Comprehensive Class Notes: Creating a Movies and Theatres API in a MERN Stack Application

This document serves as comprehensive notes on the concepts covered in the class focused on implementing the Movies and Theatres API in a MERN (MongoDB, Express, React, Node.js) stack application. The class was primarily concerned with setting up the backend infrastructure to support CRUD operations on movies and organizing an admin, partner, and user page structure.

Project Overview

Roles in the Application

1. **Admin**:
   - Can add movies and approve theatres owned by partners.
   - Has functionalities to manage movie data through a MovieList component and theatre requests through a TheatresTable component.
2. **Partner**:
   - Can request approval from admins for showing movies in their theatres.
3. **User**:
   - Can browse movies, book tickets, and enjoy entertainment offerings.

Application Structure

The application requires creating individual pages for each user role with specific functionalities. The directories are structured as follows:

- `Partner/`
- `User/`

Each directory contains an `index.jsx` file acting as the root file for rendering basic components specific to the role.

Creating the Movie API

Setting Up the Model and Routes

1. **Movie Model**:
   - Defined using Mongoose, this model includes fields like `title`, `description`, `duration`, `genre`, `language`, `release date`, and `poster`.
2. **Routes Configuration**:
   - **Add Movie Route**: Configured as a POST request to `/add-movie` to handle adding new movies through data provided via `req.body`.
   - **Get All Movies Route**: Configured to retrieve all movies from the database.
   - **Update Movie Route**: Allows for updating a movie's details.
   - **Delete Movie Route**: Used for removing a movie entry from the database.

Implementing CRUD Operations

- The movie routes and handlers are established in `movieRoute.js` and associated with controllers for each action:
  - **Add a Movie**: Uses `Movie.save()` to persist new movie data.
  - **Get All Movies**: Retrieves all movies using `Movie.find()`.
  - **Update Movie**: Uses `Movie.findByIdAndUpdate()` for modifying movie details.

Client-side Integration

React Components

1. **MoviesList Component**:
   - Displays a list of movies utilizing Ant Design's `Table` component.
   - Capable of opening modals for adding, editing, and deleting movies .
2. **MovieForm Component**:
   - Used for both creating and editing movie entries.
   - Integrated with state management for handling outputs from the form interactions .
3. **DeleteMovieModal Component**:
   - Confirms movie deletion from the database by displaying a modal dialog .

State Management and API Calls

- Leveraging Axios for API interactions, the client-side code includes helper functions in the `movies.js` file to manage movie data through HTTP requests:
  - `getAllMovies` , `addMovie` , `updateMovie` , and `deleteMovie` functions .

UI Enhancements using Ant Design

- Ant Design components like `Table` and `Modal` are heavily utilized for UI consistency and functionality.
- Tabs are used on the admin page to navigate between managing movies and theatres .

Completing the backend setup for movie management in a MERN stack involves defining clear data models, setting up Express routes, and establishing controller logic for CRUD operations. For the client, React components interact with the backend using Axios, facilitated by organized state management with Redux for a seamless administrative user experience.

With this foundational setup, the next step involves extending permissions and functionalities for partner roles, focusing on theatre approvals and listings .