# CDAC Mumbai

---

**Module: WPT**

**Topic: Assignment - 9**

---

## Objective

Develop a Role-Based CRUD Application using **ReactJS** for the frontend, **NodeJS** for the backend, and **MongoDB** for the database. Implement secure login for different roles and enforce role-specific access to features and data.

---

## Requirements

1. **Role-Based Login and Authentication**
    - **Roles**: Implement two roles: **User** and **Admin**.
    - **Authentication & Authorization**: Use **JWT** (JSON Web Token) for secure login, with token-based authorization for all protected routes.
    - **State Management**: Manage user session, login status, and access permissions using a state management solution (e.g., **Redux** or React Context).
2. **Functional Requirements**
    - **User Role Functionalities**
        - **CRUD Operations**: Users should be able to perform Create, Read, Update, and Delete operations on their assigned items (e.g., tasks, posts, orders).
        - **Data Dependency**: Each item created by a User should be associated with a parent entity (e.g., a project or category) to simulate a dependency.
        - **Restricted Access**: Users should only view and modify their own items.
    - **Admin Role Functionalities**
        - **CRUD Operations on All Items**: Admins can perform CRUD operations on any item across the platform.
        - **User Management**: Admins should be able to manage (create, update, delete) users and assign items to them.
        - **Data Dependency Management**: Admins can assign or modify items for Users under specific categories/projects.
3. **Database Relations**
    - **Item-User Dependency**: Each item (e.g., task, order) must be tied to a specific user. Set up MongoDB schemas to enforce this relationship.

    - **Parent-Item Dependency**: Each item should belong to a parent entity (e.g., project, category). Establish this dependency in MongoDB.
4. **Frontend Components**
    - **Role-Based Views**: Display different UI components based on the User or Admin role.
    - **Item List**: Show a list of items relevant to the logged-in user (all items for Admin, user-specific items for Users).
    - **User & Item Assignment**: Allow Admins to assign items and manage users from the frontend interface.

---

## Additional Requirements

1. **Error Handling and Security**
    - Show error messages for issues like invalid login, unauthorized access, or CRUD operation failures.
    - Use JWT-based access control for protected API routes, ensuring only authenticated users can perform actions.
2. **Example Usage**
   **Suggested Example Themes**:
    - Project Management (Projects and Tasks)
    - Inventory System (Categories and Items)
    - Content Management (Blogs and Posts)

*Choose your own theme and apply these functionalities accordingly.*

---

## Example Theme: Event Planning and Ticketing System

---

## Objective

Create an **Event Planning and Ticketing System** where **Event Managers** can create and manage events, while **Users** (attendees) can browse events, book tickets, and view their bookings.

---

## Features and Requirements

1. **Roles**:
    - **Event Manager**:
        - Can create, edit, and delete events.
        - Can view attendees for each event.
        - Can manage ticket availability and event capacity.
    - **User (Attendee)**:
        - Can view upcoming events.
        - Can book tickets, view their bookings, and cancel bookings if needed.
2. **Database Structure**:
    - **Users Collection**: Stores details about each user, including a role field to distinguish Event Managers from Attendees.
    - **Events Collection**: Contains event details like event name, date, time, location, capacity, and ticket price.
    - **Bookings Collection** (Dependency): Tracks ticket bookings, linking users to events and storing booking details (like number of tickets and total cost).
3. **Functional Requirements**:
    - **Event Manager Functionalities**:
        - **Manage Events**: Create, update, and delete events, set event capacity and ticket prices.
        - **View Bookings**: See a list of attendees for each event with booking details (number of tickets booked).
        - **Update Ticket Availability**: Adjust ticket availability based on capacity and cancellations.
    - **User Functionalities**:
        - **Browse Events**: View a list of upcoming events with details like date, location, and price.
        - **Book Tickets**: Book tickets for an event (if tickets are available) and get a booking confirmation.

        - **View Bookings**: Access a personal bookings list to review or cancel upcoming bookings.
4. **Security**:
    - Implement **JWT Authentication** to restrict access based on roles.
    - **Authorization**: Ensure that only Event Managers can access and manage event data, while Users can only view and manage their own bookings.
5. **Example Usage**:
    - **Event Manager Login**: An Event Manager logs in, creates a new event, manages ticket capacity, and views attendees for each event.
    - **User Login**: A User logs in, views available events, books tickets for an interesting event, and reviews their booking history.