https://binnysjewellery.com/

MERN Stack Movie Application: Role-Based Access Control

Objective:

Develop a movie web application with the following functionality:

1. **User Features**:
   - View movie details fetched from IMDb's Top 250 Movies.
     - **Reference URL**: https://www.imdb.com/chart/top?ref_=nv_mv_250
   - Search movies by name or description.
   - Sort movie details by name, rating, release date, and duration.
2. **Admin Features**:
   - Add new movie details.
   - Edit or delete existing movies.

Requirements:

Frontend

- **Framework**: React.js
- **Styling**: Material-UI for CSS and responsiveness
- **Features**:
  - **User Pages**:
    - Home Page: Displays all movies with pagination.
    - Search Page: Allows filtering and searching for movies by name or description.
  - **Admin Pages**:
    - Add Movie Page: Form for adding new movies.

        - Edit/Delete Movie Page: Admin functionalities for modifying movie details.
    - **Authentication**:
        - Implement JWT-based authentication for user login and role-based access control.
    - **Routing**:
        - Use react-router-dom to protect admin routes and manage navigation.
    - **State Management**:
        - Use Context API or Redux for managing application state.

Backend

- **Framework**: Node.js with Express.js
- **Database**: MongoDB
- **Features**:
    - **REST API Endpoints**:
        - GET /movies: Retrieve all movies.
        - GET /movies/sorted: Get sorted movies by name, rating, release date, or duration.
        - GET /movies/search: Search movies by name or description.
        - POST /movies: Add a new movie (admin only).
        - PUT /movies/:id: Edit movie details (admin only).
        - DELETE /movies/:id: Delete a movie (admin only).
    - **Authentication & Authorization**:
        - Secure endpoints with JWT authentication.
        - Middleware for role-based access control.
    - **Data Handling**:
        - Use a distributed queue for lazy insertion of data into the database.
        - Ensure database concurrency and performance.
    - **Error Handling**:

        - Implement robust error handling for unauthorized access, invalid inputs, and crashes.

Additional Instructions:

1. **Concurrency and Performance**:
    - Design the system with scalability in mind.
    - Optimize API calls and database queries for high performance.
2. **Queue Implementation**:
    - Add movie data to a distributed queue and implement lazy insertion into the database.
3. **Crash Recovery**:
    - Implement a mechanism to retrieve unprocessed messages in case of application or server crashes.
    - For example, use a message broker like RabbitMQ or Kafka for message persistence.
4. **Version Control**:
    - Commit all code to a public GitHub repository.
    - Include a detailed README.md file with setup instructions, API documentation, and a live application URL.
5. **Deployment**:
    - Deploy the frontend using platforms like Vercel or Netlify.
    - Deploy the backend using Heroku, AWS, or Railway.
    - Use MongoDB Atlas for hosting the database.

Evaluation Criteria:

1. **Authentication & Authorization**:
    - Secure user login and role-based access control for admin functionality.

2. **Frontend Design**:
    - Responsive UI design using Material-UI.
3. **Backend Implementation**:
    - Efficient and secure REST API implementation.
4. **Scalability**:
    - Ability to handle concurrent users and large datasets.
5. **Deployment**:
    - Fully deployed and functional application with a live URL.
6. **Code Quality**:
    - Well-documented code with proper version control practices.