BỘ THÔNG TIN VÀ TRUYỀN THÔNG

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÁO CÁO BÀI TẬP LỚN
MÔN PHÁT TRIỂN HỆ THỐNG TMDT

Đề tài: "**Website bán vé xem phim**"

Giảng viên hướng dẫn : **Kim Ngọc Bách**

Sinh viên thực hiện: **Doanh Văn Vũ - B21DCCN794**

Nguyễn Châu Long – B21DCCN495

Nguyễn Đăng Quang – B21DCCN628

Nhóm: **11**

Hà Nội, 2025

LỜI CẢM ƠN

Chúng em xin chân thành gửi lời cảm ơn đến thầy Kim Ngọc Bách, người đã tận tình hướng dẫn, chỉ bảo và hỗ trợ chúng em trong suốt quá trình thực hiện bài tập lớn môn Phát triển hệ thống Thương mại điện tử.

Nhờ sự chỉ dẫn quý báu của thầy, chúng em không chỉ nắm vững kiến thức chuyên môn về xây dựng hệ thống web hiện đại mà còn học hỏi được nhiều kỹ năng quan trọng trong quá trình làm việc nhóm.

Cuối cùng, chúng em xin cảm ơn các thành viên trong nhóm vì sự nỗ lực, tinh thần trách nhiệm và tinh thần hợp tác cao, giúp dự án hoàn thành đúng tiến độ và đạt được kết quả như mong đợi.

Mục lục

I. Giới thiệu đề tài:

1.1. Giới thiệu Website đặt vé xem phim:

Nhóm 11 tạo Website đặt vé xem phim cho các rạp chiếu phim. Khách hàng khi muốn mua vé xem phim mà không muốn đến tận rạp có thể đặt trước tại website này.

Các rạp chiếu sẽ dùng trang web này để cập nhật các bộ phim mà rạp sẽ chiếu hàng ngày để người dùng có thể theo dõi lịch chiếu phim.

Bài tập lớn này được xây dựng dựa trên các công nghệ phổ biến trong phát triển các ứng Web ngày như Node.js, React, MongoDB cùng với Framework Express.

Về bố cục của bài tập lớn, mã nguồn của dự án được chia thành 2 phần là Backend và Frontend.

Trong đó phần Backend xây dựng API cho ứng dụng web, các Model trong MongoDB, xác minh đăng nhập cho người dùng. Phần Frontend xây dựng giao diện người dùng với thư viện MUI (Material UI), sử dụng Redux để quản lý trạng thái của trang web, đồng thời gọi API và thực hiện các yêu cầu HTTP thông qua thư viện Axios để hiển thị phim, thông tin người dùng, vé đã .

1.2. Công nghệ sử dụng:

Node.js

Được phát hành vào năm 2009, NodeJS, hay còn được biết với tên gọi chính thức là Node.js, là môi trường thời gian chạy (runtime environment) JavaScript đa nền tảng và mã nguồn mở. NodeJS cho phép các lập trình viên tạo cả ứng dụng Frontend và Backend bằng JavaScript.

Express.js

Expressjs là một framework được xây dựng trên nền tảng của Nodejs. Nó cung cấp các tính năng mạnh mẽ để phát triển web hoặc mobile. Expressjs hỗ trợ các method HTTP và midleware tạo ra API vô cùng mạnh mẽ và dễ sử dụng.

MongoDB

MongoDB là một hệ quản trị cơ sở dữ liệu mã nguồn mở, là CSDL thuộc NoSql và là một database hướng tài liệu (document), các dữ liệu được lưu trữ trong document kiểu JSON thay vì dạng bảng như CSDL quan hệ nên truy vấn sẽ rất nhanh. Với CSDL quan hệ chúng ta có khái niệm bảng, các cơ sở dữ liệu quan hệ (như MySQL hay SQL Server...) sử dụng các bảng để lưu dữ liệu thì với MongoDB chúng ta sẽ dùng khái niệm là collection thay vì bảng.

React

React là thư viện JavaScript phổ biến nhất để xây dựng giao diện người dùng (UI) . Nó cho tốc độ phản hồi tuyệt vời khi user nhập liệu bằng cách sử dụng phương pháp mới để render trang web.

Redux

Redux là một thư viện quản lý trạng thái (state management) tương thích với các ứng dựng web, phổ biến trong việc phát triển ứng dụng Frontend sử dụng JavaScript và ReactJS.

Đây là một công cụ hữu ích giúp xây dựng các ứng dụng có tính nhất quán, hoạt động linh hoạt trên nhiều môi trường

Cách Redux hoạt động rất đơn giản. Có một "store" trung tâm chứa toàn bộ trạng thái của ứng dụng. Mỗi thành phần có thể truy cập trạng thái được lưu trữ mà không phải gửi từ thành phần này sang thành phần khác.

Axios

Axios là thư viện giúp client tương tác với server thông qua giao thức HTTP dựa trên các Promises. Trong dự án web này, ta sử dụng Axios để giao tiếp với backend và xử lý phản hồi từ đó.

Material UI

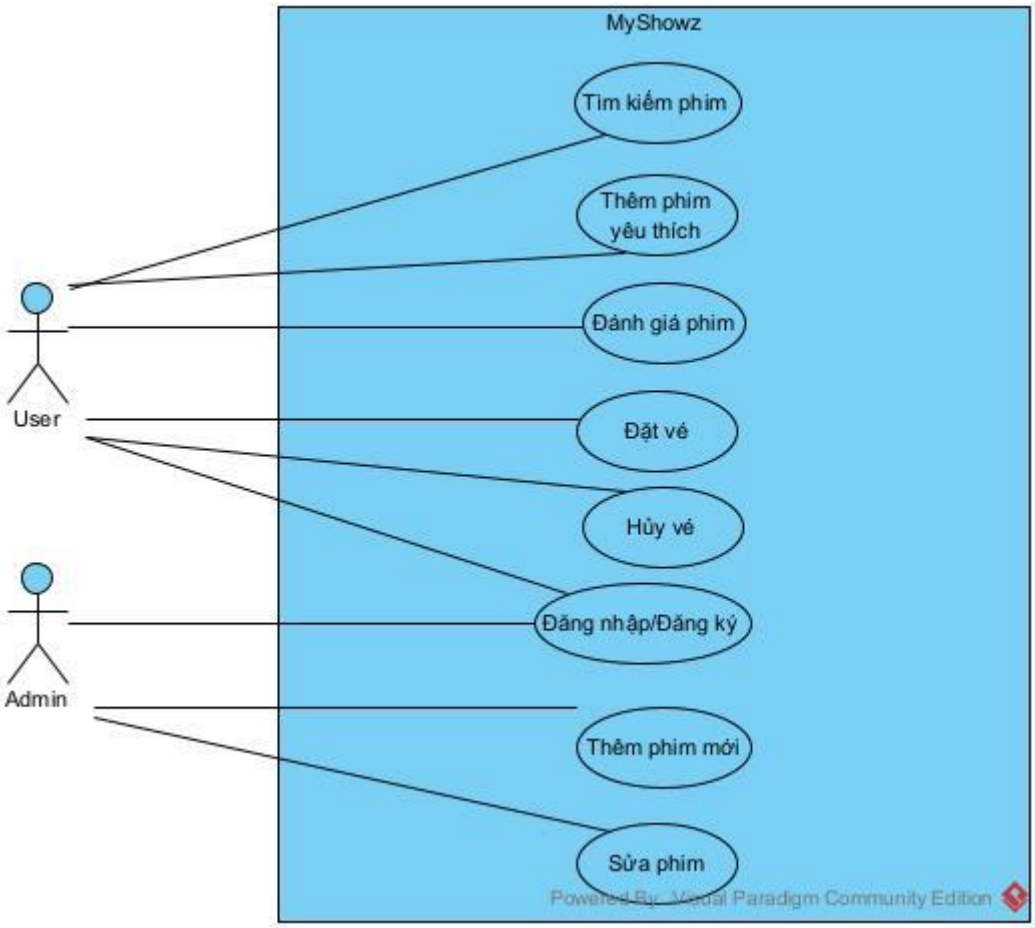
Material-UI xuất phát điểm là một thư viện các React Component đã được tích hợp thêm cả Google's Material Design. Material UI đem đến cho bạn và trang web của bạn một giao diện hoàn toàn mới, với những button, textfield, toggle… được design theo một phong cách mới lạ.

II. Phân tích hệ thống

2.1Xác định và mô tả tác nhân

- Quản trị hệ thống (Admin) : Người có quyền quản lý đơn hàng, quản lý các bộ phim có trong website bán vé
- Khách hàng (Customer) : Người có thể mua vé thông qua website bằng các hình thức thanh toán có sẵn trên web

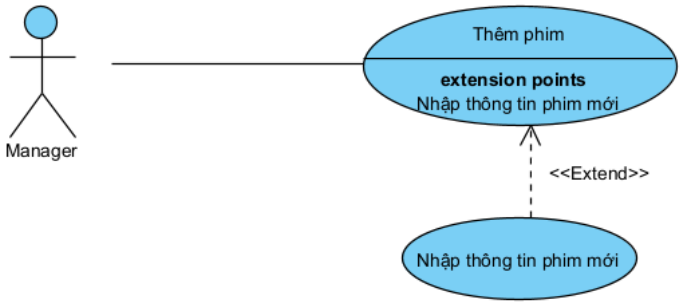
2.2.Use case tổng quan



2.3.Use case chi tiết và kịch bản các chức năng chính

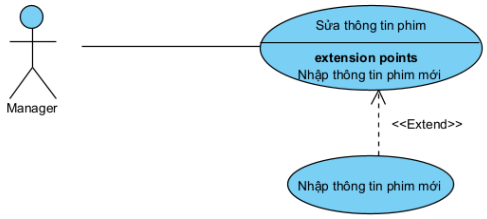
2.3.1.Chức năng chính của người quản trị

a. Thêm phim



Use case	Thêm phim
Actor	Người quản trị
Tiền điều kiện	Người quản trị đăng nhập thành công
Hậu điều kiện	Phim mới được thêm thành công
Kịch bản chính	1.Người quản trị đăng nhập thành công 2.Hệ thống hiển thị giao diện trang chủ người quản trị 3. Người quản trị chọn chức năng thêm phim 4. Hệ thống hiện giao diện thêm phim mới bao gồm các ô nhập - Tên phim - Giới thiệu - Poster - Ngày phát hành - Lịch chiếu phim Nút Thêm phim 5. Người quản trị nhập thông tin phim muốn thêm sau đó chọn nút Thêm phim 6. Hệ thống hiện thông báo Thêm phim thành công và quay trở lại giao diện trang chủ
Ngoại lệ	5.Người quản trị chỉ nhập tên phim 5.1. Hệ thống hiện thông báo yêu cầu nhập thêm thông tin vào các ô thông tin

b. Sửa thông tin phim

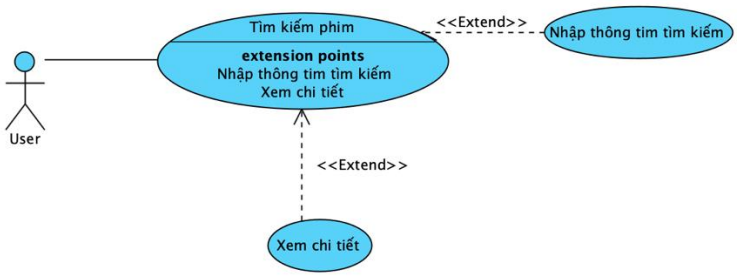


Use case	Sửa thông tin phim
Actor	Người quản trị

Tiền điều kiện	Người quản trị đăng nhập thành công
Hậu điều kiện	Thông tin phim được sửa thành công
Kịch bản chính	1.Người quản trị đăng nhập thành công 2.Hệ thống hiển thị giao diện trang chủ người quản trị 3. Người quản trị chọn chức năng sửa thông tin phim 4. Hệ thống hiện giao diện sửa thông tin phim bao gồm các ô nhập - Tên phim - Giới thiệu - Poster - Ngày phát hành - Lịch chiếu phim Nút Sửa phim 5. Người quản trị nhập thông tin phim muốn sửa sau đó chọn nút Sửa phim 6. Hệ thống hiện thông báo Sửa phim thành công và quay trở lại giao diện trang chủ
Ngoại lệ	5.Người quản trị chỉ nhập một ô và xóa hết thông tin các ô còn lại 5.1. Hệ thống hiện thông báo yêu cầu nhập thêm thông tin vào các ô thông tin

2.3.2.Chức năng chính của khách hàng

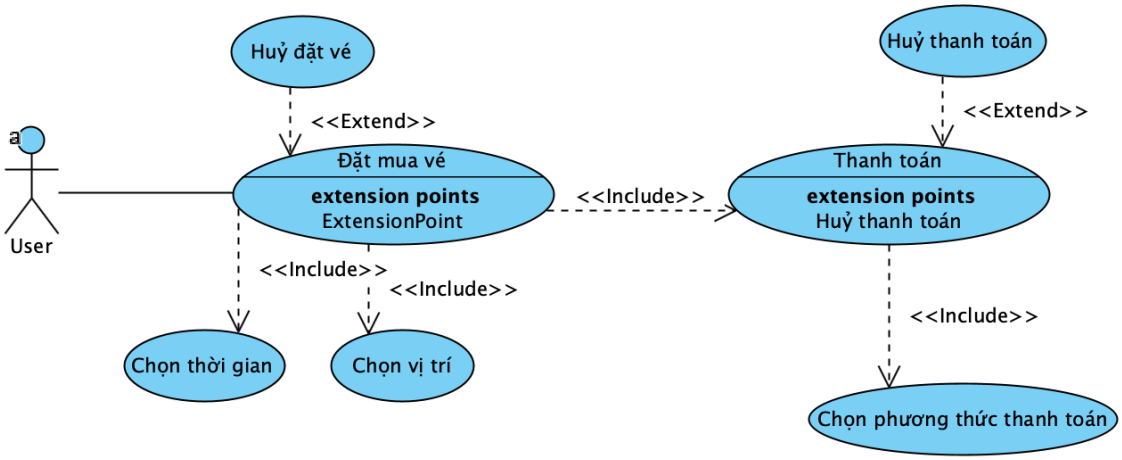
a. Tìm kiếm phim



Use case	Tìm kiếm phim
Actor	Người dung
Tiền điều kiện	Người dùng truy cập và đăng nhập thành công trang web
Hậu điều kiện	Tìm và xem thông tin chi tiết của bộ phim tìm kiếm
Kịch bản chính	1. Người dùng truy cập trang web thành công.

<table>
<tr><td></td><td>2. Hệ thống hiển thị giao diện trang chủ.
3. Người dùng chọn chức năng "Tìm kiếm phim".
4. Hệ thống hiển thị giao diện tìm kiếm.
<table><tr><td>Nhập để tìm kiếm</td></tr><tr><td>-
-
-</td></tr></table>
5. Người dùng nhập từ khóa là tên phim muốn tìm.
6. Hệ thống xử lý và hiển thị danh sách các phim liên quan đến từ khóa.
<table><tr><td>Nhà</td></tr><tr><td>-Nhà bà Loan
-Ngôi nhà ma ám</td></tr></table>
7. Người dùng chọn một phim cụ thể trong danh sách.
8. Hệ thống hiển thị giao diện thông tin chi tiết của phim được chọn.</td></tr>
<tr><td>Ngoại lệ</td><td>5.Người dùng không nhập từ khóa tìm kiếm
5.1. Giao diện không hiển thị phim
6. Hệ thống không tìm thấy phim chứa từ khóa tìm kiếm
6.1. Giao diện không hiển thị thông tin phim</td></tr>
</table>

b. Đặt vé



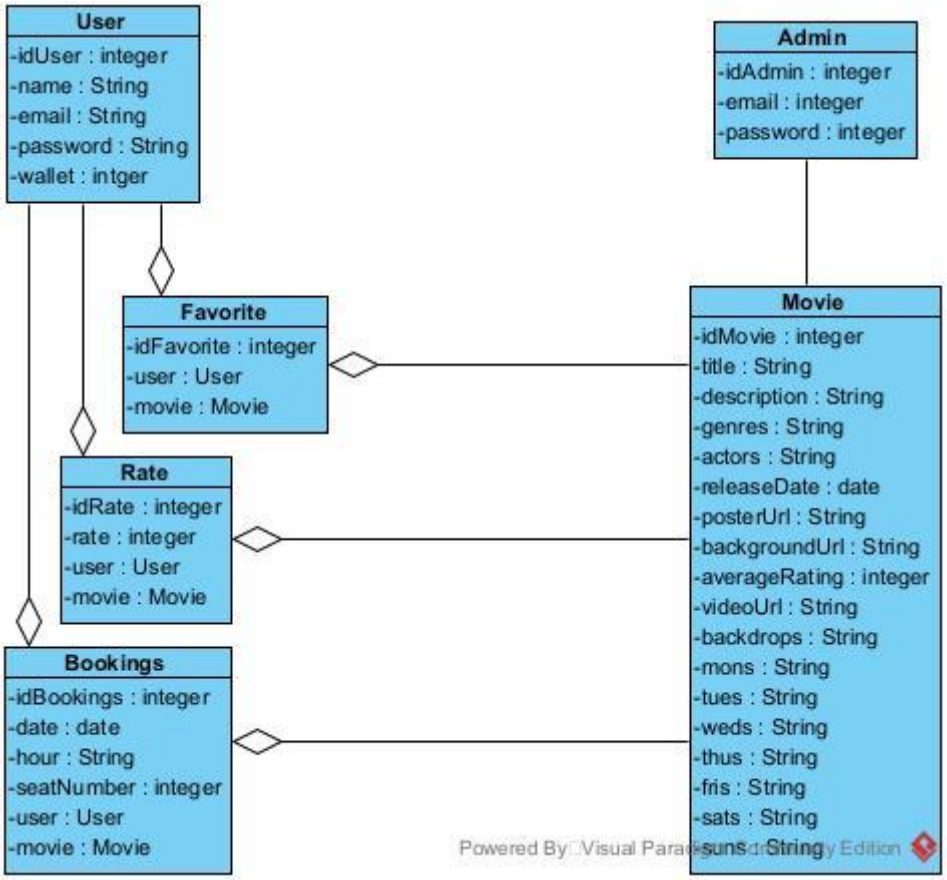
Use case	Đặt mua vé
Actor	Người dùng
Tiền điều kiện	Người dùng đăng nhập thành công
Hậu điều kiện	Đặt mua thành công
Kịch bản chính	1.Người dùng chọn 1 phim từ giao diện chính 2. Giao diện thông tin phim hiện lên và có nút đặt vé 3. Người dùng nhấn nút đặt vé 4.Giao diện kéo xuống hiển thị chọn ngày Chọn ngày đặt vé 12/6 | 13/6 5.Người dùng chọn ngày cần đặt 6.Giao diện cập nhập hiển thị thêm chọn giờ Chọn ngày đặt vé 12/6 | 13/6 Chọn giờ

	<table><tr><td>16:00</td><td>18:00</td><td>20:00</td><td>22:00</td></tr></table> 7. Người dùng chọn giờ đặt vé 8.Giao diện cập nhập hiển thị thêm chọn vị trí <table><tr><td colspan="4">Chọn ngày đặt vé</td></tr><tr><td colspan="2">12/6</td><td colspan="2">13/6</td></tr><tr><td colspan="4">Chọn giờ</td></tr><tr><td colspan="4"><table><tr><td>16:00</td><td>18:00</td><td>20:00</td><td>22:00</td></tr></table></td></tr><tr><td colspan="4">Chọn vị trí</td></tr><tr><td colspan="4"><table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr></table></td></tr><tr><td colspan="4">Đặt vé</td></tr></table> 9.Người dùng chọn vị trí và click Đặt vé 10.Giao diện chọn phương thức thanh toán hiện lên 11.Người dùng nhấn chọn 1 phương thức thanh toán 12.Giao diện thanh toán hiện lên 13.Người dùng thực hiện thanh toán 14.Thông báo thanh toán thành công hiện lên.
Ngoại lệ	10. Người dùng nhấn huỷ thanh toán 10.1. Hệ thống quay về trang chủ 13. Tài khoản người dùng không đủ tiền

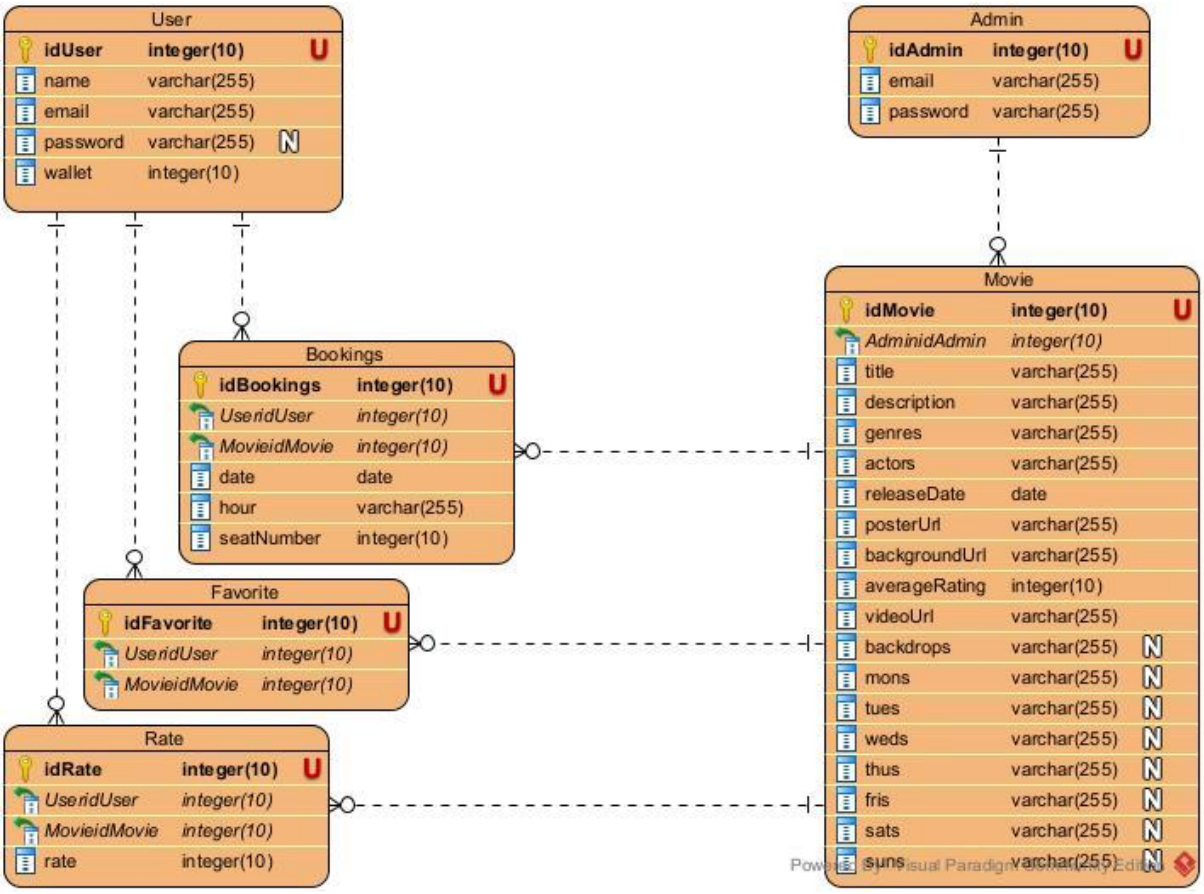
	13.1.Thông báo tài khoản hoặc ví của khách hàng không đủ tiền hiện lên và nút Xác nhận 13.2. Khách hàng chọn Xác nhận 13.3. Hệ thống quay trở lại giao diện thanh toán

III. Thiết kế hệ thống

3.1.Sơ đồ lớp thực thể toàn hệ thống



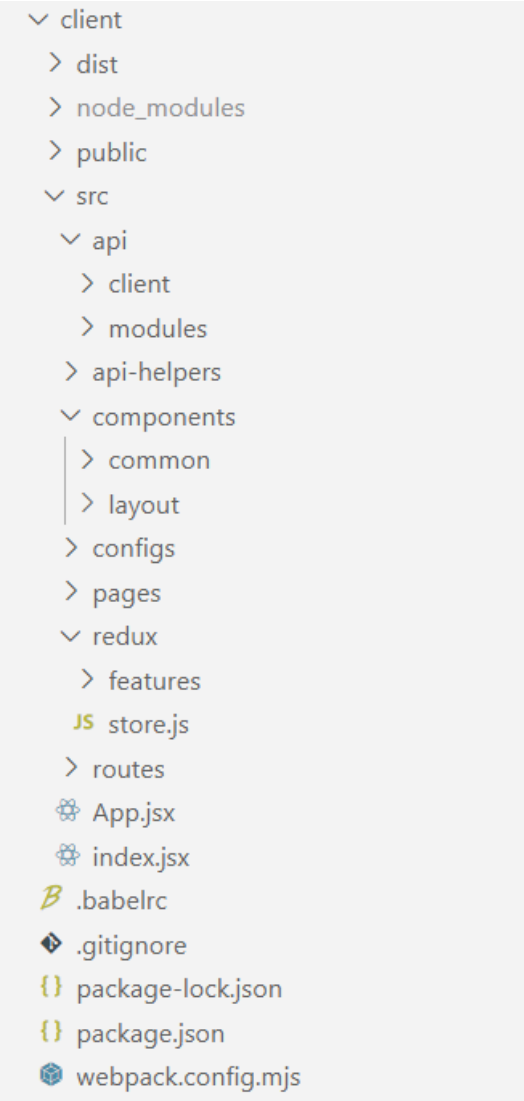
3.2.Thiết kế cơ sở dữ liệu



IV. Cài đặt hệ thống

4.1.Frontend (client)

Để xây dựng Frontend cho website, nhóm sử dụng công nghệ React.

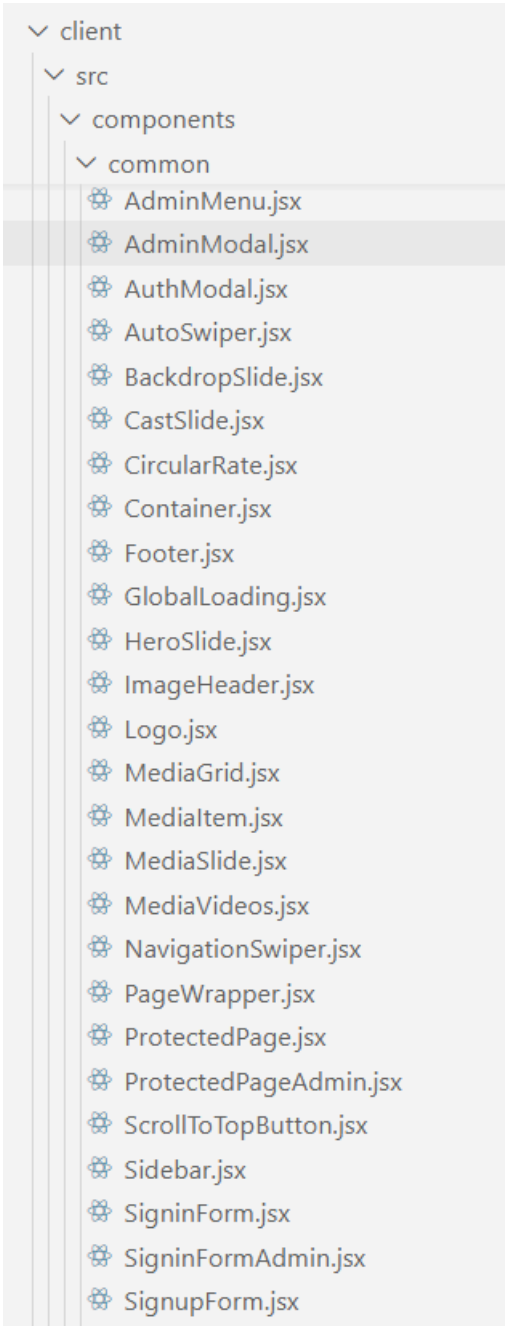


Hình 4.1.1. Cấu trúc frontent (client)

4.1.1.Components

Giao diện trang web được chia thành các phần nhỏ trong thư mục components/common như thanh menu, logo, form đăng ký,...

Ngoài ra file MainLayout quyết định cấu trúc chính mà các phần nhỏ được thể hiện trên giao diện trang web.



Hình 4.1.2. components

4.1.2. Pages:

Thư mục page chứa giao diện của các trang khác nhau trong web như trang FavoriteList.jsx chứa các bộ phim ưa thích của người dùng, trang thể hiện các vé đã đặt như BookingList.jsx, hay trang tìm kiếm MediaSearch.jsx.

MediaSearch.jsx

client > src > pages > MediaSearch.jsx > ...

```jsx
const MediaSearch = () => {
  const [movies, setMovies] = useState([]);
  const [medias, setMedias] = useState([]);
  const [searchType, setSearchType] = useState(searchTypes[0]);
  const onChange = (selected) => setSearchType(selected);
  useEffect(() => {
    const getMovies = async () => {
      await getAllMovies()
        .then((data) => setMovies(data.movies))
        .catch((err) => console.log(err));
    };
    getMovies();
  }, []);
  const onQueryChange = (e) => {
    const newQuery = e.target.value;
    if(searchType == "tên phim") setMedias(movies.filter(movie => movie.title.toLowerCase().includes(newQuery.toLowerCase())));
    else if(searchType == "thể loại") setMedias(movies.filter(movie => movie.genres.some(genre => genre.toLowerCase().includes(newQuery
    else setMedias(movies.filter(movie => movie.actors.some(actor => actor.toLowerCase().includes(newQuery.toLowerCase()))));
    if(newQuery == "") setMedias([]);
  };
  return (
    <>
      <Toolbar />
      <Box sx={{ ...uiConfigs.style.mainContent }}>
        <Stack spacing={2}>
          <Stack
            spacing={2}
            direction="row"
            justifyContent="center"
            sx={{ width: "100%" }}
          >
            {searchTypes.map((item, index) => (
              <Button
                size="large"
                key={index}
                variant={searchType === item ? "contained" : "text"}
```

Hình 4.1.3. MediaSearch.jsx

4.1.3.Giao tiếp với server

Để giao tiếp với phần Backend của trang web và lấy dữ liệu từ server, trong Frontend, ta sử dụng thư viện *Axios* để thực hiện các yêu cầu HTTP tới server.

Các hàm request sử dụng *Axios* được lưu trữ trong file *api-helpers.js*

api-helpers.js

client > src > api-helpers > api-helpers.js > getTicket

```
import axios from "axios";
export const getAllMovies = async () => {
  const res = await axios.get("/movie").catch((err) => console.log(err));

  if (res.status !== 200) {
    return console.log("No Data");
  }

  const data = await res.data;
  return data;
};

export const sendAdminAuthRequest = async (data) => {
  const res = await axios
    .post("/admin/login", {
      email: data.email,
      password: data.password,
    })
    .catch((err) => console.log(err));

  if (res.status !== 200) {
    return console.log("Unexpectyed Error");
  }

  const resData = await res.data;
  return resData;
};

export const getMovieDetails = async (id) => {
  const res = await axios.get(`/movie/${id}`).catch((err) => console.log(err));
  if (res.status !== 200) {
    return console.log("Unexpected Error");
  }
  const resData = await res.data;
  return resData;
};
export const getBookings = async (id) => {
```

Hình 4.1.4. api-helpers

Lấy dữ liệu tất cả các bộ phim:

```
export const getAllMovies = async () => {
  const res = await axios.get("/movie").catch((err) => console.log(err));

  if (res.status !== 200) {
    return console.log("No Data");
  }

  const data = await res.data;
  return data;
};
```

Hình 4.1.5. getAllMovies

Gửi yêu cầu đặt phim:

```
export const newBooking = async (data) => {
  const res = await axios
    .post("/booking", {
      movie: data.movie,
      seatNumber: data.seatNumber,
      hour: data.hour,
      date: data.date,
      user: localStorage.getItem("userId"),
    })
    .catch((err) => console.log(err));

  if (res.status !== 201) {
    return console.log("Unexpected Error");
  }
  const resData = await res.data;
  return resData;
};
```

Hình 4.1.6. newBooking

4.1.4.Redux và React Hooks:

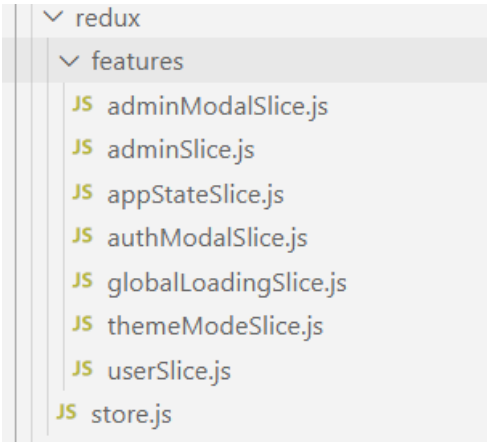
Như đã nói ở phần trên, Redux là một thư viện quản lý trạng thái (state) cho phép các Component truy cập state mà không phải chuyển từ component này sang component khác. “state” ở đây có thể hiểu là các dữ liệu thay đổi khi ứng dụng web hoạt động.

React Hooks cũng là một công cụ để quản lý trạng thái, giúp lập trình viên sử dụng các tính năng của React trong các hàm component chức năng (functional components) thay vì chỉ trong các class component.

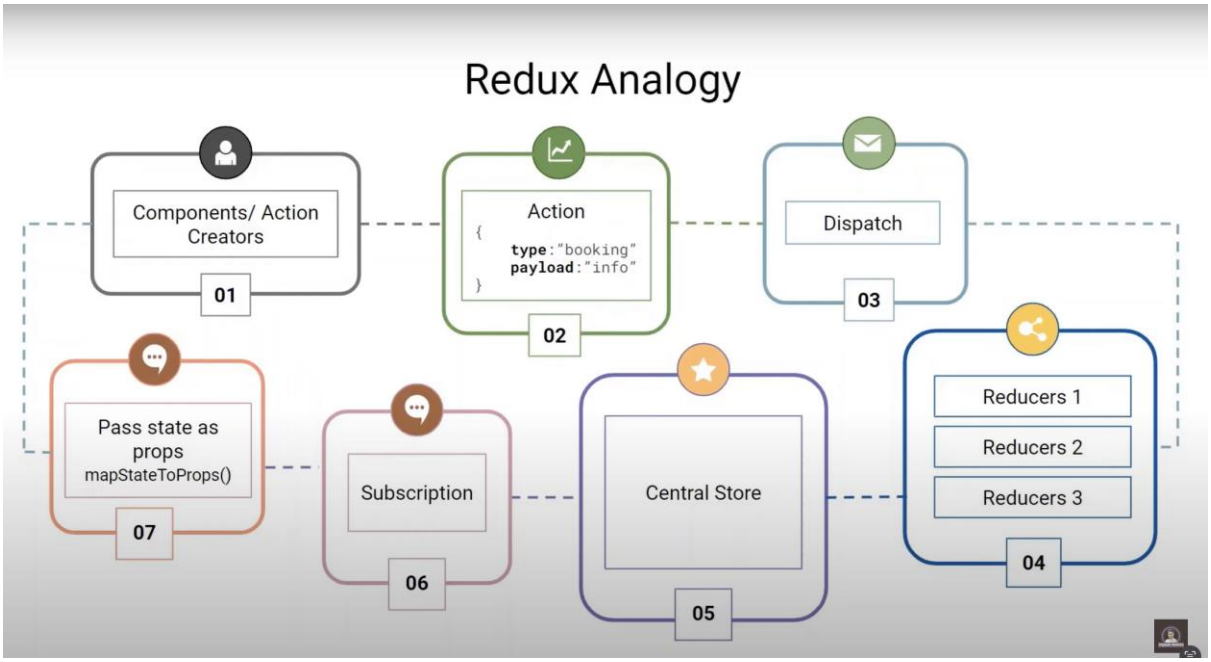
Trong ứng dụng này web, ta sử dụng cả hai tính năng trên để quản lý state cho ứng dụng web.

4.1.4.1. Redux

Cách redux hoạt động:



Hình 4.1.7. store.js và các slice



Hình 4.1.8. Cơ chế hoạt động của redux

Trong Redux, trạng thái của ứng dụng được lưu trữ trong một đối tượng gọi là "store". Store là một kho chứa trạng thái duy nhất và không thể thay đổi trực tiếp từ bất kỳ nơi nào. Thay vào đó, để thay đổi trạng thái trong store, Redux theo một quy trình được gọi là "action dispatch và reducer".

1. *Components* (Các thành phần): Components trong Redux là các phần tử giao diện người dùng (UI) của ứng dụng. Chúng có thể là các thành phần React hoặc bất kỳ thành phần nào khác.

2. *Actions* (Các hành động): Actions là các đối tượng JavaScript đơn giản, mô tả sự kiện xảy ra trong ứng dụng. Ví dụ, một action có thể là "Thêm một sản phẩm vào giỏ hàng" hoặc "Đăng nhập người dùng". Mỗi action phải có một thuộc tính bắt buộc gọi là "type" để xác định loại hành động.

3. *Action Dispatch* (Phân phát hành động): Khi một thành phần (component) muốn thay đổi trạng thái trong store, nó sẽ gửi một action tới Redux bằng cách gọi

một hàm gọi là "dispatch". Hàm dispatch sẽ nhận một action làm đối số và chuyển nó đến reducer.

4. *Reducer* (Trình xử lý): Reducer là một hàm JavaScript nhận vào trạng thái hiện tại và một action, sau đó trả về trạng thái mới. Reducer xác định cách thay đổi trạng thái dựa trên loại hành động được gửi đến. Mỗi lần một action được dispatch, reducer sẽ được gọi để xử lý hành động và trả về một trạng thái mới.

5. *Store* (Kho chứa): Store là nơi lưu trữ trạng thái của ứng dụng và quản lý việc gọi reducer để cập nhật trạng thái. Bạn có thể truy cập vào trạng thái hiện tại trong store và cập nhật nó thông qua các phương thức của store như getState, dispatch, và subscribe.

Để thay đổi trạng thái trong Redux, một thành phần gửi một action thông qua hàm dispatch. Action được chuyển đến reducer, nơi mà trạng thái hiện tại được xử lý và cập nhật. Sau đó, Redux thông báo cho các thành phần khác biết rằng trạng thái đã thay đổi, và các thành phần có thể truy cập vào trạng thái mới từ store để cập nhật giao diện người dùng.

Trong ứng dụng web trên, để xây dựng store, ta sử dụng khái niệm *slice* Redux Toolkit để tạo ra một phần của store, bao gồm reducer và các action tương ứng. Nó giúp tổ chức mã nguồn Redux một cách dễ đọc, dễ quản lý và tái sử dụng.

Để tạo một slice, ta sử dụng hàm createSlice từ Redux Toolkit. Hàm này nhận vào một đối tượng mô tả các trạng thái ban đầu, các reducer và các action tương ứng.

userSlice.js

client > src > redux > features > userSlice.js > userSlice > reducers > setUser

```
1   import { createSlice } from "@reduxjs/toolkit";
2
3   export const userSlice = createSlice({
4     name: "User",
5     initialState: {
6       user: null
7     },
8     reducers: {
9       setUser: (state, action) => {
10        if (action.payload === null) {
11          localStorage.removeItem("actkn");
12          localStorage.removeItem("userId");
13
14        } else {
15          if (action.payload.token)
16          {
17            localStorage.setItem("actkn", action.payload.token);
18            localStorage.setItem("userId", action.payload.id);
19          }
20        }
21
22        state.user = action.payload;
23      }
24    }
25  });
26
27  export const {
28    setUser
29  } = userSlice.actions;
30
31  export default userSlice.reducer;
```

Hình 4.1.9. userSlice.js

4.1.4.2. ReactHooks

Một vài Hooks quan trọng được sử dụng trong ứng dụng web là *useEffect, useDispatch, useSelector, useState.*

a. useEffect

useEffect là một hook trong React được sử dụng để thực hiện các thao tác phụ (side effects) sau mỗi lần render của component. Thao tác phụ có thể bao gồm gọi API, tương tác với DOM, đăng ký và hủy đăng ký các sự kiện, và nhiều hơn nữa.

Cú pháp của useEffect như sau:

```
useEffect(() => {
  // Thực hiện các thao tác phụ ở đây
```

```
    return () => {
      // Thực hiện các thao tác dọn dẹp (cleanup) (tuỳ
chọn)
    };
  }, [dependency]);
```

Hàm callback truyền vào useEffect sẽ được thực thi sau mỗi lần component được render.

Nếu không có dependency truyền vào (không có mảng dependency), callback sẽ được thực thi sau mỗi lần component được render lại.

Nếu có một mảng dependency truyền vào, callback sẽ chỉ được thực thi khi giá trị của các dependency thay đổi.

Nếu callback trả về một hàm, hàm đó sẽ được thực thi khi component bị hủy (unmount) hoặc dependency thay đổi trước khi callback mới được thực thi.

```
useEffect(() => {
  const getFavorites = async () => {
    dispatch(setGlobalLoading(true));
    await getUserFavorite()
      .then((res) => {
        setCount(res.favorites.length);
        setFavorites([...res.favorites]);
        setFilteredFavorites([...res.favorites].splice(0, skip));
      })
      .catch((err) => console.log(err));
    dispatch(setGlobalLoading(false));
  };

  getFavorites();
}, []);
```

Hình 4.1.10. useEffect()

Trong ví dụ trên, ta sử dụng *useEffect* để lấy danh sách người dùng cho lần đầu render bởi mảng dependency là một mảng rỗng. Điều này có nghĩa là callback trong useEffect chỉ được thực thi một lần sau khi component được render lần đầu tiên và không có dependency nào được theo dõi để kích hoạt lại callback.

b. useState

useState là một hook trong React được sử dụng để khởi tạo và quản lý trạng thái (*state*) trong *functional components*.

Cú pháp của useState như sau:

```
const [state, setState] = useState(initialState);
```

`state` là biến chứa giá trị của trạng thái.

`setState` là hàm được sử dụng để cập nhật giá trị của trạng thái.

`initialState` là giá trị ban đầu của trạng thái.

```
const BookingList = () => {
  const [bookings, setBookings] = useState([]);
  const [activeBookings, setActiveBookings] = useState([]);
  const [expiredBookings, setExpiredBookings] = useState([]);
  const [filteredActive, setFilteredActive] = useState([]);
  const [filteredExpired, setFilteredExpired] = useState([]);
  const [pageActive, setPageActive] = useState(1);
  const [pageExpired, setPageExpired] = useState(1);
```

Hình 4.1.11. useState()

Trong ví dụ trên, ta khởi tạo state bookings :

`const [bookings, setBookings] = useState([]);`

Sau đó sử dụng *setBookings* để cập nhật state của *bookings* bằng hàm *setBookings* với giá trị truyền vào là *bookings* trả về từ hàm *getUserBooking* trong *api-helper.js*.

c.useSelector

useSelector là một hook trong Redux Toolkit và React Redux được sử dụng để truy cập và lấy state từ Redux store trong functional components.

Cú pháp của useSelector như sau:

`const result = useSelector(selectorFunction);`

`selectorFunction` là một hàm selector được sử dụng để truy xuất và lấy dữ liệu từ Redux store.

`result` là kết quả trả về từ `selectorFunction`, chứa giá trị của state mà chúng ta quan tâm.

Mỗi khi state trong Redux store thay đổi, useSelector sẽ tự động kích hoạt việc render lại component và cập nhật giao diện với giá trị mới của `result`.

```
const App = () => {
  const { themeMode } = useSelector((state) => state.themeMode);
  return (
```

Hình 4.1.12. useSelector()

Ví dụ trên sử dụng *useSelector* để truy cập tới state *themeMode* trong App.js. Mỗi khi state *themeMode* thay đổi (dark hoặc light), trang web sẽ được render lại với theme sáng hoặc tối.

d. useDispatch:

useDispatch là một hook trong Redux Toolkit và React Redux được sử dụng để gửi các actions đến Redux store từ functional components.

Cú pháp của useDispatch như sau:

```
const dispatch = useDispatch();
```

dispatch là một hàm được trả về bởi *useDispatch*, được sử dụng để gửi các actions đến Redux store.

Khi *dispatch* được gọi, Redux store sẽ chạy *reducer* tương ứng và cập nhật state. Sau khi state thay đổi, React Redux sẽ tự động kích hoạt việc render lại component và cập nhật giao diện với state mới.

```
const HomePage = () => {
  const dispatch = useDispatch();
  const [movies, setMovies] = useState([]);
  useEffect(() => {
    const getMovies = async () => {
      window.scrollTo(0, 0);
      dispatch(setGlobalLoading(true));
      await getAllMovies()
        .then((data) => {setMovies(data.movies);
        localStorage.setItem("movies", data.movies);
        })
        .catch((err) => console.log(err));
      dispatch(setGlobalLoading(false));
    };
    getMovies();
  }, []);
```

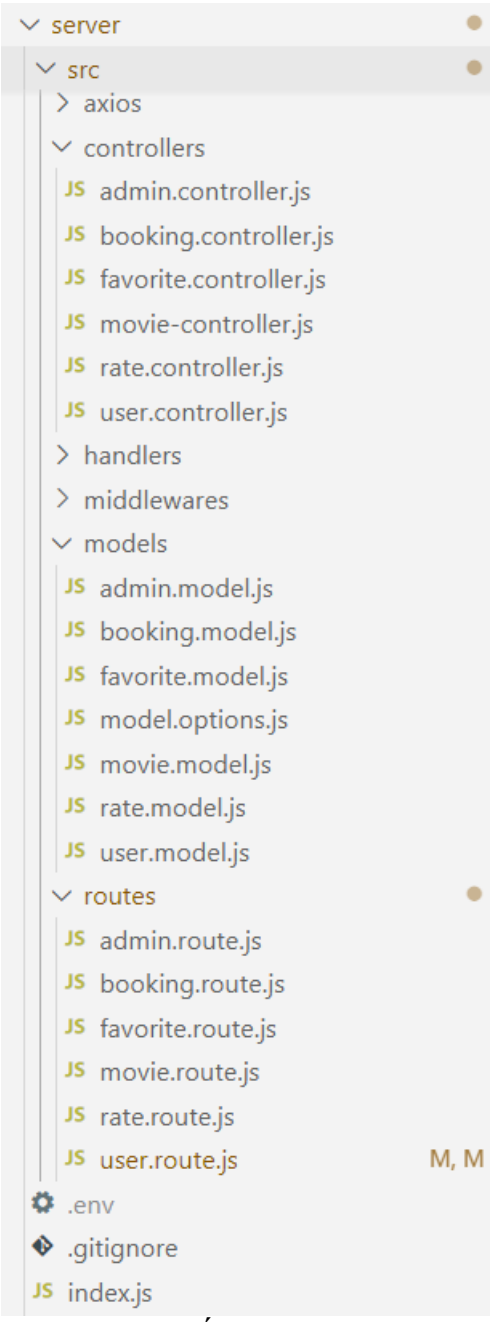
Hình 4.1.13. useDispatch()

Trong ví dụ trên, *dispatch* gửi action *setThemeMode* đê thay đổi state *themeMode* trong store bằng với giá trị cua *theme*.

4.2.Backend (server):

Phần Backend của website sử dụng *ExpressJs* để tạo API và thiết lập cấu trúc và ràng buộc quy tắc cho các documents trong cơ sở dữ liệu *MongoDB*.

Cấu trúc Backend:



Hình 4.2.1. Cấu trúc thư mục backend

Trong *index.js*, thư viện ODM *mongoose* được sử dụng để kết nối ứng dụng với MongoDB.

Đồng thời, quy định các đường dẫn URL và xử lý các yêu cầu theo đường dẫn đó thông qua các Route được định nghĩ trong folder *routes.*

Các Route trong routes sẽ điều hướng các yêu cầu cần được xử lý tới *controller* thích hợp. Sau đó *controller* sẽ trả lại các phản hồi dựa trên kết quả logic của hàm xử lý cho yêu cầu được gọi.

index.js

server > index.js > catch() callback

```
import express from "express";
import cors from "cors";
import mongoose from "mongoose";
import "dotenv/config";
import userRoute from "./src/routes/user.route.js";
import movieRoute from "./src/routes/movie.route.js";
import adminRoute from "./src/routes/admin.route.js";
import bookingRoute from "./src/routes/booking.route.js";
import favoriteRoute from "./src/routes/favorite.route.js";
import rateRoute from "./src/routes/rate.route.js";

const app = express();

app.use(cors());
app.use(express.json());
app.use("/user", userRoute);
app.use("/movie", movieRoute);
app.use("/admin", adminRoute);
app.use("/booking", bookingRoute);
app.use("/favorite", favoriteRoute);
app.use("/rate", rateRoute);

mongoose.connect(`mongodb+srv://lynk64te:${process.env.MONGODB_PASSWORD}@cluster0.zlxkken.mongodb.net/?retryWrites=true&w=majority&appNa
).then(() => {
  console.log("Mongodb connected");
  app.listen(process.env.PORT_ENV, () => {
    console.log(`Server is listening on port ${process.env.PORT_ENV}`);
  });
}).catch((err) => {
  console.log({ err });
});

```

Hình 4.2.2. index.js

4.2.1. Models:

Trong *models*, các documents tham gia vào ứng dụng web được định nghĩa thông qua *mongoose.Schema*. Các đối tượng cụ thể của mỗi model sẽ được lưu trong cơ sở dữ liệu MongoDB.

Admin model gồm 3 thuộc tính, trong đó *addMovies* là một mảng, trong đó các phần tử là các đối tượng thuộc *model Movie* được thêm vào bởi *Admin*.

admin.model.js

server > src > models > admin.model.js > ...

```
import mongoose from "mongoose";
import modelOptions from "./model.options.js";
const adminSchema = new mongoose.Schema({
  email: {
    type: String,
    unique: true,
    required: true,
  },
  password: {
    type: String,
    required: true,
    minLength: 6,
  },
  addedMovies: [
    {
      type: mongoose.Types.ObjectId,
      ref: "Movie",
    },
  ],
});

export default mongoose.model("Admin", adminSchema);

```

Hình 4.2.3. Model Admin

User model gồm 4 thuộc tính, trong đó *bookings* là một mảng mà các phần tử là các đối tượng thuộc model *Bookings*, thể hiện các vé được đặt của người dùng đó.

JS user.model.js ×

server > src > models > JS user.model.js > userSchema > password

```
1   import mongoose from "mongoose";
2   import modelOptions from "./model.options.js";
3   import crypto from "crypto";
4
5   const userSchema = new mongoose.Schema({
6     name: {
7       type: String,
8       required: true
9     },
10    email: {
11      type: String,
12      required: true,
13      unique: true
14    },
15    password: {
16      type: String,
17      required: false,
18      select: false
19    },
20    salt: {
21      type: String,
22      required: false,
23      select: false
24    },
25    bookings: [{ type: mongoose.Types.ObjectId, ref: "Booking" }],
26    wallet: {
27      type: Number,
28      default: 0
29    },
30  }, modelOptions);
31
32  userSchema.methods.setPassword = function (password) {
33    this.salt = crypto.randomBytes(16).toString("hex");
34
35    this.password = crypto.pbkdf2Sync(
36      password,
37      this.salt,
```

Hình 4.2.4. Model User

Movie model gồm các thuộc tính thể hiện thông tin về bộ phim, trong đó *bookings* là mảng chứa các đối tượng thuộc model *Bookings*.

movie.model.js

server > src > models > movie.model.js > ...

```
1  import mongoose from "mongoose";
2  import modelOptions from "./model.options.js";
3  const movieSchema = new mongoose.Schema({
4    title: {
5      type: String,
6      required: true,
7    },
8    description: {
9      type: String,
10     required: true,
11   },
12   genres: [{
13     type: String,
14     required: true
15   }],
16   actors: [{ type: String, required: true }],
17   releaseDate: {
18     type: Date,
19     required: true,
20   },
21   posterUrl: {
22     type: String,
23     required: true,
24   },
25   backgroundUrl: {
26     type: String,
27     required: true,
28   },
29   averageRating: {
30     type: Number,
31     required: true,
32   },
33   videoUrl: {
34     type: String,
35     required: true,
36   },
37   backdrops: [{ type: String }],
```

Hình 4.2.5. Model Movie

Bookings model gồm năm thuộc tính trong đó gồm user là người đặt và movie là phim được đặt là các đối tượng thuộc model *User* và *Movie*.

booking.model.js

server > src > models > booking.model.js > ...

```
1  import mongoose, { Schema } from "mongoose";
2  import modelOptions from "./model.options.js";
3
4  export default mongoose.model(
5    "Booking",
6    mongoose.Schema({
7      movie: {
8        type: mongoose.Types.ObjectId,
9        ref: "Movie",
10       required: true,
11     },
12     date: {
13       type: Date,
14       required: true,
15     },
16     hour: {
17       type: String,
18       required: true,
19     },
20     seatNumber: {
21       type: Number,
22       required: true,
23     },
24     user: {
25       type: mongoose.Types.ObjectId,
26       ref: "User",
27       required: true,
28     },
29   }, modelOptions)
30 );
```

Hình 4.2.6. Model Bookings

Model Favorite là document chứa danh sách các bộ phim ưa thích của người dùng User.

favorite.model.js

server > src > models > favorite.model.js > ...

```
1  import mongoose, { Schema } from "mongoose";
2  import modelOptions from "./model.options.js";
3
4  export default mongoose.model(
5    "Favorite",
6    mongoose.Schema({
7      user: {
8        type: Schema.Types.ObjectId,
9        ref: "User",
10       required: true
11     },
12     movie: {
13       type: mongoose.Types.ObjectId,
14       ref: "Movie",
15       required: true,
16     },
17   }, modelOptions)
18 );
```

Hình 4.2.7. Model Favorite

Model rate chứa thông tin đánh giá của người dùng về bộ phim

rate.model.js

server > src > models > rate.model.js > ...

```
1  import mongoose, { Schema } from "mongoose";
2  import modelOptions from "./model.options.js";
3
4  export default mongoose.model(
5    "Rate",
6    mongoose.Schema({
7      user: {
8        type: Schema.Types.ObjectId,
9        ref: "User",
10       required: true
11     },
12     rate: {
13       type: Number,
14       required: true,
15     },
16     movie: {
17       type: mongoose.Types.ObjectId,
18       ref: "Movie",
19       required: true,
20     },
21   }, modelOptions)
22 );
```

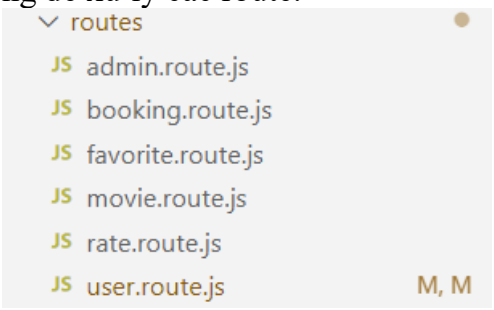
Hình 4.2.8. Model rate

4.2.2. Routes và Controllers:

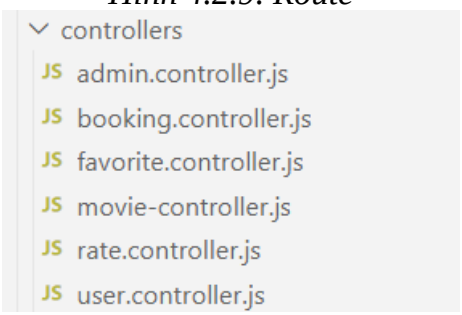
Tương ứng với các document trong database, có các file xử lý route cho từng đối tượng là *admin-route, user-route, booking-route, movie-route, favorite-route và rate-route.*

Các routes cho từng đối tượng được khởi tạo với *express.Router().*

Express Router cho phép định nghĩa các routes và cách xử lý tương ứng cho từng route. Ở đây ta sử dụng các hàm điều khiển (*controller functions*) trong folder *controllers* cho từng đối tượng để xử lý các route.



Hình 4.2.9. Route



Hình 4.2.10.Controller

Ví dụ với user-route và user-controller

user.route.js M, M

server > src > routes > user.route.js > ...

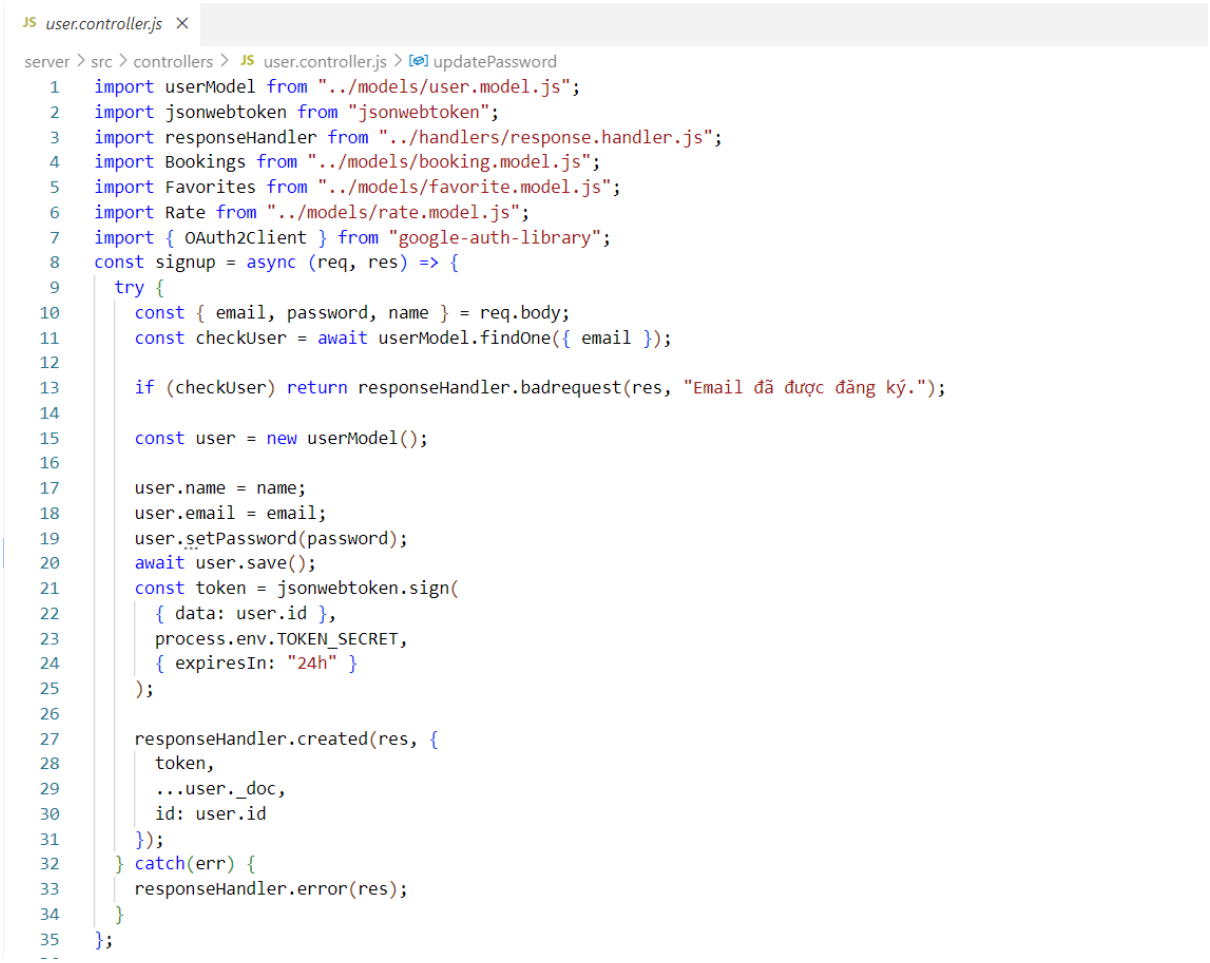
```
import express from "express";
import { body } from "express-validator";
import userController from "../controllers/user.controller.js";
import requestHandler from "../handlers/request.handler.js";
import userModel from "../models/user.model.js";
import tokenMiddleware from "../middlewares/token.middleware.js";

const router = express.Router();

router.post(
  "/signup",
  body("email")
    .exists().withMessage("Hãy nhập email.")
    .custom(async value => {
      const user = await userModel.findOne({ email: value });
      if (user) return Promise.reject("Email đã được đăng ký.");
    }),
  body("password")
    .exists().withMessage("Hãy nhập mật khẩu.")
    .isLength({ min: 8 }).withMessage("Mật khẩu phải có ít nhất 8 kí tự."),
  body("confirmPassword")
    .exists().withMessage("Hãy nhập mật khẩu xác nhận.")
    .isLength({ min: 8 }).withMessage("Mật khẩu xác nhận phải có ít nhất 8 kí tự.")
    .custom((value, { req }) => {
      if (value !== req.body.password) throw new Error("Mật khẩu xác nhận không khớp.");
      return true;
    }),
  body("name")
    .exists().withMessage("Hãy nhập tên người dùng.")
    .isLength({ min: 4 }).withMessage("Tên người dùng phải có ít nhất 4 kí tự."),
  requestHandler.validate,
  userController.signup
);

router.post(
  "/signin",
  body("email")
```

Hình 4.2.11. user-route



Hình 4.2.12. user-controller

Ví dụ với *user-routes* và yêu cầu */signup*, hàm xử lý *signup* được định nghĩa trong file *user-controller* sẽ được gọi để xử lý quá trình đăng ký tài khoản cho người dùng.

Trong đó, các hàm xử lý yêu cầu và phản hồi sẽ được quy định trong các file *requesthandler.js* và *respondhandler.js.*

Khi *user* đặt vé cho một bộ phim, hàm controller *newBooking* sẽ được gọi.

```
export const newBooking = async (req, res) => {
  const { movie, date, hour, seatNumber, user } = req.body;

  try {
    const existingMovie = await Movie.findById(movie);
    const existingUser = await User.findById(user);

    if (!existingMovie) {
      return res.status(404).json({ message: "Movie Not Found With Given ID" });
    }
    if (!existingUser) {
      return res.status(404).json({ message: "User not found with given ID" });
    }

    const booking = new Bookings({
      movie,
      date: new Date(date),
      hour,
      seatNumber,
      user,
    });

    const session = await mongoose.startSession();
    session.startTransaction();

    existingUser.bookings.push(booking);
    existingMovie.bookings.push(booking);

    await existingUser.save({ session });
    await existingMovie.save({ session });
    await booking.save({ session });

    await session.commitTransaction();

    return res.status(201).json({ booking });
  } catch (err) {
    console.error(err);
```

Hình 4.2.13. Hàm xử lý đặt vé từ người dùng

Trong hàm này, sau khi khởi tạo một đối tượng thuộc model Bookings, một MongoDB session sẽ được khởi tạo. Session này cho phép gộp nhiều thao tác trên database trong một lần hành động.

Tại đây, thuộc tính bookings (mảng gồm các đối tượng thuộc model Bookings) của hai đối tượng *user* và *movie* được đặt sẽ được thêm vào đối tượng Bookings mới được tạo:

```
existingUser.bookings.push(booking);
existingMovie.bookings.push(booking);
```

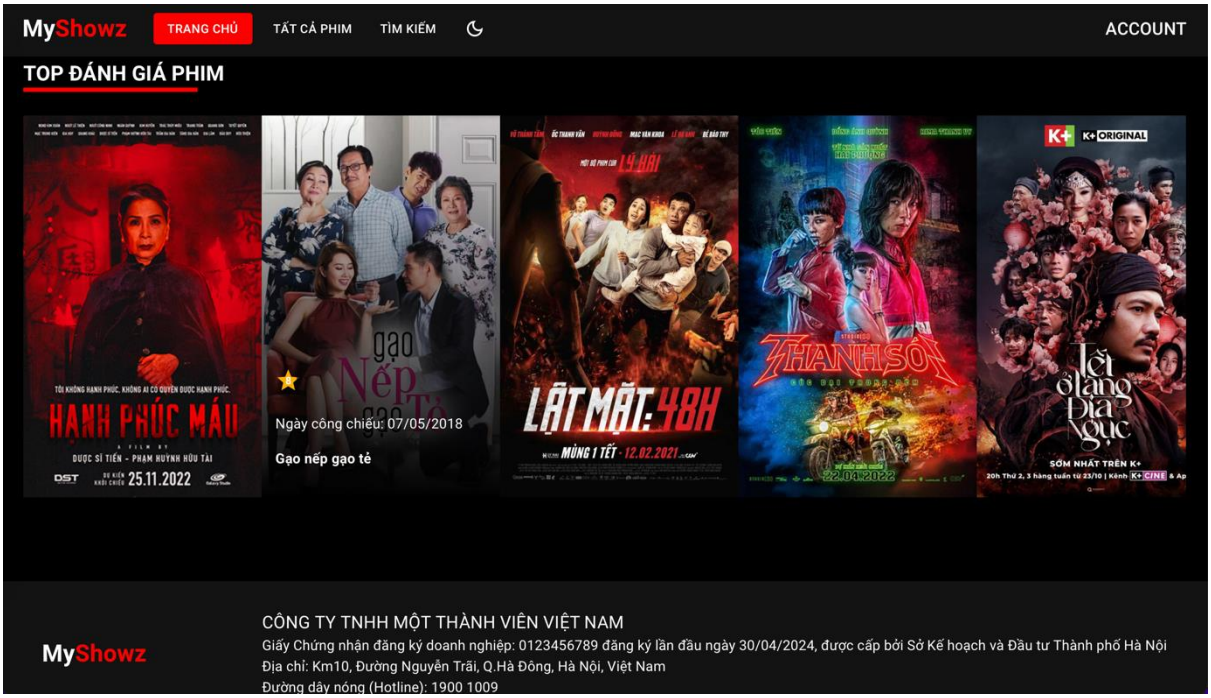
Sau đó các thay đổi này sẽ được lưu vào cùng session đã được khởi tạo và được commit trong một lần.

```
await existingUser.save({session});
await existingUser.save({session});
await booking.save({session});
```

```
session.commitTransaction();
```

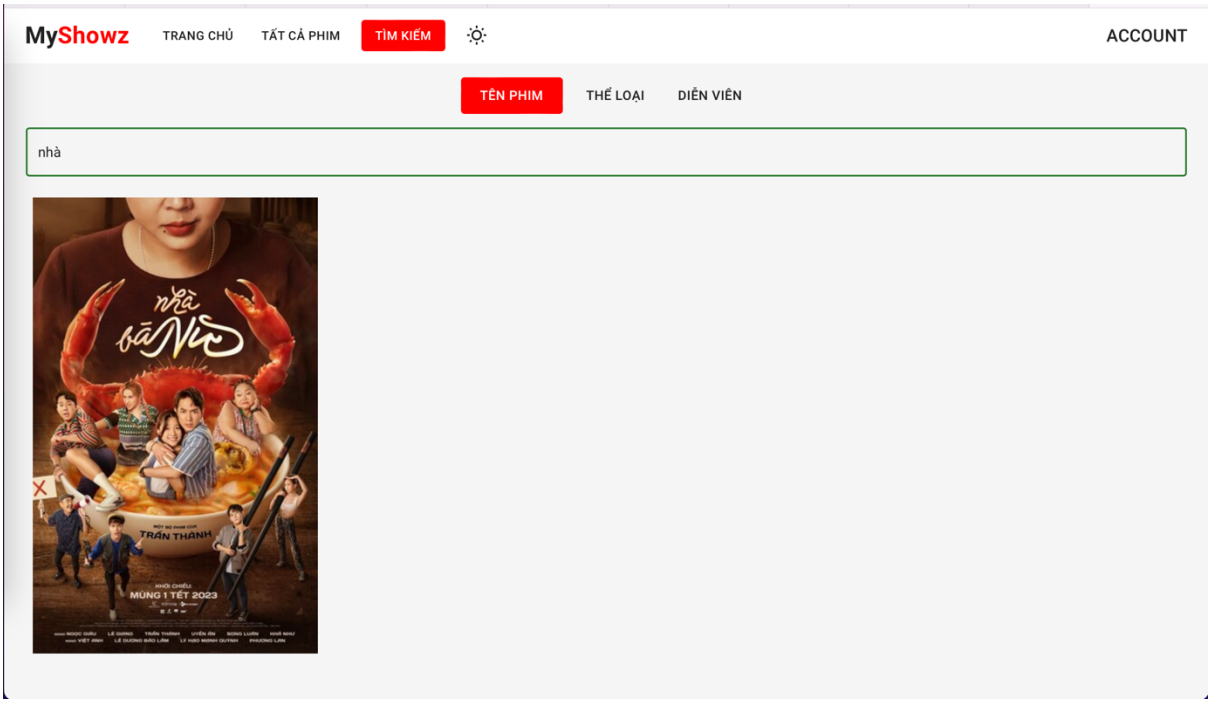
Các phần khác trong backend (server) cũng hoạt động theo cách tương tự.

4.3.Giao diện và chức năng của trang web:

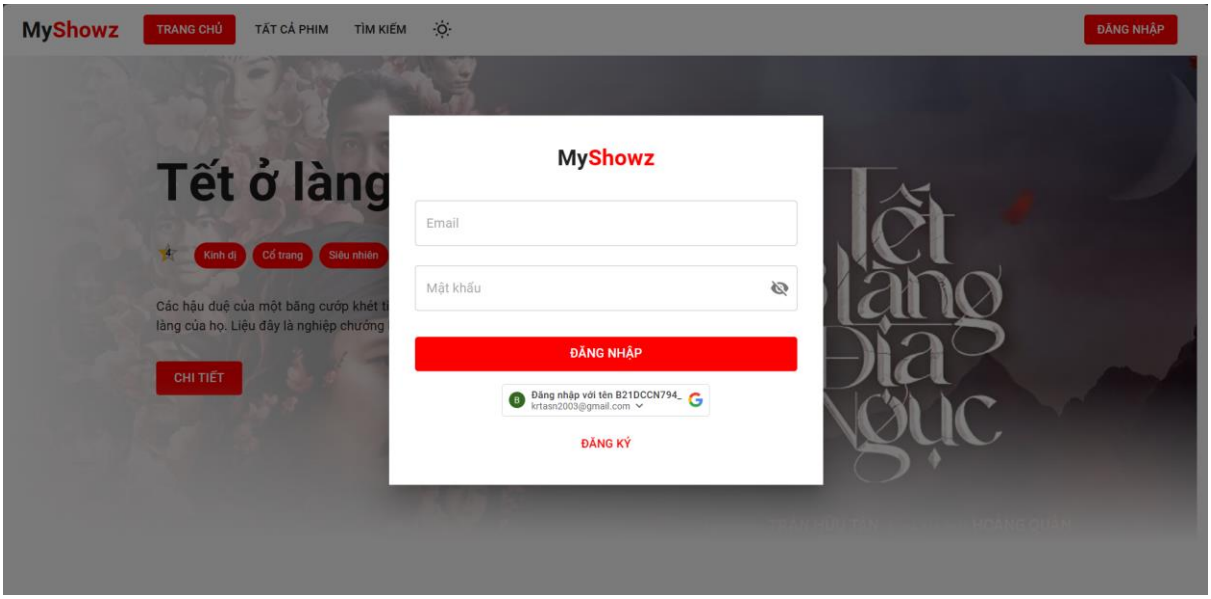


Hình 4.3.1. Giao diện trang chủ (nền tối)

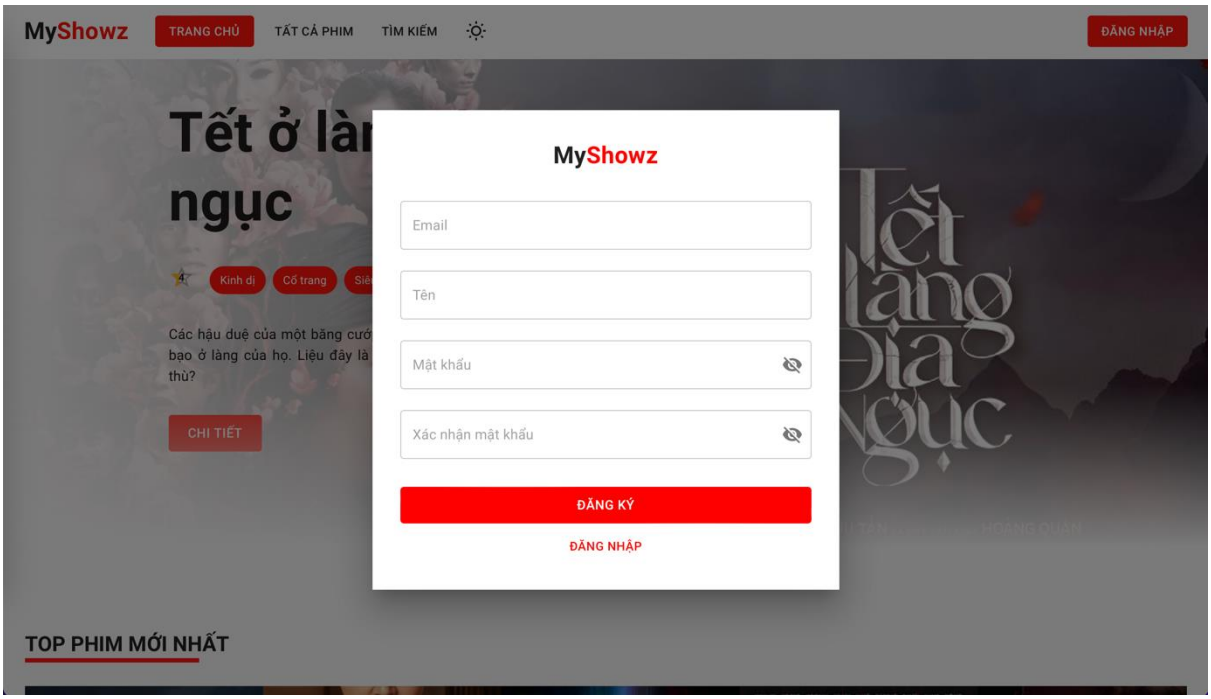
Hình 4.3.2. Giao diện với nền sáng



Hình 4.3.3. Giao diện tìm kiếm

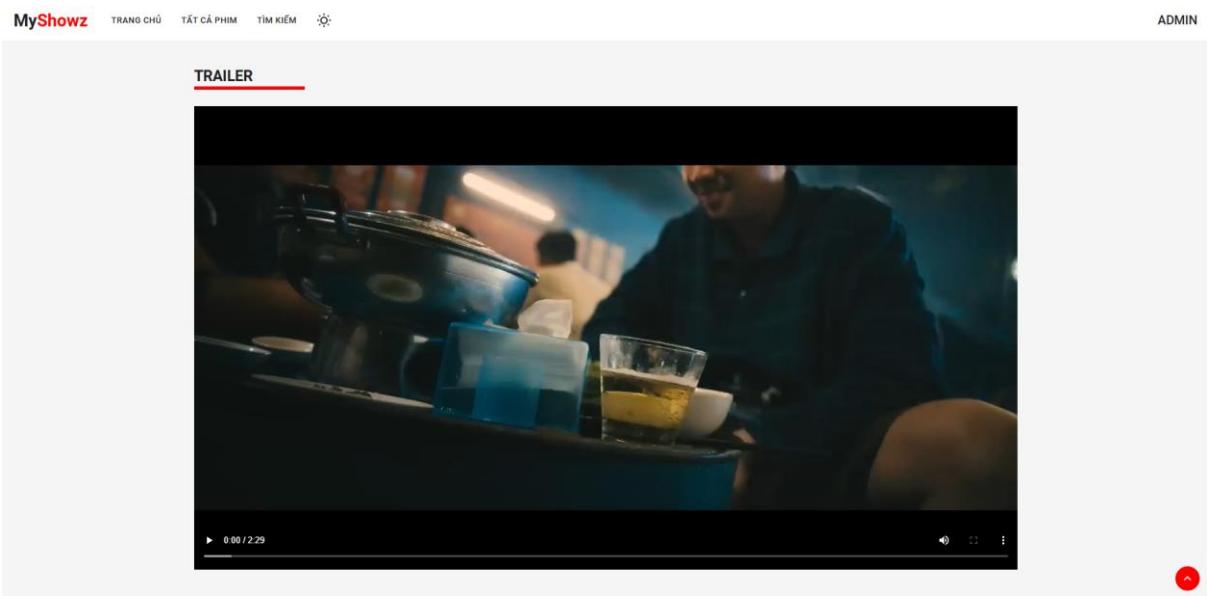


Hình 4.3.4. Giao diện đăng nhập

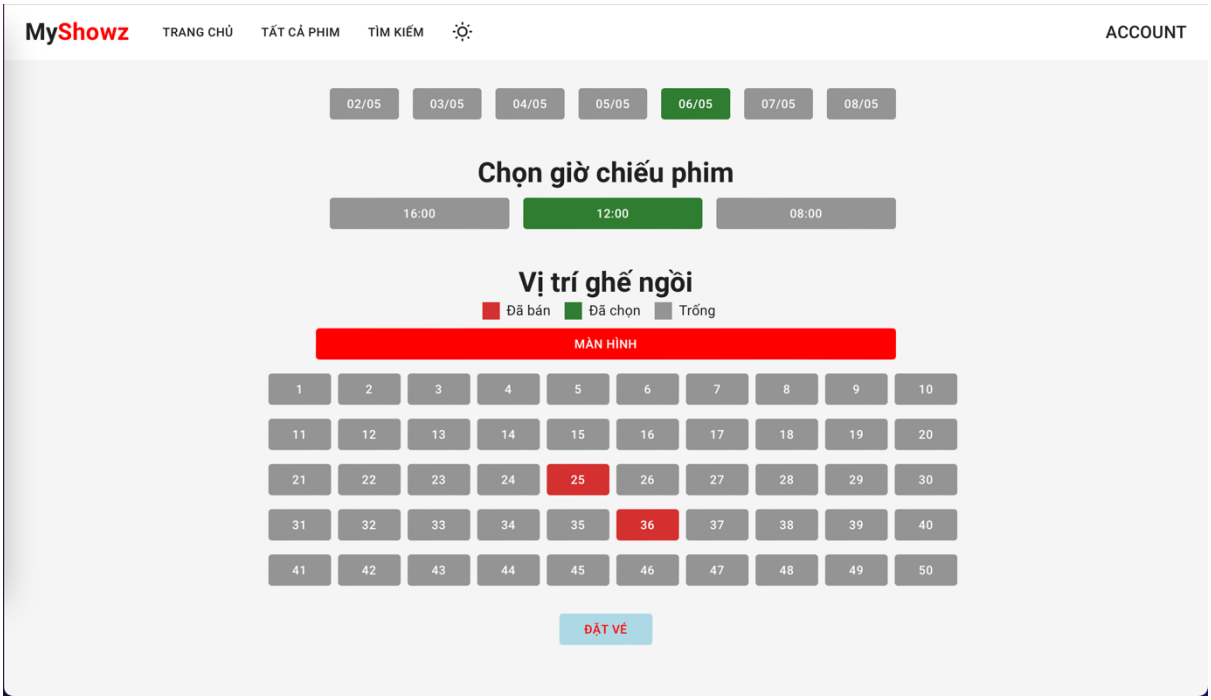


Hình 4.3.5. Giao diện đăng ký

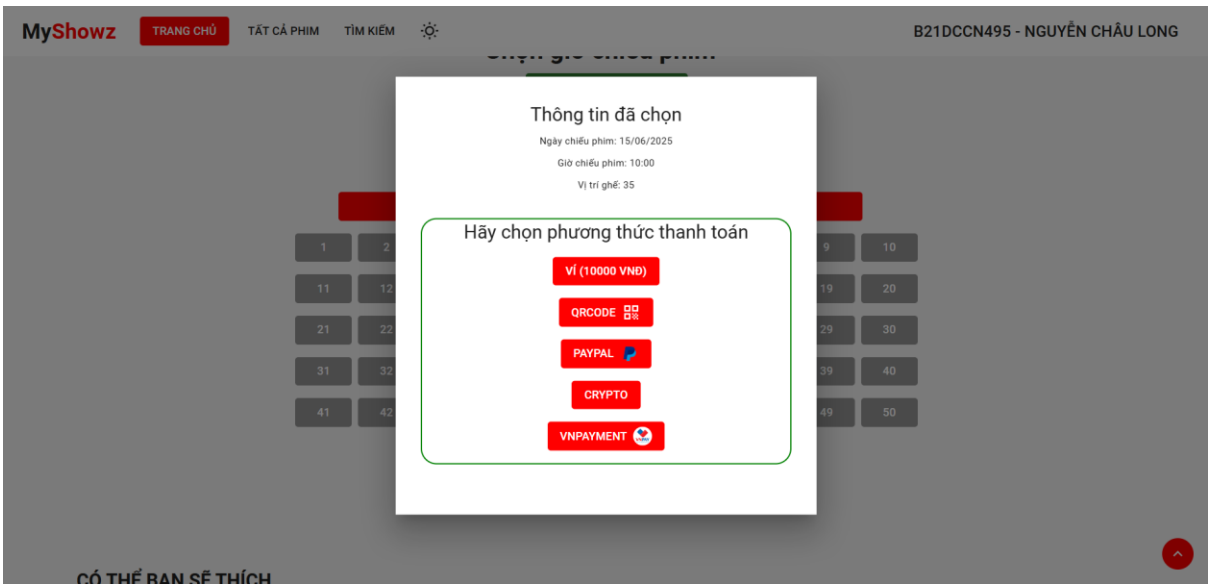
Hình 4.3.6. Giao diện chi tiết phim



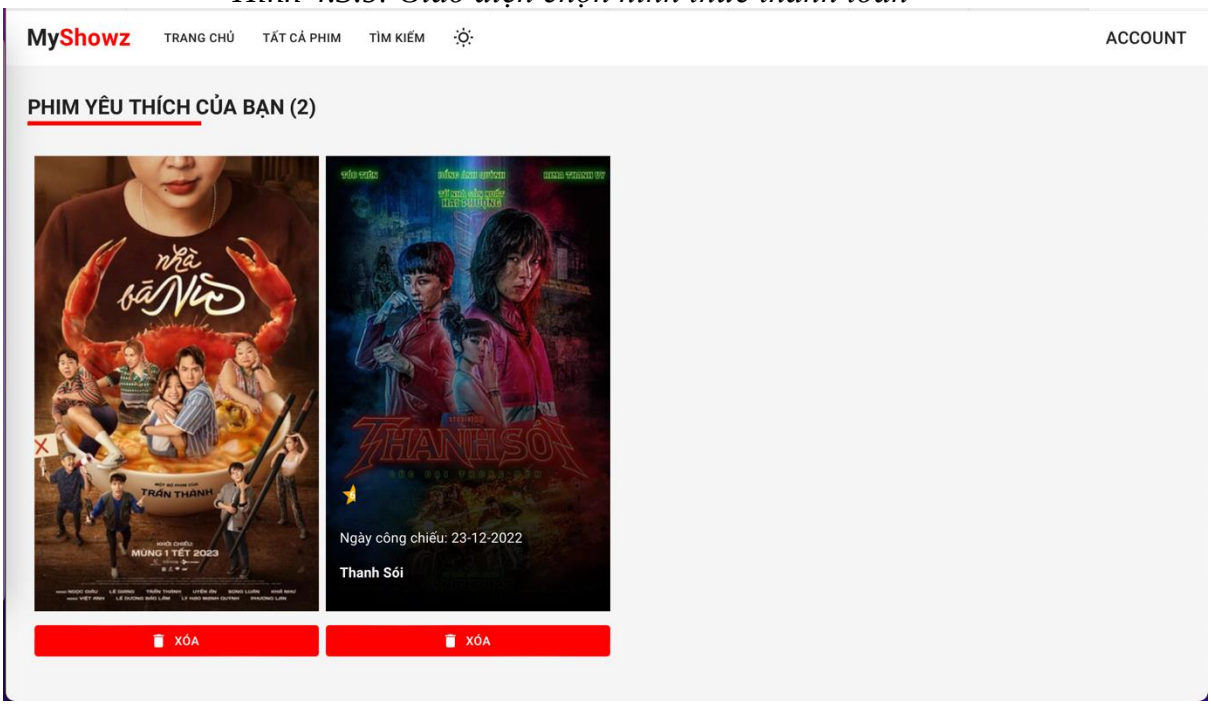
Hình 4.3.7. Giao diện chi tiết phim (tiếp)



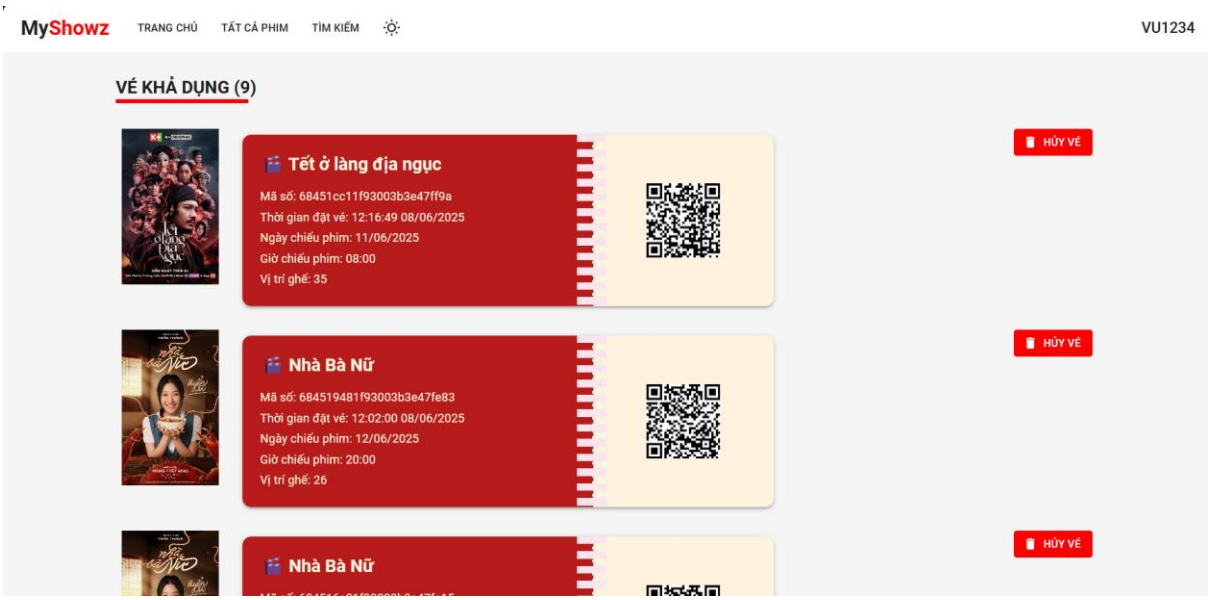
Hình 4.3.8. Giao diện đặt vé



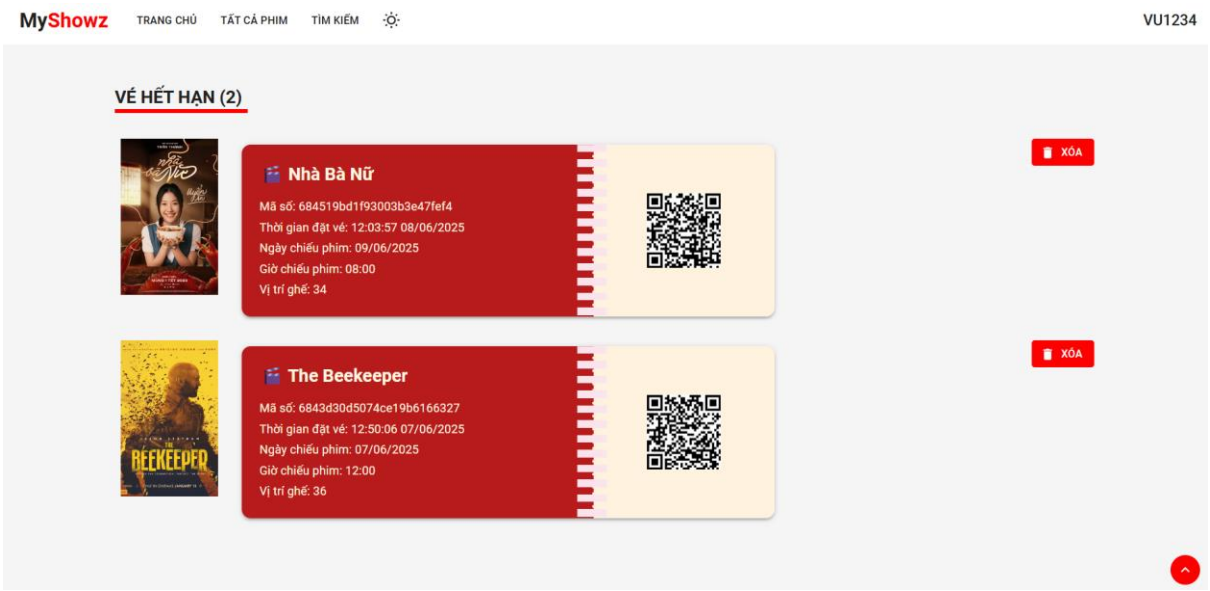
Hình 4.3.9. Giao diện chọn hình thức thanh toán



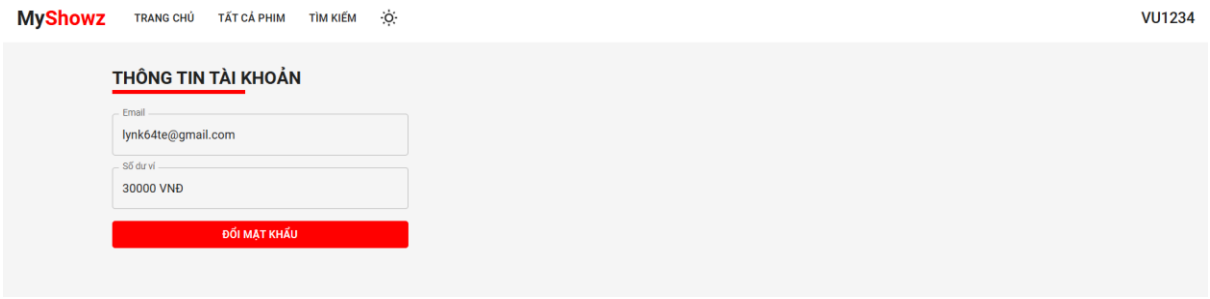
Hình 4.3.10. Danh sách phim yêu thích của user



Hình 4.3.11. Danh sách vé đã đặt



Hình 4.3.12. Danh sách vé đã đặt(tiếp)



Hình 4.3.13.Giao diện thông tin tài khoản

V.Kết luận và định hướng phát triển

5.1.Kết luận

Qua quá trình nghiên cứu và triển khai, nhóm đã hoàn thành xây dựng hệ thống Website bán vé xem phim trực tuyến với các tính năng cơ bản và cần thiết cho cả người dùng (khách hàng) và quản trị viên (admin). Ứng dụng được phát triển theo mô hình tách biệt frontend và backend:

- **Frontend** sử dụng ReactJS kết hợp Redux để quản lý trạng thái và Material UI để thiết kế giao diện, mang lại trải nghiệm trực quan, mượt mà cho người dùng.
- **Backend** sử dụng Node.js cùng với Express để xây dựng các API và MongoDB để lưu trữ dữ liệu một cách hiệu quả.
- Hệ thống cho phép người dùng đăng nhập, tìm kiếm phim, đặt vé, xem lịch sử vé, thêm phim yêu thích và đánh giá phim.
- Quản trị viên có thể thêm, sửa thông tin phim, giúp dễ dàng quản lý nội dung hiển thị trên hệ thống.
- Ngoài ra, dự án đã tích hợp đa dạng các phương thức thanh toán như VNPAY, PayPal, thanh toán bằng ví nội bộ và xác minh bằng ví crypto, giúp người dùng có nhiều lựa chọn linh hoạt khi thanh toán.

Thông qua quá trình thực hiện, nhóm đã củng cố kiến thức về phát triển hệ thống web, kỹ năng làm việc nhóm, cũng như hiểu rõ hơn về quy trình phát triển phần mềm từ thiết kế, lập trình, đến kiểm thử.

5.2.Định hướng phát triển

Mặc dù hệ thống đã đáp ứng được yêu cầu cơ bản, nhưng vẫn còn nhiều điểm có thể cải thiện và mở rộng trong tương lai:

1. **Tăng cường bảo mật**:
    - Thêm các lớp xác thực bảo vệ API.
2. **Phát triển tính năng nâng cao**:
    - Gửi email thông báo khi đặt vé thành công.
    - Tích hợp hệ thống đánh giá phim bằng hình ảnh và bình luận.
3. **Mở rộng tích hợp thanh toán**:
    - Tích hợp thêm các cổng thanh toán quốc tế khác như Stripe, Momo,...
    - Bổ sung khả năng theo dõi trạng thái giao dịch theo thời gian thực.
4. **Kiểm thử người dùng**:
    - Thu thập phản hồi từ người dùng để cải tiến hệ thống.
5. **Quản lý ghế và rạp nâng cao**:
    - Cho phép tùy chỉnh sơ đồ ghế ngồi theo từng rạp.
    - Hỗ trợ nhiều rạp và phòng chiếu khác nhau.