MERN 10 - Project Part 6 -

Buidling the Home page which will show Currently Running Movies

Buidling the Shows API

Partners can add shows in their approved theatres

Adding Shows in Theatres

## title:Building the Home page for Movies

In your Home page i.e pages->Home.js add this code

import { useEffect, useState } from "react";

import { HideLoading, ShowLoading } from "../../redux/loaderSlice";

import { useDispatch } from "react-redux";

import { getAllMovies } from "../../api/movie";

import { message, Row, Col, Input } from "antd";

import { useNavigate } from "react-router-dom";

import { SearchOutlined } from "@ant-design/icons";

import moment from "moment";

State variables

const Home = () => {

const [movies, setMovies] = useState(null);

const [searchText, setSearchText] = useState("");

const dispatch = useDispatch();

const navigate = useNavigate();

}

Method to get movie data

const getData = async () => {

try {

dispatch(ShowLoading());

const response = await getAllMovies();

if (response.success) {

setMovies(response.data);

} else {

message.error(response.message);

}

dispatch(HideLoading());

} catch (err) {

message.error(err.message);

dispatch(HideLoading());

}

};

useEffect(() => {

getData();

}, []);

Handle search

const handleSearch = (e) => {

setSearchText(e.target.value);

console.log(searchText);

};

Markup

return (

<>

<Row className="justify-content-center w-100">

<Col xs={{ span: 24 }} lg={{ span: 12 }}>

<Input

placeholder="Type here to search for movies"

onChange={handleSearch}

prefix={<SearchOutlined />}

/>

<br />

<br />

<br />

</Col>

</Row>

<Row

className="justify-content-center"

gutter={{

xs: 8,

sm: 16,

md: 24,

lg: 32,

}}

>

{movies &&

movies

.filter((movie) =>

movie.title.toLowerCase().includes(searchText.toLowerCase())

)

.map((movie) => (

<Col

className="gutter-row mb-5"

key={movie.\_id}

span={{

xs: 24,

sm: 24,

md: 12,

lg: 10,

}}

>

<div className="text-center">

<img

onClick={() => {

navigate(

`/movie/${movie.\_id}?date=${moment().format(

"YYYY-MM-DD"

)}`

);

}}

className="cursor-pointer"

src={movie.poster}

alt="Movie Poster"

width={200}

style={{ borderRadius: "8px" }}

/>

<h3

onClick={() => {

navigate(

`/movie/${movie.\_id}?date=${moment().format(

"YYYY-MM-DD"

)}`

);

}}

className="cursor-pointer"

>

{movie.title}

</h3>

</div>

</Col>

))}

</Row>

</>

);

};

export default Home;

A Row and Col structure from Ant Design is used to create a responsive layout.

An Input field is provided for searching movies, with the SearchOutlined icon as a prefix.

onChange={handleSearch} binds the search input to the handleSearch function.

## title:Partners can add shows in their approved theatres

Now if you look in your Partner's Page , when the admin approves the theatre , the property isActive is set to true that means the theatre now is Active and can run shows , whenever a theatre will be approved we will show a buttom of Add shows beside it

You can see we have already added this in the TheatreList of the partner

Add one component as ShowModel.js which wil just be a basic component for now

Partner -> ShowModal.js

import React, { useEffect } from "react";

function ShowModal() {

return (

<>

<h1>Show Modal</h1>

</>

);

}

export default ShowModal;

Add below code in TheatreList to open ShowModal

import { ShowLoading, HideLoading } from "../../redux/loaderSlice";

import ShowModal from "./ShowModal";

const [theatres, setTheatres] = useState([]);

const [isShowModalOpen, setIsShowModalOpen] = useState(false);

const dispatch = useDispatch();

Add this at the last in the markup

{isDeleteModalOpen && (

<DeleteTheatreModal

isDeleteModalOpen={isDeleteModalOpen}

selectedTheatre={selectedTheatre}

setIsDeleteModalOpen={setIsDeleteModalOpen}

setSelectedTheatre={setSelectedTheatre}

getData={getData}

/>

)}

{isShowModalOpen && (

<ShowModal

isShowModalOpen={isShowModalOpen}

setIsShowModalOpen={setIsShowModalOpen}

selectedTheatre={selectedTheatre}

/>

)}

</>

Add this in the column config definition

{

title: "Action",

dataIndex: "action",

render: (text, data) => {

return (

<div className="d-flex align-items-center gap-10">

<Button

onClick={() => {

setIsModalOpen(true);

setFormType("edit");

setSelectedTheatre(data);

}}

>

<EditOutlined />

</Button>

<Button

onClick={() => {

setIsDeleteModalOpen(true);

setSelectedTheatre(data);

}}

>

<DeleteOutlined />

</Button>

{data.isActive && (

<Button

onClick={() => {

setIsShowModalOpen(true);

setSelectedTheatre(data);

}}

>

+ Shows

</Button>

)}

</div>

);

},

},

## title:Show API

Now we need to create CRUD api for Show, So first we will create a Show Model

showModel.js

const mongoose = require("mongoose");

const showSchema = new mongoose.Schema(

{

name: {

type: String,

required: true,

},

date: {

type: Date,

required: true,

},

time: {

type: String,

required: true,

},

movie: {

type: mongoose.Schema.Types.ObjectId,

ref: "movies",

required: true,

},

ticketPrice: {

type: Number,

required: true,

},

totalSeats: {

type: Number,

required: true,

},

bookedSeats: {

type: Array,

default: [],

},

theatre: {

type: mongoose.Schema.Types.ObjectId,

ref: "theatre",

required: true,

},

},

{ timestamps: true }

);

module.exports = mongoose.model("shows", showSchema);

## title:Creating Routes for Shows

const router = require("express").Router();

const Show = require("../models/showModel");

// Add Show

router.post("/add-show", async (req, res) => {

try {

const newShow = new Show(req.body);

await newShow.save();

res.send({

success: true,

message: "New show has been added!",

});

// console.log(req.body, res.success, res.message);

} catch (err) {

res.send({

status: false,

message: err.message,

});

}

});

router.post("/delete-show", async (req, res) => {

try {

await Show.findByIdAndDelete(req.body.showId);

res.send({

success: true,

message: "The show has been deleted!",

});

} catch (err) {

res.send({

status: false,

message: err.message,

});

}

});

// Update mshow

router.put("/update-show", async (req, res) => {

try {

await Show.findByIdAndUpdate(req.body.showId, req.body);

res.send({

success: true,

message: "The show has been updated!",

});

} catch (err) {

res.send({

success: false,

message: err.message,

});

}

});

router.post("/get-all-shows-by-theatre", async (req, res) => {

try {

const shows = await Show.find({ theatre: req.body.theatreId }).populate(

"movie"

);

res.send({

success: true,

message: "All shows fetched",

data: shows,

});

// console.log(req.body, res.data, shows)

} catch (err) {

res.send({

success: false,

message: err.message,

});

}

});

// Get all theatres by movie which has some shows

router.post("/get-all-theatres-by-movie", async (req, res) => {

try {

/\*\*

\* This route handles a POST request to /get-all-theatres-by-movie.

It expects the request body (req.body) to contain movie and date.

It retrieves all shows of the specified movie and date from the database (await Show.find({ movie, date }).populate('theatre')).

It then filters out unique theatres and organizes shows under each unique theatre.

It sends a success response with the fetched theatres and associated shows.

If there's an error (e.g., database error), it sends a failure response with an error message.

\*/

const { movie, date } = req.body;

// First get all the shows of the selected date

const shows = await Show.find({ movie, date }).populate("theatre");

// Filter out the unique theatres now

let uniqueTheatres = [];

shows.forEach((show) => {

let isTheatre = uniqueTheatres.find(

(theatre) => theatre.\_id === show.theatre.\_id

);

if (!isTheatre) {

let showsOfThisTheatre = shows.filter(

(showObj) => showObj.theatre.\_id == show.theatre.\_id

);

uniqueTheatres.push({

...show.theatre.\_doc,

shows: showsOfThisTheatre,

});

}

});

res.send({

success: true,

message: "All theatres fetched!",

data: uniqueTheatres,

});

} catch (err) {

res.send({

success: false,

message: err.message,

});

}

});

router.post("/get-show-by-id", async (req, res) => {

try {

const show = await Show.findById(req.body.showId)

.populate("movie")

.populate("theatre");

res.send({

success: true,

message: "Show fetched!",

data: show,

});

} catch (err) {

res.send({

success: false,

message: err.message,

});

}

});

module.exports = router;

/get-all-theatres-by-movie

**Explanation**

The code filters out unique theatres from an array of show objects and groups the shows by their respective theatres. It checks if a theatre has already been added to the uniqueTheatres array, and if not, it adds the theatre along with all its shows. The final list of unique theatres, each containing their respective shows, is then sent in the response. The .\_doc property is used to extract the raw theatre data from Mongoose documents.

/get-show-by-id

It retrieves a specific show by its showId from the database (await Show.findById(req.body.showId).populate('movie').populate('theatre')).

It populates the retrieved show with details of the associated movie and theatre.

## Axios calls corresponding to all the routes

shows.js

import { axiosInstance } from ".";

export const addShow = async (payload) => {

try {

const response = await axiosInstance.post("/api/shows/add-show", payload);

return response.data;

} catch (err) {

return err.message;

}

};

export const updateShow = async (payload) => {

try {

const response = await axiosInstance.put("/api/shows/update-show", payload);

console.log(payload, response);

return response.data;

} catch (err) {

return err.response;

}

};

export const getShowsByTheatre = async (payload) => {

try {

const response = await axiosInstance.post(

"/api/shows/get-all-shows-by-theatre",

payload

);

return response.data;

} catch (err) {

return err.response;

}

};

export const deleteShow = async (payload) => {

try {

const response = await axiosInstance.post(

"/api/shows/delete-show",

payload

);

return response.data;

} catch (err) {

return err.response;

}

};

export const getAllTheatresByMovie = async (payload) => {

try {

const response = await axiosInstance.post(

"/api/shows/get-all-theatres-by-movie",

payload

);

return response.data;

} catch (err) {

return err.response;

}

};

export const getShowById = async (payload) => {

try {

const response = await axiosInstance.post(

"/api/shows/get-show-by-id",

payload

);

return response.data;

} catch (err) {

return err.message;

}

};

## Now we need to create our showModel page and add shows for Theatres

showModal under Partner

import {

Col,

Modal,

Row,

Form,

Input,

Button,

Select,

Table,

message,

} from "antd";

import { ShowLoading, HideLoading } from "../../redux/loaderSlice";

import { useDispatch } from "react-redux";

// import { addTheatre, updateTheatre } from '../../apicalls/theatres';

import {

ArrowLeftOutlined,

EditOutlined,

DeleteOutlined,

} from "@ant-design/icons";

import { useEffect, useState } from "react";

// import { useSelector } from 'react-redux';

import { getAllMovies } from "../../api/movie";

import {

addShow,

deleteShow,

getShowsByTheatre,

updateShow,

} from "../../api/shows";

import moment from "moment";

const ShowModal = ({

isShowModalOpen,

setIsShowModalOpen,

selectedTheatre,

}) => {

const [view, setView] = useState("table");

const [movies, setMovies] = useState(null);

const [selectedMovie, setSelectedMovie] = useState(null);

const [shows, setShows] = useState(null);

const [selectedShow, setSelectedShow] = useState(null);

const dispatch = useDispatch();

const getData = async () => {

try {

dispatch(ShowLoading());

const movieResponse = await getAllMovies();

if (movieResponse.success) {

setMovies(movieResponse.data);

} else {

message.error(movieResponse.message);

}

const showResponse = await getShowsByTheatre({

theatreId: selectedTheatre.\_id,

});

if (showResponse.success) {

setShows(showResponse.data);

} else {

message.error(showResponse.message);

}

dispatch(HideLoading());

} catch (err) {

message.error(err.message);

dispatch(HideLoading());

}

};

const onFinish = async (values) => {

try {

dispatch(ShowLoading());

let response = null;

if (view === "form") {

response = await addShow({ ...values, theatre: selectedTheatre.\_id });

} else {

// console.log(view, selectedTheatre, selectedTheatre.\_id);

response = await updateShow({

...values,

showId: selectedShow.\_id,

theatre: selectedTheatre.\_id,

});

}

if (response.success) {

getData();

message.success(response.message);

setView("table");

} else {

message.error(response.message);

}

dispatch(HideLoading());

} catch (err) {

message.error(err.message);

dispatch(HideLoading());

}

};

const handleCancel = () => {

setIsShowModalOpen(false);

};

const handleDelete = async (showId) => {

try {

dispatch(ShowLoading());

const response = await deleteShow({ showId: showId });

if (response.success) {

message.success(response.message);

getData();

} else {

message.error(response.message);

}

dispatch(HideLoading());

} catch (err) {

message.error(err.message);

dispatch(HideLoading());

}

};

const columns = [

{

title: "Show Name",

dataIndex: "name",

key: "name",

},

{

title: "Show Date",

dataIndex: "date",

render: (text, data) => {

return moment(text).format("MMM Do YYYY");

},

},

{

title: "Show Time",

dataIndex: "time",

render: (text, data) => {

return moment(text, "HH:mm").format("hh:mm A");

},

},

{

title: "Movie",

dataIndex: "movie",

render: (text, data) => {

return data.movie.title;

},

},

{

title: "Ticket Price",

dataIndex: "ticketPrice",

key: "ticketPrice",

},

{

title: "Total Seats",

dataIndex: "totalSeats",

key: "totalSeats",

},

{

title: "Available Seats",

dataIndex: "seats",

render: (text, data) => {

return data.totalSeats - data.bookedSeats.length;

},

},

{

title: "Action",

dataIndex: "action",

render: (text, data) => {

return (

<div className="d-flex align-items-center gap-10">

<Button

onClick={() => {

setView("edit");

setSelectedMovie(data.movie);

setSelectedShow({

...data,

date: moment(data.date).format("YYYY-MM-DD"),

});

console.log(selectedMovie && selectedMovie.title);

}}

>

<EditOutlined />

</Button>

<Button onClick={() => handleDelete(data.\_id)}>

<DeleteOutlined />

</Button>

{data.isActive && (

<Button

onClick={() => {

setIsShowModalOpen(true);

}}

>

+ Shows

</Button>

)}

</div>

);

},

},

];

useEffect(() => {

getData();

}, []);

return (

<Modal

centered

title={selectedTheatre.name}

open={isShowModalOpen}

onCancel={handleCancel}

width={1200}

footer={null}

>

<div className="d-flex justify-content-between">

<h3>

{view === "table"

? "List of Shows"

: view === "form"

? "Add Show"

: "Edit Show"}

</h3>

{view === "table" && (

<Button type="primary" onClick={() => setView("form")}>

Add Show

</Button>

)}

</div>

{view === "table" && <Table dataSource={shows} columns={columns} />}

{(view === "form" || view === "edit") && (

<Form

className=""

layout="vertical"

style={{ width: "100%" }}

initialValues={view === "edit" ? selectedShow : null}

onFinish={onFinish}

>

<Row

gutter={{

xs: 6,

sm: 10,

md: 12,

lg: 16,

}}

>

<Col span={24}>

<Row

gutter={{

xs: 6,

sm: 10,

md: 12,

lg: 16,

}}

>

<Col span={8}>

<Form.Item

label="Show Name"

htmlFor="name"

name="name"

className="d-block"

rules={[

{ required: true, message: "Show name is required!" },

]}

>

<Input

id="name"

type="text"

placeholder="Enter the show name"

></Input>

</Form.Item>

</Col>

<Col span={8}>

<Form.Item

label="Show Date"

htmlFor="date"

name="date"

className="d-block"

rules={[

{ required: true, message: "Show date is required!" },

]}

>

<Input

id="date"

type="date"

placeholder="Enter the show date"

></Input>

</Form.Item>

</Col>

<Col span={8}>

<Form.Item

label="Show Timing"

htmlFor="time"

name="time"

className="d-block"

rules={[

{ required: true, message: "Show time is required!" },

]}

>

<Input

id="time"

type="time"

placeholder="Enter the show date"

></Input>

</Form.Item>

</Col>

</Row>

</Col>

<Col span={24}>

<Row

gutter={{

xs: 6,

sm: 10,

md: 12,

lg: 16,

}}

>

<Col span={8}>

<Form.Item

label="Select the Movie"

htmlFor="movie"

name="movie"

className="d-block"

rules={[{ required: true, message: "Movie is required!" }]}

>

<Select

id="movie"

placeholder="Select Movie"

defaultValue={selectedMovie && selectedMovie.title}

style={{ width: "100%", height: "45px" }}

options={movies.map((movie) => ({

key: movie.\_id,

value: movie.\_id,

label: movie.title,

}))}

/>

</Form.Item>

</Col>

<Col span={8}>

<Form.Item

label="Ticket Price"

htmlFor="ticketPrice"

name="ticketPrice"

className="d-block"

rules={[

{ required: true, message: "Ticket price is required!" },

]}

>

<Input

id="ticketPrice"

type="number"

placeholder="Enter the ticket price"

></Input>

</Form.Item>

</Col>

<Col span={8}>

<Form.Item

label="Total Seats"

htmlFor="totalSeats"

name="totalSeats"

className="d-block"

rules={[

{ required: true, message: "Total seats are required!" },

]}

>

<Input

id="totalSeats"

type="number"

placeholder="Enter the number of total seats"

></Input>

</Form.Item>

</Col>

</Row>

</Col>

</Row>

<div className="d-flex gap-10">

<Button

className=""

block

onClick={() => {

setView("table");

}}

htmlType="button"

>

<ArrowLeftOutlined /> Go Back

</Button>

<Button

block

type="primary"

htmlType="submit"

style={{ fontSize: "1rem", fontWeight: "600" }}

>

{view === "form" ? "Add the Show" : "Edit the Show"}

</Button>

</div>

</Form>

)}

</Modal>

);

};

export default ShowModal;