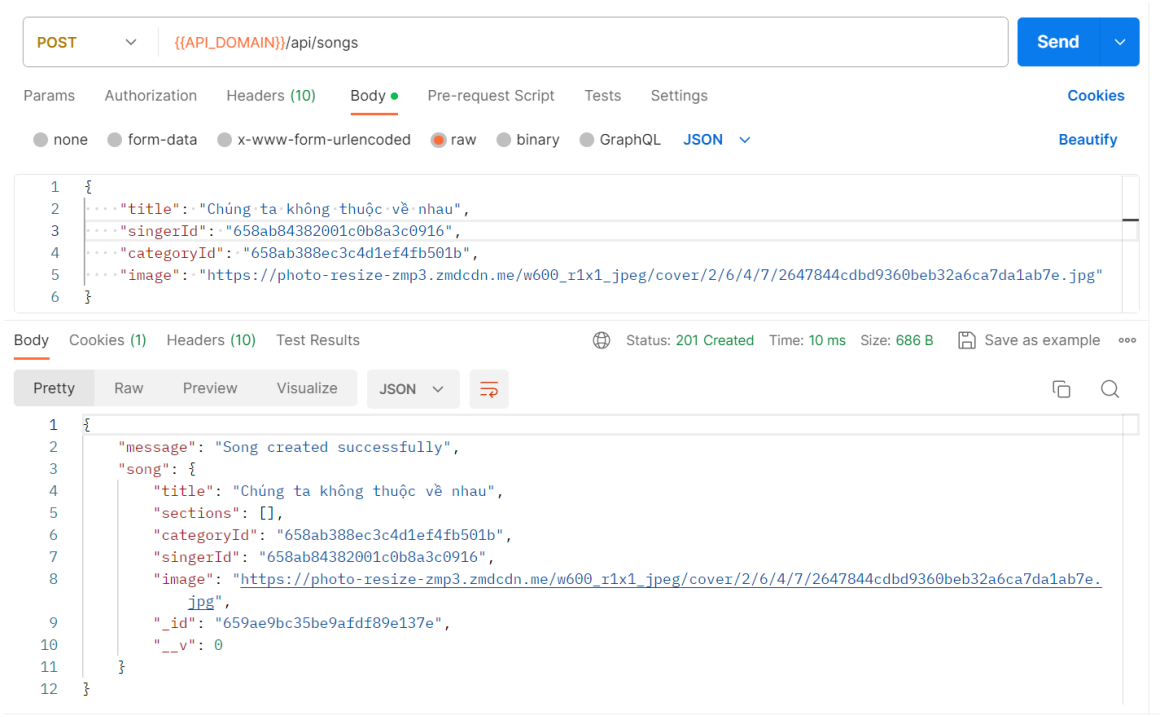
4.1.2 Các API liên quan đến bài hát



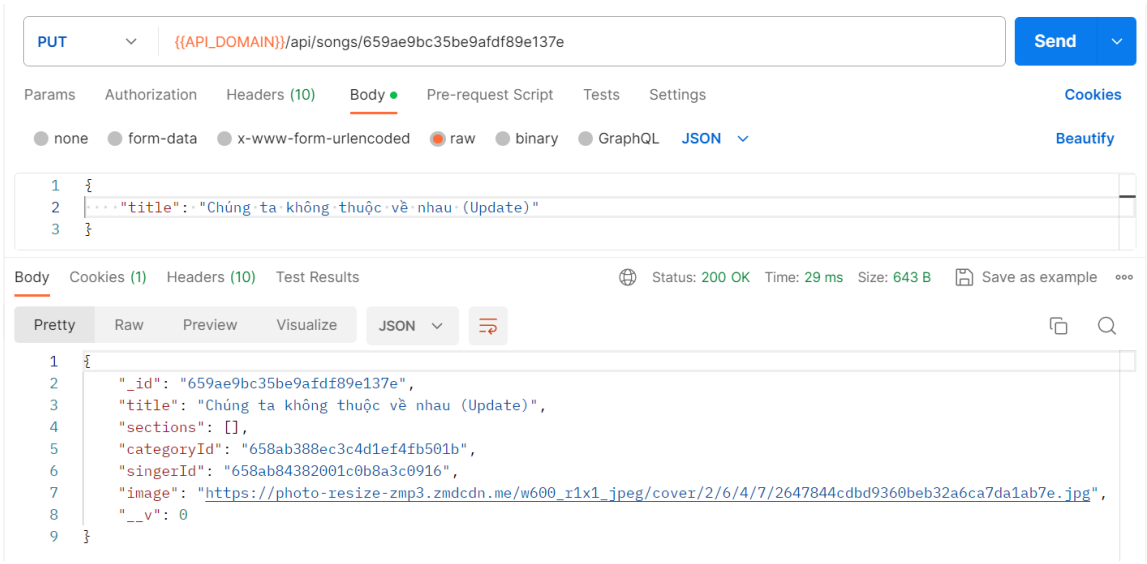
Hình 12. Thử nghiệm API lấy ra danh sách bài hát với Postman



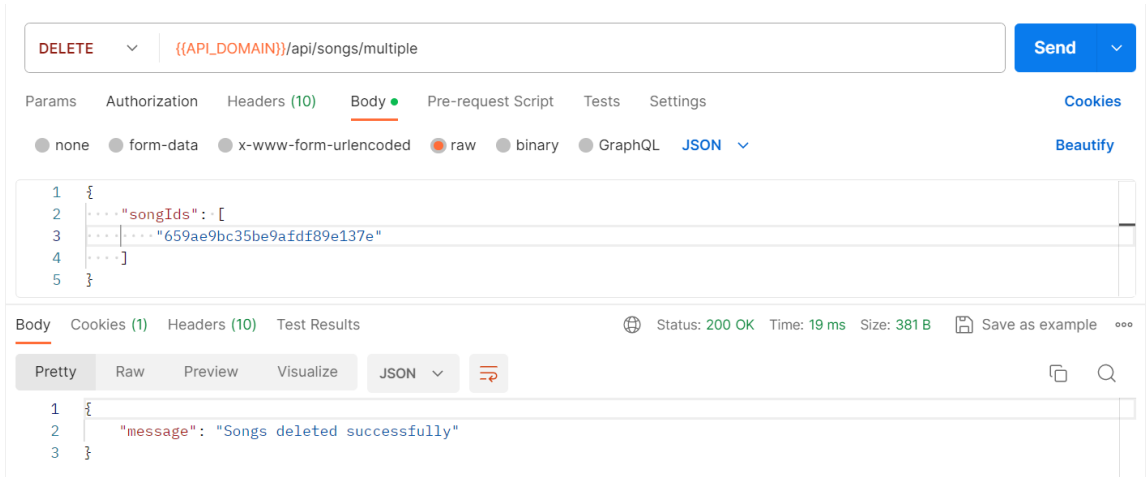
Hình 13. Thử nghiệm API lấy ra thông tin bài hát theo ID với Postman



Hình 14. Thử nghiệm API tạo bài hát mới với Postman

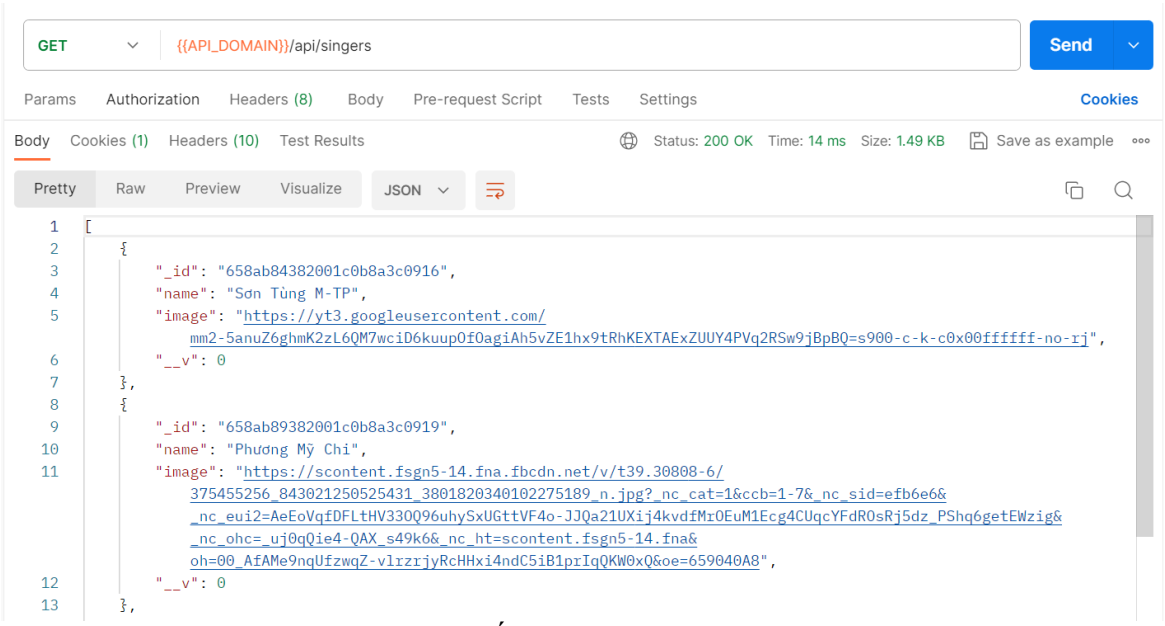


Hình 15. Thử nghiệm API cập nhật thông tin bài hát theo ID với Postman

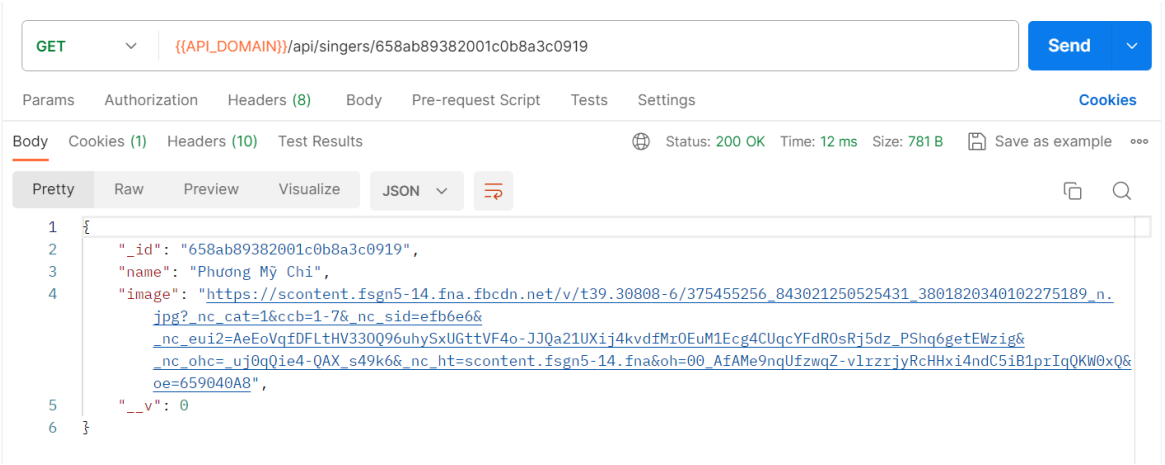


Hình 16. Thử nghiệm API xoá bài hát theo danh sách ID với Postman

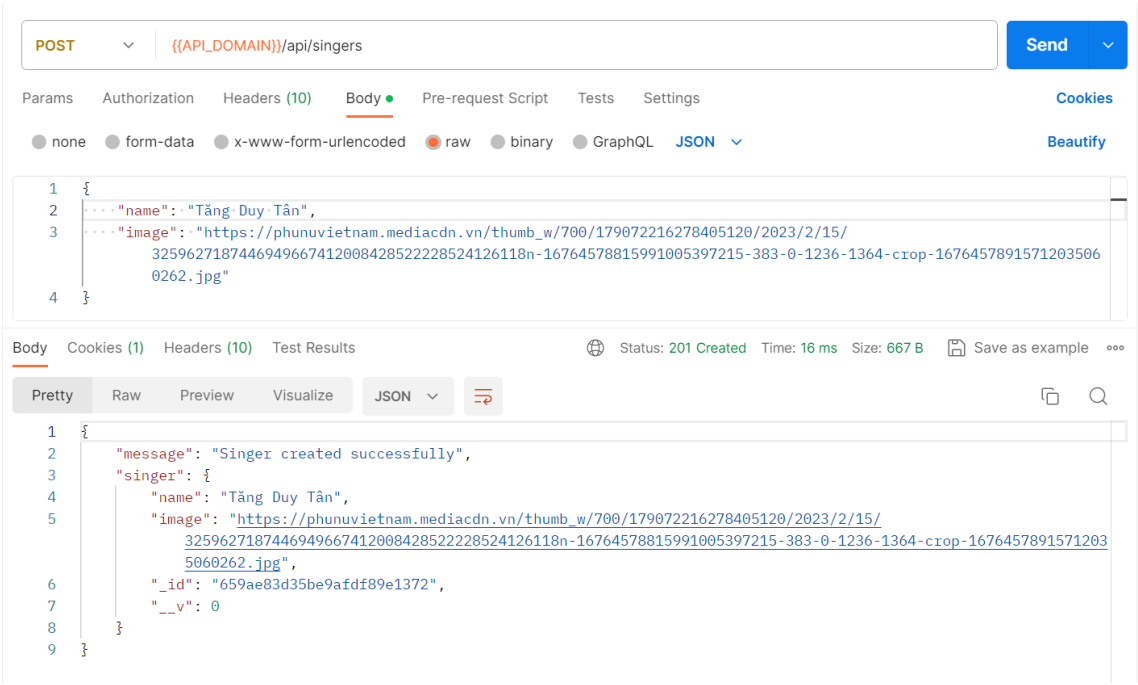
4.1.3 Các API liên quan đến ca sĩ



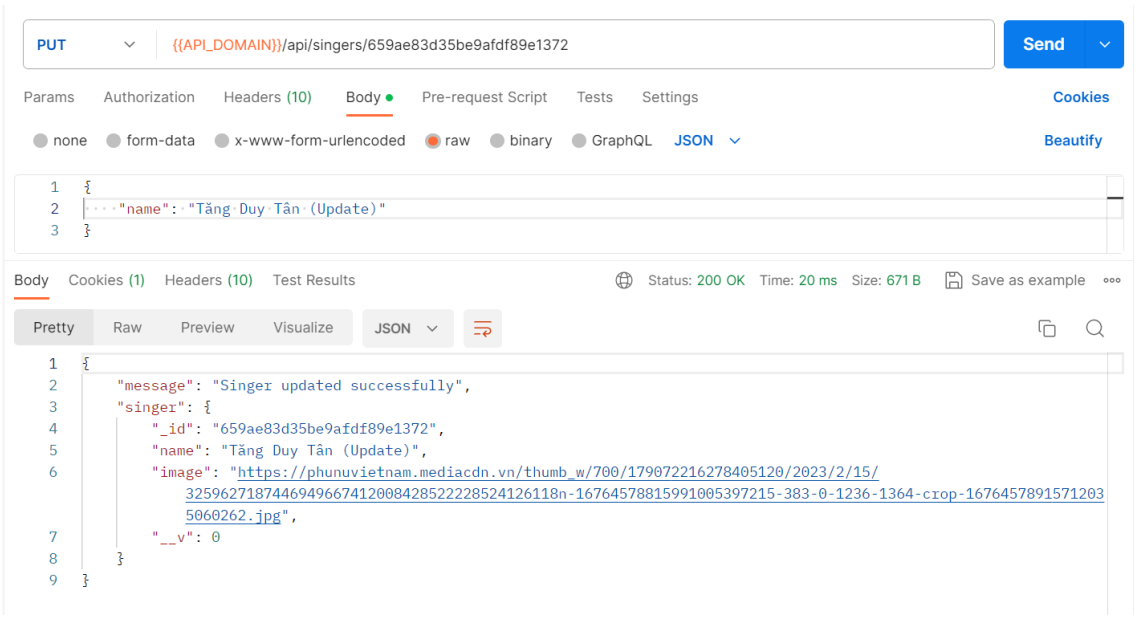
Hình 17. Thử nghiệm API lấy ra thông tin toàn bộ ca sĩ với Postman



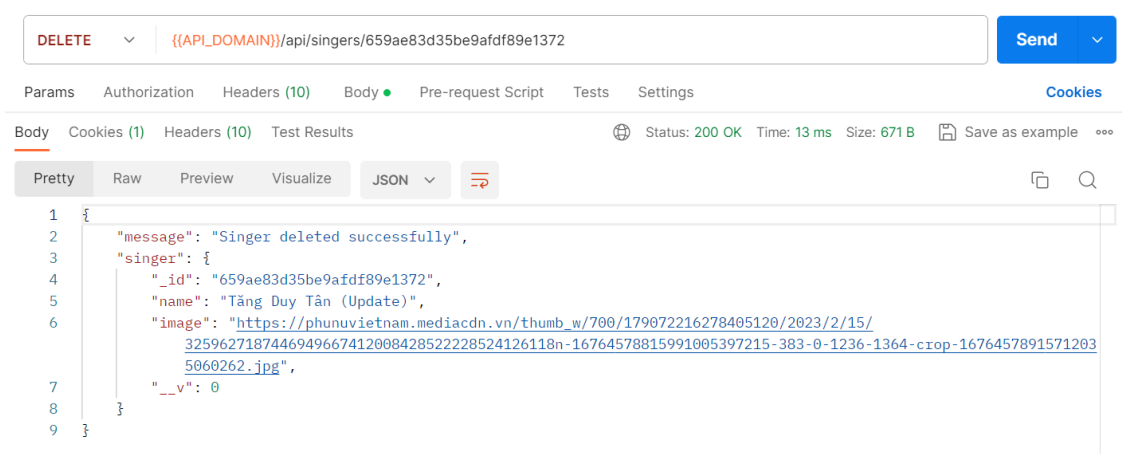
Hình 18. Thử nghiệm API lấy ra thông tin ca sĩ theo ID với Postman



Hình 19. Thử nghiệm API tạo ca sĩ mới với Postman

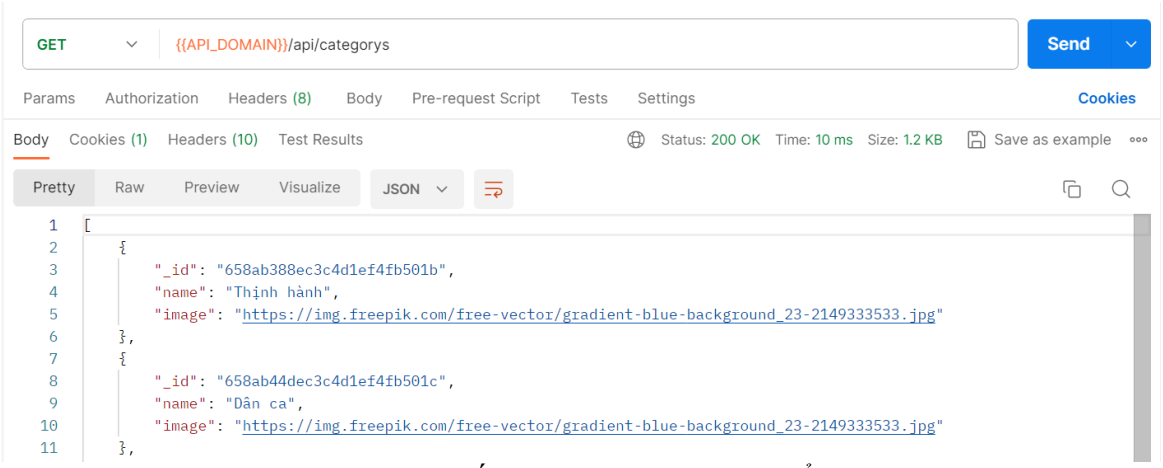


Hình 20. Thử nghiệm API sửa thông tin ca sĩ theo ID với Postman

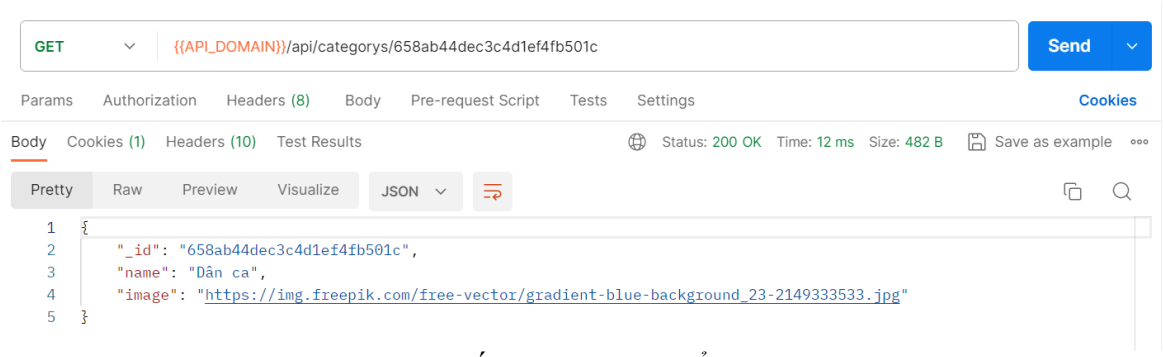


Hình 21. Thử nghiệm API xoá thông tin ca sĩ theo ID với Postman

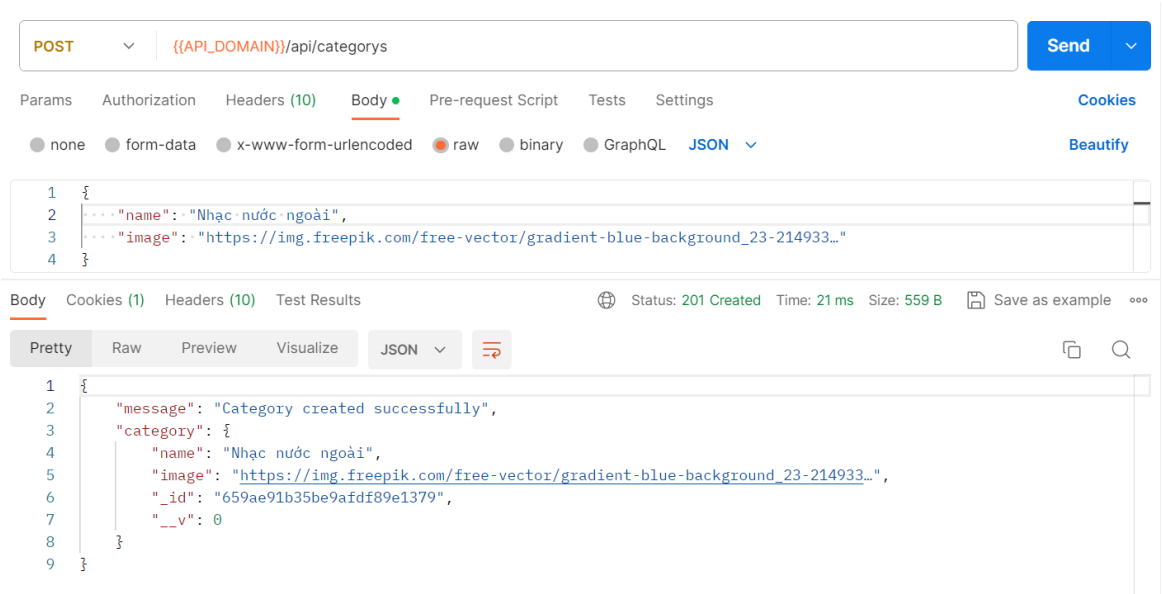
4.1.4 Các API liên quan đến thể loại



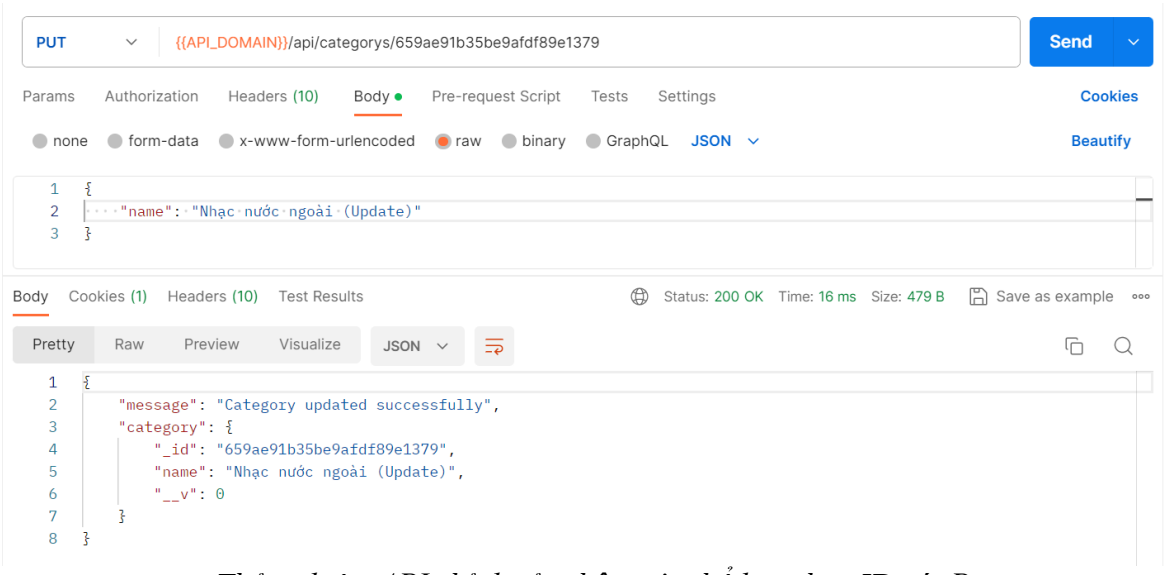
Hình 22. Thử nghiệm API lấy ra thông tin toàn bộ thể loại với Postman



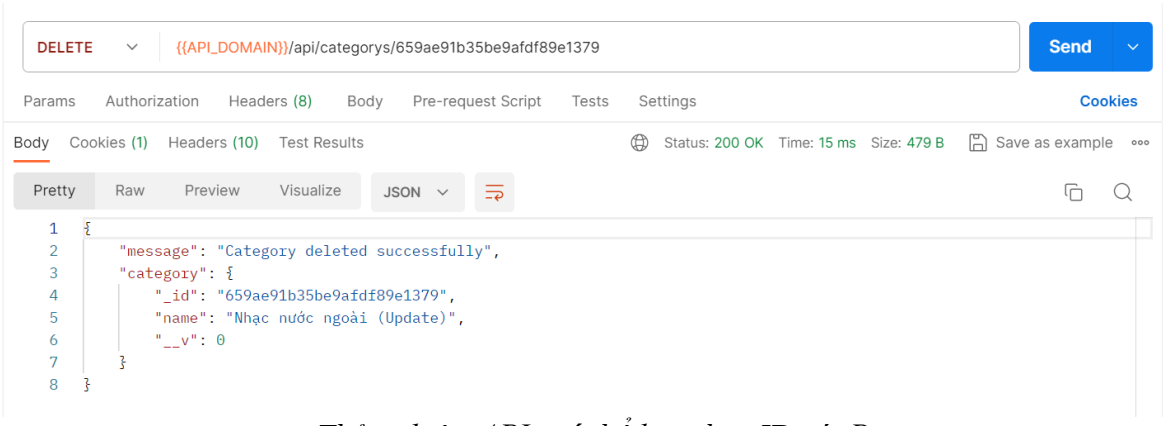
Hình 23. Thử nghiệm API lấy ra thông tin thể loại theo ID với Postman



Hình 24. Thử nghiệm API tạo thể loại mới với Postman

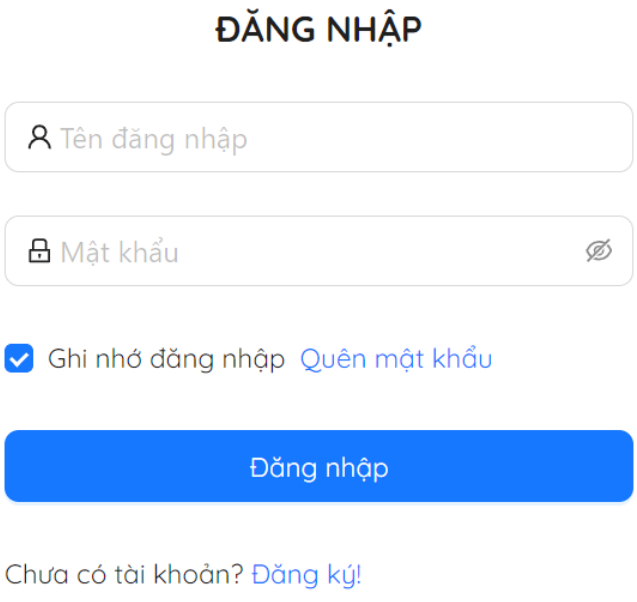


Hình 25. Thử nghiệm API chỉnh sửa thông tin thể loại theo ID với Postman

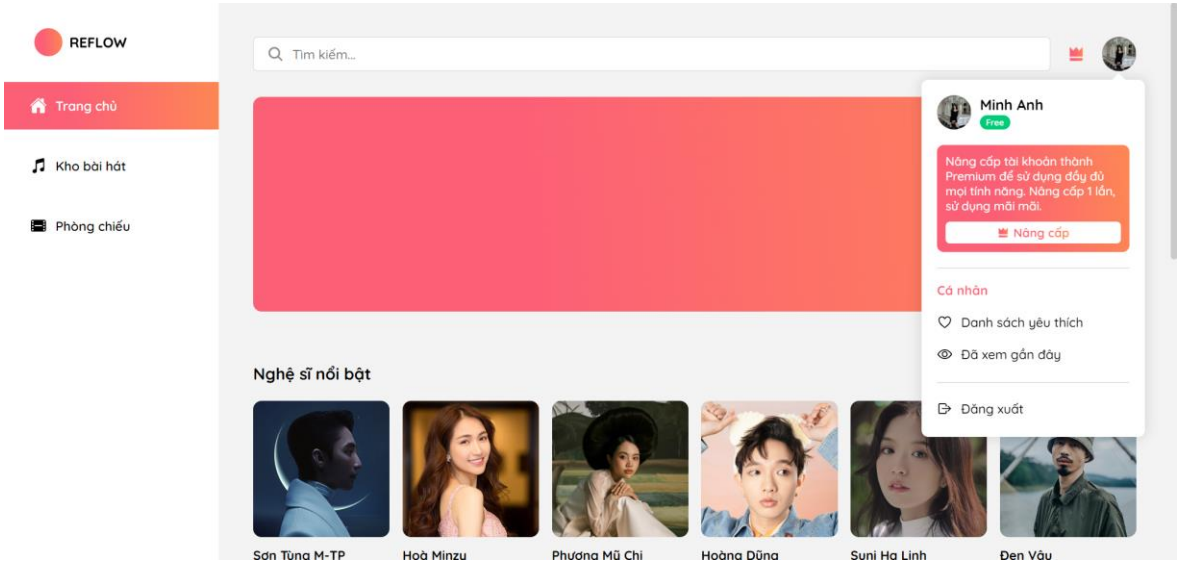


Hình 26. Thử nghiệm API xoá thể loại theo ID với Postman

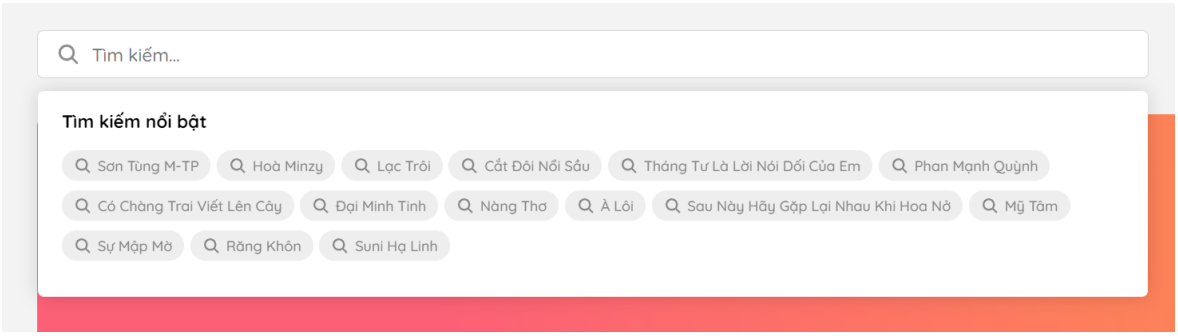
4.2 Giao diện chức năng phía người dùng



Hình 27. Giao diện đăng nhập Reflow



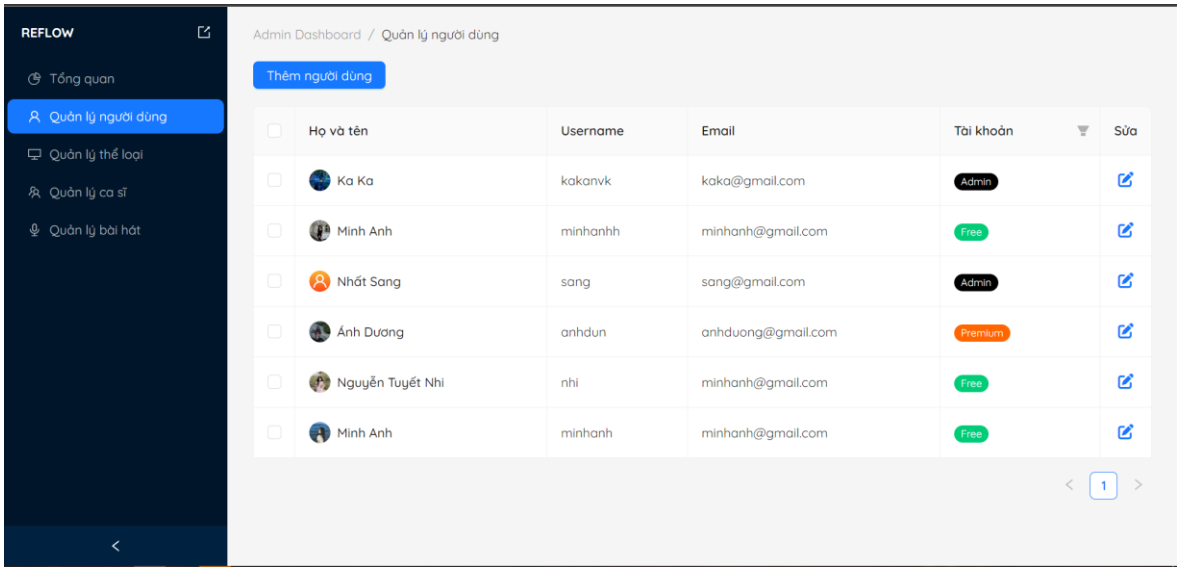
Hình 28. Giao diện Trang chủ Reflow



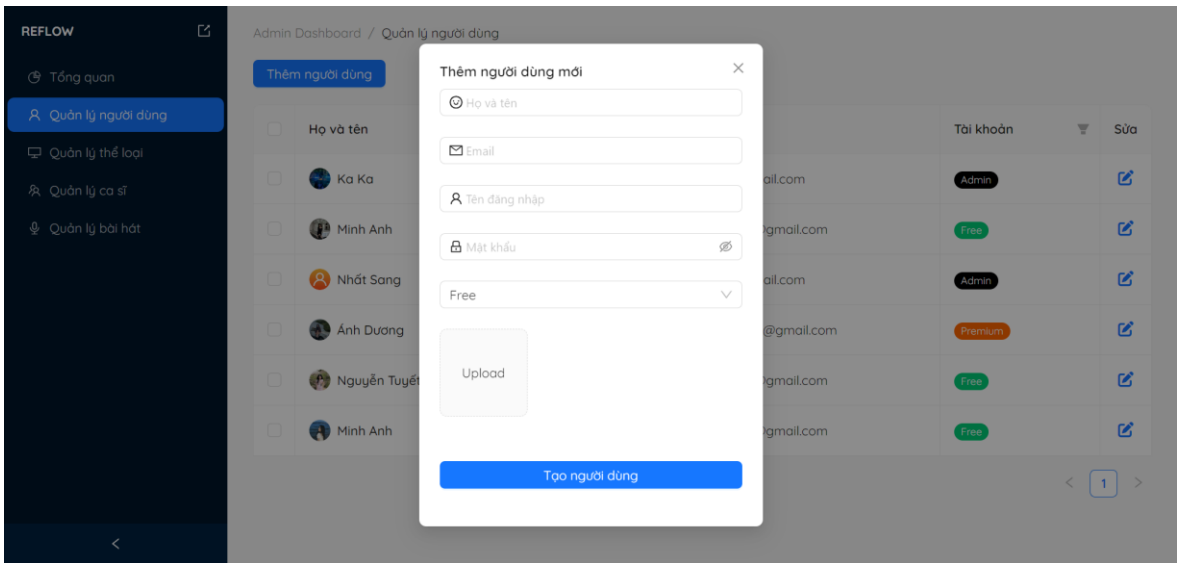
Hình 29. Giao diện đề xuất tìm kiếm

4.3 Giao diện chức năng phía người quản trị

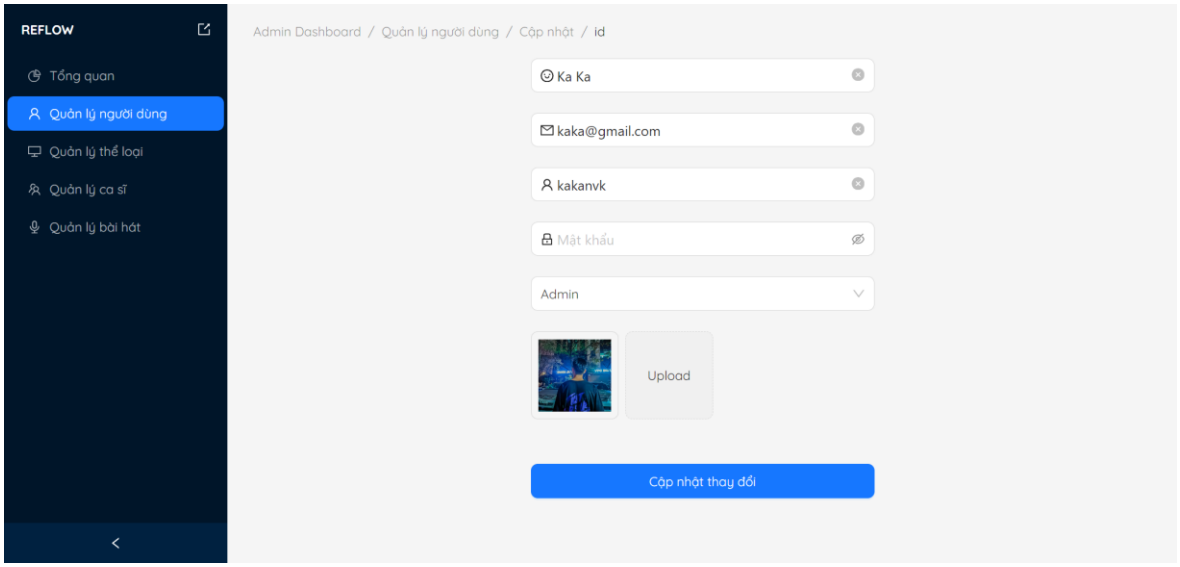
4.3.1 Giao diện quản lý người dùng



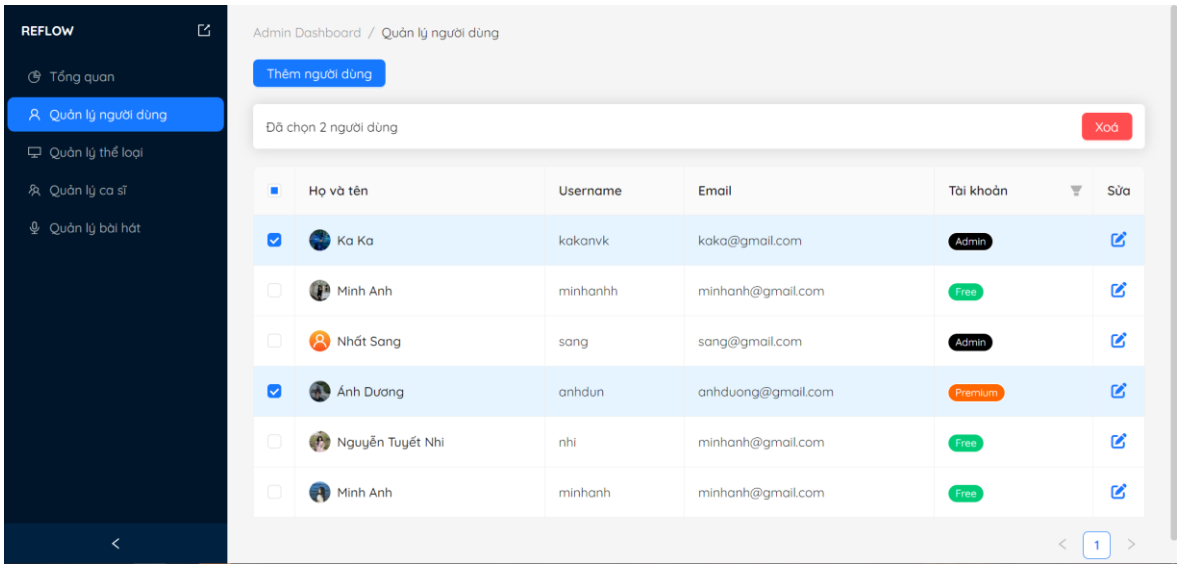
Hình 30. Giao diện trang quản lý người dùng



Hình 31. Giao diện chức năng tạo người dùng mới



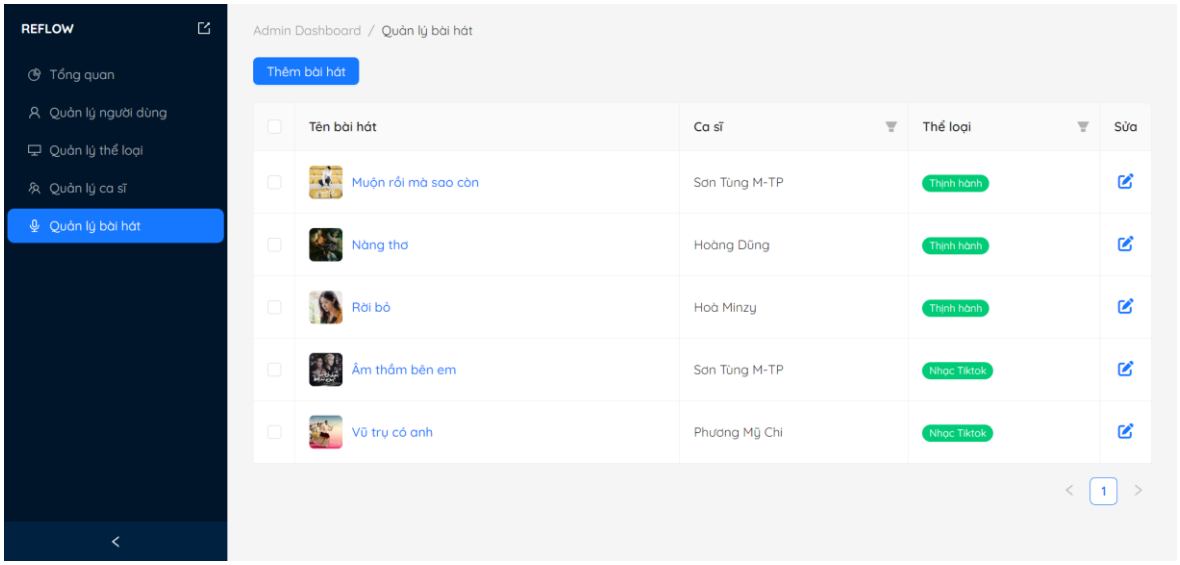
Hình 32. Giao diện chức năng sửa thông tin người dùng



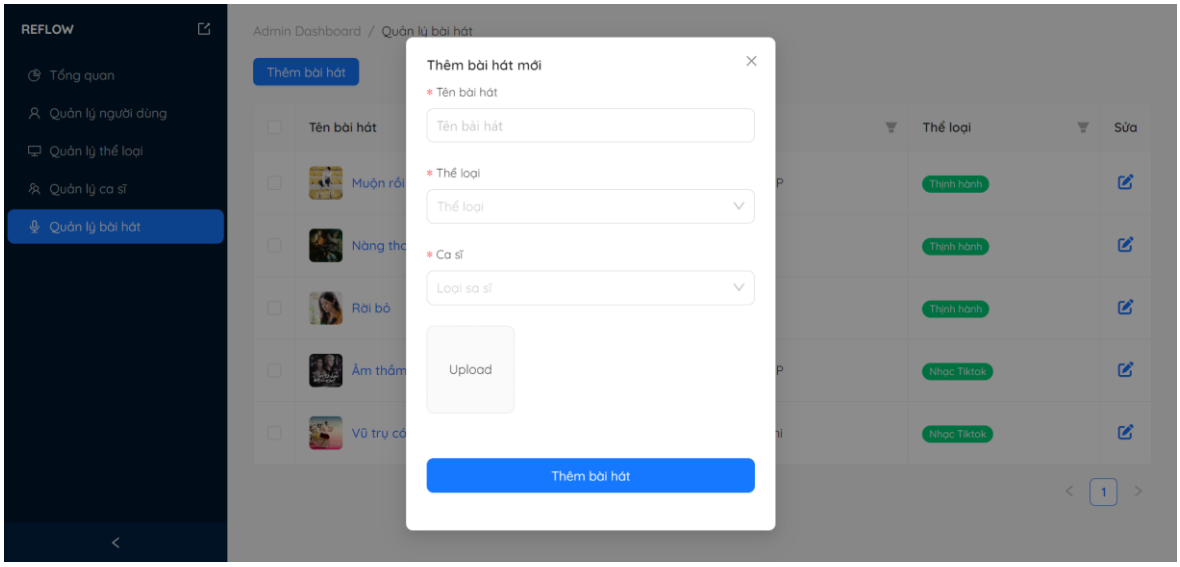
Hình 33. Giao diện chức năng chọn xoá nhiều người dùng

4.3.2 Giao diện quản lý bài hát

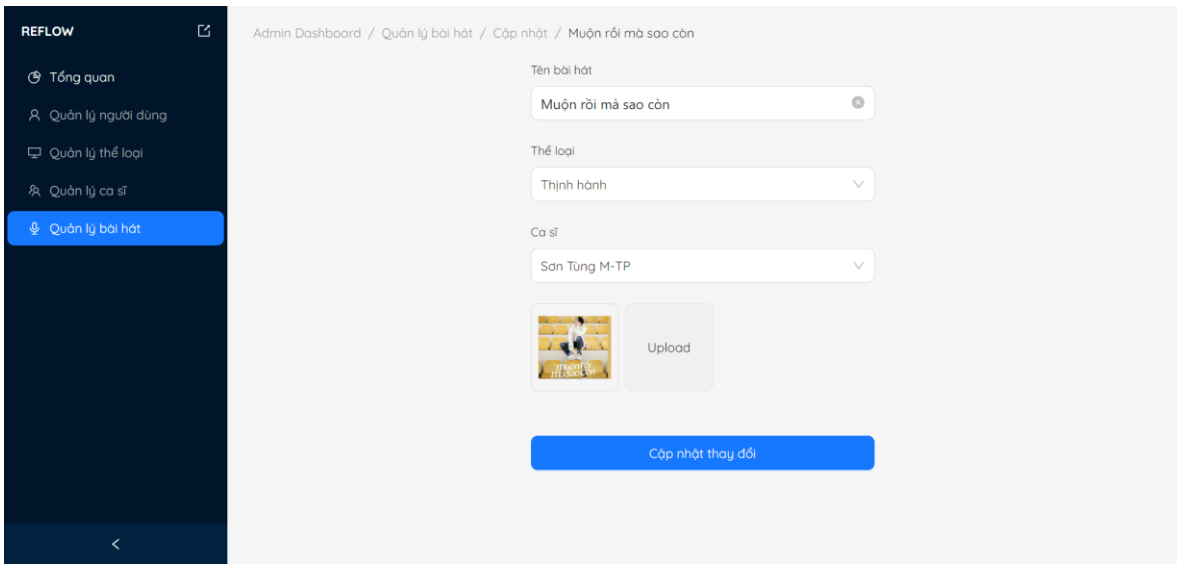
Người quản trị có thể sử dụng trang quản lý bài hát để thực hiện các thao tác với bài hát như thêm, sửa, xoá bài hát. Ngoài ra, khi chọn vào một bài hát bất kỳ, giao diện quản lý đoạn của bài hát đó sẽ xuất hiện, cho phép thêm, sửa, xoá thông tin lời bài hát, tên của một đoạn. Ngoài ra, còn có thể kéo thả để tuỳ chỉnh thứ tự của các đoạn.



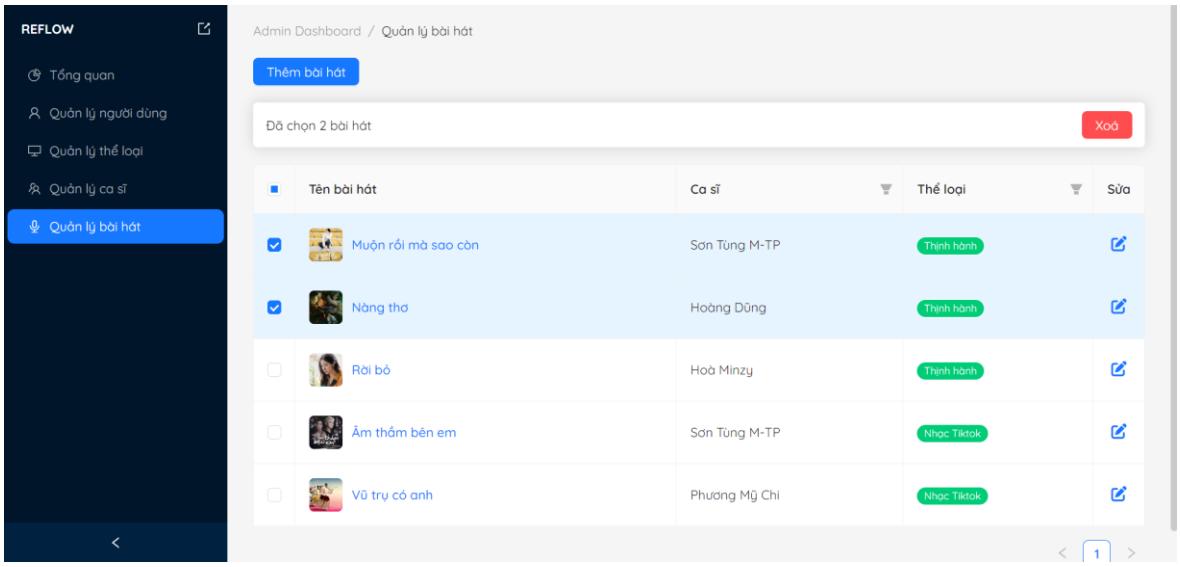
Hình 34. Giao diện quản lý bài hát



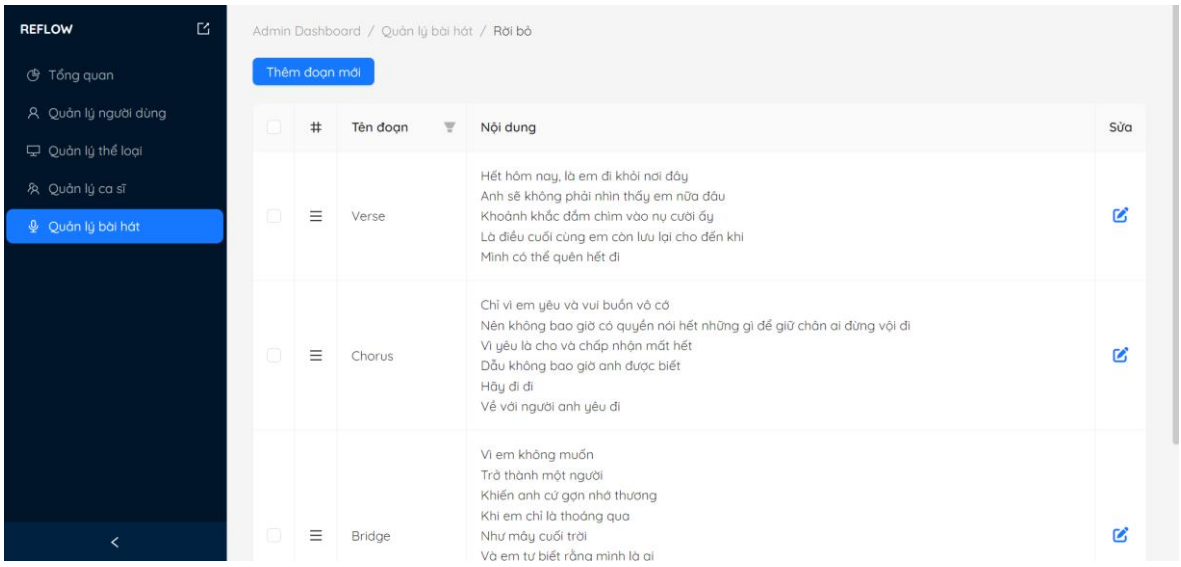
Hình 35. Giao diện chức năng thêm bài hát mới



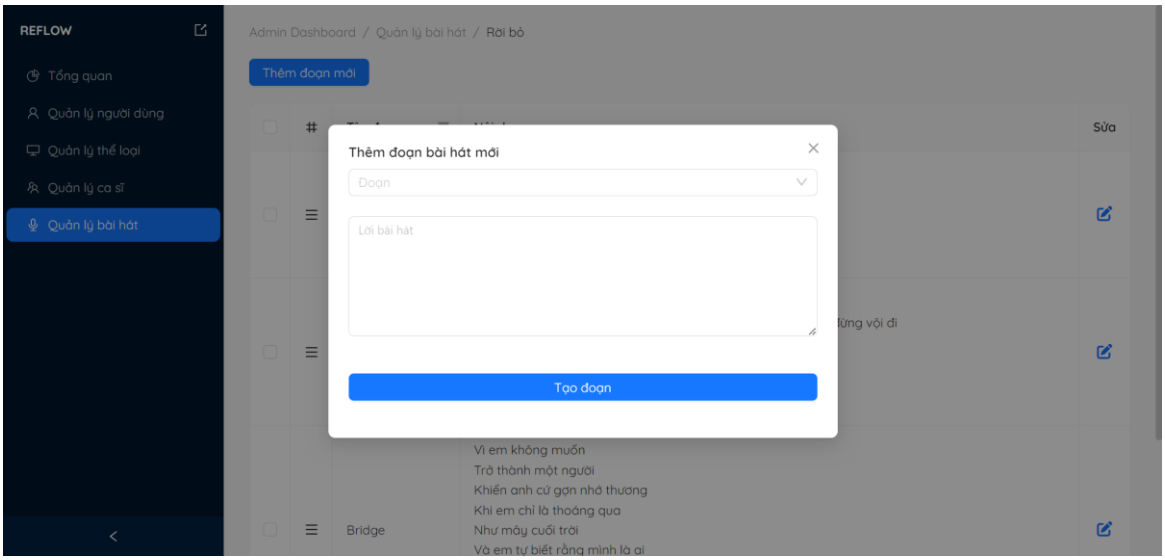
Hình 36. Giao diện chức năng sửa thông tin bài hát



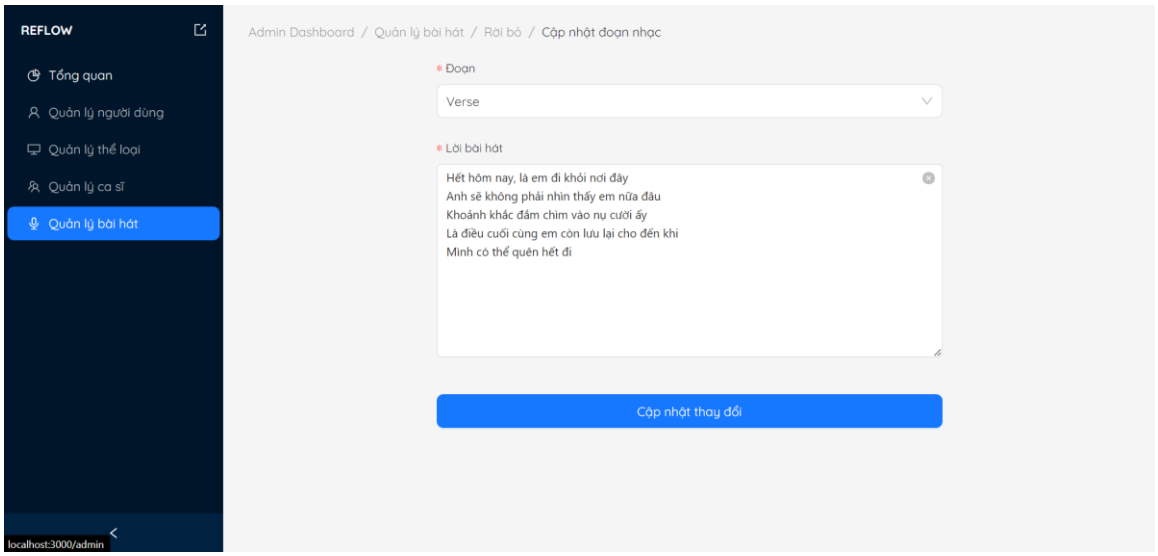
Hình 37. Giao diện chức năng chọn xoá nhiều bài hát



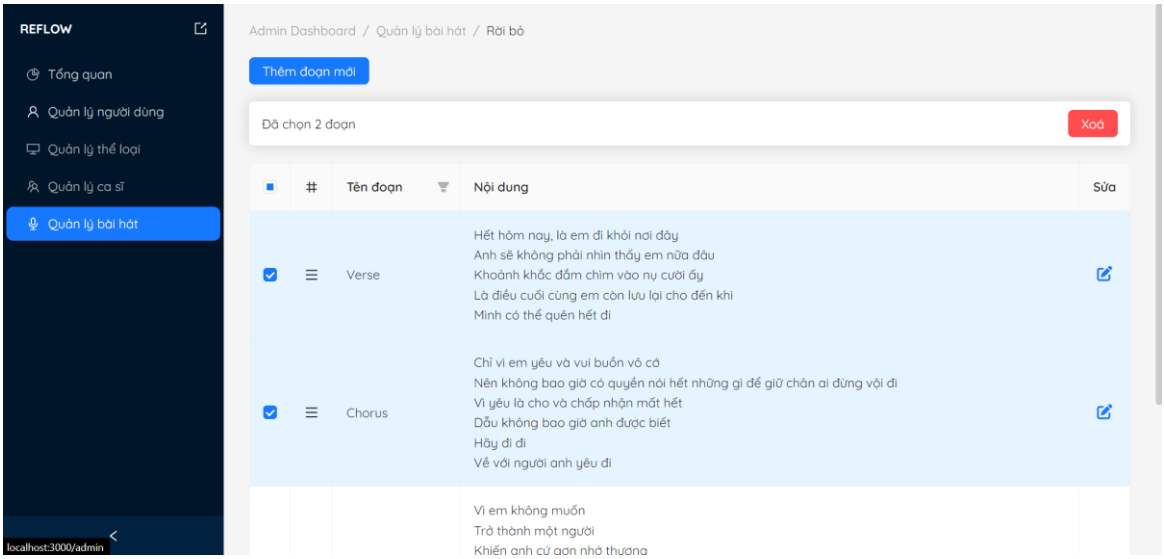
Hình 38. Giao diện chức năng quản lý các đoạn trong bài hát



Hình 39. Giao diện chức năng thêm đoạn bài hát mới



Hình 40. Giao diện chức năng sửa thông tin đoạn bài hát



Hình 41. Giao diện chức năng chọn xoá nhiều đoạn của bài hát

CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1 Kết quả đạt được:

Nắm vững khái niệm, nguyên tắc, và cơ chế hoạt động của Restful API cùng với công nghệ NodeJS.

Áp dụng các nguyên tắc và công nghệ trên để xây dựng thành công trang web trình chiếu lời bài hát Reflow với các chức năng mà đề bài đã đặt ra.

Triển khai thành công lên môi trường mạng Internet.

5.2 Hướng phát triển:

Cải thiện hơn nữa vấn đề bảo mật thông tin, xác thực người dùng để tránh các mối đe doạ từ bên ngoài.

Phát triển các tính năng độc đáo, tăng độ nhận diện thương hiệu, thu hút lượng người dùng ổn định.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] AWS, "Restful API," 2023. [Online]. Available: https://aws.amazon.com/vi/what-is/restful-api/.

[2] Wikipedia, "React (software)," 2023. [Online]. Available: https://en.wikipedia.org/wiki/React_(software).

[3] IBM, "What is a REST API," 2023. [Online]. Available: https://www.ibm.com/topics/rest-apis.

[4] Wikipedia, "Node.js," 2023. [Online]. Available: https://en.wikipedia.org/wiki/Node.js.