1. Project Overview

The goal is to develop a full-stack web application that serves as a task management system with **Role-Based Access Control (RBAC)**. This application must allow users to register, log in, manage tasks, and perform different actions based on their assigned role (e.g., **Admin, Manager, Employee**).

Technology Stack (Suggested: MERN Stack)

- **Frontend:** React.js (with React Router for navigation)
- **Backend:** Node.js (with Express.js)
- **Database:** MongoDB (with Mongoose for Object Data Modeling)
- **Authentication:** JSON Web Tokens (JWT) for secure, stateless authentication
- **Password Hashing:** `bcryptjs`

---

2. Core Features & Requirements

2.1 User Authentication & Authorization

Feature	Details
Sign Up/Register	Secure registration form. Passwords must be **hashed** before storage.
Log In/Sign In	Users must be validated against the database. Successful login should issue a **JWT** token.
Authorization	Check the user's `role` before granting access to specific API routes.

2.2 Task Management

Feature	Details (CRUD Operations)
Task Model	Must include fields: `title`, `description`, `status` (e.g., To Do, In Progress, Done), `priority` (e.g., High, Medium, Low), `dueDate`, `assignedTo` (User ID), and `createdBy` (User ID).
Create Task	Form to create a new task. The `createdBy` field is automatically set to the logged-in user.
Read/View Tasks	Users should only be able to see tasks relevant to them based on their role (see RBAC below).

Feature	Details (CRUD Operations)
Update Task	Functionality to edit a task's details.
Delete Task	Functionality to remove a task.

2.3 Role-Based Access Control (RBAC) Logic

The application must enforce the following access rules:

User Role	Task Management Permissions	User Management Permissions	View Access
Employee	**CRUD** on **only** tasks created by them. **Update** status/priority on tasks *assigned* to them.	Read-only access to their own profile.	View all tasks assigned to them.
Manager	**CRUD** on **all** tasks. Can **assign** tasks to any Employee.	Can view and update roles of **Employees** only.	View all tasks in the system.
Admin	**CRUD** on **all** tasks. Can **assign** tasks to any user.	**CRUD** on **all** users (including other Managers/Admins). Can change any user's role.	View all tasks in the system.

---

3. Technical Requirements & Best Practices

Backend (Node/Express/MongoDB)

1. **RESTful API:** Design clear, versioned API endpoints (e.g., `/api/v1/tasks`, `/api/v1/auth`).
2. **Modular Structure:** Separate routes, controllers, models, and middleware into distinct files.
3. **Input Validation:** Validate incoming request data before interacting with the database.
4. **Error Handling:** Implement a global error-handling middleware to catch and return structured error responses (with appropriate HTTP status codes).

Frontend (React)

1. **Component Structure:** Use a clear, modular component hierarchy.
2. **State Management:** Use modern React features (e.g., Context API or Redux) for managing global state like user authentication and tasks.

3. **UI/UX:** Create a clean, responsive user interface. A Kanban-style board (Todo, In Progress, Done columns) for task viewing is a plus.
4. **Conditional Rendering:** Use the user's role to conditionally render components or UI elements (e.g., showing an "Admin Dashboard" link only to Admins).

Source Code & Documentation

1. **Version Control:** Use Git and host the code in a **public GitHub repository**. Demonstrate consistent, meaningful commit history.
2. **README.md:** A detailed README file is mandatory, including:
    - Project Title and Description.
    - Setup/Installation Instructions (for both client and server).
    - List of Technologies Used.
    - A test account for each role (**Admin, Manager, Employee**).
