MERN-8: Project Part 4 - (Implementing Movies and Theatre API)

1. Creating Admin Partner And User Page
2. Creating the Movie Api (CRUD) and testing via Postman to DB
3. Creating the Movie Form which can perform CRUD from the client side

title:Creating Admin Partner And User Page

In our app there are Three roles

Admin - Admin can add movies in the app and can approve partner Theatres who wants to run their shows for the movies

Partner - If a user logs in as a partner that means they can request the admin to get their theatres on the app , if the admin approves then they will be able to run shows for the movies in their approved Theatres

User - This role is a customer role , where people can go and book tickets for a currently showing movie , go watch eat popcorn and have fun!

So to create these three roles we will have to create their separate pages where they can do the task that is assigned to them

Admin features will be different , Partner features will be different and User feature will be different

So for this we will create three separate components in our Pages folder like this

Create three directories inside the Pages Directory Admin Partner User create a index.jsx file inside all of them which will act as root file and create three basic components

Index.js for Admin

```
import React from "react";

function Admin() {
 return (
   <div>
     <h1>Admin Page</h1>
   </div>
 );
}

export default Admin;
```

Index.js for Partner

```
import React from "react";
```

```
function Partner() {
 return (
   <div>
     <h1>Partner Page</h1>
   </div>
 );
}

export default Partner;
```

Index.js for User

```
import React from "react";

function Profile() {
 return (
   <div>
     <h1>Profile Page</h1>
   </div>
 );
}

export default Profile;
```

Admin Page

At first we will start working with the admin Page

The admin will have these two rights

The admin can add movies of which shows will run
The admin can approve or decline Theatre request from a Partner
So the admin component will have two sub components

MovieList Component - Responsible for Movies CRUD

TheatresTable - Theatre request will come here for approval

So now create two Basic components again inside the admin Directory

MovieList.js

```
import React from "react";
import { Table } from "antd";

function MovieList() {
 return (
   <div>
     <Table />
   </div>
 );
}

export default MovieList;
```

TheatresTable.js

```
import { Table } from "antd";
import React from "react";

function TheatresTable() {
 return (
   <div>
     <Table />
   </div>
 );
}

export default TheatresTable;
```

In your admin (index.js file) Now put this code

```
// import React from "react";

import { Tabs } from "antd";
import MovieList from "./MovieList";
import TheatresTable from "./TheatresTable";

function Admin() {
 const tabItems = [
   {
     key: "1",
     label: "Movies",
     children: <MovieList />,
   },

   {
     key: "2",
     label: "Theatres",
     children: <TheatresTable />,
   },
 ];

 return (
   <div>
     <h1>Admin Page</h1>

     <Tabs items={tabItems} />
   </div>
 );
}

export default Admin;
```

This code sets up a tabbed interface using the Tabs component from the Ant Design library. Here is a detailed explanation of what each part of the code does:

title:designing the Movies List by Adding static Data

```
import { Table } from "antd";

function MovieList() {
 const fakeMovies = [
   {
     key: "1",
     poster: "Image1",
     description: "Wolverine Vs Deadpool",
     duration: 120,
     genre: "Action",
     language: "English",
     releaseDate: "2024-08-01",
     name: "Wolverine Vs Deadpool",
   },
   {
     key: "2",
     poster: "Image2",
     description: "Wolverine Vs Deadpool",
     duration: 120,
     genre: "Action",
     language: "English",
     releaseDate: "2024-08-01",
     name: "Wolverine Vs Deadpool 2",
   },
 ];


 const tableHeadings = [
   {
     title: "Poster",
   },
   {
```

```
    title: "Movie Name",
    dataIndex: "name",
  },
  {
    title: "Description",
    dataIndex: "description",
  },
  {
    title: "Duration",
    dataIndex: "duration",
  },
  {
    title: "Genre",
    dataIndex: "genre",
  },
  {
    title: "Language",
    dataIndex: "language",
  },
  {
    title: "Release Date",
    dataIndex: "releaseDate",
  },
  {
    title: "Action",
  },
];

 return (
   <div>
     <Table dataSource={movies} columns={tableHeadings} />
   </div>
 );
}

export default MovieList;
```

title:Buidling the Movie API

To add a Movie we will first need to create a Movie Model

Go to server -> Models and create a movieModel.js file

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

const movieSchema = new mongoose.Schema({
 movieName: {
   type: String,
   required: true,
 },
 description: {
   type: String,
   required: true,
 },
 duration: {
   type: Number,
   required: true,
 },
 genre: {
   type: String,
   required: true,
 },
 language: {
   type: String,
   required: true,
 },
 releaseDate: {
   type: Date,
   required: true,
 },
 poster: {
   type: String,
   required: true,
 },
});

const Movies = mongoose.model("movies", movieSchema);
```

```
module.exports = Movies;
```

The Mongoose library is imported. Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a straightforward, schema-based solution to model application data.

title:Buidling the Movie Route

Now in Your Route folder Create a movieRoute.js file and create you first Route

```
const movieRouter = require("express").Router();
const Movie = require("../models/movieModel");

// Add a Movie

movieRouter.post("/add-movie", async (req, res) => {
 try {
   const newMovie = new Movie(req.body);
   await newMovie.save();
   res.send({
     success: true,
     message: "New movie has been added!",
   });
 } catch (error) {
   res.send({
     success: false,
     message: err.message,
   });
 }
});

module.exports = movieRouter;
```

Add this to the main server file

```
/** Routes */
app.use("/api/users", userRouter);
app.use("/api/movies", movieRouter);
```

Create movieController.js and move the handler to this file

```
const Movie = require("../model/movieModel");

const addMovie = async (req, res) => {
 try {
   const newMovie = new Movie(req.body);
   await newMovie.save();
   res.send({
     success: true,
     message: "New movie has been added!",
   });
 } catch (error) {
   res.send({
     success: false,
     message: err.message,
   });
 }
};

module.exports = { addMovie };
```

Update the route in movieRoute.js

```
movieRouter.post("/add-movie", addMovie);
```

title:Implementing Update Delete and Get Routes

Add these three routes after the add a movie is done

```
const movieRouter = require("express").Router();
const {
 addMovie,
 getAllMovies,
 updateMovie,
 deleteMovie,
} = require("../controllers/movieController");

// Add a Movie

movieRouter.post("/add-movie", addMovie);
// Get all the movies

movieRouter.get("/get-all-movies", getAllMovies);

// Update a movie

// Update a movie
movieRouter.put("/update-movie", updateMovie);

movieRouter.put("/delete-movie", deleteMovie);

module.exports = movieRouter;
```

Update movie controller

```
const Movie = require("../model/movieModel");

const addMovie = async (req, res) => {
 try {
   const newMovie = new Movie(req.body);
   await newMovie.save();
   res.send({
     success: true,
     message: "New movie has been added!",
```

```
    });
  } catch (error) {
    res.send({
      success: false,
      message: err.message,
    });
  }
};

const getAllMovies = async (req, res) => {
  try {
    const allMovies = await Movie.find();
    res.send({
      success: true,
      message: "All movies have been fetched!",
      data: allMovies,
    });
  } catch (error) {
    res.send({
      success: false,
      message: error.message,
    });
  }
};

const updateMovie = async (req, res) => {
  try {
    await Movie.findByIdAndUpdate(req.body.movieId, req.body);
    res.send({
      success: true,
      message: "Movie updated",
    });
  } catch (error) {
    res.send({
      success: false,
      message: error.message,
    });
  }
};


const deleteMovie = async (req, res) => {
```

```
 try {
   await Movie.findByIdAndDelete(req.body.movieId);
   console.log(req.body.movieId);
   res.send({
     success: true,
     message: "The movie has been deleted!",
   });
 } catch (err) {
   res.send({
     success: false,
     message: err.message,
   });
 }
};

module.exports = { addMovie, getAllMovies, updateMovie, deleteMovie };
```

Now we need to write axios Instance calls for our movies

In the api directory under client folder - create movies.js file and add these instances which will work in accordnce with differnent routes

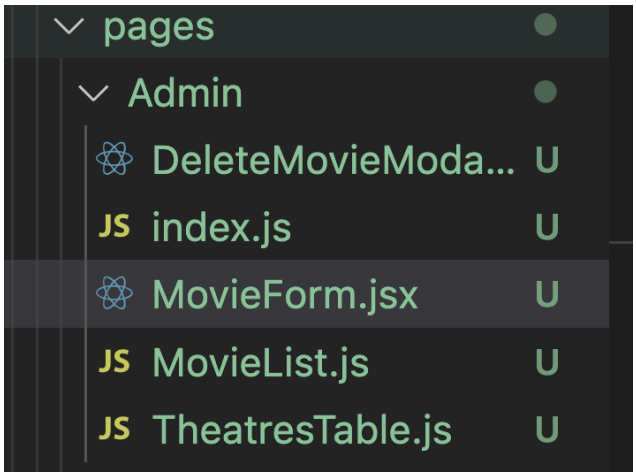
```
import { axiosInstance } from "./index";

//get all Movies
export const getAllMovies = async () => {
 try {
   const response = await axiosInstance.get("api/movies/get-all-movies");
   return response.data;
 } catch (error) {
   console.error(error);
 }
};

// Add a movie
```

```
export const addMovie = async (values) => {
 try {
   const response = await axiosInstance.post("api/movies/add-movie", values);
   return response.data;
 } catch (error) {
   console.error(error);
 }
};

export const updateMovie = async (payload) => {
 try {
   const response = await axiosInstance.put(
     "/api/movies/update-movie",
     payload
   );
   return response.data;
 } catch (err) {
   return err.message;
 }
};

// Delete a movie
export const deleteMovie = async (payload) => {
 try {
   const response = await axiosInstance.put(
     "/api/movies/delete-movie",
     payload
   );
   return response.data;
 } catch (err) {
   return err.message;
 }
};
```

Movies action on front end

Now let's go back to MoviesList Component and create the form for adding Movies

Now we need to create a movie Form , In the Admin directory create a Movie Form Component and a deletemovieModal



1. Updating MoviesList Component

```
import React, { useEffect, useState } from "react";
import { Button, Table } from "antd";
import MovieForm from "./MovieForm";
import { HideLoading, ShowLoading } from "../../redux/loaderSlice";
import { getAllMovies } from "../../api/movies";
import { useDispatch } from "react-redux";
import moment from "moment";
import { EditOutlined, DeleteOutlined } from "@ant-design/icons";
import DeleteMovieModal from "./DeleteMovieModal";
```

a. Create some state variables

```
const [isModalOpen, setIsModalOpen] = useState(false);
const [movies, setMovies] = useState(FakeMovies);
const [selectedMovie, setSelectedMovie] = useState(null);
const [formType, setFormType] = useState("add");
const [isDeleteModalOpen, setIsDeleteModalOpen] = useState(false);
const dispatch = useDispatch();
```

b. Now let us create table headings for our table antd table component

```
const tableHeadings = [
  {
    title: "Poster",
    dataIndex: "poster",
    render: (text, data) => {
      return (
        <img
          width="75"
          height="115"
          style={{ objectFit: "cover" }}
          src={data.poster}
        />
      );
    },
  },
  {
    title: "Movie Name",
    dataIndex: "title",
  },
  {
    title: "Description",
    dataIndex: "description",
  },
  {
    title: "Duration",
    dataIndex: "duration",
    render: (text) => {
      return `${text} Min`;
    },
  },
  {
    title: "Genre",
    dataIndex: "genre",
  },
  {
    title: "Language",
```

```
    dataIndex: "language",
  },
  {
    title: "Release Date",
    dataIndex: "releaseDate",
    render: (text, data) => {
      return moment(data.releaseDate).format("MM-DD-YYYY");
    },
  },
  {
    title: "Action",
    render: (text, data) => {
      return (
        <div>
          <Button
            onClick={() => {
              setIsModalOpen(true);
              setSelectedMovie(data);
              setFormType("edit");
            }}
          >
            <EditOutlined />
          </Button>
          <Button
            onClick={() => {
              setIsDeleteModalOpen(true);
              setSelectedMovie(data);
            }}
          >
            <DeleteOutlined />
          </Button>
        </div>
      );
    },
  },
];
```

i. In the Ant Design Table component, the render function is used to customize how the data in a column is displayed. The render

function takes two parameters, typically referred to as text and record (or data in your example). Here's a breakdown of what these parameters represent and how they are passed:

ii. text Parameter:

The text parameter represents the data value of the current cell in the column.
This value corresponds to the property specified by the dataIndex field for that column.

iii. record (or data) Parameter:

The record parameter represents the entire data object for the current row.
This allows you to access other properties of the current row's data object, which can be useful for more complex render logic.

iv. The dataIndex field in the Ant Design Table component is used to specify the key in the data source object that should be used for a particular column. This field helps the table component to know which data from the data source should be displayed in each column. Here's a breakdown of why dataIndex is necessary:

Mapping Data to Columns: The dataIndex field tells the table which property of the data source object should be displayed in that column. For example, if the data source object is { poster: 'url', title: 'Movie Title' }, the dataIndex: 'poster' in the column definition ensures that the value of the poster key is used for that column.

Simplifies Data Handling: By defining the dataIndex, you don't need to manually extract and render the data for each cell unless you want to customize the rendering using the render function. The table automatically handles the data extraction based on the dataIndex.

c. getData function

```
const getData = async () => {
  dispatch(ShowLoading());
  const response = await getAllMovies();
  const allMovies = response.data;
  setMovies(
    allMovies.map(function (item) {
      return { ...item, key: `movie${item._id}` };
    })
  );
  dispatch(HideLoading());
};

useEffect(() => {
  getData();
}, []);
```

d. Implement the markup

```
<>
    <div className="d-flex justify-content-end">
      <Button
        onClick={() => {
          setIsModalOpen(true);
          setFormType("add");
        }}
      >
        Add Movie
      </Button>
    </div>
    <Table dataSource={movies} columns={tableHeadings} />
  </>
```

Extra for now:

In case you want to see the Movie List, add this in App.js

```
<Route
            path="/admin"
            element={
              <ProtectedRoute>
                <Admin />
              </ProtectedRoute>
            }
          />
```

Remember: you will need to go to MongoDB and change the role of the user to admin

Add/ Update Movie and Delete Movie form

```
<Table dataSource={movies} columns={tableHeadings} />
    {isModalOpen && (
      <MovieForm
        isModalOpen={isModalOpen}
        setIsModalOpen={setIsModalOpen}
        selectedMovie={selectedMovie}
        formType={formType}
```

```
        setSelectedMovie={setSelectedMovie}
        getData={getData}
      />
    )}

    {isDeleteModalOpen && (
      <DeleteMovieModal
        isDeleteModalOpen={isDeleteModalOpen}
        selectedMovie={selectedMovie}
        setIsDeleteModalOpen={setIsDeleteModalOpen}
        setSelectedMovie={setSelectedMovie}
        getData={getData}
      />
    )}
```

MovieForm Component

```
import { Col, Modal, Row, Form, Input, Select, Button, message } from "antd";
import TextArea from "antd/es/input/TextArea";
import { ShowLoading, HideLoading } from "../../redux/loaderSlice";
import { useDispatch } from "react-redux";
import { addMovie, updateMovie } from "../../api/movies";
import moment from "moment";

const MovieForm = ({
 isModalOpen,
 setIsModalOpen,
 selectedMovie,
 setSelectedMovie,
 formType,
 getData,
}) => {
 const dispatch = useDispatch();

 if (selectedMovie) {
   selectedMovie.releaseDate = moment(selectedMovie.releaseDate).format(
     "YYYY-MM-DD"
   );
 }
```

```
const onFinish = async (values) => {
  try {
    dispatch(ShowLoading());
    let response = null;
    if (formType === "add") {
      response = await addMovie(values);
    } else {
      response = await updateMovie({ ...values, movieId: selectedMovie._id });
    }
    if (response.success) {
      getData();
      message.success(response.message);
      setIsModalOpen(false);
    } else {
      message.error(response.message);
    }
    setSelectedMovie(null);
    dispatch(HideLoading());
  } catch (err) {
    dispatch(HideLoading());
    message.error(err.message);
  }
};

const handleCancel = () => {
  setIsModalOpen(false);
  setSelectedMovie(null);
};

return (
  <Modal
    centered
    title={formType === "add" ? "Add Movie" : "Edit Movie"}
    open={isModalOpen}
    onCancel={handleCancel}
    width={800}
    footer={null}
  >
    <Form layout="vertical" initialValues={selectedMovie} onFinish={onFinish}>
      <Row gutter={{ xs: 6, sm: 10, md: 12, lg: 16 }}>
        <Col span={24}>
          <Form.Item
```

```
            label="Movie Name"
            name="title"
            rules={[{ required: true, message: "Movie name is required!" }]}
          >
            <Input placeholder="Enter the movie name" />
          </Form.Item>
        </Col>
        <Col span={24}>
          <Form.Item
            label="Description"
            name="description"
            rules={[{ required: true, message: "Description is required!" }]}
          >
            <TextArea rows="4" placeholder="Enter the description" />
          </Form.Item>
        </Col>
        <Col span={24}>
          <Row gutter={{ xs: 6, sm: 10, md: 12, lg: 16 }}>
            <Col span={8}>
              <Form.Item
                label="Movie Duration (in min)"
                name="duration"
                rules={[
                  { required: true, message: "Movie duration is required!" },
                ]}
              >
                <Input type="number" placeholder="Enter the movie duration" />
              </Form.Item>
            </Col>
            <Col span={8}>
              <Form.Item
                label="Select Movie Language"
                name="language"
                rules={[
                  { required: true, message: "Movie language is required!" },
                ]}
              >
                <Select
                  placeholder="Select Language"
                  options={[
                    { value: "English", label: "English" },
                    { value: "Hindi", label: "Hindi" },
```

```
                { value: "Punjabi", label: "Punjabi" },
                { value: "Telugu", label: "Telugu" },
                { value: "Bengali", label: "Bengali" },
                { value: "German", label: "German" },
              ]}
            />
          </Form.Item>
        </Col>
        <Col span={8}>
          <Form.Item
            label="Release Date"
            name="releaseDate"
            rules={[
              {
                required: true,
                message: "Movie Release Date is required!",
              },
            ]}
          >
            <Input type="date" />
          </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 Movie Genre"
            name="genre"
            rules={[
              { required: true, message: "Movie genre is required!" },
            ]}
          >
            <Select
              placeholder="Select Movie"
              options={[
                { value: "Action", label: "Action" },
                { value: "Comedy", label: "Comedy" },
                { value: "Horror", label: "Horror" },
                { value: "Love", label: "Love" },
                { value: "Patriot", label: "Patriot" },
```

```
                    { value: "Bhakti", label: "Bhakti" },
                    { value: "Thriller", label: "Thriller" },
                    { value: "Mystery", label: "Mystery" },
                  ]}
                />
              </Form.Item>
            </Col>
            <Col span={16}>
              <Form.Item
                label="Poster URL"
                name="poster"
                rules={[
                  { required: true, message: "Movie Poster is required!" },
                ]}
              >
                <Input placeholder="Enter the poster URL" />
              </Form.Item>
            </Col>
          </Row>
        </Col>
      </Row>
      <Form.Item>
        <Button
          block
          type="primary"
          htmlType="submit"
          style={{ fontSize: "1rem", fontWeight: "600" }}
        >
          Submit the Data
        </Button>
        <Button className="mt-3" block onClick={handleCancel}>
          Cancel
        </Button>
      </Form.Item>
    </Form>
  </Modal>
 );
};

export default MovieForm;
```

Explanation

1. Props:
    a. isModalOpen, setIsModalOpen: Controls the visibility of the modal.
    b. selectedMovie, setSelectedMovie: The movie selected for editing.
    c. formType: Determines if the form is for adding a new movie or editing an existing one.
    d. getData: Function to refresh the movie list.
2. State Management:
    a. Uses dispatch to trigger Redux actions for showing and hiding loading indicators.
3. Gutter prop in row
    a. The gutter prop in the Row component of Ant Design (antd) is used to define the spacing between the columns within that row. This prop can accept various types of values to control the horizontal and vertical spacing, providing a flexible way to design responsive layouts.
    b. In our code we passed an object. This sets different gutter values for different screen sizes:
        i. 8 pixels for extra small screens (xs)
        ii. 16 pixels for small screens (sm)
        iii. 24 pixels for medium screens (md)
        iv. 32 pixels for large screens (lg)

DeleteMovieModal Component

```
import { Modal, message } from "antd";
import { deleteMovie } from "../../api/movies";
import { ShowLoading, HideLoading } from "../../redux/loaderSlice";
import { useDispatch } from "react-redux";

const DeleteMovieModal = ({
 isDeleteModalOpen,
 setIsDeleteModalOpen,
 selectedMovie,
 setSelectedMovie,
 getData,
}) => {
 const dispatch = useDispatch();

 const handleOk = async () => {
   try {
     dispatch(ShowLoading());
     const movieId = selectedMovie._id;
     const response = await deleteMovie({ movieId });
     if (response.success) {
       message.success(response.message);
       getData();
     } else {
       message.error(response.message);
     }
     setSelectedMovie(null);
     setIsDeleteModalOpen(false);
     dispatch(HideLoading());
   } catch (err) {
     dispatch(HideLoading());
     setIsDeleteModalOpen(false);
     message.error(err.message);
   }
 };

 const handleCancel = () => {
   setIsDeleteModalOpen(false);
   setSelectedMovie(null);
 };
```

```
 return (
   <Modal
     title="Delete Movie?"
     open={isDeleteModalOpen}
     onOk={handleOk}
     onCancel={handleCancel}
   >
     <p className="pt-3 fs-18">Are you sure you want to delete this movie?</p>
     <p className="pb-3 fs-18">
       This action can't be undone and you'll lose this movie data.
     </p>
   </Modal>
 );
};

export default DeleteMovieModal;
```

Comment the useEffect in the MovieList and see the forms