HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CUỐI KỲ THỰC TẬP CƠ SỞ

Xây dựng web xem phim trực tuyến MOIVEO

Giảng viên hướng dẫn : **Kim Ngọc Bách**
Lớp : **E22CQCN02-B**
Sinh viên thực hiện : **Lê Quang Thanh**
Mã sinh viên : **B22DCVT509**

Hà Nội – 2025

MỤC LỤC

1. Xác định vấn đề

1.1. Giới thiệu

- Trong bối cảnh công nghệ số phát triển mạnh mẽ và nhu cầu giải trí trực tuyến ngày càng tăng cao, các nền tảng xem phim trực tuyến đã trở thành một phần không thể thiếu trong đời sống giải trí của người dân. Website xem phim là dịch vụ cung cấp nội dung điện ảnh qua internet, cho phép người dùng truy cập và xem các bộ phim, chương trình truyền hình mà không cần phải tải xuống. Mô hình này đã tạo ra cuộc cách mạng trong ngành công nghiệp giải trí, thay đổi thói quen tiêu dùng của khán giả từ việc xem truyền hình truyền thống sang nền tảng số.

- MOIVEO là nền tảng xem phim trực tuyến được thiết kế nhằm mang đến trải nghiệm giải trí tuyệt vời cho người dùng Việt Nam. Với kho phim đa dạng từ những blockbuster Hollywood đến các tác phẩm điện ảnh châu Á, MOIVEO cam kết cung cấp nội dung chất lượng cao với giao diện thân thiện và dễ sử dụng. Trang web được phát triển với mục tiêu đơn giản hóa việc tìm kiếm và thưởng thức phim ảnh. Người dùng có thể dễ dàng tìm kiếm theo thể loại, năm sản xuất hoặc quốc gia, đồng thời được hỗ trợ phụ đề tiếng Việt cho hầu hết các bộ phim. MOIVEO không chỉ tập trung vào số lượng mà còn chú trọng đến chất lượng hình ảnh và âm thanh để mang lại trải nghiệm xem phim tốt nhất.

1.2. Hệ thống đề xuất

a. Kiến trúc hệ thống:

- Backend : Node.js + Express.js
- Frontend : React, TailwindCss
- Cơ sở dữ liệu : MongoDB
- Xác thực : JWT (JSON Web Token)
- Quản lí trạng thái: Redux

b. Các chức năng chính của hệ thống đề xuất

- Quản lý người dùng
  - Đăng ký và đăng nhập tài khoản (AuthPage.js)
  - Quản lý thông tin cá nhân (ProfilePage.js)
  - Khôi phục mật khẩu (ForgotPasswordPage.js)
- Xem và tương tác với phim
  - Trang chủ hiển thị banner và danh sách phim (Home.js, BannerHome.js)
  - Tìm kiếm phim (SearchPage.js)
  - Xem chi tiết phim (DetailsPage.js)

- Xem thông tin diễn viên (PersonPage.js)
- Khám phá phim theo thể loại (ExplorPage.js)
- Phát video phim (VideoPlay.js)

- Tính năng cá nhân hóa
  - Lưu phim yêu thích (FavouritePage.js)
  - Thông báo thời gian thực (Notification)
  - Bình luận về phim (CommentSection.js)
- Chatbot thông minh (ChatboxPage.js)
  - Tích hợp Google Gemini API
  - Tương tác thông minh với người dùng
  - Widget chat có thể truy cập từ mọi trang
- Giao diện và trải nghiệm người dùng
  - Responsive design với mobile navigation (MobileNavigation.js)
  - Hiển thị phim dạng card với scroll ngang (HorizontalScollCard.js, MovieCard.js)
  - Header và Footer nhất quán (Header.js, Footer.js)
- Quản lý trạng thái và dữ liệu
  - Sử dụng Redux để quản lý state (store/movieSlice.js)
  - Context API cho xác thực (context/authContext.js)
  - Custom hooks cho fetch data (useFetch.js, useFetchDetails.js)
- Backend Services
  - REST API endpoints cho:
    - Xác thực người dùng
    - Quản lý bình luận
    - Thông báo realtime
  - Tích hợp MongoDB để lưu trữ dữ liệu
  - Socket.IO cho tính năng realtime

c. Các điểm mạnh của hệ thống

- Kiến trúc hệ thống hoàn chỉnh
  - Tách biệt rõ ràng giữa Frontend và Backend
  - Cấu trúc thư mục được tổ chức tốt (components, pages, services, utils)
  - Sử dụng các design pattern phổ biến như Context API, Redux cho state management

- Routing được cấu hình rõ ràng và linh hoạt

- Tích hợp AI thông minh
  - Sử dụng Google Gemini API cho chatbot
  - Tương tác thông minh với người dùng qua ChatboxWidget
  - Widget chat có thể truy cập từ mọi trang trong ứng dụng
  - Cải thiện trải nghiệm người dùng với AI assistant
- Trải nghiệm người dùng tốt
  - Giao diện responsive cho cả desktop và mobile
  - Navigation thuận tiện với MobileNavigation cho thiết bị di động
  - Hiển thị phim dạng card với scroll ngang dễ sử dụng
  - Tính năng tìm kiếm và lọc phim linh hoạt
- Bảo mật và xác thực
  - Hệ thống đăng nhập/đăng ký đầy đủ
  - JWT authentication cho bảo mật API
  - Chức năng khôi phục mật khẩu
  - Quản lý session và token hiệu quả
- Real-time Features
  - Sử dụng Socket.IO cho tương tác realtime
  - Thông báo realtime cho người dùng
  - Chat realtime với AI chatbot
  - Cập nhật bình luận realtime
- Tính mở rộng và bảo trì
  - Sử dụng các công nghệ hiện đại (React, Node.js, MongoDB)
  - Code được tổ chức theo module, dễ mở rộng
  - Sử dụng các custom hooks để tái sử dụng logic
  - Tách biệt rõ ràng giữa business logic và UI
- Tính năng phong phú cho người dùng
  - Lưu trữ phim yêu thích
  - Theo dõi lịch sử xem
  - Bình luận và đánh giá phim
  - Xem thông tin chi tiết phim và diễn viên
- Performance Optimization

- Sử dụng React Router cho Single Page Application
- Lazy loading cho các components
- Tối ưu hiệu suất với các hooks tùy chỉnh
- Caching dữ liệu phía client

- Data Management
- Tích hợp với TMDb API cho nguồn dữ liệu phim phong phú
- Sử dụng MongoDB cho lưu trữ dữ liệu linh hoạt
- API endpoints được thiết kế RESTful
- Xử lý state management hiệu quả với Redux

1.3. Mục tiêu

a. Đối tượng sử dụng :

- Người dùng phổ thông (Regular Users)

- Đối tượng chính sử dụng hệ thống
- Nhu cầu: Giải trí, xem phim, tìm kiếm thông tin
- Có thể:
    - Xem thông tin và trailer phim
    - Tìm kiếm phim theo nhiều tiêu chí
    - Bình luận và đánh giá phim
    - Lưu phim yêu thích
    - Tương tác với AI Chatbot

- Người dùng đam mê phim (Movie Enthusiasts)

- Đối tượng quan tâm sâu về phim ảnh
- Nhu cầu: Tìm hiểu chi tiết về phim và diễn viên
- Có thể:
    - Xem thông tin chi tiết về diễn viên
    - Theo dõi lịch sử xem
    - Tương tác trong cộng đồng qua bình luận
    - Khám phá phim theo thể loại
    - Trao đổi với AI Chatbot về kiến thức phim

- Người dùng di động (Mobile Users)

- Đối tượng xem phim trên thiết bị di động
- Nhu cầu: Trải nghiệm mượt mà trên mobile

- Được hỗ trợ:
    - Giao diện responsive
    - Navigation tối ưu cho mobile
    - Tính năng xem offline

b. Phạm vi áp dụng :

- Hệ thống chỉ hỗ trợ phát trực tiếp
- Hỗ trợ tiếng Việt và tiếng Anh trong giai đoạn đầu
- Chưa có thêm các đặc quyền riêng dành cho người dùng đăng ký Gói dịch vụ Premium hoặc sử dụng trang web với tần suất cao.

c. Giới hạn kỹ thuật :

- Hệ thống hoạt động trên nền web, chưa có ứng dụng di động riêng
- Chưa hỗ trợ chế độ online

2. Phân tích yêu cầu

2.1. Yêu cầu chức năng

- Đăng ký, đăng nhập và quản lý tài khoản
    - Đăng ký tài khoản.
    - Đăng nhập và cập nhật thông tin cá nhân (ảnh đại diện, tên hiển thị, mật khẩu, v.v.).
    - Khôi phục mật khẩu thông qua mã OTP với nodemailer.
- Xem và Tìm kiếm Phim
    - Xem danh sách phim mới và phổ biến trên trang chủ
    - Tìm kiếm phim theo tên, thể loại, năm phát hành
    - Khám phá phim theo nhiều thể loại khác nhau
    - Xem banner phim nổi bật
    - Cuộn ngang để xem nhiều phim trong cùng danh mục
- Xem Chi Tiết Phim
    - Xem thông tin chi tiết của phim
    - Xem trailer và video liên quan
    - Xem thông tin diễn viên
    - Đọc tóm tắt nội dung phim
    - Xem đánh giá và điểm số của phim
- Tương tác với Phim
    - Bình luận về phim
    - Đánh giá phim
    - Chia sẻ phim với người khác
    - Tương tác với bình luận của người khác
    - Báo cáo nội dung không phù hợp
- Quản lý Tài Khoản
    - Đăng ký tài khoản mới

- Đăng nhập vào hệ thống
- Khôi phục mật khẩu
- Cập nhật thông tin cá nhân
- Quản lý cài đặt tài khoản

- Tính năng Cá nhân hóa
  - Lưu phim vào danh sách yêu thích
  - Xem lịch sử phim đã xem
  - Tạo danh sách phát riêng
  - Nhận đề xuất phim phù hợp
  - Quản lý danh sách phim cá nhân
- Chat với AI Assistant
  - Hỏi đáp thông tin về phim
  - Nhận gợi ý phim dựa trên sở thích
  - Tương tác với chatbot thông minh
  - Nhận hỗ trợ tìm kiếm phim
  - Widget chat có thể truy cập từ mọi trang
- Hệ thống Thông báo
  - Nhận thông báo về phim mới
  - Thông báo khi có người tương tác
  - Cập nhật về bình luận mới
  - Thông báo về hoạt động tài khoản
  - Tùy chỉnh cài đặt thông báo
- Xem thông tin Diễn viên
  - Xem tiểu sử diễn viên
  - Xem danh sách phim của diễn viên
  - Xem hình ảnh và thông tin chi tiết
  - Theo dõi diễn viên yêu thích
  - Nhận thông báo về phim mới của diễn viên

2.2. Yêu cầu phi chức năng

- **Hiệu năng (Performance)**
  - Tải trang nhanh và mượt mà
  - Tối ưu hóa load ảnh và video
  - Phản hồi người dùng nhanh chóng
  - Xử lý đồng thời nhiều request
  - Tối ưu hóa băng thông với lazy loading
- **Bảo mật (Security)**
  - Xác thực người dùng an toàn với JWT
  - Mã hóa dữ liệu nhạy cảm
  - Bảo vệ chống tấn công XSS và CSRF
  - Quản lý phiên làm việc an toàn
  - Kiểm soát quyền truy cập API
- **Khả năng mở rộng (Scalability)**
  - Kiến trúc module dễ mở rộng

- Có thể thêm tính năng mới dễ dàng
- Hỗ trợ tăng số lượng người dùng
- Khả năng xử lý dữ liệu lớn
- Tích hợp thêm API bên thứ ba

- Độ tin cậy (Reliability)

- Hệ thống hoạt động ổn định
- Xử lý lỗi gracefully
- Backup và khôi phục dữ liệu
- Monitoring hệ thống
- Logging đầy đủ

- Trải nghiệm người dùng (Usability)

- Giao diện thân thiện, dễ sử dụng
- Responsive trên mọi thiết bị
- Thời gian phản hồi nhanh
- Thông báo lỗi rõ ràng
- Hướng dẫn sử dụng trực quan

3. Tổng quan về hệ thống

3.1. Các công nghệ và công cụ liên quan

3.2.1 MERN Stack

- MERN Stack là một tập hợp các công nghệ phát triển web phổ biến, bao gồm bốn thành phần chính: MongoDB**,** Express.js**,** React.js **và** Node.js. Tên gọi MERN được tạo thành từ chữ cái đầu của từng công nghệ này. Ưu điểm lớn nhất của MERN Stack là sử dụng JavaScript xuyên suốt cả frontend và backend, giúp developers dễ dàng chuyển đổi giữa các tầng của ứng dụng. Stack này đặc biệt phù hợp cho việc phát triển Single Page Applications (SPA) và các ứng dụng web hiện đại có tính tương tác cao:

➢ **MongoDB (Database Layer)**

- Tính linh hoạt của cấu trúc dữ liệu MongoDB sử dụng định dạng BSON (Binary JSON) cho phép lưu trữ dữ liệu không có cấu trúc cố định. Điều này giúp developers dễ dàng thay đổi schema mà không cần migrate dữ liệu phức tạp như cơ sở dữ liệu quan hệ truyền thống.
- Hiệu suất cao Với khả năng xử lý dữ liệu lớn và tốc độ truy vấn nhanh, MongoDB phù hợp cho các ứng dụng cần xử lý real-time. Hệ thống indexing mạnh mẽ giúp tối ưu hóa các truy vấn phức tạp.
- Khả năng mở rộng tuyệt vời MongoDB hỗ trợ horizontal scaling (sharding) một cách tự nhiên, cho phép phân tán dữ liệu trên nhiều server để xử lý lượng dữ liệu khổng lồ và lưu lượng truy cập cao.
- Dễ học và sử dụng Cú pháp truy vấn của MongoDB gần với JavaScript, giúp developers frontend dễ dàng làm quen. Việc làm việc với JSON objects cũng rất quen thuộc với các lập trình viên web.
- Replica Sets và High Availability MongoDB cung cấp tính năng sao lưu tự động và failover, đảm bảo ứng dụng luôn hoạt động ổn định ngay cả khi có sự cố phần cứng.
- Trong dự án này MongoDB lưu trữ:
    - Thông tin người dùng
    - Bình luận về phim
    - Thông báo
    - Danh sách yêu thích

➢ **Express.js:** Là một web framework minimalist và linh hoạt cho Node.js, được thiết kế để xây dựng các ứng dụng web và API một cách nhanh chóng và hiệu quả. Được mệnh danh là "fast, unopinionated, minimalist web framework", Express cung cấp các tính năng cơ bản cần thiết mà không áp đặt cấu trúc cứng nhắc.

- Đơn giản và dễ học Express có cú pháp rõ ràng, dễ hiểu, giúp developers nhanh chóng tạo ra các route và xử lý HTTP requests. Chỉ với vài dòng code, bạn có thể tạo một server hoạt động.

- Hiệu suất cao Được xây dựng trên Node.js, Express kế thừa khả năng xử lý bất đồng bộ mạnh mẽ, cho phép xử lý hàng nghìn requests đồng thời mà không bị block.
- Middleware system linh hoạt Hệ thống middleware của Express cho phép xử lý requests theo chuỗi, dễ dàng thêm các tính năng như authentication, logging, CORS mà không làm phức tạp code chính.
- Hệ sinh thái phong phú Với hàng nghìn packages trên NPM, Express có thể tích hợp dễ dàng với các công cụ khác như database drivers, template engines, và authentication libraries.
- RESTful API friendly Express được thiết kế đặc biệt tốt cho việc xây dựng RESTful APIs, hỗ trợ đầy đủ các HTTP methods và route parameters.
- Các API Endpoints trong dự án:
    - User routes (/api/users/*)
    - Comment routes (/api/comments/*)
    - Notification routes (/api/notifications/*)
- Middleware trong dự án :
    - Authentication
    - Error handling
    - CORS
    - Body parsing
    - Route handling

➢ **React:** Là Frontend framework trong dự án , React sử dụng kiến trúc component, cho phép chia nhỏ giao diện thành các thành phần độc lập và có thể tái sử dụng. Điều này giúp code dễ bảo trì, mở rộng và kiểm thử.

- Virtual DOM React sử dụng Virtual DOM để tối ưu hóa hiệu suất. Thay vì cập nhật trực tiếp DOM, React so sánh sự thay đổi và chỉ cập nhật những phần cần thiết, giúp ứng dụng chạy nhanh hơn.
- JSX - JavaScript XML JSX cho phép viết HTML trong JavaScript một cách tự nhiên, giúp code dễ đọc và dễ hiểu hơn. Developers có thể kết hợp logic JavaScript với markup một cách liền mạch.
- One-way Data Flow Dữ liệu trong React chảy theo một hướng từ parent component xuống child component, giúp debug dễ dàng và ứng dụng có tính dự đoán cao.
- Hooks System React Hooks cho phép sử dụng state và lifecycle methods trong functional components, giúp code ngắn gọn hơn và dễ tái sử dụng logic.
- Hệ sinh thái mạnh mẽ Với cộng đồng lớn và nhiều thư viện hỗ trợ như React Router, Redux, Material-UI, React có thể xây dựng từ ứng dụng đơn giản đến phức tạp.
- Components chính:
    - Pages:
        - Home.js: Trang chủ

        - AuthPage.js: Đăng nhập/Đăng ký
        - ChatboxPage.js: Trang chat AI
        - DetailsPage.js: Chi tiết phim
        - ProfilePage.js: Trang cá nhân
    - Components:
        - Header.js: Navigation
        - MovieCard.js: Hiển thị phim
        - ChatboxWidget.js: Widget chat
        - CommentSection.js: Phần bình luận
    - Context & Store:
        - authContext.js: Quản lý authentication
        - movieSlice.js: Redux slice cho phim
    - Custom Hooks:
        - useFetch.js: Hook fetch data
        - useFetchDetails.js: Hook fetch chi tiết

- **Node.js:** Node.js là một runtime environment mã nguồn mở cho phép chạy JavaScript ở phía server. Được xây dựng trên V8 JavaScript engine của Google Chrome, Node.js ra đời năm 2009 bởi Ryan Dahl và đã cách mạng hóa cách phát triển ứng dụng web bằng cách cho phép sử dụng JavaScript cho cả frontend và backend.

- Event-driven và Non-blocking I/O Node.js sử dụng mô hình bất đồng bộ (asynchronous) và event-driven, cho phép xử lý hàng nghìn kết nối đồng thời mà không bị block. Điều này làm cho Node.js cực kỳ hiệu quả với các ứng dụng I/O intensive.
- JavaScript toàn stack Developers có thể sử dụng cùng một ngôn ngữ (JavaScript) cho cả frontend và backend, giảm thiểu việc chuyển đổi ngữ cảnh và tăng hiệu quả phát triển.
- NPM - Package Manager mạnh mẽ Node Package Manager (NPM) là kho thư viện lớn nhất thế giới với hàng triệu packages, giúp developers dễ dàng tìm và sử dụng các module có sẵn.
- Hiệu suất cao Được xây dựng trên V8 engine, Node.js có tốc độ thực thi JavaScript rất nhanh, đặc biệt phù hợp cho các ứng dụng real-time như chat, gaming, hoặc live streaming.
- Features:
    - RESTful API
    - Socket.IO server
    - JWT Authentication
    - File handling
    - API Integration

3.2.2 Các công cụ liên quan

- Development Tools:

a) Project Setup Tools

- Create React App (thể hiện qua cấu trúc dự án)
- npm (có package.json ở cả frontend và backend)
- Environment variables (file .env)

b) Code Editor & Extensions
- ESLint (mặc định từ CRA)
- Browser DevTools cho React
- VS Code (khuyến nghị)

c) Build Tools
- Webpack (built-in từ CRA)
- Babel (built-in từ CRA)
- PostCSS (cho Tailwind - tailwind.config.js)

- Development Environment:

a) Frontend Development Server
- Local development server (npm start - port 3000)
- Hot reload
- Error reporting
- Source maps cho debugging

b) Backend Development
- Nodemon (giả định cho development)
- Express server
- API testing environment

3.2. Cấu trúc dự án

3.2.1. Frontend Architecture

```
movieapp-main/
├── public/                 # Static files
│   ├── favicon.ico
│   ├── index.html          # HTML template
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
│
├── src/                    # Source code
│   ├── assets/             # Media files
│   │   ├── login.png
│   │   ├── logo.png
│   │   └── user.png
│   │
│   ├── components/         # Reusable components
│   │   ├── BannerHome.js       # Banner component cho trang
chủ
│   │   ├── Card.js             # Component card chung
│   │   ├── CommentSection.js   # Phần comments
```

```
    ├── Divider.js          # Component phân chia
    ├── Footer.js           # Footer component
    ├── Header.js           # Header/Navigation bar
    ├── HorizontalScollCard.js  # Card cuộn ngang
    ├── MobileNavigation.js     # Navigation cho mobile
    ├── MovieCard.js            # Card hiển thị phim
    └── VideoPlay.js            # Component phát video

├── constants/          # Các hằng số
│   └── navigation.js       # Các routes và navigation configs

├── context/            # React Context
│   └── authContext.js      # Context quản lý authentication

├── hooks/              # Custom React hooks
│   ├── useFetch.js         # Hook để fetch data
│   └── useFetchDetails.js # Hook fetch chi tiết

├── pages/              # Các trang chính
│   ├── authContext.js          # Auth context page
│   ├── AuthPage.js             # Trang đăng nhập/đăng ký
│   ├── ChatboxPage.js          # Trang chat
│   ├── DetailsPage.js          # Trang chi tiết phim
│   ├── ExplorePage.js          # Trang khám phá
│   ├── FavouritePage.js        # Trang yêu thích
│   ├── ForgotPasswordPage.js   # Trang quên mật khẩu
│   ├── Home.js                 # Trang chủ
│   ├── PersonPage.js           # Trang thông tin người dùng
│   ├── ProfilePage.js          # Trang profile
│   └── SearchPage.js           # Trang tìm kiếm

├── routes/             # Route configuration
│   └── index.js            # Định nghĩa các routes

├── store/              # Redux store
│   ├── movieoSlice.js      # Movie reducer/actions
│   └── store.js            # Store configuration

├── App.js              # Root component
├── App.css             # Styles cho App component
├── index.js            # Entry point
├── index.css           # Global styles
├── App.test.js         # Tests cho App component
├── setupTests.js       # Setup cho testing
└── reportWebVitals.js # Performance monitoring
```

```
├── build/                  # Production build
│   ├── static/
│   │   ├── css/
│   │   │   ├── main.c766444b.css
│   │   │   └── main.c766444b.css.map
│   │   └── js/
│   │       ├── 453.ca28609e.chunk.js
│   │       ├── 453.ca28609e.chunk.js.map
│   │       ├── main.468e8659.js
│   │       ├── main.468e8659.js.LICENSE.txt
│   │       └── main.468e8659.js.map
│   ├── asset-manifest.json
│   ├── index.html
│   ├── manifest.json
│   └── robots.txt
│
├── package.json            # Dependencies và scripts
├── README.md               # Project documentation
└── tailwind.config.js      # Tailwind CSS configuration
```

3.2.2. Backend Architecture

```
backend/
├── server.js           # Entry point của backend
└── package.json        # Dependencies và scripts của backend
```

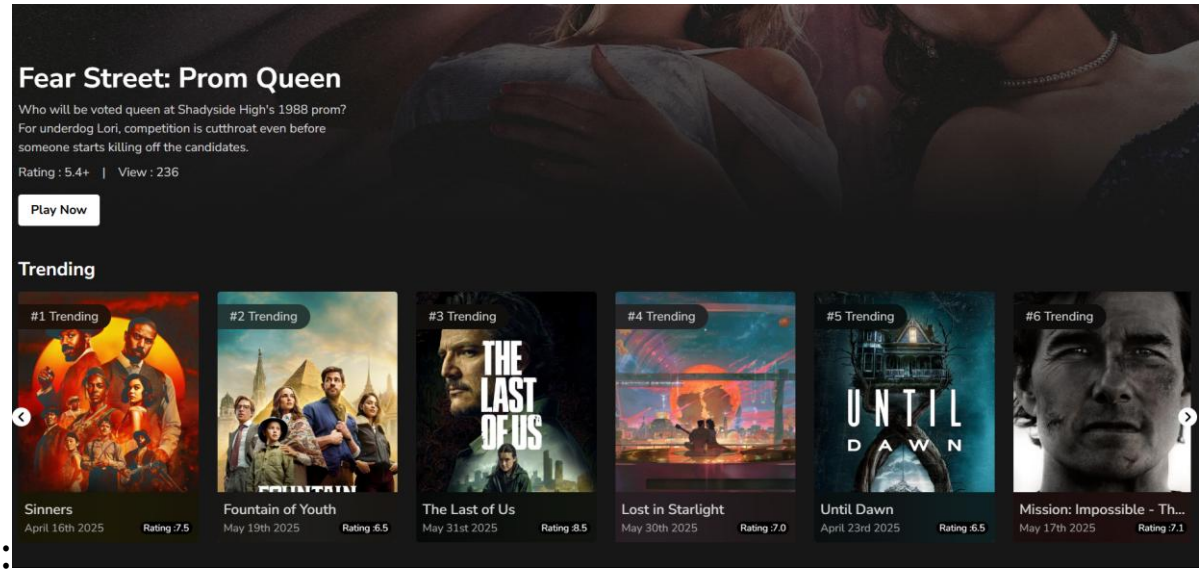
4. Thiết kế hệ thống

4.1. Thiết kế Giao diện người dùng (UX/UI)

4.1.1. Trang Chủ (Home.js)

- **Banner phim nổi bật**

- **Danh sách phim mới**

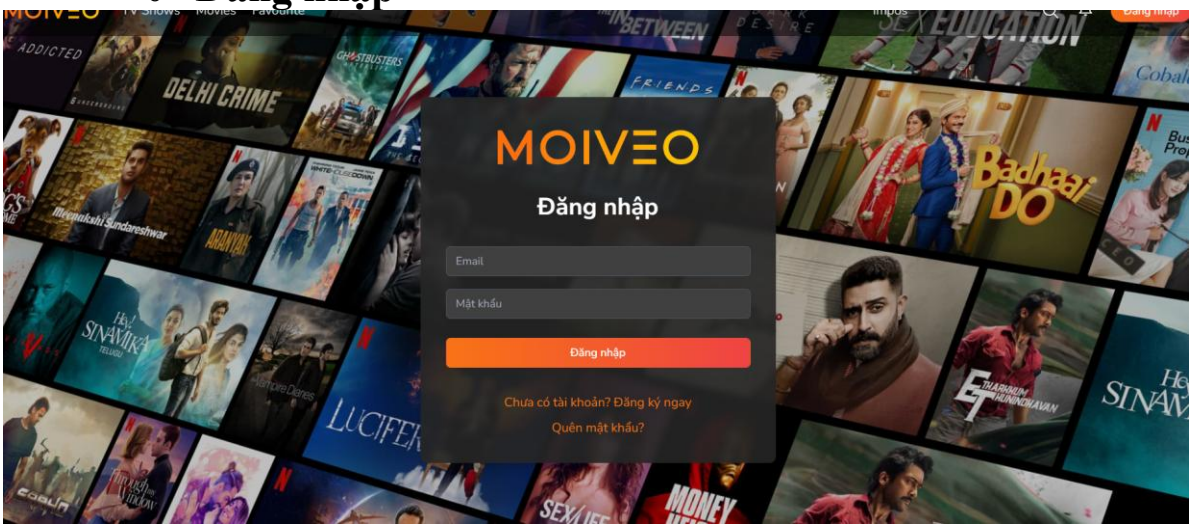


- ***Header***

- ***Footer***

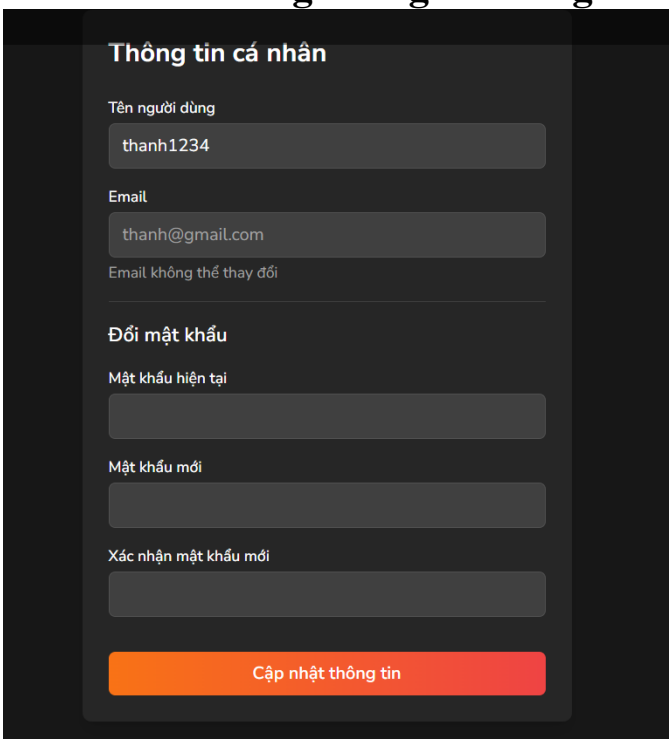
4.1.2.Quản lý Tài khoản

- **AuthPage.js - Trang xác thực:**
  - **Đăng nhập**

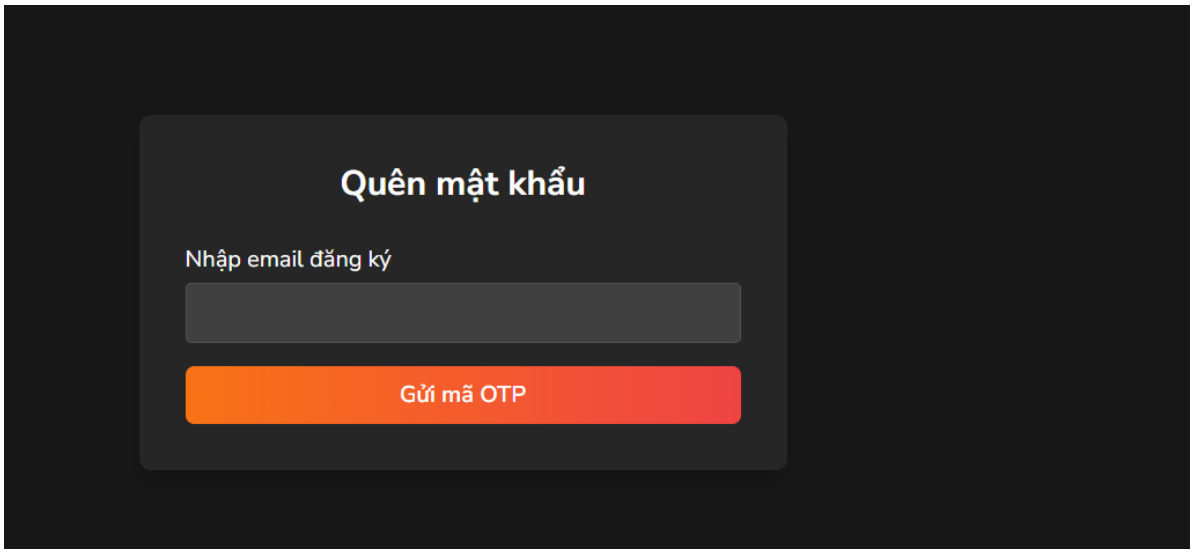


  - **Đăng ký**

- **ProfilePage.js - Trang cá nhân:**
  - **Thông tin người dùng**

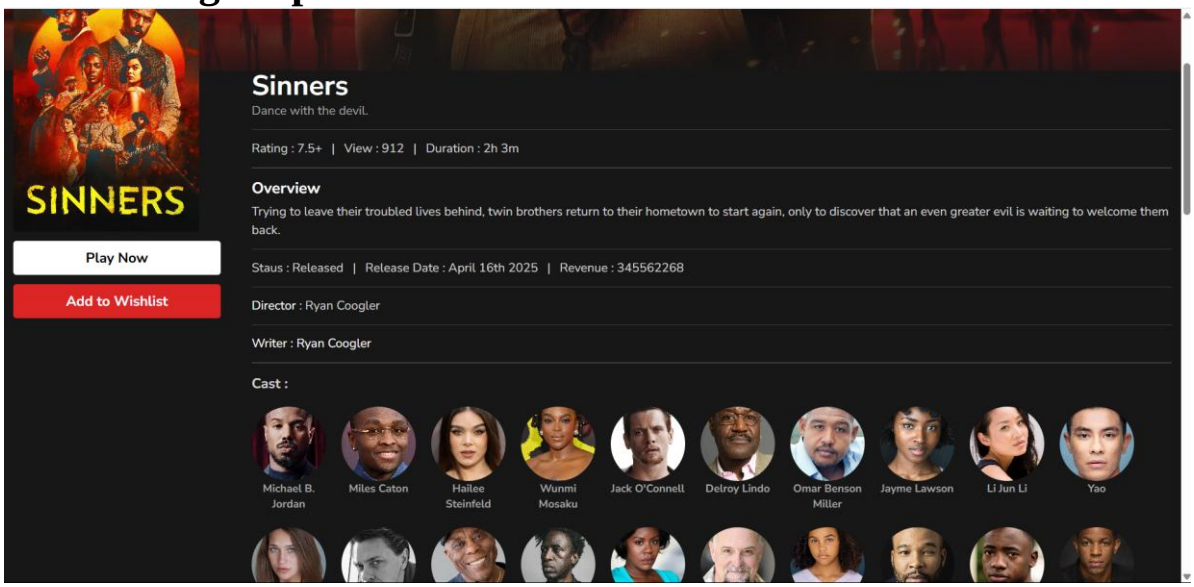


- **ForgotPasswordPage.js - Khôi phục mật khẩu**

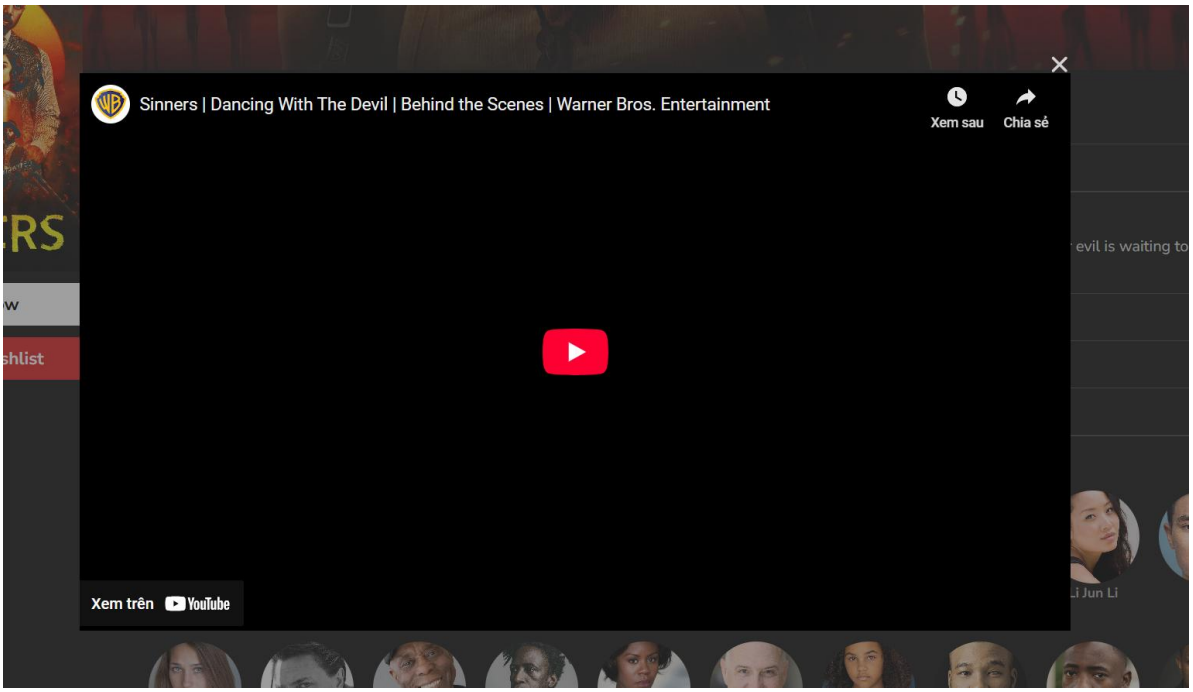


4.1.3. Tương tác với Phim

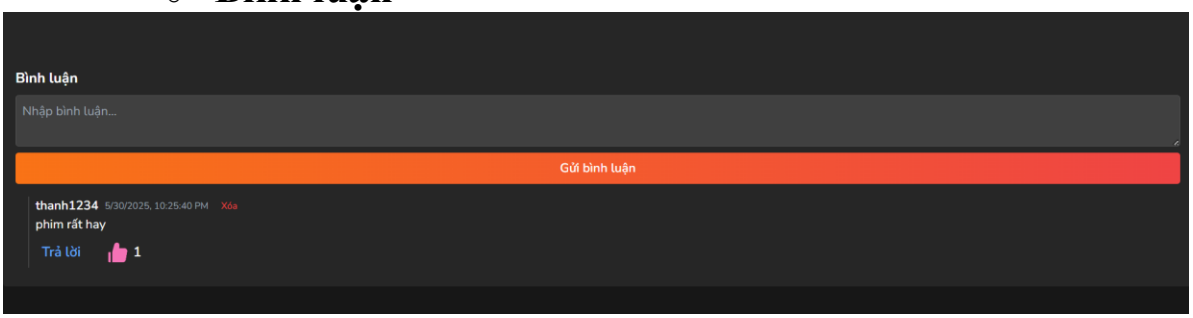
- ***DetailsPage.js - Chi tiết phim:***
  - **Thông tin phim**



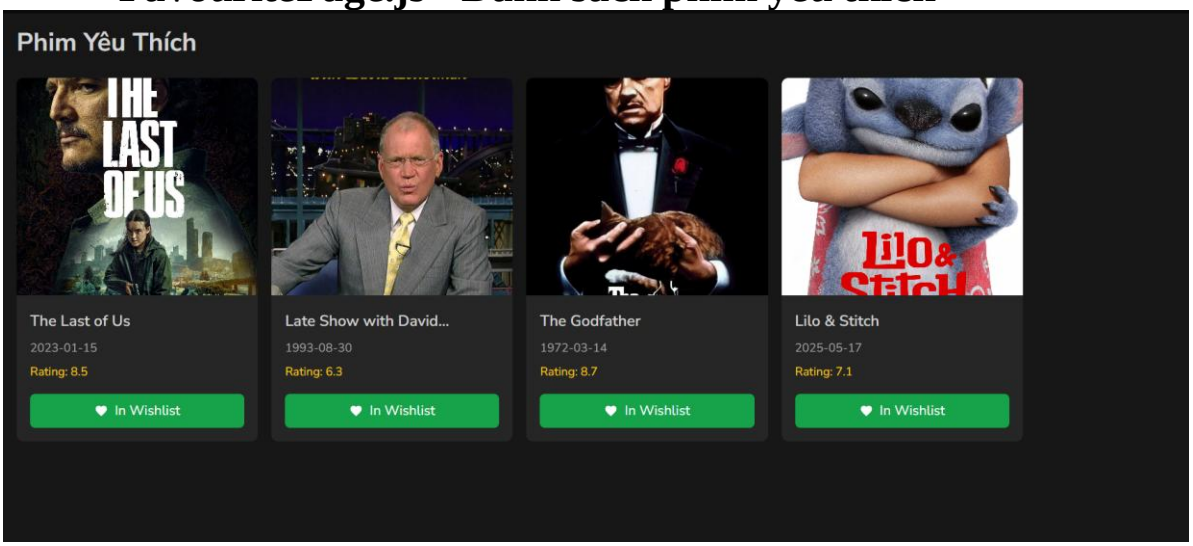
  - **VideoPlay**



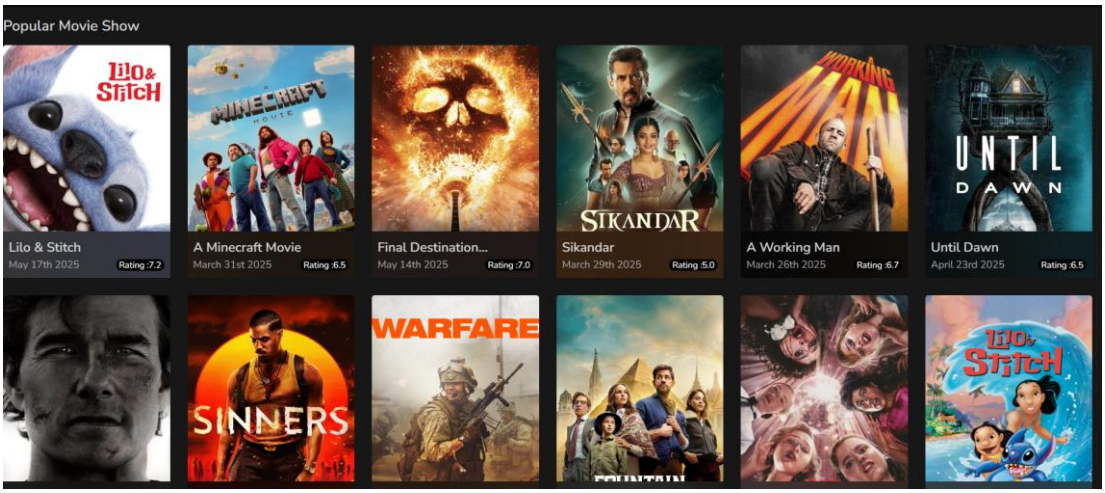
- **Bình luận**



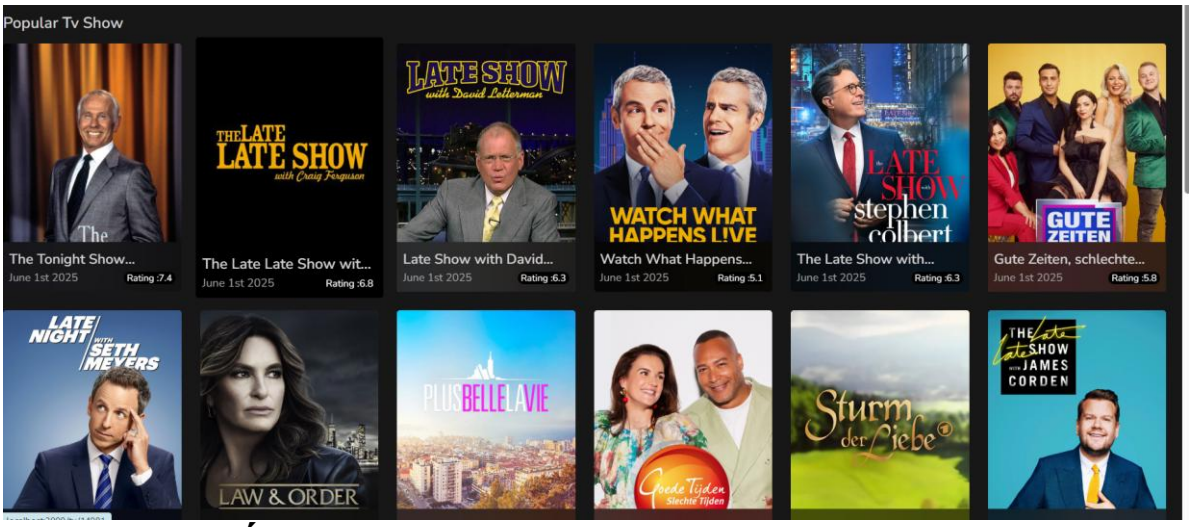
- **FavouritePage.js - Danh sách phim yêu thích**



- **Movie**

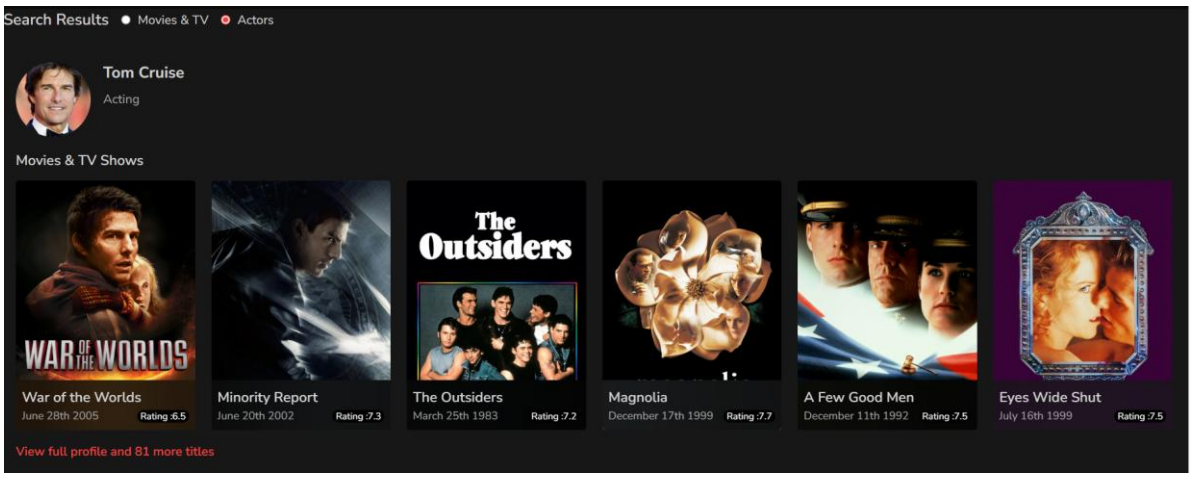


- **TV Shows**

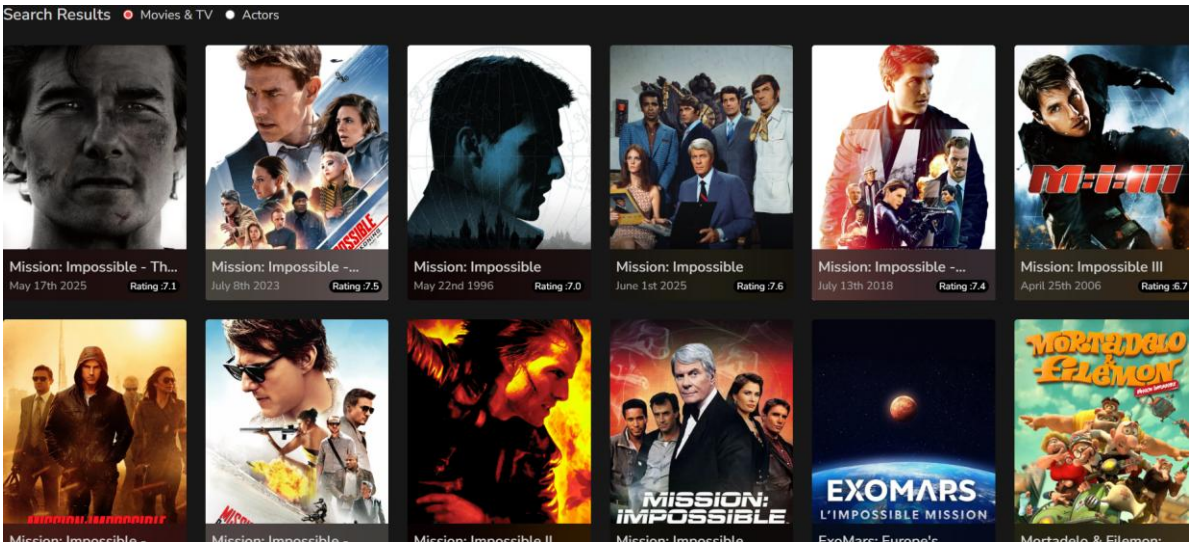


4.1.4.Tìm kiếm và Thông tin

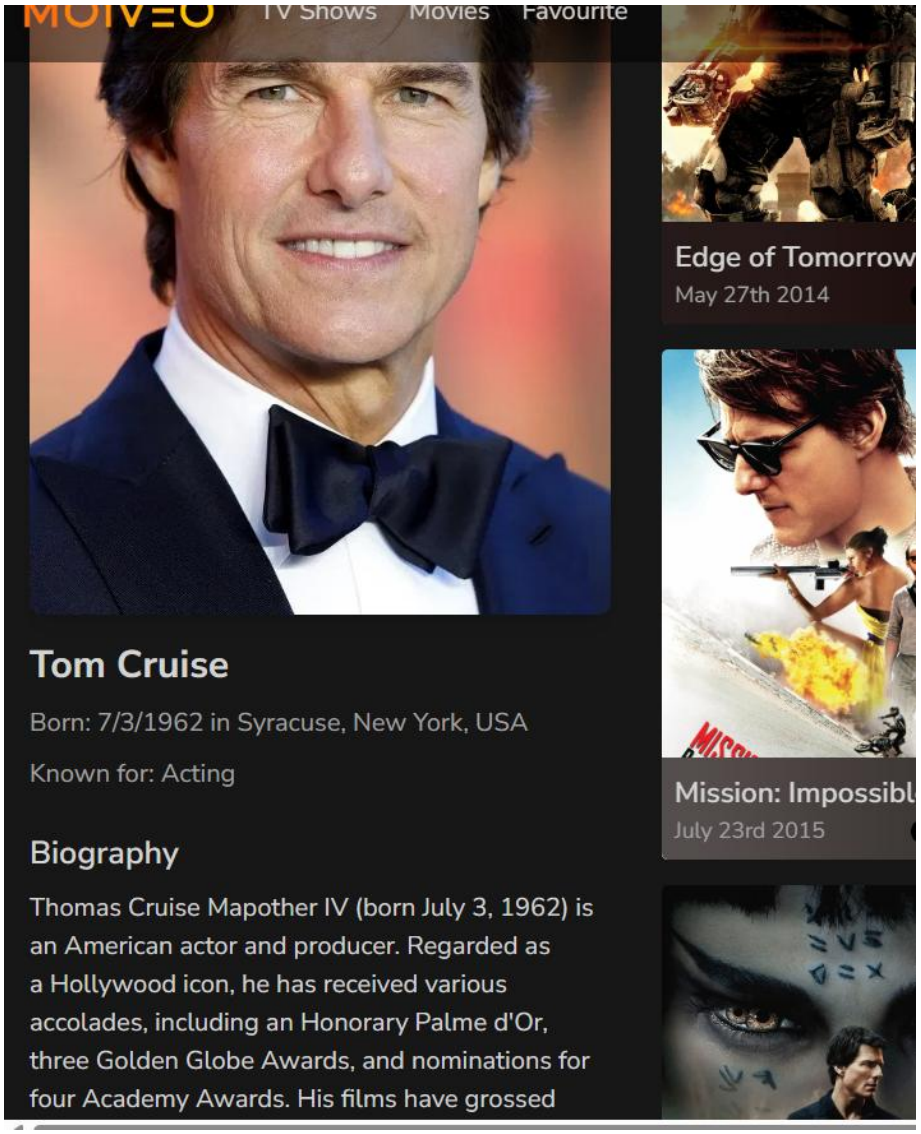
- **SearchPage.js - Trang tìm kiếm:**
  - **Tìm theo tên diễn viên**



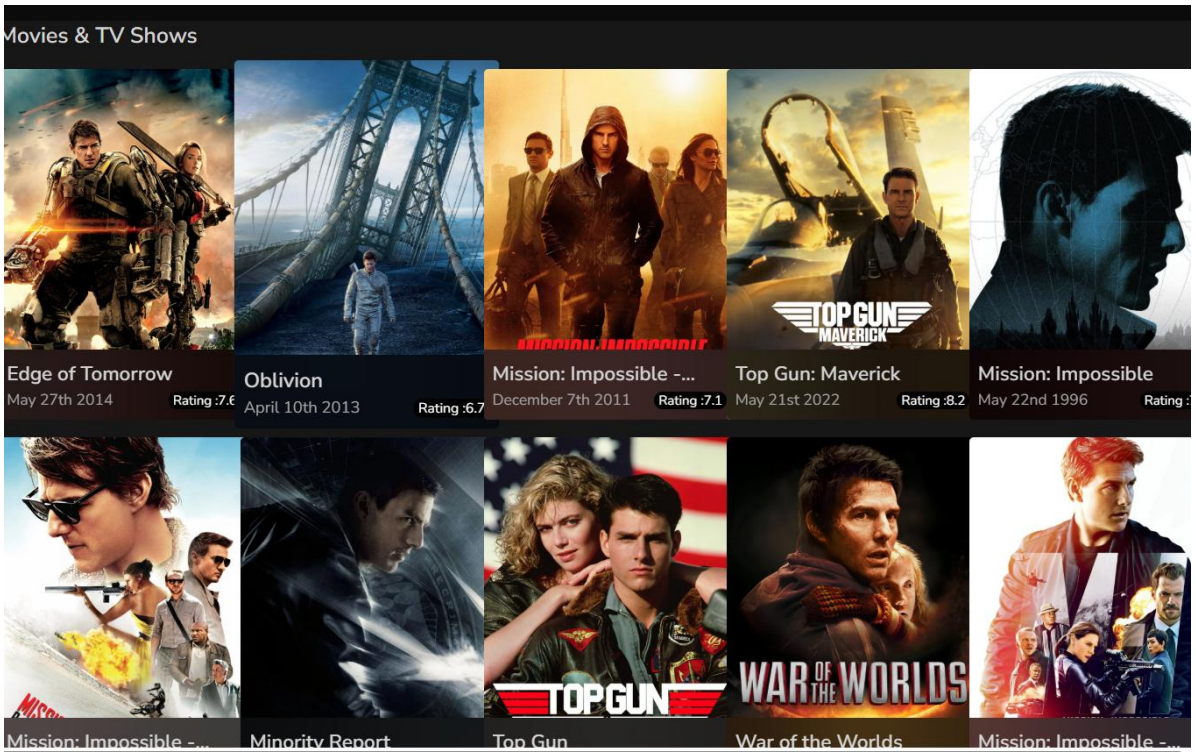
  - **Lọc theo thể loại phim**



- **PersonPage.js - Trang thông tin diễn viên:**
  - **Tiểu sử**

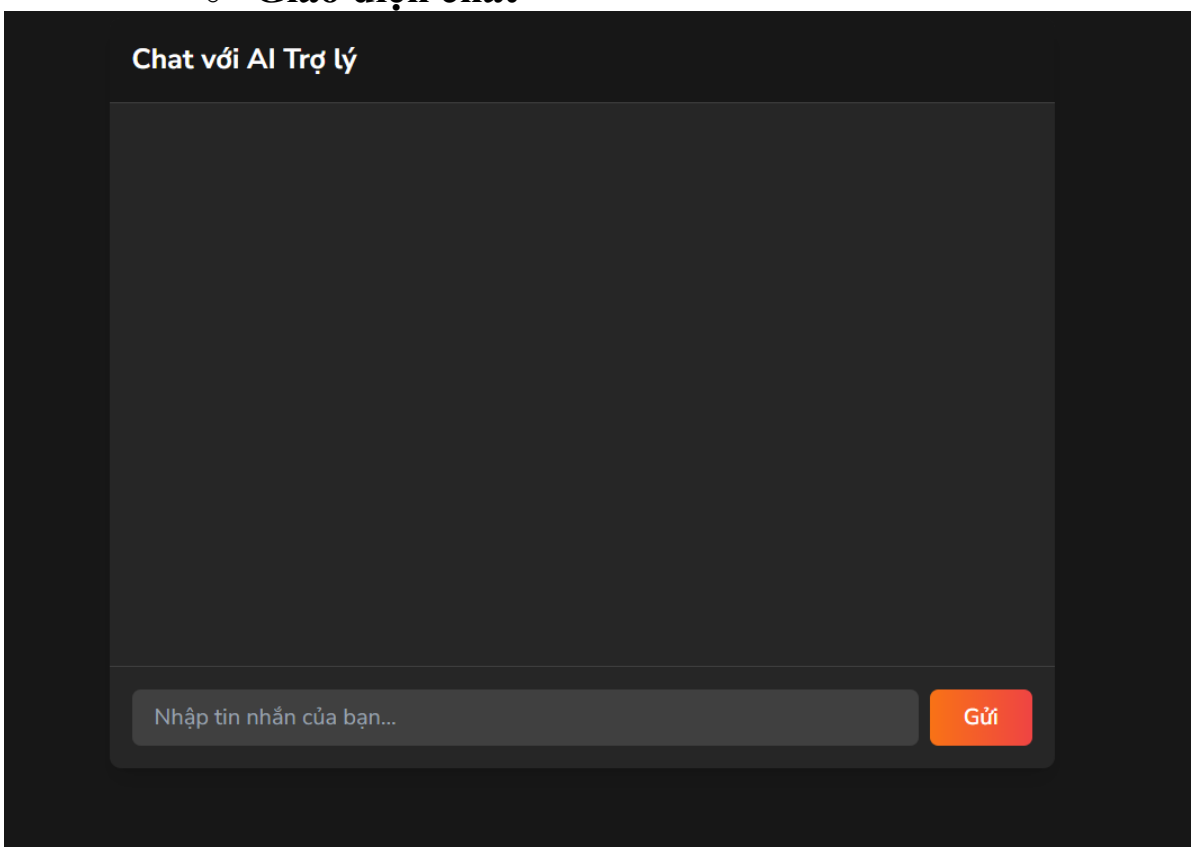


  - **Phim đã tham gia**

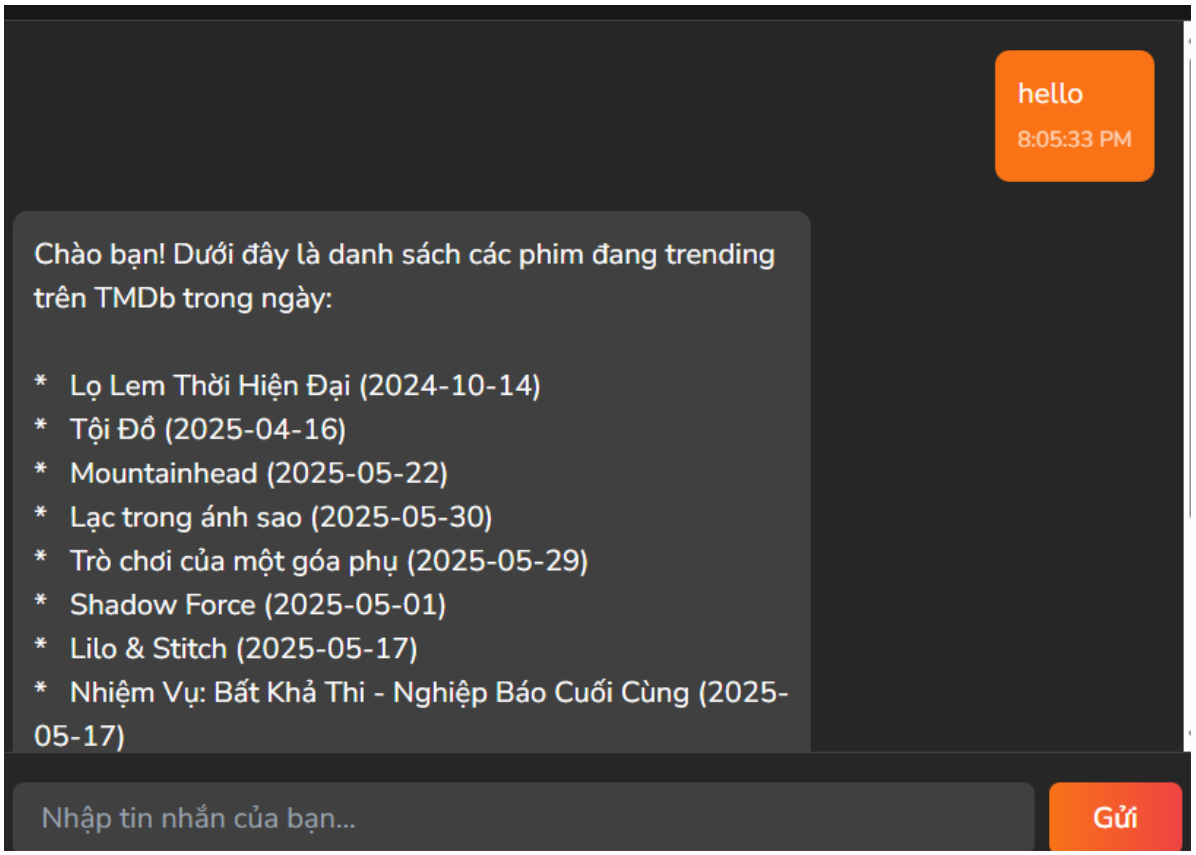


4.1.5.Tính năng AI và Chat

- **ChatboxPage.js - Trang chat với AI:**
  - **Giao diện chat**



  - **Tương tác với Gemini AI**



4.2. Thiết kế cơ sở dữ liệu

4.2.1.User Model

```
// User Model
const UserSchema = new mongoose.Schema({
  wishlist: [
    {
      movieId: String,
      title: String,
      poster_path: String,
      vote_average: Number,
      release_date: String,
      media_type: String
    }
  ],
  username: {
    type: String,
    required: true,
    unique: true
  },
  email: {
    type: String,
    required: true,
    unique: true
  },
  password: {
    type: String,
    required: true
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
});

const User = mongoose.model('User', UserSchema);
```

4.2.2.OTP Model

```
// OTP Model
const OtpSchema = new mongoose.Schema({
  email: { type: String, required: true },
  otp: { type: String, required: true },
  expiresAt: { type: Date, required: true }
});
const Otp = mongoose.model('Otp', OtpSchema);

// model cmt
const CommentSchema = new mongoose.Schema({
  movieId: { type: String, required: true },
  user: {
    id: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
    username: String
  },
  content: { type: String, required: true },
  parentId: { type: mongoose.Schema.Types.ObjectId, ref: 'Comment', default: null },
  createdAt: { type: Date, default: Date.now },
  likes: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }]
});
const Comment = mongoose.model('Comment', CommentSchema);
```

4.2.3.Comment Model

```
// model cmt
const CommentSchema = new mongoose.Schema({
  movieId: { type: String, required: true },
  user: {
    id: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
    username: String
  },
  content: { type: String, required: true },
  parentId: { type: mongoose.Schema.Types.ObjectId, ref: 'Comment', default: null },
  createdAt: { type: Date, default: Date.now },
  likes: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }]
});
const Comment = mongoose.model('Comment', CommentSchema);
```

4.2.4.Notification Model

```
// model Notification
const NotificationSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  message: { type: String, required: true },
  createdAt: { type: Date, default: Date.now },
  read: { type: Boolean, default: false }
});
const Notification = mongoose.model('Notification', NotificationSchema);
```

4.3. Thiết kế Backed

4.3.1. API Design

a) Authentication APIs:

/api/register - Đăng ký tài khoản mới
/api/login - Đăng nhập
/api/auth/send-otp - Gửi mã OTP
/api/auth/verify-otp - Xác thực OTP
/api/auth/reset-password - Đặt lại mật khẩu

b) User APIs:

/api/user - Lấy thông tin user
/api/user/profile - Cập nhật thông tin user
/api/wishlist - CRUD danh sách phim yêu thích

c) Comment APIs:

/api/comments/:movieId - Lấy comments của phim
/api/comments - Thêm comment mới
/api/comments/:commentId - Xóa comment
/api/comments/:id/like - Like/unlike comment

d) Notification APIs:

/api/notifications - Lấy/thêm thông báo
/api/notifications/:id - Xóa thông báo

4.3.2. Các chức năng nổi bật

- **Xây dựng xác thực OTP với nodemailer:**

a. Quy trình hoạt động tổng thể:

- User nhập email -> Kiểm tra email tồn tại
- Gửi OTP -> Tạo OTP -> Lưu DB -> Gửi email
- User nhập OTP -> Kiểm tra OTP hợp lệ
- User đặt mật khẩu mới -> Cập nhật mật khẩu -> Xóa OTP

b. Cấu hình nodemailer

- Tải thư viện nodemailer

```
"dependencies": {
  "bcryptjs": "^2.4.3",
  "cors": "^2.8.5",
  "dotenv": "^16.3.1",
  "express": "^4.18.2",
  "jsonwebtoken": "^9.0.2",
  "mongoose": "^8.0.3",
  "nodemailer": "^7.0.3"
},
```

- Cấu hình nodemailer:

```
const transporter = nodemailer.createTrans
  service: 'gmail',
  auth: {
    user: process.env.EMAIL_USER,  // Emai
    pass: process.env.EMAIL_PASS   // Mật
  }
});
```

c. Các luồng hoạt động :

- API gửi OTP

```
app.post('/api/auth/send-otp', async (req,
  try {
    const { email } = req.body;
```

- Tạo mã OTP ngẫu nhiên 6 số và thời gian hết hạn

```
const otp = Math.floor(100000 + Math.random() * 900000).toString();

// 2. Tạo thời gian hết hạn (5 phút)
const expiresAt = new Date(Date.now() + 5 * 60 * 1000);
```

- Xóa OTP cũ nếu có

```
// 3. Xóa OTP cũ nếu có
await Otp.deleteMany({ email });
```

- Gửi email chứa OTP

```
  await transporter.sendMail({
    from: process.env.EMAIL_USER,  // Email gửi đi
    to: email,                     // Email người nhận
    subject: 'Mã OTP xác thực Movie App',
    text: `Mã OTP của bạn là: ${otp}. Mã có hiệu lực trong 5 phút.`
  });

  res.json({ message: 'Đã gửi mã OTP về email.' });
} catch (err) {
  console.error('Lỗi gửi OTP:', err);
  res.status(500).json({ message: 'Không thể gửi OTP.' });
}
;
```

- Cơ chế xác thực của OTP:

```
pp.post('/api/auth/verify-otp', async (req, res) => {
 try {
   const { email, otp } = req.body;

   // 1. Tìm record OTP trong database
   const record = await Otp.findOne({ email, otp });
   if (!record) {
     return res.status(400).json({ message: 'OTP không đúng' });
   }

   // 2. Kiểm tra thời gian hết hạn
   if (record.expiresAt < new Date()) {
     return res.status(400).json({ message: 'OTP đã hết hạn' });
   }

   res.json({ message: 'OTP hợp lệ' });
 } catch (err) {
   res.status(500).json({ message: 'Không thể xác thực OTP' });
 }
);
```

- Reset mật khẩu sau khi xác thực OTP:

```
app.post('/api/auth/reset-password', async (req, res) => {
  try {
    const { email, otp, newPassword } = req.body;

    // 1. Kiểm tra OTP có hợp lệ
    const record = await Otp.findOne({ email, otp });
    if (!record || record.expiresAt < new Date()) {
      return res.status(400).json({ message: 'OTP không hợp lệ hoặc đã hết hạn' });
    }

    // 2. Tìm user và cập nhật mật khẩu
    const user = await User.findOne({ email });
    const salt = await bcrypt.genSalt(10);
    user.password = await bcrypt.hash(newPassword, salt);
    await user.save();

    // 3. Xóa OTP đã sử dụng
    await Otp.deleteMany({ email });

    res.json({ message: 'Đổi mật khẩu thành công!' });
  } catch (err) {
    res.status(500).json({ message: 'Không thể đổi mật khẩu.' });
  }
```

- **Xây dựng chức năng đăng nhập đăng ký với JWT Authentication**

a. Giao diện người dùng

- Sử dụng form chung cho cả đăng nhập và đăng ký
- Có thể chuyển đổi giữa 2 form thông qua nút toggle
- Form đăng nhập có 2 trường: email và password
- Form đăng ký có 4 trường: username, email, password và confirmPassword
- Có link "Quên mật khẩu" trong form đăng nhập

b. Validation

- Kiểm tra username không được trống khi đăng ký
- Validate email phải đúng định dạng
- Password phải có ít nhất 6 ký tự
- Confirm password phải khớp với password khi đăng ký
- Hiển thị lỗi validation ngay dưới mỗi field

c. Các luồng đăng ký và đăng nhập:

- Luồng đăng ký:

- Frontend:

```
// 1. User điền form đăng ký
const handleRegister = async (e) => {
  e.preventDefault();
  try {
    // 2. Gửi request đến API
    const res = await axios.post('/api/register', {
      username,
      email,
      password
    });

    // 3. Lưu token vào localStorage
    localStorage.setItem('token', res.data.token);

    // 4. Cập nhật trạng thái đăng nhập
    login(res.data.user);
  } catch (err) {
    setError(err.response.data.message);
  }
}
```

- Backend:

```
// 1. Kiểm tra email đã tồn tại
let user = await User.findOne({ email });
if (user) {
  return res.status(400).json({ message: 'Email đã được sử dụng' });
}

// 2. Kiểm tra username đã tồn tại
user = await User.findOne({ username });
if (user) {
  return res.status(400).json({ message: 'Username đã tồn tại' });
}

// 3. Mã hóa mật khẩu
const salt = await bcrypt.genSalt(10);
const hashedPassword = await bcrypt.hash(password, salt);
```

```
  // 4. Tạo user mới
  user = new User({
    username,
    email,
    password: hashedPassword
  });
  await user.save();

  // 5. Tạo JWT token
  const token = jwt.sign(
    { id: user._id },
    process.env.JWT_SECRET,
    { expiresIn: '1d' }
  );

  // 6. Trả về token và thông tin user
  res.status(201).json({
    token,
    user: {
      id: user._id,
      username: user.username,
      email: user.email
    }
  });
} catch (err) {
```

- Luồng đăng nhập:

- Frontend:

```
const handleLogin = async (e) => {
  e.preventDefault();
  try {
    // 1. Gửi request đăng nhập
    const res = await axios.post('/api/login', {
      email,
      password
    });

    // 2. Lưu token
    localStorage.setItem('token', res.data.token);

    // 3. Cập nhật context
    login(res.data.user);
  } catch (err) {
    setError(err.response.data.message);
  }
}
```

- Backend:

```
// 1. Tìm user theo email
const user = await User.findOne({ email });
if (!user) {
  return res.status(400).json({ message: 'Email không tồn tại' });
}

// 2. Kiểm tra mật khẩu
const isMatch = await bcrypt.compare(password, user.password);
if (!isMatch) {
  return res.status(400).json({ message: 'Mật khẩu không đúng' });
}

// 3. Tạo JWT token
const token = jwt.sign(
  { id: user._id },
  process.env.JWT_SECRET,
  { expiresIn: '1d' }
);
```

```
    // 4. Trả về token và thông tin user
    res.json({
      token,
      user: {
        id: user._id,
        username: user.username,
        email: user.email
      }
    });
  } catch (err) {
    res.status(500).json({ message: 'Lỗi server' });
  }
});
```

d. Auth Middleware: Bảo vệ các route cần xác thực

```
function authMiddleware(req, res, next) {
  const token = req.header('x-auth-token');
  if (!token) {
    return res.status(401).json({ message: 'Không có token' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    res.status(401).json({ message: 'Token không hợp lệ' });
  }
}
```

➢ *Chức năng thay đổi thông tin cá nhân*

a) Frontend:

- **Hiển thị form với thông tin hiện tại**

```
const ProfilePage = () => {
  const navigate = useNavigate();  const { user, isAuthenticated, updateProfile }
  const [formData, setFormData] = useState({
    username: user?.username || '',
    email: user?.email || '',
    currentPassword: '',
    newPassword: '',
    confirmNewPassword: ''
  });
  const [loading, setLoading] = useState(false);
  const [message, setMessage] = useState({ type: '', content: '' });
```

- **Validate input khi submit**

```
if (formData.newPassword) {
  if (formData.newPassword.length < 6) {
    setMessage({ type: 'error', content: 'Mật khẩu mới phải có ít nhất 6 ký tự' });
    return;
  }
  if (formData.newPassword !== formData.confirmNewPassword) {
    setMessage({ type: 'error', content: 'Mật khẩu mới không khớp' });
    return;
  }
  if (formData.newPassword === formData.currentPassword) {
    setMessage({ type: 'error', content: 'Hãy đổi mật khẩu khác' });
    return;
  }
```

- **Gọi updateProfile từ authContext**

```
    setLoading(true);
    const result = await updateProfile({
      username: formData.username,
      currentPassword: formData.currentPassword,
      newPassword: formData.newPassword || undefined
    });
```

- **Xử lý response và hiển thị thông báo**

```
  if (result.success) {
    setMessage({ type: 'success', content: result.message });
    if (window.notify) window.notify('Đổi mật khẩu/thông tin thành công!');
```

- **Điều hướng nếu thành công**

b) Backend:

- **Nhận request PUT với token**

```
app.put('/api/user/profile', async (req, res) => {
  try {
```

- **Xác thực token và tìm user**

```
try {
  const token = req.header('x-auth-token');
  if (!token) {
    return res.status(401).json({ message: 'Không có token, quyền truy cập bị từ chối' });
  }

  const decoded = jwt.verify(token, process.env.JWT_SECRET || 'movieappsecret');
  let user = await User.findById(decoded.id);

  if (!user) {
    return res.status(404).json({ message: 'Không tìm thấy người dùng' });
  }

  const { username, currentPassword, newPassword } = req.body;

  // Kiểm tra xem username mới có bị trùng không (nếu có thay đổi)
  if (username !== user.username) {
    const existingUser = await User.findOne({ username });
    if (existingUser) {
      return res.status(400).json({ message: 'Tên người dùng đã tồn tại' });
    }
  }
```

- **Kiểm tra username mới có bị trùng**

```
// Kiểm tra xem username mới có bị trùng không (nếu có thay đổi)
if (username !== user.username) {
  const existingUser = await User.findOne({ username });
  if (existingUser) {
    return res.status(400).json({ message: 'Tên người dùng đã tồn tại' });
  }
}
```

- **Nếu đổi mật khẩu:**
  - **Verify mật khẩu hiện tại**

```
if (newPassword) {
  // Xác thực mật khẩu hiện tại
  const isMatch = await bcrypt.compare(currentPassword, user.password);
  if (!isMatch) {
    return res.status(400).json({ message: 'Mật khẩu hiện tại không đúng' });
  }
```

  - **Mã hóa mật khẩu mới**

```
  // Mã hóa mật khẩu mới
  const salt = await bcrypt.genSalt(10);
  const hashedPassword = await bcrypt.hash(newPassword, salt);
  user.password = hashedPassword;
}
```

- **Lưu thay đổi vào database**

```
    // Cập nhật username
    user.username = username;

    // Lưu thay đổi
    await user.save();

    res.json({
      message: 'Cập nhật thông tin thành công',
      user: {
        id: user._id,
        username: user.username,
        email: user.email
      }
    });

  } catch (err) {
    console.error('Lỗi khi cập nhật profile:', err);
    res.status(500).json({ message: 'Lỗi server' });
  }
});
```

c. Bảo mật:

- Token xác thực cho mọi request
- Kiểm tra mật khẩu hiện tại
- Validate đầy đủ ở cả 2 phía
- Mật khẩu được hash trước khi lưu
- Email không được phép thay đổi

➢ **Chức năng chatbox với AI**

a. Luồng hoạt động

1. Xác thực user.
2. Phân tích message: hỏi về diễn viên hay không.
3. Lấy dữ liệu phim phù hợp từ TMDb.
4. Tạo prompt và gửi cho Google Gemini.
5. Nhận câu trả lời AI và trả về frontend.

b. Giải thích

+ Xác thực và nhận message từ client

```
app.post('/api/chat', authMiddleware, async (req, res) => {
  try {
    const { message } = req.body;
    if (!message) {
      return res.status(400).json({ message: 'Message is required' });
    }
```

- Sử dụng middleware authMiddleware để kiểm tra token đăng nhập.
- Lấy message từ body request. Nếu không có, trả về lỗi 400.

+ **Xử lý phân tích message: hỏi về diễn viên hay phim trending**

```
// Phân tích message: nếu hỏi về diễn viên thì lấy phim theo diễn viên
const actorRegex = /diễn viên ([^\n\r\d.,!?]+)|phim của ([^\n\r\d.,!?]+)|([^\n\r\d.,!?]+) đóng phim gì/i;
let actorName = null;
const match = message.match(actorRegex);
if (match) {
  actorName = match[1] || match[2] || match[3];
  if (actorName) {
    actorName = actorName.trim();
    try {
```

- Tạo regex để phát hiện nếu user hỏi về diễn viên.
- Nếu có, lấy tên diễn viên từ message.

+ **Nếu hỏi về diễn viên, lấy danh sách phim của diễn viên từ TMDb**

```
try {
  // Tìm kiếm diễn viên trên TMDb
  const searchRes = await axios.get('https://api.themoviedb.org/3/search/person', {
    params: { api_key: process.env.TMDB_API_KEY, query: actorName, language: 'vi-VN' }
  });
  if (searchRes.data.results && searchRes.data.results.length > 0) {
    const person = searchRes.data.results[0];
    // Lấy danh sách phim của diễn viên
    const creditsRes = await axios.get(`https://api.themoviedb.org/3/person/${person.id}/movie_credits`, {
      params: { api_key: process.env.TMDB_API_KEY, language: 'vi-VN' }
    });
    const movies = creditsRes.data.cast ? creditsRes.data.cast.slice(0, 15) : [];
    movieList = movies.map(m => `${m.title} (${m.release_date || ''})`).join(', ');
    usedActor = true;
    prompt = `Dưới đây là danh sách phim của diễn viên ${actorName} trên TMDb: ${movieList}\n\nUser: ${message
  }
} catch (err) {
  console.error('TMDb actor search error:', err);
}
```

- Gọi TMDb API để tìm diễn viên.
- Lấy ID diễn viên, gọi tiếp API để lấy danh sách phim của diễn viên đó.
- Tạo prompt cho AI dựa trên danh sách phim vừa lấy.

+ **Nếu không hỏi về diễn viên, lấy phim trending**

```
// Nếu không phải hỏi về diễn viên hoặc không tìm được diễn viên, lấy phim trending như cũ
if (!usedActor) {
  try {
    const tmdbRes = await axios.get('https://api.themoviedb.org/3/trending/all/day?language=en-US', {
      params: { api_key: process.env.TMDB_API_KEY, language: 'vi-VN' }
    });
    const movies = tmdbRes.data.results.slice(0, 15);
    movieList = movies.map(m => `${m.title} (${m.release_date || ''})`).join(', ');
  } catch (tmdbErr) {
    console.error('TMDb error:', tmdbErr);
    movieList = '';
  }
  prompt = `Dưới đây là danh sách phim trending trong ngày trên TMDb: ${movieList}\n\nUser: ${message}\nAssistant:
}
```

- Nếu không phải hỏi về diễn viên, gọi TMDb API lấy danh sách phim trending.
- Tạo prompt cho AI dựa trên danh sách phim trending.

+ **Gọi Google Gemini để sinh câu trả lời**

```
const model = genAI.getGenerativeModel({ model: "gemini-2.5-flash-preview-05-20" });
const result = await model.generateContent(prompt);
const response = await result.response;
const text = response.text();
```

- Gửi prompt (có danh sách phim) cho Google Gemini.
- Lấy kết quả trả về từ AI.

+ **Trả kết quả về cho client**

```
res.json({
  message: text,
  timestamp: new Date()
});
```

5. *Cài đặt và triển khai*

5.1. Cài đặt môi trường phát triển

a) Yêu cầu cấu hình

- Node.js (phiên bản 14.x trở lên)
- NPM (Node Package Manager)
- MongoDB (phiên bản 4.x trở lên)
- Trình duyệt web hiện đại (Chrome, Firefox, Edge,...)
- Ít nhất 4GB RAM
- Ổ cứng còn trống ít nhất 1GB

b) Các bước cài đặt

- B1: Clone repository

git clone https://github.com/thanh12a2/ThucTapCoSo1.git

cd movieapp

- B2: Cài đặt dependencies

+) Frontend:

cd movieapp-main

npm install

+) Backend:

cd backend

npm install

- B3: Cấu hình file .env

+) Frontend:

REACT_APP_ACCESS_TOKEN = Your_TMDB_access_token

REACT_APP_API_URL=http://localhost:5000/api

JWT_SECRET=your_JWT_key

MONGODB_URI=mongodb://localhost:27017/your_database_name

+) Backend:

MONGODB_URI=mongodb://localhost:27017/webmoive

JWT_SECRET=your_JWT_key

EMAIL_USER=your_email_user

EMAIL_PASS=your_email_password_code

GEMINI_API_KEY = your_gemini_api_key

TMDB_API_KEY=your_TMDB_api_key

5.2. Triển khai ứng dụng

- Mở MongoDB và kết nối

-Mở 2 terminal:

+) Terminal 1: Chạy backend

cd backend

node server.js

+) Terminal 2: Chạy frontend

cd movieapp-main

npm start

6. Đánh giá ưu, nhược điểm và khả năng ứng dụng của hệ thống

6.1. Ưu điểm

- Giao diện thân thiện: Hệ thống được thiết kế với giao diện người dùng trực quan, dễ sử dụng. Đã áp dụng các nguyên tắc thiết kế UI/UX hiện đại như:

- Layout phù hợp với thói quen đọc
- Phối màu hài hòa, dễ chịu
- Typography rõ ràng, dễ đọc
- Responsive design cho các kích thước màn hình phổ biến
- Các yếu tố tương tác (nút, form) được thiết kế trực quan

- Tối ưu hiệu suất: Đã áp dụng nhiều kỹ thuật tối ưu hiệu suất:

- Caching dữ liệu thời tiết để giảm số lượng API calls
- Tối ưu truy vấn database bằng cách sử dụng index và pagination

- Bảo mật thông tin: Hệ thống được xây dựng với các biện pháp bảo mật cơ bản:

- Sử dụng cơ chế xác thực JWT Authentication
- Sử dụng mã OTP để khôi phục mật khẩu với nodemailer

- Tính năng đa dạng: Website đáp ứng đầy đủ các nhu cầu cơ bản của một trang xem phim:

- Quản lý người dùng
- Quản lý phim đa dạng
- Tương tác xã hội (thích, bình luận)
- Tìm kiếm và phân loại nội dung
- Thống kê và báo cáo
- Thông báo thời gian thực

- Tích hợp API bên ngoài: Việc tích hợp API của The Movie Database không chỉ cung cấp nhiều thông tin về phim, diễn viên cho người dùng mà còn chứng minh khả năng tương tác với các dịch vụ bên ngoài của hệ thống.

- Khả năng mở rộng: Kiến trúc hệ thống được thiết kế theo mô hình MVC (Model-View-Controller) rõ ràng, giúp dễ dàng thêm các tính năng mới mà không ảnh hưởng đến cấu trúc hiện tại.

6.2. Nhược điểm

- Chưa hỗ trợ đa ngôn ngữ: Hiện tại, hệ thống chỉ hỗ trợ tiếng Việt và tiếng Anh. Điều này hạn chế khả năng tiếp cận người dùng quốc tế. Để khắc phục được điều này, cần xây dựng hệ thống đa ngôn ngữ với các file ngôn ngữ riêng biệt và cơ chế chuyển đổi ngôn ngữ.

- Chưa có tính năng phân tích dữ liệu nâng cao: Hệ thống hiện tại chỉ cung cấp các thống kê cơ bản. Việc bổ sung các công cụ phân tích chuyên sâu về hành vi người dùng sẽ giúp hiểu rõ hơn nhu cầu và cải thiện trải nghiệm.

- **Chưa có phân quyền người dùng**: Hệ thống hiện tại mọi người dùng đều có thể truy cập được. Việc bổ sung thêm phân quyền người dùng như : admin, nhà làm phim có thể giúp trang web được bổ sung thêm nhiều bộ phim hoặc các thông tin về diễn viên được đa dạng hơn.

6.3. Khả năng ứng dụng thực tế

- **Nền tảng giải trí cá nhân**: Website xem phim trực tuyến có thể trở thành một nền tảng giải trí độc lập phục vụ cá nhân hoặc cộng đồng nhỏ. Người dùng có thể dễ dàng truy cập, tìm kiếm và thưởng thức các bộ phim yêu thích ở bất kỳ đâu, bất kỳ lúc nào mà không cần phụ thuộc vào các dịch vụ trả phí lớn như Netflix hay FPT Play. Với khả năng tùy chỉnh, người dùng có thể tạo danh sách phim cá nhân, lưu lịch sử xem, đánh giá phim hoặc chia sẻ với bạn bè.

- **Ứng dụng trong giáo dục và đào tạo**: Không chỉ dừng lại ở mục đích giải trí, hệ thống có thể được tái sử dụng để phát triển thành nền tảng học tập trực tuyến. Các video bài giảng, khóa học, nội dung đào tạo có thể được quản lý và phát theo cách tương tự như phim. Ngoài ra, tính năng phân quyền người dùng giúp phân biệt vai trò của học viên và giảng viên một cách rõ ràng, giúp nâng cao hiệu quả giảng dạy.

- **Xây dựng hệ thống quản lý nội dung cho doanh nghiệp**: Website có thể được mở rộng để phục vụ như một CMS (Content Management System) cho doanh nghiệp trong lĩnh vực truyền thông, giải trí hoặc giáo dục. Hệ thống có thể cho phép quản trị viên đăng tải phim, phân loại, gắn nhãn độ tuổi, kiểm duyệt nội dung và theo dõi số lượng người xem, từ đó hỗ trợ quyết định kinh doanh hiệu quả hơn.

7. Kết luận và hướng phát triển

7.1. Kết luận chung

- Sau quá trình thực hiện đồ án, em đã xây dựng thành công hệ thống website xem phim trực tuyến MOIVEO với đầy đủ các chức năng cơ bản và một số tính năng nâng cao như tích hợp AI Chatbot (Google Gemini), thông báo thời gian thực, và lưu trữ cá nhân hóa.

- Hệ thống được phát triển trên nền tảng MERN Stack (MongoDB, Express, React, Node.js), sử dụng kiến trúc tách biệt Frontend–Backend, giúp đảm bảo khả năng mở rộng, bảo trì và nâng cấp về sau. Ngoài ra, các nguyên tắc như MVC, RESTful API, JWT Authentication, Socket.IO cũng đã được áp dụng để đảm bảo hiệu suất, bảo mật và trải nghiệm người dùng.

- Việc triển khai thành công các tính năng như xác thực người dùng, phát video trực tuyến, lưu yêu thích, bình luận, và đặc biệt là chatbot AI đã chứng minh tính khả thi và tiềm năng ứng dụng của hệ thống trong thực tế.

7.2. Hướng phát triển

7.2.1. Cải thiện trải nghiệm người dùng

- Hỗ trợ đa ngôn ngữ: Thiết kế lại hệ thống để hỗ trợ nhiều ngôn ngữ, bao gồm:

- Tạo file ngôn ngữ riêng biệt (language files)
- Xây dựng cơ chế chuyển đổi ngôn ngữ
- Tách nội dung và giao diện để dễ dàng dịch

- Chế độ tối (Dark mode): Phát triển chế độ giao diện tối để người dùng có thể lựa chọn tùy theo sở thích và điều kiện ánh sáng. Việc này bao gồm:

- Thiết kế bảng màu cho dark mode
- Lưu trữ preference của người dùng
- Tự động chuyển đổi dựa trên thời gian hoặc cài đặt hệ thống

- **Cá nhân hóa giao diện**: Cho phép người dùng tùy chỉnh giao diện theo sở thích cá nhân:

- Thay đổi bố cục trang chủ
- Lựa chọn danh mục ưa thích để hiển thị
- Điều chỉnh kích thước font chữ và các yếu tố hiển thị

- **Quản lí người dùng đa dạng hơn**: Cho phép admin có thể phê duyệt phim, bình luận; nhà làm phim có thể đăng các bộ phim mới lên website; người dùng có thể có các gói Premium để có quyền xem các bộ phim có trả phí…

7.2.2 Mở rộng tính năng

- **Tính năng đăng ký nhận bản tin**: Phát triển hệ thống email marketing để người dùng có thể đăng ký nhận:

- Thông báo về bộ phim mới
- Thông báo về sự kiện hoặc cập nhật quan trọng

- **Hỗ trợ đa phương tiện**: Mở rộng khả năng đăng tải và hiển thị các loại nội dung:

- Video (tích hợp với YouTube hoặc tự host)
- Audio (podcast, đọc bài viết)
- Slideshow và gallery ảnh
- Tài liệu nhúng (PDF, presentations)

- **Tính năng tìm kiếm nâng cao**: Bổ sung các bộ lọc và tùy chọn tìm kiếm chuyên sâu:

- Tìm kiếm theo khoảng thời gian
- Tìm kiếm theo độ phổ biến
- Tìm kiếm theo độ liên quan
- Gợi ý tìm kiếm thông minh

7.2.3. Cải thiện bảo mật và hiệu suất

- **Triển khai HTTPS**: Đảm bảo tất cả kết nối đều được mã hóa bằng cách:

- Cài đặt SSL certificate
- Chuyển hướng tất cả request HTTP sang HTTPS
- Thiết lập HSTS (HTTP Strict Transport Security)

- **Tối ưu hóa SEO**: Cải thiện khả năng hiển thị trên các công cụ tìm kiếm:

- Tối ưu cấu trúc URL
- Tạo sitemap.xml và robots.txt

- Bổ sung metadata (Open Graph, Twitter Cards)
- Cải thiện tốc độ tải trang
- Tối ưu nội dung cho từ khóa

- **Triển khai CDN**: Sử dụng mạng phân phối nội dung để:

- Phân phối static assets (hình ảnh, CSS, JavaScript)
- Giảm độ trễ cho người dùng ở các vị trí địa lý khác nhau
- Giảm tải cho server chính
- Cung cấp layer bảo vệ DdoS