TRƯỜNG ĐẠI HỌC TRÀ VINH

TRƯỜNG KỸ THUẬT VÀ CÔNG NGHỆ

KHOA CÔNG NGHỆ THÔNG TIN

----------🕮----------

BÁO CÁO KẾT THÚC MÔN

MÔN: CÔNG NGHỆ PHẦN MỀM

HỆ THỐNG ĐẶT VÉ XEM PHIM TRỰC TUYẾN

Giảng viên hướng dẫn:

TS. Nguyễn Bảo Ân

Sinh viên thực hiện:

Thạch Thị Xuân Linh

(110122013 - DA22TTA)

Kiên Thị Bé Hai

(110122218 - DA22TTA)

Phan Nguyễn Hoàng Hân

(110122003 - DA22TTA)

Vĩnh Long, tháng 7 năm 2025

TRƯỜNG ĐẠI HỌC TRÀ VINH

TRƯỜNG KỸ THUẬT VÀ CÔNG NGHỆ

KHOA CÔNG NGHỆ THÔNG TIN

----------🕮----------

BÁO CÁO KẾT THÚC MÔN

MÔN: CÔNG NGHỆ PHẦN MỀM

HỆ THỐNG ĐẶT VÉ XEM PHIM TRỰC TUYẾN

Giảng viên hướng dẫn:

TS. Nguyễn Bảo Ân

Sinh viên thực hiện:

Thạch Thị Xuân Linh

(110122013 - DA22TTA)

Kiên Thị Bé Hai

(110122218 - DA22TTA)

Phan Nguyễn Hoàng Hân

(110122003 - DA22TTA)

Vĩnh Long, tháng 7 năm 2025

LỜI CẢM ƠN

Chúng em xin gửi lời cảm ơn chân thành đến thầy vì những kiến thức và kinh nghiệm quý báu mà thầy đã giảng dạy trong suốt khoá học "Công nghệ phần mềm". Nhờ sự hướng dẫn tận tâm của thầy, chúng em đã có cơ hội tiếp cận và làm quen với những lĩnh vực, công nghệ mới mẻ và hấp dẫn.

Những khái niệm từ cơ bản đến nâng cao đều được thầy giảng dạy một cách rõ ràng và dễ hiểu, giúp chúng em tự tin hơn trong việc ứng dụng những kiến thức này vào thực tế. Ngoài ra, chúng em cũng muốn bày tỏ lòng biết ơn về cách thầy tạo ra môi trường học tập thân thiện và truyền cảm hứng cho chúng em. Sự tận tâm và nhiệt huyết của thầy trong việc truyền đạt kiến thức đã khiến cho quá trình học tập trở nên thú vị và đáng nhớ.

Một lần nữa, chúng em xin chân thành cảm ơn thầy Nguyễn Bảo Ân vì sự đóng góp to lớn và những điều tuyệt vời thầy đã mang đến cho cuộc sống học tập của chúng em.

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Trà Vinh, ngày…...tháng…...năm 2025
GIÁO VIÊN CHẤM BÁO CÁO
(ký, ghi rõ họ tên)

NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC TỪ VIẾT TẮT

Từ viết tắt	Diễn giải
API	Application Programming Interface – Giao diện lập trình ứng dụng
CI/CD	Continuous Integration / Continuous Deployment – Tích hợp liên tục / Triển khai liên tục
CRUD	Create, Read, Update, Delete – Tạo, Đọc, Cập nhật, Xóa
ERD	Entity Relationship Diagram – Sơ đồ thực thể quan hệ
JWT	JSON Web Token – Chuỗi mã xác thực người dùng
UI/UX	User Interface / User Experience – Giao diện người dùng / Trải nghiệm người dùng
HTTP	HyperText Transfer Protocol – Giao thức truyền siêu văn bản
JSON	JavaScript Object Notation – Định dạng dữ liệu dạng đối tượng
SQL	Structured Query Language – Ngôn ngữ truy vấn có cấu trúc
NoSQL	Not only SQL – Cơ sở dữ liệu không quan hệ
ORM	Object-Relational Mapping – Ánh xạ đối tượng - quan hệ
CRUD	Create, Read, Update, Delete – Tạo, Đọc, Sửa, Xóa
RESTful	Representational State Transfer – Kiến trúc dịch vụ web
IDE	Integrated Development Environment – Môi trường phát triển tích hợp
CSS	Cascading Style Sheets – Ngôn ngữ định kiểu
HMR	Hot Module Replacement – Thay thế mô-đun nóng (không cần reload trang)

NPM	Node Package Manager – Trình quản lý gói cho Node.js
CORS	Cross-Origin Resource Sharing – Chia sẻ tài nguyên khác nguồn
CRUD	Create, Read, Update, Delete – Các thao tác cơ bản với dữ liệu
JWT	JSON Web Token – Token xác thực người dùng
ESLint	ECMAScript Lint – Công cụ kiểm tra và định dạng mã JavaScript

Chương 1. GIỚI THIỆU

1.1 Giới thiệu chủ đề

Trong bối cảnh xã hội hiện đại, nhu cầu giải trí của con người ngày càng đa dạng, trong đó xem phim tại rạp là một trong những hình thức phổ biến và được ưa chuộng nhất. Sự phát triển nhanh chóng của công nghệ thông tin và truyền thông đã thúc đẩy quá trình chuyển đổi số trong nhiều lĩnh vực dịch vụ, đặc biệt là trong ngành công nghiệp điện ảnh. Theo đó, các rạp chiếu phim hiện đại không chỉ cạnh tranh về chất lượng hình ảnh, âm thanh mà còn về tính tiện lợi trong khâu phục vụ khách hàng, đặc biệt là quá trình đặt vé.

Hệ thống đặt vé xem phim trực tuyến ra đời như một giải pháp hiệu quả nhằm tối ưu hóa trải nghiệm người dùng, giảm thiểu thời gian chờ đợi và hạn chế các rủi ro phát sinh trong quá trình mua vé truyền thống. Hệ thống cho phép người dùng tìm kiếm phim theo nhiều tiêu chí như thể loại, tiêu đề, ngày chiếu, khu vực rạp hoặc giá vé. Đặc biệt, tính năng định vị vị trí địa lý giúp hệ thống gợi ý các rạp gần người dùng, giúp rút ngắn thời gian tìm kiếm và đưa ra lựa chọn phù hợp hơn.

Ngoài ra, hệ thống còn được thiết kế để tự động cập nhật lịch chiếu và khóa chức năng đặt vé đối với các suất chiếu đã hoặc sắp diễn ra, nhằm đảm bảo tính chính xác, tránh việc đặt vé trễ gây ảnh hưởng đến vận hành thực tế tại rạp. Nhìn chung, hệ thống không chỉ nâng cao hiệu quả quản lý mà còn góp phần xây dựng trải nghiệm dịch vụ chuyên nghiệp và hiện đại hơn cho khách hàng.

1.2 Mục tiêu của ứng dụng

Ứng dụng đặt vé xem phim trực tuyến được phát triển với mục tiêu chính là cung cấp một nền tảng hiện đại, tiện lợi và thân thiện với người dùng, đáp ứng nhu cầu đặt vé mọi lúc, mọi nơi thông qua giao diện trực quan và thao tác đơn giản. Người dùng có thể dễ dàng đăng nhập, duyệt phim, chọn suất chiếu, vị trí ghế và thanh toán chỉ trong vài bước. Đồng thời, hệ thống cũng hỗ trợ việc quản lý tập trung dành cho quản trị viên trong việc cập nhật dữ liệu phim, lên lịch chiếu, cấu hình phòng chiếu và theo dõi vé bán.

Về mặt kỹ thuật, hệ thống được xây dựng theo kiến trúc client/server với giao tiếp thông qua các API tuân thủ nguyên tắc RESTful. Phần frontend được phát triển

bằng ReactJS, cho phép tạo ra giao diện tương tác linh hoạt, phản hồi nhanh, thân thiện với người dùng. Phần backend được đảm nhiệm bởi nền tảng NodeJS, xử lý các tác vụ nghiệp vụ như xác thực người dùng, quản lý phim, suất chiếu và vé đặt. Cơ sở dữ liệu được lưu trữ trong MySQL với các mối quan hệ rõ ràng, đảm bảo tính toàn vẹn dữ liệu.

Hệ thống còn được tích hợp quy trình CI/CD sử dụng GitHub Actions để tự động hóa việc kiểm thử và triển khai phần mềm mỗi khi có thay đổi mã nguồn. Việc đóng gói toàn bộ ứng dụng bằng Docker giúp đảm bảo tính nhất quán của môi trường phát triển, thử nghiệm và triển khai thực tế.

Dưới đây là các công cụ và công nghệ chính được sử dụng trong quá trình phát triển hệ thống:

- Jira: Quản lý dự án theo phương pháp Scrum, lập kế hoạch Sprint, phân chia và theo dõi tiến độ công việc nhóm.

- Figma: Thiết kế UI/UX, tạo mẫu và chia sẻ giao diện trực quan.

- GitHub: Lưu trữ và quản lý mã nguồn, phân nhánh và kiểm soát phiên bản.

- GitHub Actions: Thiết lập CI/CD để kiểm thử và triển khai tự động.

- Postman: Kiểm thử các API và mô phỏng dữ liệu trong quá trình phát triển.

- Swagger: Tài liệu hóa hệ thống API, hỗ trợ mô tả cấu trúc request/response và phương thức HTTP.

- Docker: Đóng gói, triển khai các thành phần hệ thống trên môi trường độc lập, giảm thiểu lỗi môi trường.

1.3 Lý do chọn đề tài

Trong thực tế, quá trình bán vé tại nhiều rạp chiếu phim vẫn còn mang tính thủ công hoặc phụ thuộc vào các hệ thống truyền thống, thiếu tính linh hoạt và khả năng mở rộng. Khách hàng thường xuyên phải xếp hàng chờ đợi để mua vé, đặc biệt vào các khung giờ cao điểm hoặc ngày lễ, dẫn đến tình trạng quá tải, trải nghiệm không thoải mái và tiềm ẩn nguy cơ bỏ lỡ suất chiếu mong muốn. Bên cạnh đó, việc quản lý lịch chiếu, tình trạng ghế và phân phối phòng chiếu của các rạp cũng gặp nhiều khó khăn nếu không có hệ thống hỗ trợ chuyên biệt.

Xuất phát từ những vấn đề nêu trên, nhóm quyết định lựa chọn đề tài **"Hệ thống đặt vé xem phim trực tuyến"** nhằm hiện thực hóa một nền tảng giải pháp số hóa toàn diện cho hoạt động đặt vé, đồng thời nâng cao hiệu quả vận hành cho các đơn vị rạp chiếu. Bên cạnh tính ứng dụng thực tiễn cao, đề tài còn là cơ hội để nhóm vận dụng kiến thức đã học trong môn Công nghệ Phần mềm, bao gồm: phân tích yêu cầu, thiết kế hệ thống, xây dựng giao diện người dùng, phát triển API, kiểm thử phần mềm, triển khai hệ thống và quản lý dự án theo quy trình chuyên nghiệp.

Thông qua dự án này, nhóm không chỉ hoàn thiện kỹ năng kỹ thuật mà còn rèn luyện khả năng làm việc nhóm, giao tiếp, quản lý thời gian và giải quyết vấn đề – những yếu tố thiết yếu trong thực tiễn nghề nghiệp sau này.

1.4 Khái quát về môn học

1.4.1 Các khái niệm cơ bản về công nghệ phần mềm

Công nghệ phần mềm (Software Engineering) là một ngành khoa học và kỹ thuật chuyên nghiên cứu việc xây dựng, phát triển, vận hành và bảo trì phần mềm một cách hệ thống, có kiểm soát và hiệu quả. Mục tiêu chính của công nghệ phần mềm là tạo ra phần mềm chất lượng cao, đáp ứng yêu cầu người dùng, dễ bảo trì và có khả năng mở rộng.

Các thành phần cốt lõi trong công nghệ phần mềm bao gồm:

- Quy trình phát triển phần mềm (Software Process): Là tập hợp các bước tổ chức trong suốt vòng đời phần mềm như: phân tích, thiết kế, cài đặt, kiểm thử và bảo trì.

- Kỹ thuật phần mềm (Software Engineering Techniques): Bao gồm các phương pháp phân tích yêu cầu, thiết kế hệ thống, kiểm thử phần mềm và quản lý dự án.

- Công cụ phần mềm (Software Tools): Là các ứng dụng hỗ trợ tự động hóa trong quy trình phát triển như IDE, hệ thống quản lý mã nguồn (Git), phần mềm CI/CD,...

1.4.2 Các mô hình phát triển phần mềm

Mô hình phát triển phần mềm là cách tiếp cận tổ chức các hoạt động trong quy trình phát triển. Một số mô hình tiêu biểu bao gồm:

Mô hình thác nước (Waterfall):

- Mô hình tuyến tính, các giai đoạn thực hiện tuần tự: yêu cầu → thiết kế → lập trình → kiểm thử → triển khai → bảo trì.

- Ưu điểm: Dễ hiểu, dễ quản lý. Nhược điểm: Ít linh hoạt, khó thích ứng với thay đổi.

Mô hình V (Verification & Validation):

- Mở rộng từ mô hình thác nước với trọng tâm là kiểm thử ở từng giai đoạn.
- Phù hợp với hệ thống yêu cầu độ tin cậy cao.

Mô hình xoắn ốc (Spiral):

- Kết hợp giữa thiết kế tuần tự và phát triển lặp, nhấn mạnh vào quản lý rủi ro.
- Ưu điểm: Linh hoạt, kiểm soát rủi ro tốt.

Mô hình Agile (Linh hoạt):

- Phát triển phần mềm theo hướng lặp (iterations), mỗi vòng đời ngắn gọi là sprint. Các thành viên làm việc theo nhóm nhỏ, phản hồi liên tục với khách hàng.
- Phổ biến nhất hiện nay là Scrum và Kanban.

1.4.2 Kiến trúc phần mềm dựa trên đám mây và kiến trúc Microservices

1.4.2.1. Kiến trúc phần mềm dựa trên đám mây (Cloud-based Architecture)

Đây là mô hình phát triển phần mềm tận dụng nền tảng và dịch vụ của điện toán đám mây như: hạ tầng (IaaS), nền tảng (PaaS), dịch vụ phần mềm (SaaS).

Đặc điểm:

- Phần mềm được triển khai trên các dịch vụ như AWS, Azure, Google Cloud.
- Tự động mở rộng tài nguyên (auto-scaling).
- Hỗ trợ triển khai liên tục (CI/CD) và theo dõi qua các công cụ DevOps.
- Tăng tính sẵn sàng, phục hồi nhanh, dễ tích hợp.

1.4.2.2. Kiến trúc Microservices

Kiến trúc Microservices là phương pháp thiết kế phần mềm bằng cách chia hệ thống thành các dịch vụ nhỏ, độc lập, giao tiếp qua API (thường là REST hoặc gRPC).

Đặc điểm:

- Tách biệt chức năng: Mỗi microservice đảm nhận một chức năng riêng biệt (ví dụ: user service, payment service,...).

- Triển khai độc lập: Có thể cập nhật, thay thế một dịch vụ mà không ảnh hưởng đến toàn bộ hệ thống.

- Khả năng mở rộng: Có thể mở rộng riêng từng service theo nhu cầu sử dụng.

- Microservices phù hợp với các ứng dụng có quy mô lớn, yêu cầu linh hoạt, thường xuyên thay đổi như hệ thống thương mại điện tử, đặt vé, mạng xã hội,...

Chương 2. PHÂN TÍCH YÊU CẦU

2.1. Các chức năng chính của hệ thống (Functional Requirements)

Hệ thống đặt vé xem phim trực tuyến được thiết kế nhằm hỗ trợ tối đa trải nghiệm người dùng, đồng thời cung cấp công cụ quản lý hiệu quả cho quản trị viên. Các chức năng được chia thành hai nhóm đối tượng chính: người dùng thông thường (user) và quản trị viên hệ thống (admin).

2.1.1. Đối với người dùng thông thường (user)

Đăng ký và đăng nhập tài khoản: Người dùng có thể tạo tài khoản mới với thông tin như họ tên, email và mật khẩu. Hệ thống thực hiện xác thực thông tin đăng nhập, bảo mật bằng cách mã hóa mật khẩu và sử dụng token JWT để quản lý phiên hoạt động.

Xem thông tin phim chi tiết: Bao gồm trailer, mô tả nội dung, diễn viên, thời lượng, đánh giá và lịch chiếu tương ứng. Thông tin được trình bày sinh động giúp người dùng dễ dàng đưa ra quyết định đặt vé.

Xem lịch chiếu và chọn suất: Người dùng có thể xem lịch chiếu của từng phim theo rạp và theo ngày. Tính năng lọc thông minh giúp dễ dàng xác định các suất chiếu còn khả dụng.

Đặt vé và chọn ghế: Sau khi chọn suất chiếu, người dùng tiến hành chọn ghế qua sơ đồ trực quan. Hệ thống đảm bảo rằng các ghế đã đặt sẽ được đánh dấu và không thể chọn trùng.

Thanh toán và xác nhận vé: Người dùng xác nhận thông tin đặt vé, tiến hành thanh toán (nội bộ hoặc tích hợp cổng thanh toán – mở rộng), và nhận vé điện tử qua hệ thống.

Quản lý vé đã đặt: Người dùng có thể xem lại danh sách vé đã mua, hủy vé nếu chưa đến giờ chiếu, và nhận thông báo nhắc lịch chiếu (nếu có).

Đánh giá và phản hồi: Sau khi xem phim, người dùng có thể để lại đánh giá hoặc bình luận để cải thiện chất lượng dịch vụ.

2.1.2. Đối với quản trị viên hệ thống (admin)

Quản lý người dùng: Thêm, sửa, xóa hoặc khóa tài khoản người dùng, phân quyền truy cập và theo dõi hoạt động.

Quản lý phim: Tạo mới, chỉnh sửa và xóa thông tin phim, cập nhật hình ảnh, trailer, mô tả và phân loại thể loại.

Quản lý lịch chiếu: Gán phim vào phòng chiếu, chọn thời gian chiếu và cấu hình giá vé theo ngày/giờ/rạp cụ thể.

Quản lý rạp và phòng chiếu: Tạo sơ đồ ghế động, điều chỉnh số lượng ghế, cấu hình trạng thái ghế và kiểm soát hoạt động từng phòng.

Thống kê và báo cáo: Theo dõi doanh thu theo ngày/tháng, lượng vé đã bán, suất chiếu phổ biến, hành vi người dùng...

2.2. Các yêu cầu phi chức năng (Non-functional Requirements)

Để đảm bảo hệ thống có thể vận hành ổn định, bảo mật và sẵn sàng mở rộng trong tương lai, các yêu cầu phi chức năng sau đây được xác định:

Hiệu năng (Performance): Hệ thống phải phản hồi nhanh, tối đa 2 giây cho các thao tác như tìm kiếm phim, tải lịch chiếu và hiển thị vé. Cần đảm bảo khả năng xử lý đồng thời nhiều người dùng mà không làm gián đoạn dịch vụ.

Tính sẵn sàng (Availability): Hệ thống cần hoạt động liên tục, đặc biệt vào thời điểm cao điểm như cuối tuần và dịp lễ. Cơ chế giám sát lỗi và tự phục hồi được tích hợp để hạn chế thời gian gián đoạn dịch vụ.

Bảo mật (Security): Hệ thống yêu cầu xác thực người dùng bằng token (JWT), mã hóa mật khẩu bằng thuật toán như bcrypt, đồng thời có biện pháp chống tấn công phổ biến như SQL Injection, XSS và CSRF. Quản trị viên chỉ có thể truy cập chức năng được phân quyền.

Khả năng mở rộng (Scalability): Ứng dụng cần dễ dàng mở rộng theo số lượng người dùng hoặc mở rộng chức năng như thanh toán trực tuyến, gợi ý phim bằng AI... Kiến trúc tách biệt frontend/backend và sử dụng API chuẩn REST hỗ trợ khả năng mở rộng tối ưu.

Khả năng tương thích (Compatibility): Giao diện được thiết kế responsive, đảm bảo hiển thị tốt trên mọi thiết bị từ điện thoại, máy tính bảng đến màn hình desktop. Trình duyệt được hỗ trợ gồm Chrome, Firefox, Edge.

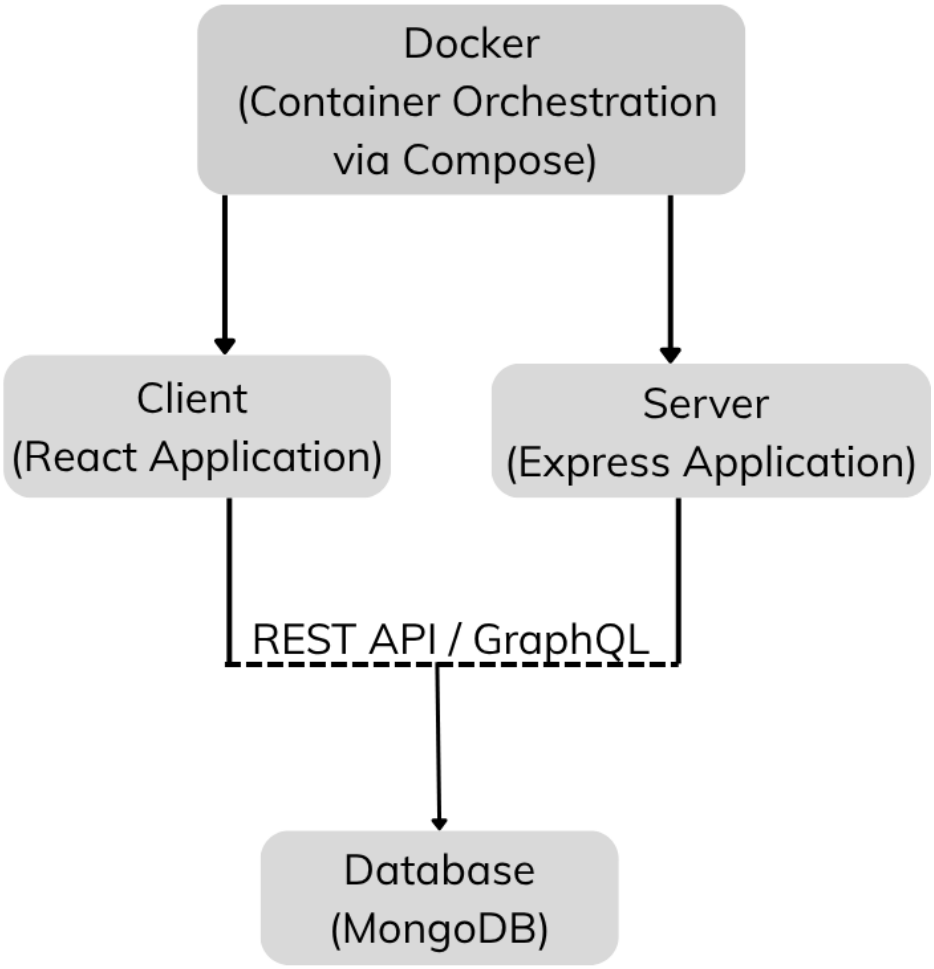
Dễ bảo trì và triển khai (Maintainability & Deployability): Ứng dụng được đóng gói bằng Docker giúp thống nhất môi trường, dễ triển khai và phục hồi. Quá trình CI/CD sử dụng GitHub Actions giúp kiểm thử, build và deploy tự động. Swagger hỗ trợ tài liệu hóa API tự động, giúp frontend dễ tích hợp, backend dễ bảo trì.

Trải nghiệm người dùng (User Experience): Giao diện tối ưu hóa thao tác đặt vé, phân bố nội dung hợp lý, màu sắc rõ ràng, và hướng dẫn dễ hiểu. Người dùng không cần nhiều bước để đặt được vé thành công, giảm thiểu thao tác không cần thiết.

Chương 3. THIẾT KẾ HỆ THỐNG

3.1. Kiến trúc tổng thể

Sơ đồ trên minh họa kiến trúc tổng thể của hệ thống theo mô hình client-server, trong đó các thành phần chính được tổ chức như sau:



Hình 3.1 Kiến trúc tổng thể của hệ thống

- Client – Ứng dụng React (Frontend): Đây là giao diện người dùng, phát triển bằng thư viện ReactJS. Thành phần này chịu trách nhiệm hiển thị dữ liệu và tương tác với người dùng cuối. Mọi yêu cầu của người dùng (chẳng hạn như xem danh sách phim, đặt vé, đăng nhập...) sẽ được gửi đến máy chủ thông qua các API sử dụng giao thức HTTP.

- API – Giao tiếp HTTP trung gian: Lớp API hoạt động như một cầu nối giữa client và server, định nghĩa các điểm cuối (endpoints) để phục vụ các chức năng của hệ

thống như xác thực người dùng, truy xuất phim, quản lý lịch chiếu, và xử lý đặt vé. Tầng này đảm bảo tính phân tách giữa giao diện người dùng và logic xử lý phía máy chủ.

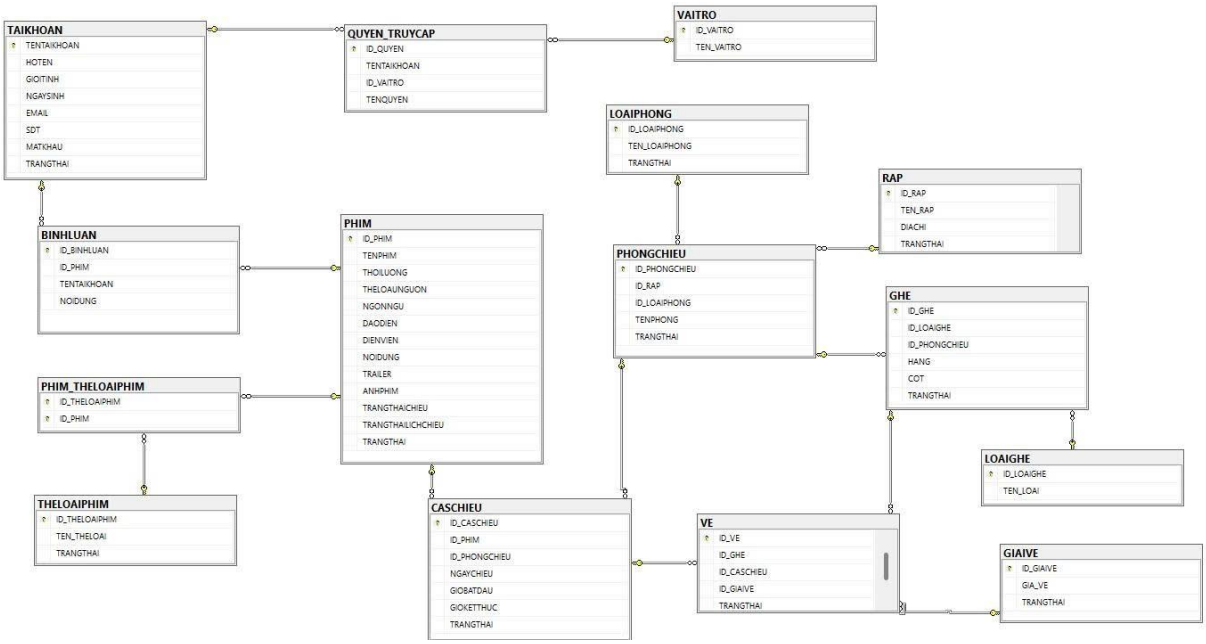
- Server – Ứng dụng Express (Backend): Đây là nơi xử lý toàn bộ logic nghiệp vụ của hệ thống. Server tiếp nhận các yêu cầu từ API, xử lý dữ liệu, xác thực thông tin, áp dụng các quy tắc nghiệp vụ và phản hồi kết quả về client. Máy chủ được triển khai bằng framework Express chạy trên Node.js.

- Cơ sở dữ liệu – MongoDB: Toàn bộ dữ liệu liên quan đến hệ thống như người dùng, phim, rạp chiếu, lịch chiếu và vé đặt được lưu trữ trong cơ sở dữ liệu NoSQL MongoDB. Server tương tác với MongoDB để thực hiện các thao tác đọc/ghi dữ liệu.

3.2 Thiết kế cơ sở dữ liệu

Cơ sở dữ liệu của hệ thống được thiết kế dựa trên mô hình quan hệ, đảm bảo tính chuẩn hóa và logic dữ liệu phục vụ cho hoạt động của hệ thống đặt vé xem phim. Hệ thống bao gồm các thực thể chính như: người dùng, phim, rạp chiếu, phòng chiếu, ghế ngồi, lịch chiếu, vé, và các bảng liên kết để quản lý phân quyền, thể loại phim, bình luận, giá vé,...

3.2.1. Mô hình quan hệ dữ liệu ERD



Hình 3.2 Mô hình quan hệ dữ liệu ERD

3.2.2. Mô tả chi tiết các thực thể

TAIKHOAN: Quản lý thông tin người dùng: tên tài khoản, họ tên, ngày sinh, giới tính, email, số điện thoại, mật khẩu và trạng thái tài khoản.

QUYEN_TRUYCAP: Liên kết tài khoản với vai trò truy cập, cho phép phân quyền hệ thống. Mỗi tài khoản có thể có một vai trò cụ thể (quản trị, người dùng, nhân viên,...).

VAITRO: Lưu thông tin vai trò người dùng trong hệ thống (ví dụ: Admin, Khách hàng, Nhân viên...).

PHIM: Chứa thông tin chi tiết của các bộ phim như: tên phim, thời lượng, thể loại, quốc gia sản xuất, đạo diễn, diễn viên, nội dung tóm tắt, trailer, trạng thái hoạt động và trạng thái lịch chiếu.

THELOAIPHIM & PHIM_THELOAIPHIM:

- THELOAIPHIM: Danh mục các thể loại phim (Hành động, Hài, Kinh dị,...).

- PHIM_THELOAIPHIM: Bảng liên kết nhiều-nhiều giữa phim và thể loại.

BINHLUAN: Lưu các bình luận của người dùng về phim, liên kết với tài khoản và mã phim tương ứng.

RAP: Quản lý thông tin rạp chiếu phim bao gồm tên rạp, địa chỉ và trạng thái hoạt động.

PHONGCHIEU: Phòng chiếu thuộc một rạp cụ thể, liên kết với loại phòng (2D, 3D, IMAX...) và có trạng thái sử dụng.

GHE & LOAIGHE:

- GHE: Thông tin từng ghế trong một phòng chiếu (vị trí hàng, cột, trạng thái).

- LOAIGHE: Phân loại ghế (thường, VIP...).

CASCHIEU: Đại diện cho các suất chiếu phim, liên kết với phim và phòng chiếu, có thông tin về ngày chiếu, giờ bắt đầu, giờ kết thúc và trạng thái.

VE: Thông tin vé đặt của người dùng cho một suất chiếu cụ thể. Vé liên kết với ghế, giá vé và lịch chiếu.

GIAVE: Lưu thông tin giá vé tương ứng với từng loại vé, phục vụ quản lý bảng giá động theo từng suất chiếu.

3.2.3. Mối quan hệ giữa các bảng

1:N: Một PHONGCHIEU có nhiều GHE; một PHIM có nhiều CASCHIEU; một TAIKHOAN có thể có nhiều BINHLUAN,...

N:M: Quan hệ giữa PHIM và THELOAIPHIM được tách thành bảng trung gian PHIM_THELOAIPHIM.

Các khóa chính và khóa ngoại được định nghĩa rõ ràng, bảo đảm tính toàn vẹn tham chiếu dữ liệu trong hệ thống.

3.2.4. Đánh giá thiết kế

Thiết kế cơ sở dữ liệu bảo đảm các yêu cầu của hệ thống:

Phù hợp với nguyên lý chuẩn hóa cơ sở dữ liệu (đặc biệt là chuẩn 3NF).

Hỗ trợ truy xuất dữ liệu hiệu quả trong các nghiệp vụ đặt vé, xem phim, quản lý người dùng và phân quyền.

Dễ mở rộng khi bổ sung tính năng mới (chẳng hạn chương trình khuyến mãi, lịch sử giao dịch...).

3.3. Thiết kế API

API (Application Programming Interface) là thành phần trung gian quan trọng giúp giao tiếp giữa frontend (React) và backend (Express.js). Các API trong hệ thống được xây dựng theo tiêu chuẩn RESTful nhằm đảm bảo khả năng mở rộng, bảo trì dễ dàng và tích hợp với các dịch vụ khác trong tương lai. Việc thiết kế API tuân thủ các nguyên tắc rõ ràng về phân tách chức năng, định danh tài nguyên và sử dụng các phương thức HTTP đúng chuẩn như GET, POST, PUT, DELETE.

3.3.1. Mô tả các endpoint chính

Hệ thống MiniCinema được thiết kế theo kiến trúc RESTful API với các nhóm endpoint chính được tổ chức theo chức năng nghiệp vụ. API được xây dựng trên nền tảng Express.js với TypeScript, sử dụng JWT cho xác thực và phân quyền.

Hệ thống API được chia thành ba nhóm chức năng chính tương ứng với các tác vụ nghiệp vụ: xác thực và người dùng, quản lý phim và lịch chiếu, đặt vé và thống kê. Dưới đây là mô tả các endpoint tiêu biểu:

Phân hệ	Endpoint	Phương thức	Mô tả
Xác thực	/api/auth/register	POST	Đăng ký tài khoản người dùng mới.
Xác thực	/api/auth/login	POST	Đăng nhập, trả về token JWT.
Người dùng	/api/users/profile	GET	Truy xuất thông tin người dùng hiện tại.
Phim	/api/movies	GET	Lấy danh sách tất cả các phim hiện có.
Phim	/api/movies/:id	GET	Lấy chi tiết phim theo mã phim.
Lịch chiếu	/api/schedule	GET	Truy xuất danh sách lịch chiếu theo ngày hoặc phim.
Lịch chiếu	/api/schedule	POST	Tạo mới lịch chiếu (admin).
Vé	/api/tickets/book	POST	Người dùng đặt vé với thông tin ghế và suất chiếu.
Vé	/api/tickets/user	GET	Xem danh sách vé đã đặt của người dùng hiện tại.
Rạp	/api/cinemas	GET	Lấy danh sách rạp đang hoạt động.
Thống kê	/api/admin/statistics	GET	Truy xuất doanh thu và số lượng vé bán (admin).

Các endpoint có phân quyền rõ ràng: người dùng thường chỉ được phép truy cập các chức năng liên quan đến trải nghiệm đặt vé; trong khi quản trị viên có thể quản lý phim, rạp, lịch chiếu và thống kê toàn hệ thống.

3.3.2. Cấu trúc request/response

Việc thiết kế cấu trúc request và response trong hệ thống đảm bảo tính nhất quán và dễ hiểu. Dữ liệu trao đổi được định dạng bằng JSON. Dưới đây là một số ví dụ điển hình:

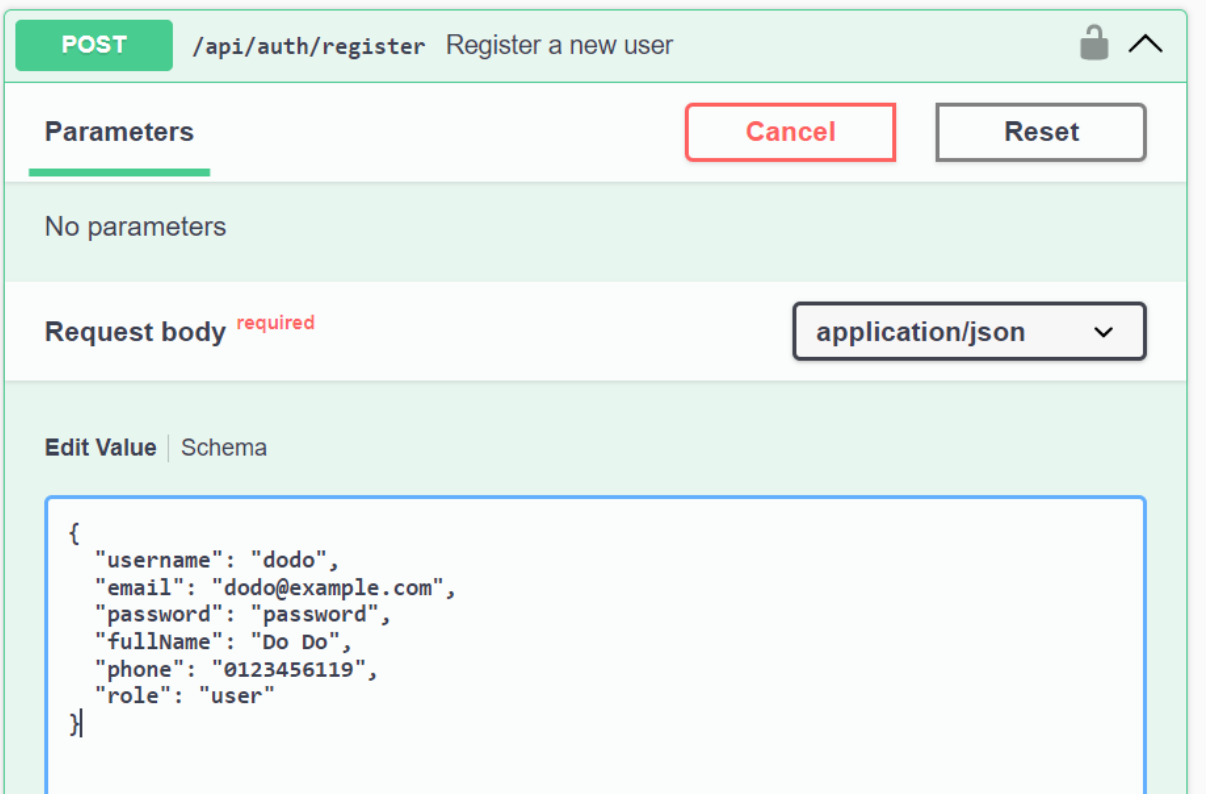
3.3.2.1. Đăng ký người dùng

Phương thức: POST

Endpoint: /api/auth/register

Request body:

```
{
  "username": "dodo",
  "email": "dodo@example.com",
  "password": "password",
  "fullName": "Do Do",
  "phone": "0123456119",
  "role": "user"
}
```



Hình 3.3. Request gửi dữ liệu đăng ký người dùng

Response:

```
{
"user": {
    "id": 978025916,
    "username": "dodo",
    "email": "dodo@example.com",
    "fullName": "Do Do",
    "role": "user"
},

"token": "eyJhbGci...." }
```

Response body


```
{
  "user": {
    "id": 978025916,
    "username": "dodo",
    "email": "dodo@example.com",
    "fullName": "Do Do",
    "role": "user"
  },
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjk3ODAyN
TkxNiwiZW1haWwiOiJkb2RvQGV4YW1wbGUuY29tIiwicm9sZSI6InVzZXIiLCJpYXQiOjE
3NTMxNTU0MDAsImV4cCI6MTc1MzI0MTgwMH0.2ffqxi21xNTx3JNeAUYhrmEqvIof7xhGp
eUMETUfVdg"
}
```


Download

Hình 3.4 Response khi đăng ký thành công

3.3.2.2. Đăng nhập và nhận JWT

Phương thức: POST

Endpoint: /api/auth/login

Request body:

```
{
  "email": "dodo@example.com",
  "password": "password"
}
```

Request body required

application/json

Edit Value | Schema

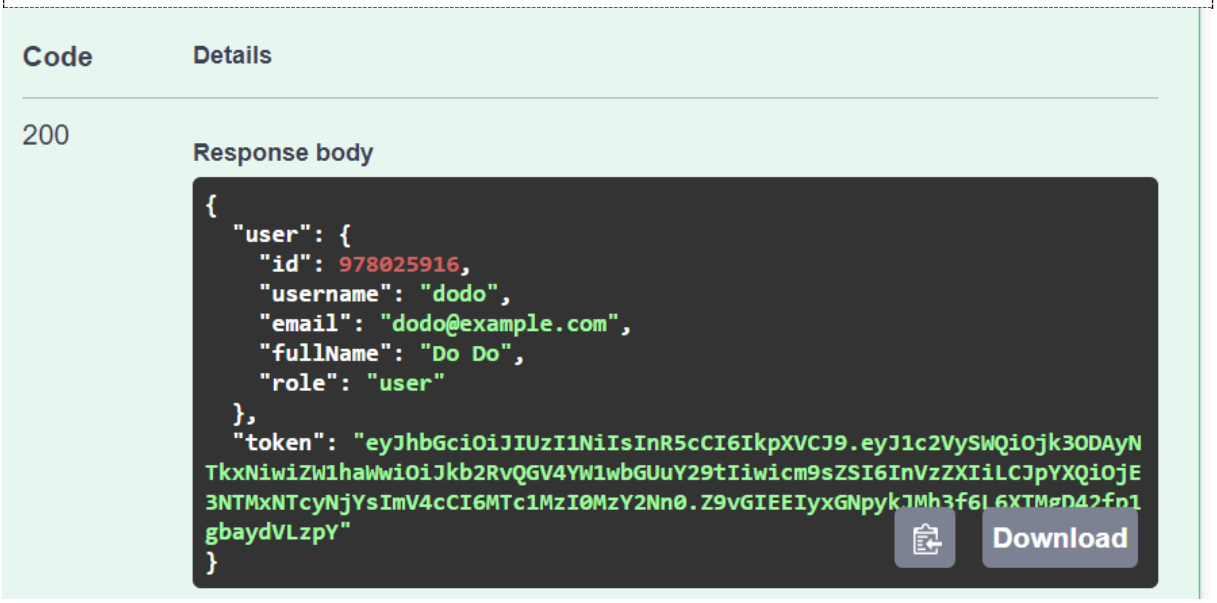
```
{
  "email": "dodo@example.com",
  "password": "password"
}
```

Hình 3.5 Request gửi dữ liệu đăng nhập

Response:

```
{
   "user": {
"id": 978025916,
```

```
"username": "dodo",
"email": "dodo@example.com",
"fullName": "Do Do",
"role": "user"
},
"token": "eyJhbGci...." }
```

Hình 3.6 Response khi đăng nhập thành công

3.3.2.3. Lấy danh sách rạp phim

Phương thức: GET

Endpoint: /api/cinema

Response:


```
[ {
"name": "Beta Cinema",
"address": "Tầng 3, TTTM Golden Palace, Mễ Trì, Nam Từ
Liêm, Hà Nội",
"phone": "1900 2610",
"createdAt": "2025-07-15T03:20:15.226Z",
"id": 2089026061
},
```

```
....
```

Code | Details

200

Response body


```
[
  {
    "name": "Beta Cinema",
    "address": "Tầng 3, TTTM Golden Palace, Mễ Trì, Nam Từ Liêm, Hà Nội",
    "phone": "1900 2610",
    "createdAt": "2025-07-15T03:20:15.226Z",
    "id": 2089026061
  },
  {
    "name": "CGV Vincom Center",
    "address": "Tầng 4, TTTM Vincom Plaza Trà Vinh, số 24 Đường Nguyễn Thị Minh Khai, Khóm 3, Phường 2, Thành phố Trà Vinh, Tỉnh Trà Vinh",
    "phone": "1900 6017",
    "createdAt": "2025-07-15T03:20:15.226Z",
    "id": 2089026019
  },
```


Hình 3.7 Response khi lấy danh sách rap phim

3.3.2.4. Lấy danh sách phim

Phương thức: GET

Endpoint: /api/movies

Response:

```
[
  {
"title": "MỘT NỬA HOÀN HẢO",
“description”: “Lucy (Dakota Johnson), một ...”
    "genre": ["Tâm lý", "Hài", “Tình cảm”],
    "duration": 117,
    ....
  },
  ...]
```

Code Details

200

Response body


```
[
  {
    "title": "MỘT NỬA HOÀN HẢO",
    "description": "Lucy (Dakota Johnson), một cô gái xinh đẹp làm
công việc mai mối ở New York. “Mát tay” trong chuyện mai mối giúp n
gười khác, nhưng trớ trêu Lucy lại “lạc lối” trong câu chuyện tình
cảm của chính mình. Bất ngờ đối mặt với ngã rẽ tình cảm khi gặp lại
người yêu cũ \"không hoàn hảo\" (Chris Evans) và một người đàn ông
\"hoàn hảo\" (Pedro Pascal). Cô nàng bị đẩy vào tình thế khó xử. Đứ
ng giữa những lựa chọn cảm xúc đầy mâu thuẫn, lúc này đây, Lucy mới
nhận ra việc tìm kiếm tình yêu đích thực chưa bao giờ là điều dễ dà
ng",
    "genre": [
      "Tâm lý",
      "Hài",
      "Tình cảm"
    ],
    "duration": 117,
    "ageRating": "16+",
    "posterUrl": "https://iguov8nhvyobj.vcdn.cloud/media/catalog/pr
oduct/cache/1/image/c5f0a1eff4c394a251036189ccddaacd/3/5/350x495-ma
terialists.jpg",
    "trailerUrl": "https://youtu.be/C_Y2nNESqSo",
    "actors": [
      "Dakota Johnson",
      "Chris Evans",
      "Pedro Pascal"
```


Download

Hình 3.8 Response khi lấy danh sách phim

3.5. Thiết kế giao diện (UI/UX)

Giao diện người dùng của hệ thống được thiết kế bằng công cụ Figma, tuân theo các nguyên tắc thiết kế hiện đại như tính nhất quán, dễ sử dụng và tối ưu cho đa thiết bị. Thiết kế tập trung vào trải nghiệm trực quan, thao tác đơn giản và phản hồi nhanh chóng nhằm nâng cao mức độ hài lòng của người dùng.

Một số màn hình chính trong thiết kế bao gồm:

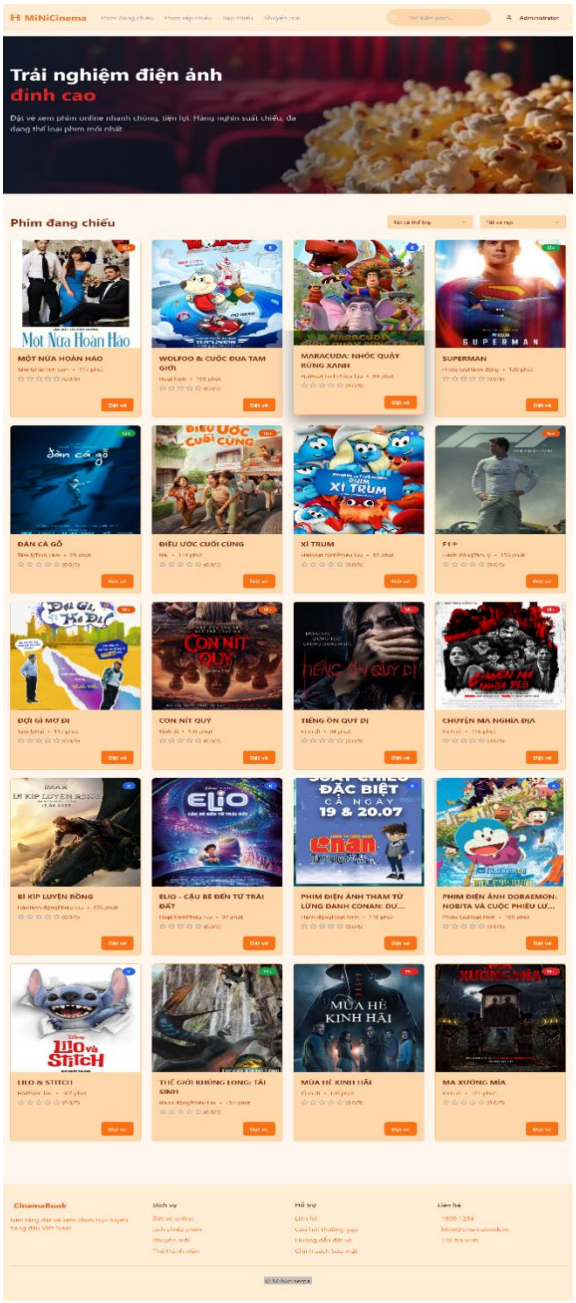
- Trang chủ: Hiển thị danh sách phim đang và sắp chiếu.

- Chi tiết phim: Cung cấp thông tin, trailer và lịch chiếu.

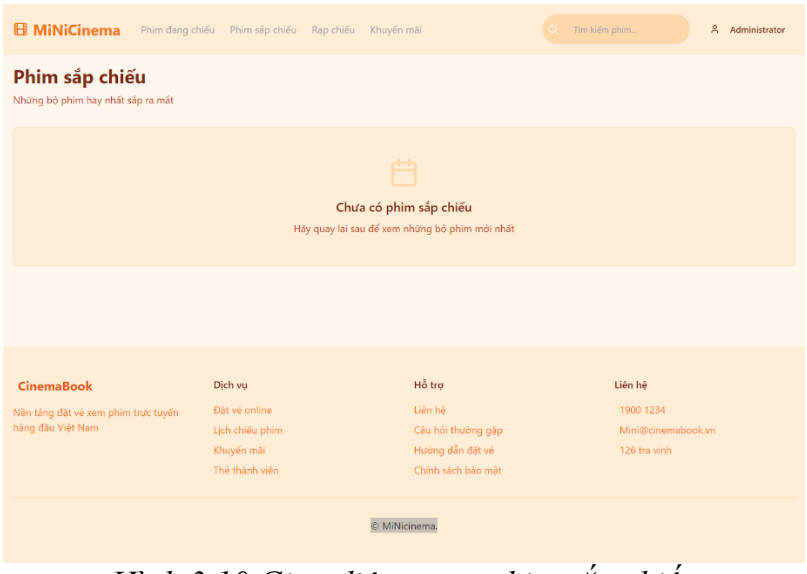
- Đặt vé: Cho phép người dùng chọn suất chiếu, ghế ngồi và thanh toán.

- Quản trị: Dành cho admin quản lý phim, suất chiếu và người dùng.

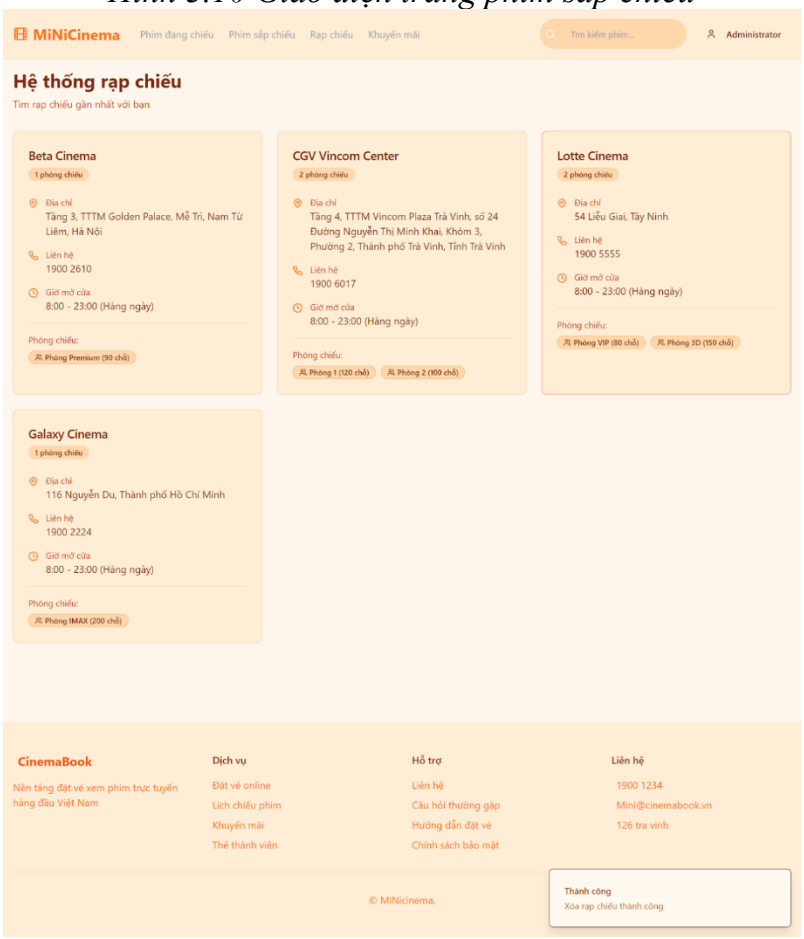
3.5.1. Các giao diện của website



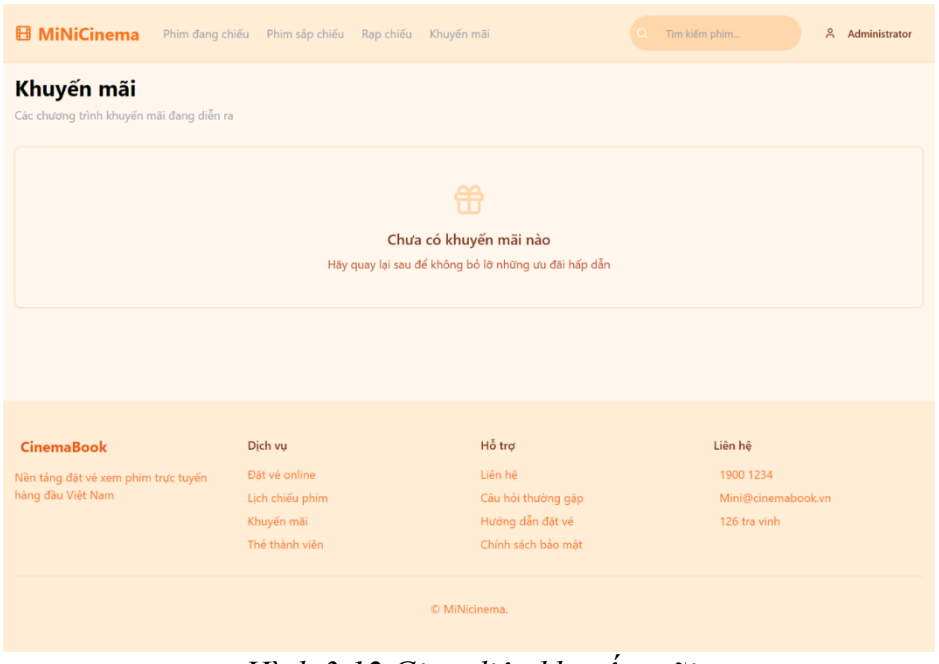
Hình 3.9 Giao diện trang chủ



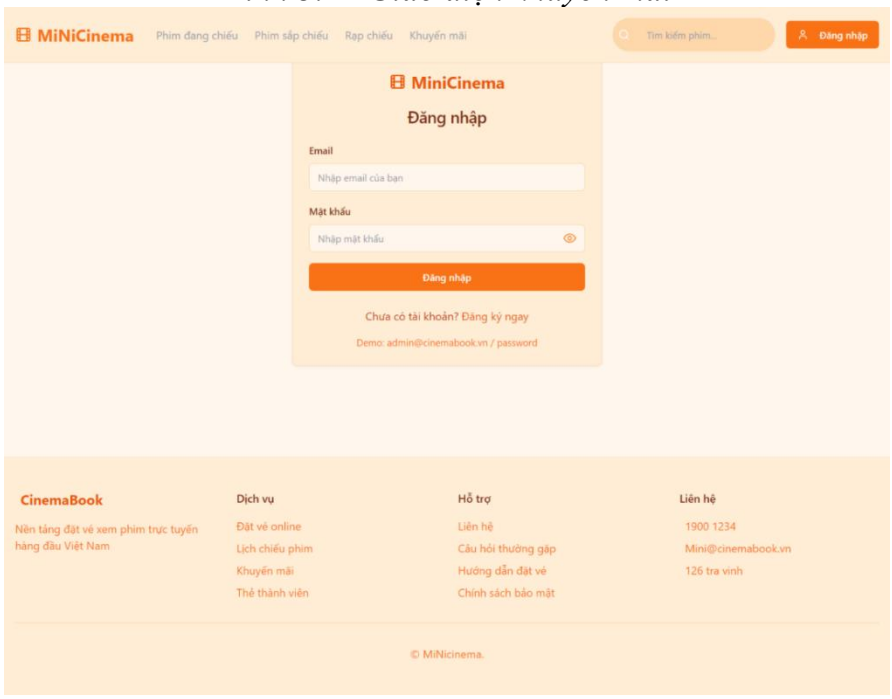
Hình 3.10 Giao diện trang phim sắp chiếu



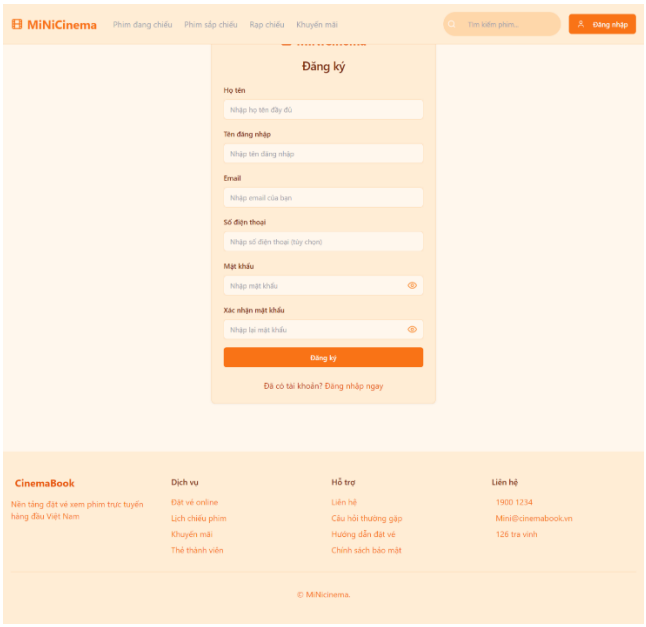
Hình 3.11 Giao diện rạp chiếu



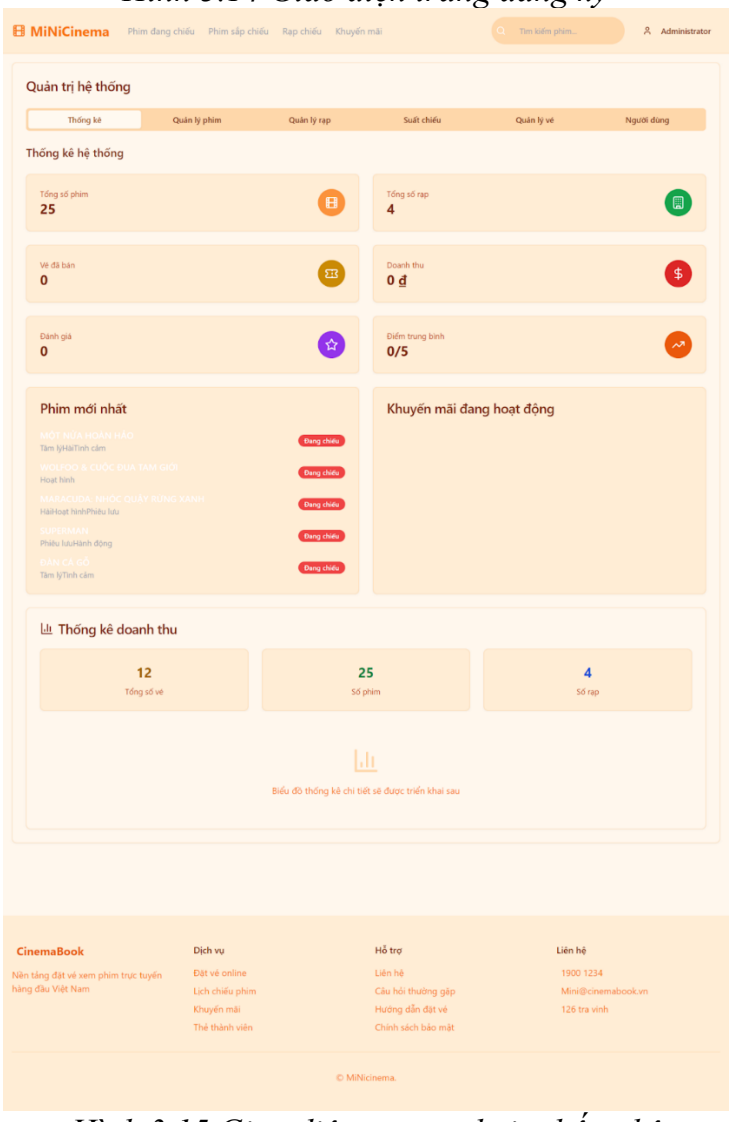
Hình 3.12 Giao diện khuyến mãi



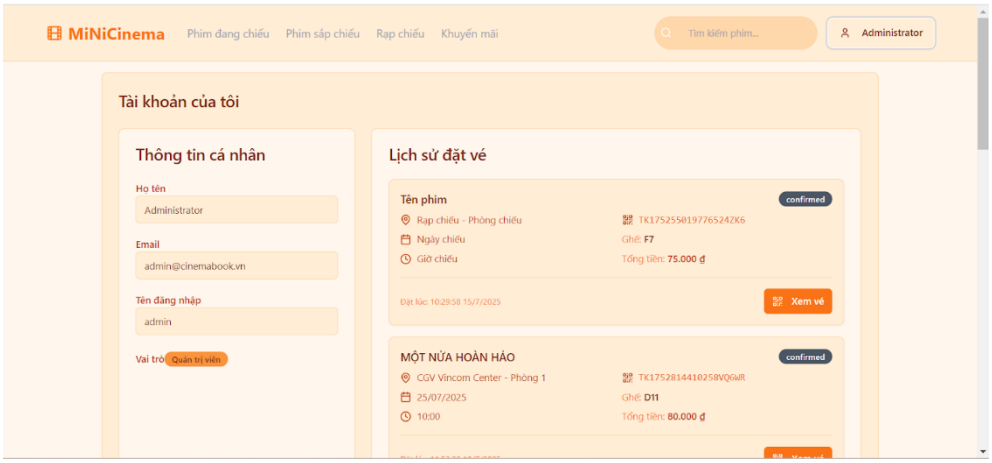
Hình 3.13 Giao diện đăng nhập



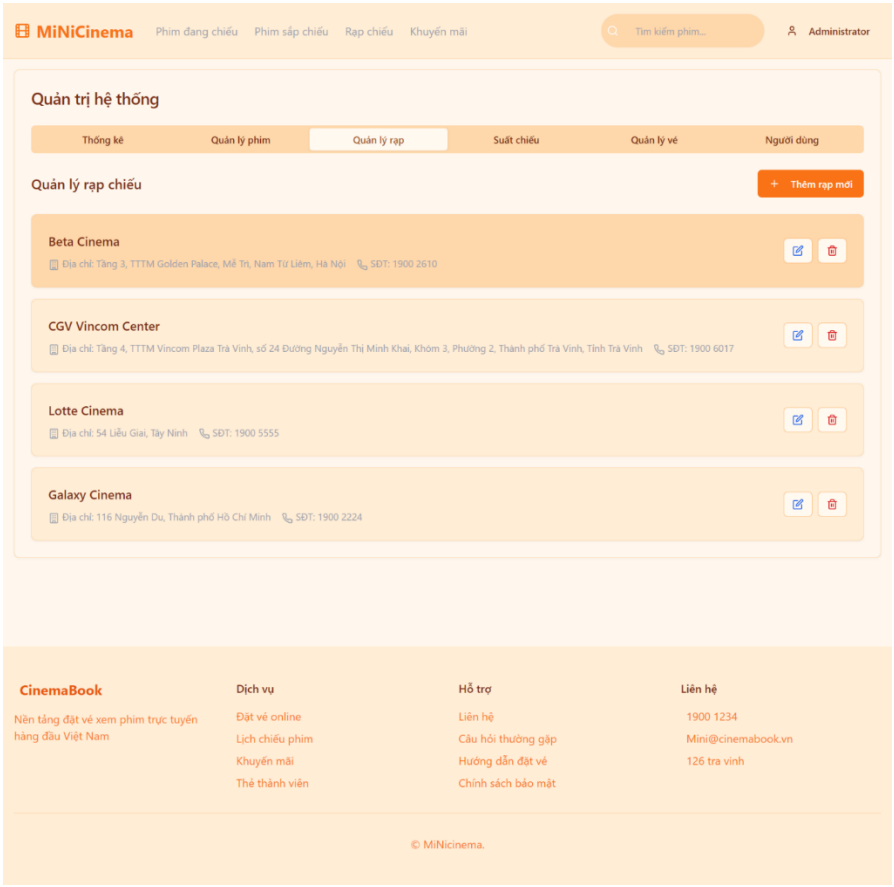
Hình 3.14 Giao diện trang đăng ký



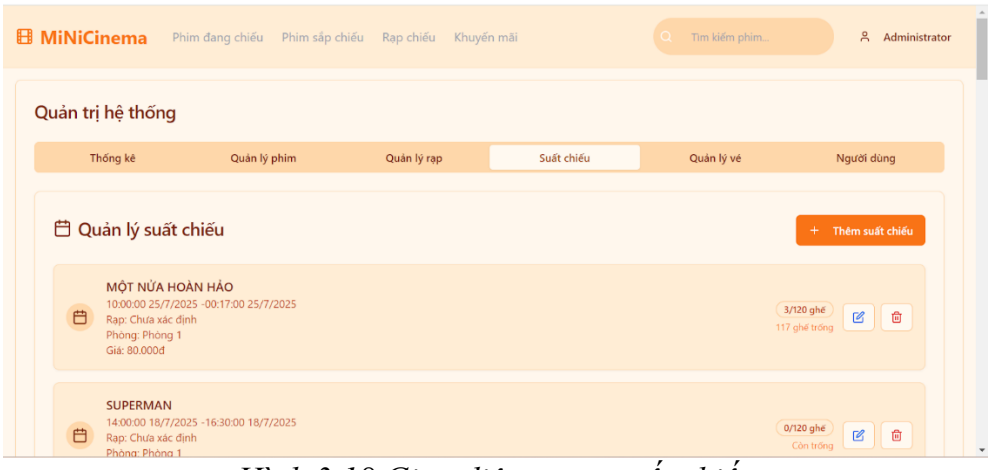
Hình 3.15 Giao diện trang admin thống kê



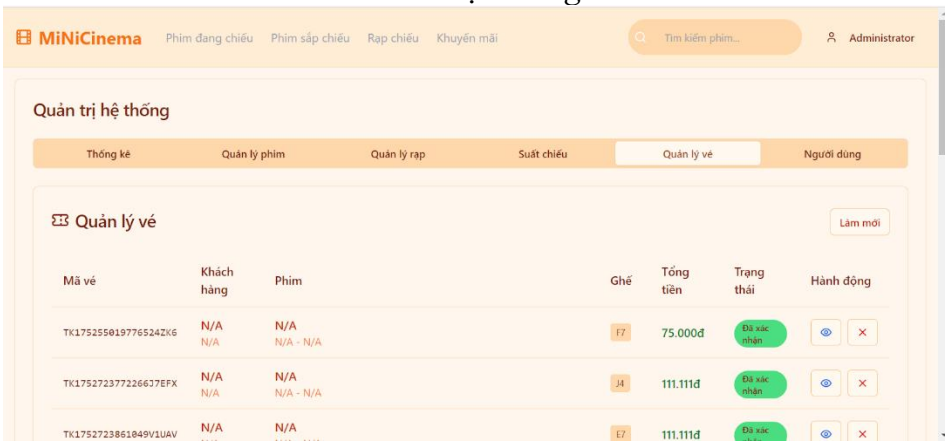
Hình 3.16 Giao diện tài khoản cá nhân



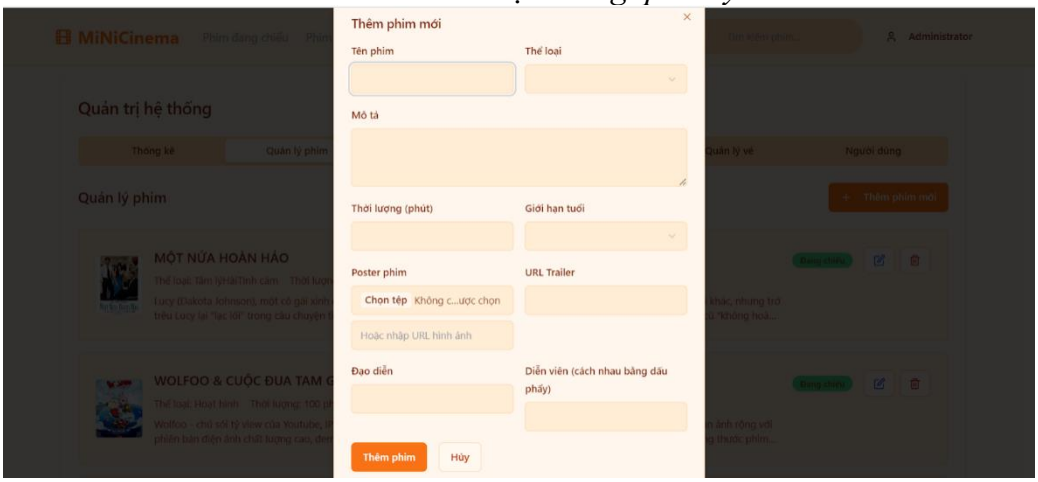
Hình 3.17 Giao diện quản lý rạp phim



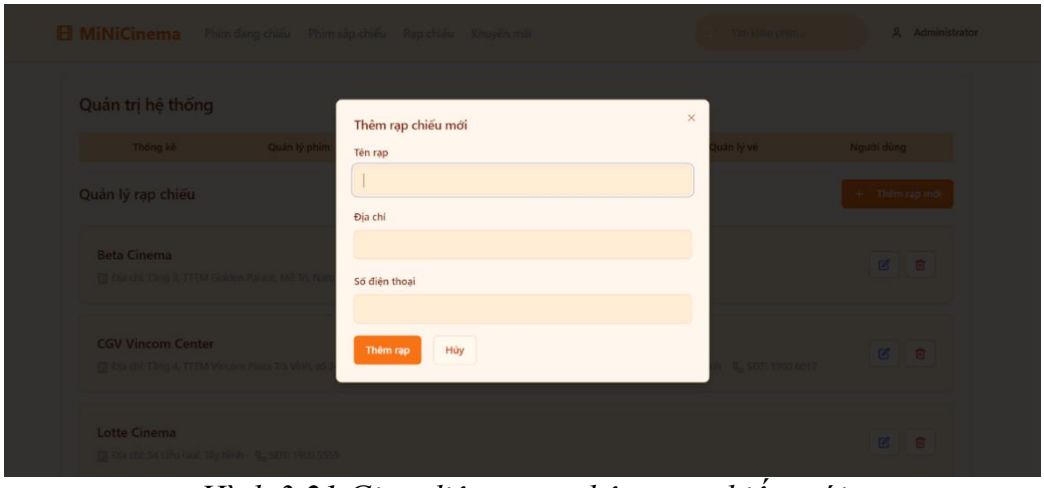
Hình 3.18 Giao diện trang suất chiếu



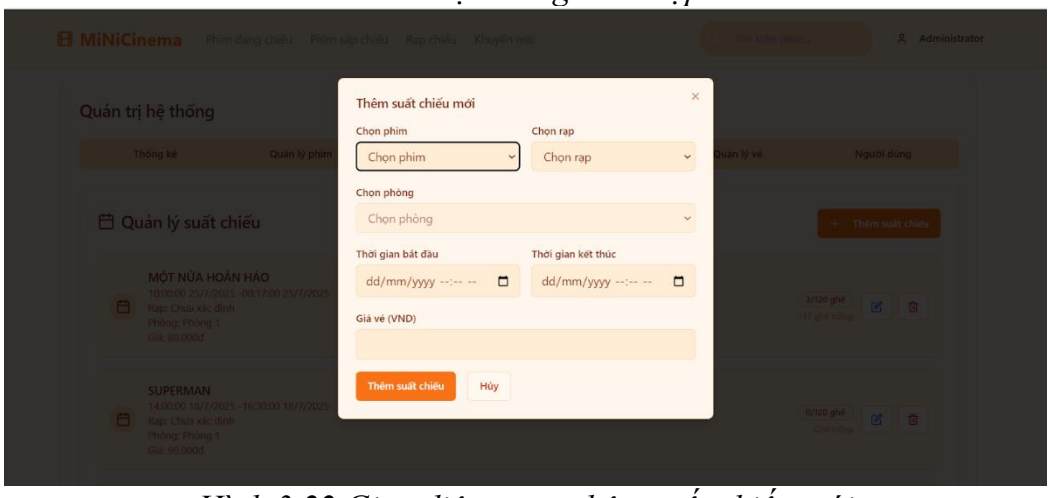
Hình 3.19 Giao diện trang quản lý vé



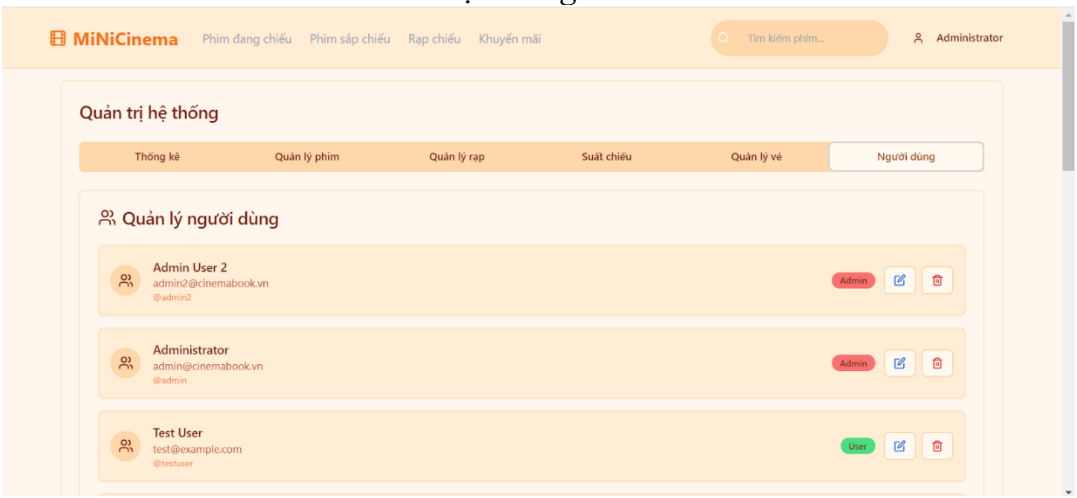
Hình 3.20 Giao diện trang thêm phim mới



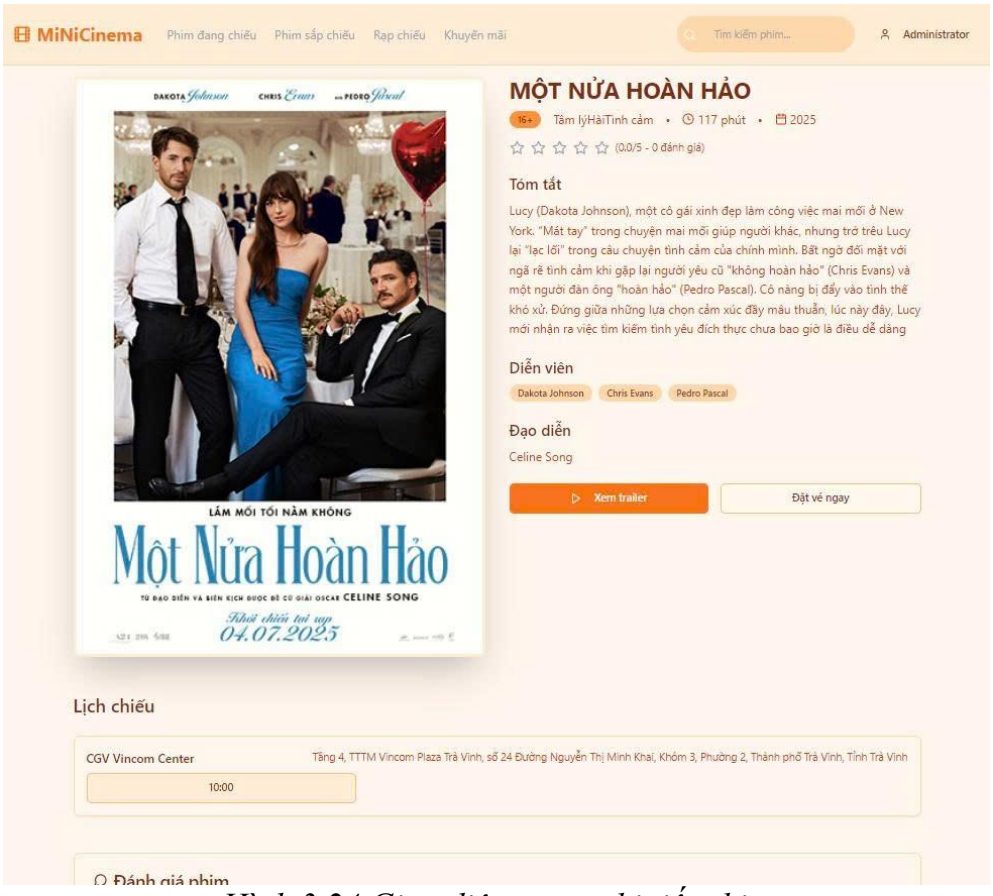
Hình 3.21 Giao diện trang thêm rạp chiếu mới



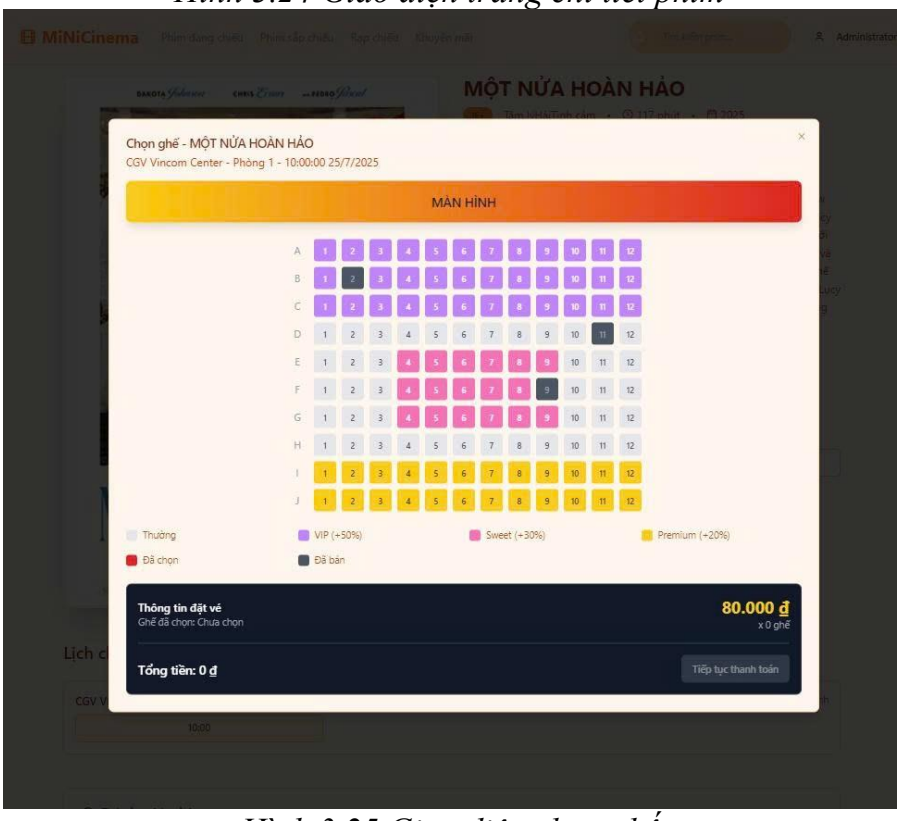
Hình 3.22 Giao diện trang thêm suất chiếu mới



Hình 3.23 Giao diện trang danh sách người dùng



Hình 3.24 Giao diện trang chi tiết phim



Hình 3.25 Giao diện chọn ghế

Chương 4. TRIỂN KHAI VÀ CÔNG NGHỆ SỬ DỤNG

4.1 Các công nghệ đã sử dụng

Trong quá trình phát triển hệ thống ứng dụng web, nhóm đã lựa chọn và triển khai một số công nghệ chủ chốt nhằm đảm bảo hiệu quả vận hành, khả năng mở rộng và bảo mật của hệ thống. Các công nghệ được phân chia theo ba lớp chính: giao diện người dùng (frontend), xử lý logic nghiệp vụ (backend), cơ sở dữ liệu (database) và xác thực bảo mật (authentication). Cụ thể:

4.1.1 Ngôn ngữ lập trình chính

4.1.1.1 TypeScript

TypeScript phiên bản 5.6.3 đóng vai trò là ngôn ngữ lập trình chính cho toàn bộ dự án NaCinema, bao gồm cả các phần frontend và backend. Việc sử dụng TypeScript mang lại nhiều lợi ích đáng kể:[4]

- An toàn kiểu dữ liệu (Type Safety): Đảm bảo tính nhất quán và chính xác của dữ liệu trên toàn bộ codebase, giảm thiểu lỗi phát sinh trong quá trình phát triển.

- IntelliSense và Tự động hoàn thành (Auto-completion): Cải thiện năng suất lập trình thông qua khả năng gợi ý mã và phát hiện lỗi cú pháp sớm.

- Phát hiện lỗi tại thời điểm biên dịch (Compile-time Error Detection): Cho phép phát hiện và sửa lỗi trước khi ứng dụng được triển khai.

- Khả năng tái cấu trúc (Refactoring Capabilities): Dễ dàng thực hiện các thay đổi cấu trúc mã nguồn một cách an toàn và hiệu quả.

- Chia sẻ kiểu dữ liệu (Shared Types): Cho phép định nghĩa và sử dụng chung các kiểu dữ liệu giữa frontend và backend, đảm bảo tính đồng bộ.

Cấu hình TypeScript được thể hiện qua tệp tsconfig.json:

```
{
  "include": ["client/src/**/*", "shared/**/*",
"server/**/*"],
```

```
  "exclude": ["node_modules", "build", "dist",
"**/*.test.ts"],
  "compilerOptions": {
    "incremental": true,
    "tsBuildInfoFile":
"./node_modules/typescript/tsbuildinfo",
    "noEmit": true,
    "module": "ESNext",
    "strict": true,
    "lib": ["esnext", "dom", "dom.iterable"],
    "jsx": "preserve",
    "esModuleInterop": true,
    "skipLibCheck": true,
    "allowImportingTsExtensions": true,
    "moduleResolution": "bundler"
  }
}
```

Shared Types Architecture: Dự án sử dụng thư mục shared/ để chia sẻ types và schema giữa frontend và backend:

Database Schema: Định nghĩa trong shared/schema.ts với Drizzle ORM

API Types: Các interface cho User, Movie, Cinema, Room, Showtime, Ticket, Review, Promotion

Validation Schemas: Sử dụng Zod để validate dữ liệu đầu vào

4.1.1.2 JavaScrip

JavaScript, đặc biệt là các tính năng của ESNext modules với cú pháp import/export, được sử dụng cho các tệp cấu hình (configuration files) và các tập lệnh

xây dựng (build scripts). Điều này giúp tận dụng tính linh hoạt của JavaScript trong các tác vụ không yêu cầu kiểm tra kiểu nghiêm ngặt.

4.1.2 Frontend

Cấu trúc thư mục:

```
├── src
│   ├── assets        # Hình ảnh, icon
│   ├── components    # Các component dùng chung
│   ├── features      # Các chức năng như login, booking,...
│   ├── pages         # Các trang chính: Home, Movies, Tickets
│   ├── routes        # Khai báo route cho ứng dụng
│   ├── redux         # Quản lý state toàn cục
│   ├── hooks         # Các custom hook
│   └── App.jsx       # File khởi động chính
├── tailwind.config.ts
│   └── vite.config.ts
```

4.1.2.1 React 18.3.1

React 18.3.1 là thư viện JavaScript chính cho việc xây dựng giao diện người dùng (UI) của MiniCinema. Các tính năng nổi bật được áp dụng bao gồm:

- Functional Components với Hooks: Giúp quản lý trạng thái và logic trong các thành phần hàm một cách hiệu quả.

- React Query: Được sử dụng để quản lý trạng thái máy chủ (server state management), cung cấp khả năng caching, cập nhật nền và xử lý lỗi tối ưu.

- Context API: Hỗ trợ quản lý trạng thái toàn cục (global state), đơn giản hóa việc chia sẻ dữ liệu giữa các thành phần.

- Suspense và Error Boundaries: Cải thiện trải nghiệm người dùng bằng cách xử lý các trường hợp tải dữ liệu và lỗi một cách linh hoạt.

4.1.2.2 Vite 5.4.19

Vite 5.4.19 đóng vai trò là công cụ xây dựng (build tool) và máy chủ phát triển (development server) chính. Vite mang lại những ưu điểm vượt trội:

- Hot Module Replacement (HMR): Cho phép cập nhật mã nguồn mà không cần tải lại toàn bộ trang, tăng tốc độ phát triển.

- Thời gian xây dựng nhanh (Fast Build Times): Sử dụng các công nghệ biên dịch hiệu quả để rút ngắn thời gian xây dựng ứng dụng.

- Hỗ trợ ES Modules (ES modules support): Tận dụng các tính năng module gốc của JavaScript.

- Hệ sinh thái plugin (Plugin Ecosystem): Cung cấp khả năng mở rộng thông qua các plugin đa dạng.

Cấu hình Vite được định nghĩa trong vite.config.ts:

```
export default defineConfig({
  plugins: [
    react(),
  ],
  resolve: {
    alias: {
      "@": path.resolve(__dirname, "client", "src"),
      "@shared": path.resolve(__dirname, "shared"),
      "@assets": path.resolve(__dirname,
"attached_assets"),
    },
  },
  root: path.resolve(__dirname, "client"),
  build: {
    outDir: path.resolve(__dirname, "dist/public"),
    emptyOutDir: true,
  }
});
```

4.1.2.3 Tailwind CSS 3.4.17

Tailwind CSS 3.4.17 là framework CSS theo hướng tiện ích (utility-first), cho phép xây dựng giao diện nhanh chóng và linh hoạt bằng cách sử dụng các lớp tiện ích được định nghĩa sẵn.[9]

Extensions: Dự án tích hợp @tailwindcss/typography để xử lý kiểu chữ và tailwindcss-animate để tạo các hiệu ứng động mượt mà.

Cấu hình Tailwind CSS:

```
export default {
  darkMode: ["class"],
  content: ["./client/index.html",
"./client/src/**/*.{js,jsx,ts,tsx}"],
  theme: {
    extend: {
      borderRadius: {
        lg: "var(--radius)",
        md: "calc(var(--radius) - 2px)",
        sm: "calc(var(--radius) - 4px)",
      },
      colors: {
        background: "var(--background)",
        foreground: "var(--foreground)",
        // ... custom color system
      }
    }
  },
```

```
  plugins: [require("tailwindcss-animate"),
require("@tailwindcss/typography")]

}
```

4.1.2.4 Thư viện Thành phần UI (UI Component Libraries)

Dự án sử dụng kết hợp nhiều thư viện để xây dựng các thành phần UI chất lượng cao:

Radix UI: Một thư viện thành phần toàn diện (hơn 25 thành phần như Dialog, Dropdown, Select, v.v.) tập trung vào khả năng tiếp cận (accessibility-first design) và cung cấp các thành phần không kiểu dáng (unstyled components) để tùy chỉnh hoàn toàn.

Lucide React: Thư viện biểu tượng với hơn 450 biểu tượng, đảm bảo tính nhất quán và đa dạng cho giao diện.

Framer Motion: Thư viện hoạt ảnh mạnh mẽ, giúp tạo ra các chuyển động mượt mà và hiệu ứng động phong phú cho giao diện người dùng.

4.1.3 Backend

Cấu trúc thư mục:

```
├── src
│   ├── config        # Cấu hình môi trường, CSDL
│   ├── controller    # Xử lý nghiệp vụ cho route
│   ├── middleware    # Xác thực, lỗi
│   ├── model         # Định nghĩa schema MongoDB
│   ├── routes        # Các route chính
│   ├── service       # Xử lý logic nghiệp vụ
│   └── utils         # Helper functions
```

4.1.3.1 Node.js với Express.js

Backend của Movie Ticket Booking được xây dựng trên Node.js và sử dụng Express.js 4.21.2 làm framework ứng dụng web. Express.js cung cấp một nền tảng mạnh mẽ và linh hoạt để xây dựng các RESTful API.

Middleware Stack: Dự án sử dụng một loạt các middleware để xử lý các chức năng quan trọng:

- express-session: Quản lý phiên người dùng.
- CORS middleware: Cho phép chia sẻ tài nguyên giữa các nguồn gốc khác nhau.
- Custom authentication middleware: Middleware xác thực tùy chỉnh để quản lý quyền truy cập.
- Error handling middleware: Xử lý lỗi tập trung cho toàn bộ ứng dụng.[1]

4.1.3.2 Cơ sở dữ liệu

- MongoDB 6.17.0: Là cơ sở dữ liệu chính của dự án, được sử dụng để lưu trữ dữ liệu phi cấu trúc (NoSQL data) như thông tin phim, người dùng,…

Cấu hình MongoDB trong Docker Compose:[8]

```
mongo:
  image: mongo:7.0
  container_name: minicinema_mongo
  restart: always
  environment:
    MONGO_INITDB_ROOT_USERNAME: admin
    MONGO_INITDB_ROOT_PASSWORD: minicinema123
    MONGO_INITDB_DATABASE: movie
  volumes:
    - mongo_data:/data/db
    - ./mongo-init.js:/docker-entrypoint-initdb.d/mongo-
init.js:ro
```

- Drizzle ORM 0.39.1: Một ORM (Object-Relational Mapper) an toàn kiểu dữ liệu, được sử dụng để tương tác với cơ sở dữ liệu. Mặc dù cấu hình hiển thị PostgreSQL, việc sử dụng Drizzle ORM với MongoDB có thể được thực hiện thông qua các adapter hoặc driver tương ứng.

Cấu hình Drizzle ORM:

```
export default defineConfig({
  out: "./migrations",
  schema: "./shared/schema.ts",
  dialect: "postgresql",
  dbCredentials: {
    url: process.env.DATABASE_URL,
  },
});
```

- Redis 7: Được sử dụng làm lớp bộ nhớ đệm (caching layer) để lưu trữ dữ liệu thường xuyên truy cập, giúp tăng tốc độ phản hồi của ứng dụng.

4.1.4 Xác thực và bảo mật

JSON Web Tokens (JWT):

- jsonwebtoken 9.0.2: Được sử dụng để triển khai xác thực dựa trên token (token-based authentication), cung cấp một phương pháp an toàn và không trạng thái (stateless) để xác minh danh tính người dùng.

-bcrypt 6.0.0: Một thư viện mạnh mẽ để băm mật khẩu (password hashing), đảm bảo rằng mật khẩu được lưu trữ an toàn trong cơ sở dữ liệu.

```
// Registration endpoint
const hashedPassword = await
bcrypt.hash(userData.password, 10);
// Login verification
const isValid = await bcrypt.compare(password,
user.password);
```

- Passport.js 0.7.0: Một middleware xác thực linh hoạt, với việc sử dụng passport-local cho chiến lược xác thực cục bộ (local authentication strategy).

Cấu hình xác thực bằng JWT (middlewares/auth.js):

```
import jwt from "jsonwebtoken";

export const authenticateToken = (req, res, next) => {

  const token = req.headers["authorization"]?.split(" ")[1];

  if (!token) return res.status(401).json({ message: "Unauthorized" });

  jwt.verify(token, process.env.JWT_SECRET, (err, user) => {

    if (err) return res.status(403).json({ message: "Forbidden" });

    req.user = user;

    next();

  });

};
```

4.2 Quy trình CI/CD với GitHub Actions

Trong quá trình phát triển hệ thống đặt vé xem phim trực tuyến, nhóm đã áp dụng quy trình CI/CD (Continuous Integration và Continuous Deployment) nhằm đảm bảo rằng phần mềm được xây dựng, kiểm thử và triển khai một cách tự động, nhanh chóng và nhất quán. Công cụ được sử dụng cho mục tiêu này là GitHub Actions, một dịch vụ CI/CD tích hợp sẵn trong nền tảng GitHub.[3]

Quy trình CI/CD được thiết kế theo các giai đoạn sau:

- Continuous Integration (CI): Mỗi khi có thay đổi mã nguồn được đẩy lên nhánh chính (main hoặc develop), GitHub Actions sẽ tự động kích hoạt workflow để

thực hiện các bước như kiểm thử mã (unit test), kiểm tra định dạng (linting), và build ứng dụng.

- Continuous Deployment (CD): Nếu quy trình CI thành công, workflow tiếp tục thực hiện bước triển khai (deploy) tự động lên môi trường máy chủ đã cấu hình trước đó (có thể là server thực tế hoặc nền tảng cloud). Việc triển khai này có thể bao gồm việc chạy Docker container hoặc cập nhật mã nguồn trên máy chủ đích.

- Quy trình CI/CD giúp phát hiện lỗi sớm, tránh xung đột trong quá trình tích hợp mã, và giảm thời gian triển khai, từ đó nâng cao chất lượng sản phẩm.

Cấu hình file:

```
name: Lint Code Base
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]
jobs:
  run-lint:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4
        with:
          # Full git history is needed to get a proper
list of changed files within `super-linter`
          fetch-depth: 0

      - name: Lint Code Base
        uses: github/super-linter@v4
        env:
          VALIDATE_ALL_CODEBASE: false
          DEFAULT_BRANCH: "main"
```

```
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Nhờ GitHub Actions, nhóm đã có thể rút ngắn đáng kể thời gian triển khai và kiểm thử thủ công, đồng thời đảm bảo rằng mỗi thay đổi đều được kiểm tra tự động trước khi đưa vào sản phẩm chính thức.

4.3 Cấu hình Docker và quy trình triển khai ứng dụng

4.3.1 Docker

Docker là nền tảng đóng gói ứng dụng và các thành phần phụ thuộc vào một container nhẹ, giúp đảm bảo ứng dụng chạy nhất quán trên mọi môi trường. Trong dự án này, nhóm đã sử dụng Docker để xây dựng và triển khai cả frontend và backend của hệ thống đặt vé xem phim.[7]

Cấu trúc triển khai Docker

Ứng dụng được chia thành hai phần chính:

Frontend: Viết bằng React, được đóng gói thành một Docker image sử dụng nginx để phục vụ giao diện người dùng.

Backend: Sử dụng Node.js với Express, đóng gói thành một image riêng để xử lý API.

Database: Sử dụng MongoDB, được khởi chạy thông qua Docker image chính thức mongo.

Toàn bộ kiến trúc được quản lý thông qua tệp docker-compose.yml, đảm bảo các dịch vụ có thể liên kết và khởi động cùng lúc.

Cấu trúc file docker-compose.yml:

```
version: '3.8'

services:

  # MongoDB Database

  mongo:
```

```
    image: mongo:7.0
    container_name: minicinema_mongo
    restart: unless-stopped
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin
      MONGO_INITDB_ROOT_PASSWORD: minicinema123
      MONGO_INITDB_DATABASE: movie
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db
      - ./mongo-init.js:/docker-entrypoint-
initdb.d/mongo-init.js:ro
    networks:
      - minicinema_network
    healthcheck:
      test: ["CMD", "mongosh", "--eval",
"db.adminCommand('ping')"]
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
  # Redis Cache
  redis:
    image: redis:7-alpine
```

```
    container_name: minicinema_redis
    restart: unless-stopped
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data
    networks:
      - minicinema_network
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 10s
  # Main Application
  app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: minicinema_app
    restart: unless-stopped
    ports:
      - "5000:5000"
    depends_on:
      mongo:
```

```
        condition: service_healthy
      redis:
        condition: service_healthy
    environment:
      - NODE_ENV=development
      -
MONGODB_URI=mongodb://admin:minicinema123@mongo:27017/mo
vie?authSource=admin
      - REDIS_URL=redis://redis:6379
      - JWT_SECRET=your-super-secret-jwt-key-change-in-
production
      - PORT=5000
    volumes:
      - ./uploads:/app/uploads:rw
      - app_logs:/app/logs
    networks:
      - minicinema_network
    healthcheck:
      test: ["CMD", "curl", "-f",
"http://localhost:5000/api/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 60s
  # Nginx reverse proxy
  nginx:
```

```
    image: nginx:alpine
    container_name: minicinema_nginx
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - ./ssl:/etc/nginx/ssl:ro
    depends_on:
      - app
    networks:
      - minicinema_network
volumes:
  mongo_data:
    driver: local
  redis_data:
    driver: local
  app_logs:
    driver: local
networks:
  minicinema_network:
    driver: bridge
```

4.3.2 Quy trình triển khai ứng dụng

Xây dựng Docker Image: Mỗi phần (frontend/backend) được viết Dockerfile để build image riêng.

Khởi chạy Container: Sử dụng docker-compose up --build để khởi tạo toàn bộ hệ thống với cấu hình đã định nghĩa.

Kiểm tra và giám sát: Theo dõi log bằng docker logs và kiểm tra trạng thái hệ thống qua docker ps.

Triển khai thực tế: Docker giúp nhóm dễ dàng chuyển toàn bộ dự án lên VPS hoặc nền tảng đám mây (như Render, Heroku, AWS...) mà không cần cấu hình lại môi trường.

Nhờ việc áp dụng Docker, hệ thống trở nên dễ triển khai, dễ phục hồi khi có lỗi và có thể chạy ổn định trên bất kỳ máy chủ nào hỗ trợ Docker mà không phụ thuộc vào môi trường cụ thể.

4.4 Triển khai và vận hành hệ thống trên nền tảng Render

4.4.1. Giới thiệu về Render

Render là một nền tảng đám mây hiện đại cho phép các lập trình viên dễ dàng triển khai các ứng dụng web, API, dịch vụ nền (background workers), cơ sở dữ liệu và trang web tĩnh. Với Render, người dùng không cần lo lắng về hạ tầng phức tạp như khi sử dụng AWS, nhưng vẫn tận dụng được nhiều tính năng mạnh mẽ như:

- Tự động triển khai từ GitHub/GitLab mỗi khi code được cập nhật.

- Hỗ trợ SSL/TLS miễn phí.

- Tích hợp CI/CD, tự động xây dựng và khởi chạy ứng dụng.

- Tự động scale và restart ứng dụng khi có sự cố.

4.4.2. Quy trình triển khai ứng dụng trên Render

Quy trình triển khai một hệ thống (ví dụ: hệ thống đặt vé phim) trên Render có thể được mô tả qua các bước chính sau:

Bước 1: Chuẩn bị mã nguồn:

Ứng dụng được phát triển với backend (Node.js, Express) và frontend (React).

Mã nguồn được quản lý trên GitHub, cấu trúc rõ ràng với thư mục client/ (React) và server/ (Express API).

File render.yaml hoặc cấu hình môi trường .env được thêm để khai báo thông số hệ thống.

Bước 2: Kết nối với Render

Đăng nhập vào https://render.com bằng tài khoản GitHub.

Chọn New Web Service để triển khai ứng dụng backend hoặc frontend.

Chọn repository cần triển khai, nhánh main hoặc production.

Cấu hình:

- Environment: Node

- Build Command: npm install hoặc npm run build

- Start Command: npm start hoặc node index.js

- Port: 3000 hoặc 5000 (tùy ứng dụng)

Bước 3: Thiết lập môi trường

Render cho phép cấu hình các biến môi trường như:

```
PORT=5000

MONGODB_URI=...

JWT_SECRET=...
```

Bước 4: Tự động triển khai (Auto Deploy)

Mỗi khi code được cập nhật trên GitHub, Render sẽ tự động:

- Lấy mã nguồn mới.

- Cài đặt dependency.

- Biên dịch hoặc khởi chạy ứng dụng.

- Cập nhật website ngay lập tức.

Chương 5. QUẢN LÝ DỰ ÁN

5.1 Cách sử dụng Jira để lập kế hoạch và theo dõi tiến độ

Trong quá trình phát triển hệ thống đặt vé xem phim, nhóm đã sử dụng Jira Software để hỗ trợ việc lập kế hoạch, tổ chức công việc và theo dõi tiến độ dự án. Jira là một công cụ quản lý dự án mạnh mẽ, phổ biến trong các nhóm phát triển phần mềm theo mô hình Agile hoặc Scrum. Việc sử dụng Jira giúp nhóm đảm bảo công việc được triển khai có hệ thống, dễ kiểm soát, và minh bạch về tiến độ.

Ban đầu, nhóm tạo một dự án mới trên Jira và lựa chọn template phù hợp với Scrum framework. Từ đó, các Epic (các mục tiêu lớn) được tạo để bao quát toàn bộ hệ thống, ví dụ như xây dựng giao diện người dùng, phát triển API, tích hợp hệ thống thanh toán, và triển khai CI/CD. Các User Story được viết dưới dạng mô tả nhu cầu thực tế của người dùng, giúp nhóm dễ hình dung các chức năng cần phát triển. Sau đó, từng User Story được chia thành các Task hoặc Sub-task để thực hiện cụ thể hơn.

Việc lập kế hoạch được thực hiện theo từng Sprint. Trước mỗi Sprint, nhóm tổ chức cuộc họp lập kế hoạch để chọn lọc các công việc ưu tiên từ backlog và kéo vào sprint backlog. Các công việc được đánh giá độ phức tạp bằng Story Point và sắp xếp thứ tự ưu tiên. Jira cung cấp bảng Sprint Board với các cột trạng thái như To Do, In Progress, In Review và Done để theo dõi trực quan quá trình phát triển phần mềm.

Trong suốt thời gian thực hiện Sprint, nhóm thường xuyên cập nhật trạng thái các công việc trên Jira. Các biểu đồ như Burndown Chart giúp theo dõi lượng công việc còn lại theo thời gian, từ đó xác định mức độ hoàn thành của Sprint và phát hiện sớm các nguy cơ trễ tiến độ. Ngoài ra, các báo cáo như Sprint Report và Velocity Chart hỗ trợ đánh giá hiệu suất làm việc của nhóm qua từng Sprint, phục vụ cho việc cải tiến quy trình ở các Sprint sau.

Việc sử dụng Jira đã giúp nhóm chủ động trong việc quản lý tiến độ, đảm bảo công việc không bị bỏ sót, đồng thời tăng khả năng phối hợp giữa các thành viên. Qua mỗi Sprint, nhóm đều tổ chức tổng kết và rút kinh nghiệm để cải thiện chất lượng phát triển trong các vòng lặp tiếp theo. Jira không chỉ là công cụ quản lý mà còn là nền tảng để nhóm học hỏi, điều chỉnh và tối ưu hóa quy trình làm việc một cách liên tục.[6]

5.2 Phân công nhiệm vụ của từng thành viên trong nhóm

5.2.1. Khởi tạo dự án, thiết lập môi trường

- Thành viên 1: Tạo project ReactJS, cấu hình Tailwind; Test fetch API đơn giản.

- Thành viên 2: Tạo project Backend, chạy Hello world; Kết nối cơ sở dữ liệu MongoDB.

- Thành viên 3: Tạo GitHub repo, setup branch convention; Viết tài liệu hướng dẫn setup, phân chia repo.

5.2.2. Chức năng cơ bản

- Thành viên 1: Cài đặt React Router, tạo layout Home, Detail, Booking, Login/Register; Trang Home: gọi API lấy danh sách phim, hiển thị gird phim; Trang chi tiết phim: lấy chi tiết phim, lịch chiếu; Trang đăng ký/đăng nhập (gọi API); Responsive giao diện.

- Thành viên 2: Thiết kế cơ sở dữ liệu (phim, lịch chiếu, ghế, vé, người dùng); API đăng ký, đăng nhập (JWT); API lấy danh sách phim; API lấy chi tiết phim, lịch chiếu; API kiểm tra ghế đã đặt.

- Thành viên 3: Review pull request, giữ code clean; Viết tài liệu API trên PostMan/Swagger; Hỗ trợ thành viên 1 test giao diện, kết nối API; Hỗ trợ thành viên 2 test API trên Postman.

5.2.3. Hoàn thiện chức năng và UI

- Thành viên 1: Trang đặt vé: chọn lịch chiếu, chọn ghế, gửi API đặt vé; Trang thanh toán: xử lý thanh toán, hiển thị thông tin vé đã đặt; Thêm thông báo toast khi thành công hoặc thất bại; Hoàn thiện giao diện đẹp (gắn ảnh, màu sắc thống nhất).

- Thành viên 2: API đặt vé; API thanh toán vé; API lấy lịch sử vé của user; Viết unit test cho các API quan trọng.

- Thành viên 3: Kiểm tra UI/UX, đề xuất cải tiến; Viết Unit test cho frontend (React Testing Library); Viết tài liệu hướng dẫn sử dụng (README); Chuẩn bị slide báo cáo tiến độ nhóm.

5.2.4. Testing, Fix bug, Triển khai demo

- Thành viên 1: Sửa bug UI; Kiểm tra responsive mobile; Tối ưu performance frontend.

- Thành viên 2: Sửa bug API; Kiểm tra bảo mật JWT, CORS; Tối ưu query cơ sở dữ liệu.

- Thành viên 3: Viết test end-to-end (Cypress/Playwright); Deploy backend (Render); Deploy frontend (Render); Gửi báo cáo demo, chuẩn bị bài thuyết trình.

Chương 6. KIỂM THỬ

6.1 Chiến lược kiểm thử và công cụ sử dụng

6.1.1. Chiến lược kiểm thử

Trong dự án hệ thống đặt vé xem phim, việc kiểm thử API đóng vai trò quan trọng nhằm đảm bảo các điểm cuối (endpoints) hoạt động ổn định, phản hồi chính xác và bảo mật dữ liệu. Nhóm đã tiến hành kiểm thử API backend viết bằng Express.js sử dụng công cụ Postman, kết hợp cả kiểm thử thủ công lẫn tự động để đánh giá chất lượng dịch vụ backend.

Việc kiểm thử API giúp đảm bảo rằng:

- Các yêu cầu HTTP (GET, POST, PUT, DELETE) được xử lý đúng logic và trả về phản hồi chính xác.
- Dữ liệu trả về đúng định dạng JSON, đúng mã trạng thái HTTP (200, 201, 400, 401, 403, 500…).
- Các luồng xử lý nghiệp vụ như đăng ký, đăng nhập, lấy danh sách phim, đặt vé, xác thực JWT... đều hoạt động ổn định.
- API được bảo vệ khỏi truy cập trái phép bằng token hợp lệ.
- Các lỗi (nếu có) được phản hồi rõ ràng giúp frontend hiển thị chính xác.

6.1.2. Công cụ sử dụng

Postman là công cụ phổ biến và mạnh mẽ dùng để kiểm thử API RESTful trong quá trình phát triển phần mềm. Trong dự án hệ thống đặt vé xem phim, nhóm đã sử dụng Postman để thực hiện kiểm thử thủ công các endpoint của backend viết bằng Node.js và Express, nhằm đảm bảo các chức năng hoạt động đúng như mong đợi.

Mục tiêu sử dụng Postman

Gửi yêu cầu HTTP đến các endpoint để kiểm tra logic xử lý.

Kiểm tra phản hồi trả về từ server (status code, body, headers).

Thử nghiệm các luồng nghiệp vụ: đăng ký, đăng nhập, đặt vé, lấy danh sách phim,...

Gửi token xác thực trong headers để kiểm tra phân quyền và bảo mật.

Tạo collection gồm nhiều request để kiểm thử toàn bộ hệ thống theo kịch bản.

Các bước sử dụng Postman

- Tạo Request mới:

Chọn phương thức HTTP như GET, POST, PUT, DELETE.

Nhập URL ví dụ: http://localhost:5000/api/auth/login.

Nếu có body (như POST), chọn tab Body, chọn raw, định dạng JSON và nhập dữ liệu.

- Gửi Request và xem phản hồi:

Nhấn nút Send để gửi yêu cầu.

Kiểm tra Status Code (như 200 OK, 400 Bad Request, 401 Unauthorized,...).

Kiểm tra Response Body và thông tin như token, thông báo lỗi...

- Thêm Header xác thực (Authorization):

Vào tab Headers, thêm key: Authorization, value: Bearer <token>.

Điều này giúp kiểm tra các route cần xác thực người dùng (như đặt vé, xem thông tin tài khoản...).

- Tạo Collection:

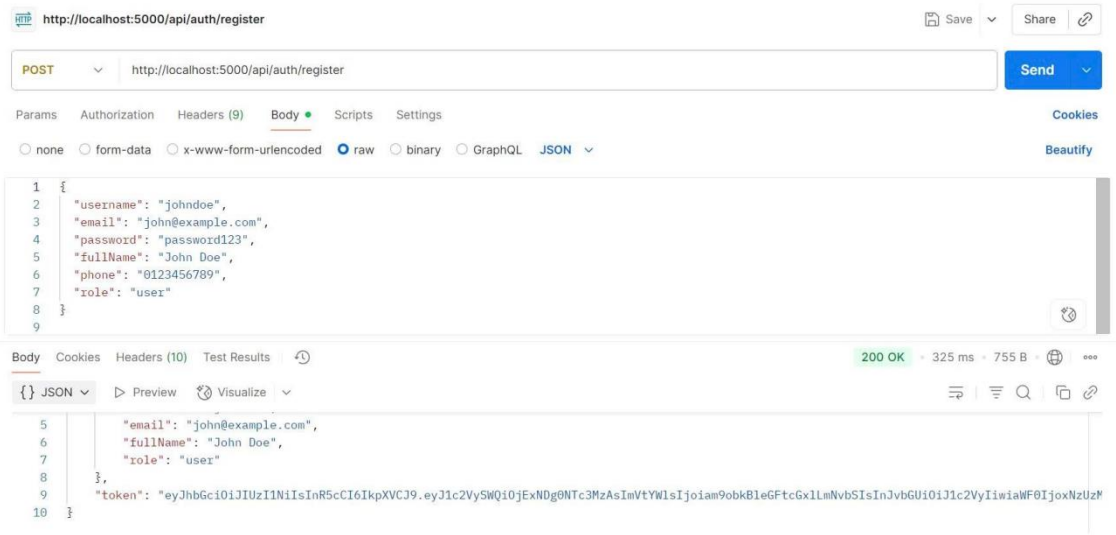
Nhóm các request thành một bộ collection để dễ dàng tái sử dụng và chia sẻ.

Có thể thêm mô tả cho từng request để giải thích mục đích kiểm thử.

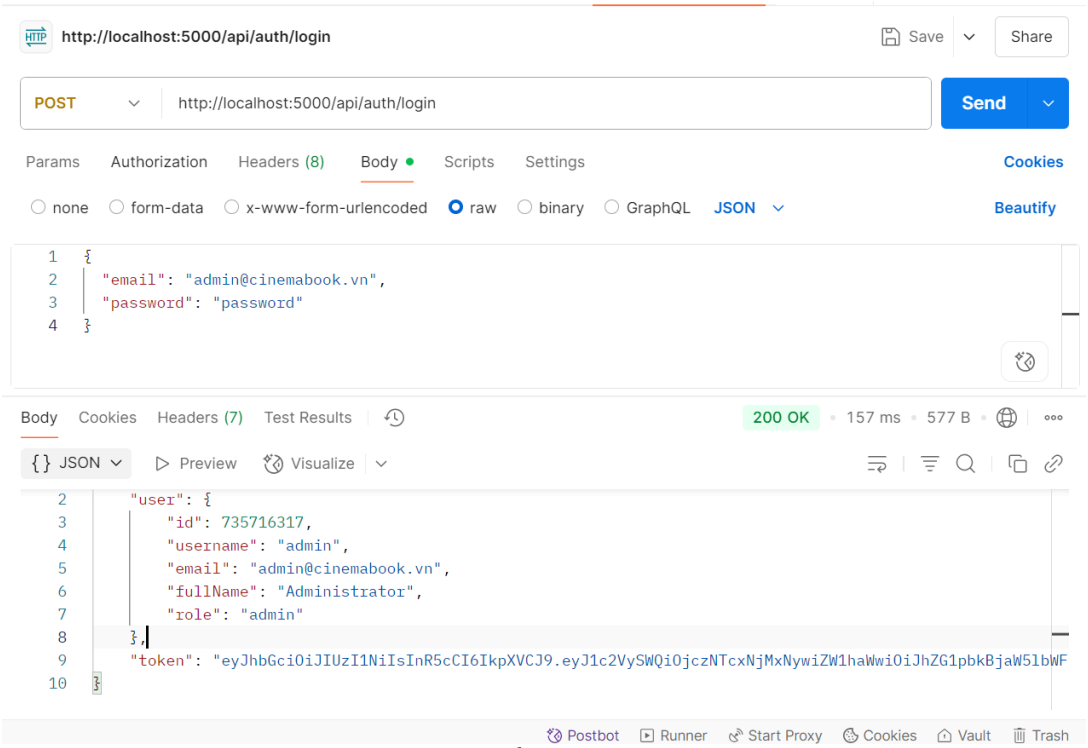
- Kiểm tra tự động bằng Tests (nếu cần):

Postman hỗ trợ viết đoạn script nhỏ (JavaScript) trong tab Tests để tự động kiểm tra điều kiện sau phản hồi.

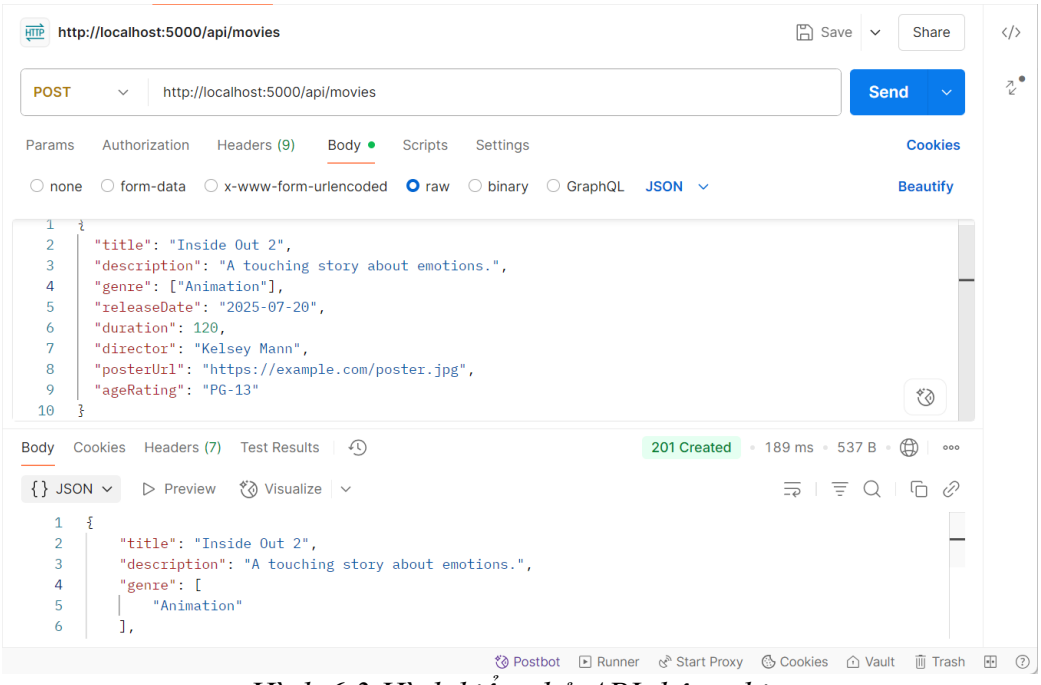
6.2 Kết quả kiểm thử API



Hình 6.1 Kết quả kiểm thử API đăng ký



Hình 6.2 Hình kiểm thử API đăng nhập



Hình 6.3 Hình kiểm thử API thêm phim

Chương 7. ĐÁNH GIÁ VÀ KẾT LUẬN

7.1 Những khó khăn gặp phải trong quá trình thực hiện

Trong quá trình xây dựng hệ thống đặt vé xem phim trực tuyến, nhóm đã gặp phải một số khó khăn chủ yếu sau:

Tiếp cận công nghệ mới: Một số thành viên chưa từng làm việc với React, Node.js, MongoDB hay công cụ CI/CD như GitHub Actions và Docker. Việc tự học và áp dụng công nghệ mới đòi hỏi thời gian tìm hiểu, thực hành và khắc phục lỗi phát sinh.

Thiết kế cơ sở dữ liệu phù hợp thực tế: Việc xây dựng một mô hình cơ sở dữ liệu phản ánh đúng nghiệp vụ thực tế như quản lý lịch chiếu, đặt ghế theo sơ đồ, phân quyền người dùng, tích hợp giá vé động,... là một thách thức đáng kể. Nhóm phải chỉnh sửa nhiều lần để đảm bảo tính logic và khả năng mở rộng của hệ thống.

Triển khai CI/CD và Docker: Dù CI/CD và Docker là những công cụ hiện đại và hỗ trợ tốt cho việc triển khai phần mềm, nhưng việc cấu hình ban đầu (workflow, container, cổng kết nối, build script...) đòi hỏi sự kiên nhẫn, thử nghiệm nhiều lần và kỹ năng tra cứu tài liệu tốt.

Phân chia và phối hợp công việc nhóm: Làm việc theo mô hình Scrum yêu cầu sự phối hợp nhịp nhàng giữa các thành viên. Ban đầu nhóm gặp khó khăn trong việc lập kế hoạch Sprint, theo dõi tiến độ và đồng bộ hóa phần việc giữa frontend, backend và kiểm thử.

Kiểm thử và xử lý lỗi API: Trong giai đoạn cuối, việc kiểm thử API với Postman đã giúp nhóm phát hiện nhiều lỗi như lỗi xác thực, sai dữ liệu đầu vào, lỗi phản hồi JSON...

7.2 Bài học rút ra và đề xuất cải thiện trong tương lai

Tư duy hệ thống và phát triển toàn diện: Thông qua dự án, các thành viên đã hiểu rõ quy trình phát triển phần mềm hiện đại, từ phân tích yêu cầu, thiết kế, triển khai đến kiểm thử và vận hành. Nhóm cũng rèn luyện được kỹ năng làm việc nhóm, tư duy logic, lập trình và giải quyết vấn đề.

Quản lý dự án phần mềm hiệu quả: Việc ứng dụng Jira giúp nhóm làm quen với cách quản lý dự án theo Sprint, phân công công việc, theo dõi tiến độ và phản hồi kịp thời. Đây là kỹ năng rất cần thiết khi làm việc thực tế tại doanh nghiệp.

Tự học và áp dụng công nghệ mới: Nhóm đã có trải nghiệm thực tế trong việc học và áp dụng các công nghệ hiện đại như React, Node.js, MongoDB, Docker và CI/CD. Điều này tạo nền tảng vững chắc cho các dự án trong tương lai.

Đề xuất cải thiện:

Tích hợp cổng thanh toán thực tế: Hiện tại hệ thống mới mô phỏng việc thanh toán, trong tương lai nên tích hợp với các cổng như VNPay, MoMo, ZaloPay để hoàn thiện trải nghiệm đặt vé.

Tối ưu giao diện người dùng: Một số giao diện hiện tại còn đơn giản, chưa hỗ trợ đầy đủ khả năng responsive trên mọi thiết bị. Nhóm đề xuất cải thiện UI/UX bằng cách sử dụng thư viện Tailwind CSS và thử nghiệm thực tế trên nhiều thiết bị.

Ứng dụng AI vào đề xuất phim: Có thể tích hợp thêm tính năng gợi ý phim dựa trên lịch sử đặt vé và hành vi người dùng để tăng tính cá nhân hóa trải nghiệm.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] *NodeJS là gì? Tổng quan kiến thức về Node.JS từ A-Z*. (2022, April 25). https://vietnix.vn/nodejs-la-gi/

[2] TopDev. (2019, July 11). *Postman là gì? Hướng dẫn API Testing với Postman – API Platform*. TopDev; TopDev - Tech Blog. https://topdev.vn/blog/postman-la-gi/

[3] Grey. (2023, July 4). *CI/CD - GitHub Actions và các kiến thức cơ bản*. Viblo. https://viblo.asia/p/cicd-github-actions-va-cac-kien-thuc-co-ban-EoW4oRMrVml

[4] Huyen, B. T. (2017, January 16). *Tìm Hiểu TypeScript và Kiến Thức Cơ Bản - Viblo*. Viblo. https://viblo.asia/p/tim-hieu-typescript-va-kien-thuc-co-ban-PmeRQpnyGoB

[5] GeeksforGeeks. (2023, November 8). *How to setup ReactJs with Vite ?* GeeksforGeeks. https://www.geeksforgeeks.org/reactjs/how-to-setup-reactjs-with-vite/

[6] Nguyễn Thu Huế. (2024, January 23). *Base Blog*. Base Blog. https://base.vn/blog/jira-la-gi/

[7] TopDev Blog. (2019, May 10). *Docker là gì? Kiến thức cơ bản về Docker*. TopDev; TopDev - Tech Blog. https://topdev.vn/blog/docker-la-gi/

[8] TopDev. (2018, May 25). *MongoDB là gì? Định nghĩa đầy đủ và chi tiết nhất về MongoDB*. TopDev. https://topdev.vn/blog/mongodb-la-gi/

[9] *Tìm hiểu về Tailwind CSS*. (2019, December 15). Viblo.asia. https://viblo.asia/p/tim-hieu-ve-tailwind-css-924lJp6WKPM