**Requirements**

**To break down the requirements for your system, let's categorize them based on the key features and functionality:**

**1. User Authentication and Authorization**

* **Sign In:** 
  + **Email & Password Verification: The system must verify the user’s email address and password during sign-in.**
  + **Error Handling: If the email or password is incorrect, or if the user does not exist, the system should display an appropriate error message.**
* **Role Management:** 
  + **After successful sign-in, the system stores the user role, user ID, and user name for future use.**
  + **The system must differentiate between Managers and Employees based on their role and allow for role-specific actions (e.g., task creation and viewing tasks).**

**2. User Registration**

* **Role-based Registration:** 
  + **Users must be able to register as either a Manager or Employee.**
  + **The Manager role allows the user to view Directors in the dropdown of possible managers.**
  + **The Employee role allows the user to view Managers in the dropdown of possible managers.**
* **User Data Storage:** 
  + **The system stores the user data (email, password, role, and other details) in the MongoDB database.**

**3. Task Management**

* **Task Creation:** 
  + **Both Managers and Employees can create tasks, but the task creation capabilities are role-dependent:** 
    - **Managers can assign tasks to themselves or their employees.**
    - **Employees can only assign tasks to themselves.**
* **Task Viewing and Editing:** 
  + **Both Managers and Employees should be able to view and edit their own tasks.**
  + **Managers should be able to view and edit tasks assigned to their employees.**
* **Task Deletion:** 
  + **Users should be able to delete their tasks if needed. Managers can delete tasks assigned to employees, while Employees can delete their own tasks.**
* **Task Filtering:** 
  + **Users should be able to filter tasks by user role (e.g., viewing tasks assigned to their employees or themselves).**

**4. Real-time Notifications**

* **Task Notification:** 
  + **When a new task is created, the system must emit a real-time notification using Socket.IO.**
  + **Notifications should be displayed on the client side in real-time without the need for a page refresh.**
* **Socket Event:** 
  + **The newTask event should be triggered from the server whenever a task is created.**
  + **The client should listen for the newTask event and display a notification on the screen.**

**5. Role-based Task Access**

* **Manager:** 
  + **Can create, view, edit, and delete tasks assigned to themselves and their employees.**
  + **Can assign tasks to employees within their team.**
* **Employee:** 
  + **Can only create, view, edit, and delete their own tasks.**
  + **Cannot assign tasks to other users (only to themselves).**

**6. Frontend Components:**

* **Homepage.js:** 
  + **Displays the options for Register and Sign In.**
  + **On sign-in, stores the user’s role, ID, and name in the application’s state.**
* **CreateTask.js:** 
  + **Allows users to create new tasks.**
  + **Displays role-based options for task assignment (Managers can assign tasks to others, Employees can only assign tasks to themselves).**
* **ViewTask.js:** 
  + **Displays details of a task when clicked.**
* **EditTask.js:** 
  + **Allows editing of task details.**
* **Real-Time Notification:** 
  + **Displays notifications when a new task is created, notifying the user in real-time.**

**7. Database (MongoDB)**

* **User Data Storage:** 
  + **Store user information including email, password, role, and other details in the MongoDB database.**
* **Task Data Storage:** 
  + **Tasks should be stored in the database with fields such as task name, description, assigned user, status, and due date.**
  + **Ensure task assignments are role-specific, so only authorized users (e.g., Managers) can assign tasks to employees.**

**8. CORS Configuration**

* **Cross-Origin Resource Sharing (CORS) must be configured for your system to allow requests from a specific origin (e.g., http://localhost:3000) for testing purposes.**
* **Ensure that CORS is set up to allow necessary HTTP methods (GET, POST, PUT, DELETE) and headers (like Content-Type) for secure and flexible communication between the client and the server.**

**9. Error Handling and Validation**

* **Form Validation:** 
  + **The system should validate that required fields (e.g., email, password) are filled before allowing the user to submit forms for sign-in and registration.**
* **Error Messages:** 
  + **Provide user-friendly error messages when the user inputs incorrect data (e.g., wrong email or password during sign-in, or invalid task details).**
  + **Handle server errors and client-side errors gracefully.**

**Summary of Requirements:**

1. **Authentication: Users can sign in and register. During sign-in, email and password are verified. User roles are set during registration and stored for role-based access.**
2. **Role-based Access: Managers can assign tasks to employees, while employees can only assign tasks to themselves. Managers can view tasks assigned to employees, while employees only see their own tasks.**
3. **Task Management: Users can create, view, edit, and delete tasks, with access rights based on roles.**
4. **Real-time Notifications: New tasks trigger real-time notifications using Socket.IO to notify users instantly when a task is created.**
5. **Database: User and task data is stored in MongoDB.**
6. **Frontend Components: Includes Homepage, CreateTask, ViewTask, and EditTask components.**
7. **CORS: Proper CORS setup for handling cross-origin requests.**
8. **Error Handling and Validation: Ensures that errors are handled gracefully with appropriate feedback for the user.**

**By implementing these requirements, your system ensures that only authorized users can perform specific actions, real-time notifications keep users updated about new tasks, and the database stores all necessary user and task information.  
Explain the components involved and how they interact.   
1. Homepage (Homepage.js):**

* **Role Selection and Authentication:** 
  + **On the homepage, users have two options: Register and Sign In.**
  + **Sign In: The user enters their email address and password. The system checks the User collection in MongoDB to see if a user exists with that email address. If the user is found:** 
    - **It verifies the password against the stored password (likely hashed).**
    - **If the email or password is incorrect, an error message is displayed.**
    - **If no user exists with the given email, an error message is displayed as well.**
  + **After successful sign-in, the system fetches and stores important details like user role, user ID, and user name in the state or context (e.g., userRole, userId, userName).**
  + **Register: During registration, the user can select their role (either Manager or Employee). If the role is Manager, they can only see Directors in the dropdown of potential managers. If the role is Employee, they can only see Managers from the User list in the MongoDB database.**

**2. User Roles:**

* **Manager Role: A manager can create tasks for themselves and their employees. They can view, edit, or delete tasks assigned to them and their employees.** 
  + **When creating a task, a manager can assign the task to themselves or any employee under their management.**
  + **The manager can see tasks that have been assigned to their employees as well as tasks they have assigned to themselves.**
* **Employee Role: An employee can only create and assign tasks to themselves. They cannot assign tasks to other employees.** 
  + **They can view their own tasks, edit them, or delete them if necessary.**

**3. Task Management:**

* **Once signed in, a user (either Manager or Employee) can create tasks.**
* **Manager: They can create tasks for themselves or assign tasks to any employee they manage.**
* **Employee: They can only create tasks for themselves.**
* **Task Viewing, Editing, and Deleting:** 
  + **When the user clicks on a task in their dashboard, they are directed to the respective component to view, edit, or delete the task.**
  + **This is handled through components like ViewTask.js and EditTask.js.**

**4. Creating Tasks (CreateTask.js):**

* **When a user clicks on Create Task, the page is redirected to the CreateTask.js component, where they can enter the task details.**
* **The system ensures that the user can only assign tasks based on their role:** 
  + **A Manager can assign tasks to both themselves and employees.**
  + **An Employee can only assign tasks to themselves.**
* **Notifications:** 
  + **After a task is successfully created, the system emits a real-time notification using Socket.IO to notify all connected users about the new task.**
  + **The notification will appear on the screen, alerting the user that a task has been created.**

**5. Components Involved:**

* **Homepage.js:**
  + **Handles the Sign In and Register functionality.**
  + **Fetches the user's role, ID, and name after successful sign-in and stores it in the state.**
  + **Provides the logic for displaying appropriate manager options based on the user role.**
* **CreateTask.js:**
  + **This component is used for creating new tasks.**
  + **It checks if the user is a Manager or Employee and displays the appropriate options for task assignment.**
  + **Once the task is created, it sends real-time notifications to other users.**
* **ViewTask.js and EditTask.js:**
  + **These components allow the user to view, edit, or delete tasks. The specific tasks shown are based on the user role and the tasks assigned to them (as a Manager or Employee).**
* **Real-time Notifications:**
  + **When a task is created, a notification is emitted via Socket.IO. The client listens for this event and displays the notification in real-time.**

**6. Real-Time Notifications:**

* **When a task is created by either a Manager or Employee, the server emits a newTask event.**
* **The client-side listens for the newTask event. Upon receiving this event, the task is shown as a real-time notification.**
* **This provides immediate feedback to the users about new tasks and keeps everyone updated without needing to refresh the page.**

**7. How the System Interacts:**

* **Sign In:**
  + **The user enters their email and password.**
  + **The server checks the User collection in MongoDB for a matching email and verifies the password.**
  + **If successful, the system stores the user’s role, user ID, and user name.**
  + **Depending on the role, the system will determine the user’s ability to create and assign tasks.**
* **Register:**
  + **The user fills in their registration form, including role selection.**
  + **If they select Manager, they can only view Directors in the manager dropdown.**
  + **If they select Employee, they can only view Managers in the manager dropdown.**
* **Task Creation:**
  + **Once signed in, the user can create tasks in CreateTask.js.**
  + **The task creation logic ensures that Managers can assign tasks to themselves and their employees, while Employees can only assign tasks to themselves.**
* **Task Viewing/Editing:**
  + **When the user clicks on a task, they are directed to either the ViewTask.js or EditTask.js component, where they can make changes to the task or view its details.**
* **Real-Time Notifications:**
  + **Once a task is created, the server emits the newTask event.**
  + **The client receives the event and displays a notification about the new task on the user’s screen.**

**Summary:**

* **Sign-In verifies the user's email and password.**
* **Register allows users to choose a role (Manager or Employee) and restricts manager options based on the role.**
* **Manager can create tasks for themselves and their employees, while Employees can only create tasks for themselves.**
* **Real-time Notifications are triggered upon task creation to notify users instantly about new tasks. This is handled via Socket.IO for real-time communication.**
* **Task Management allows viewing, editing, and deleting tasks, which can be done via components like CreateTask.js, ViewTask.js, and EditTask.js.**

**Describe how the system handles real-time notifications.**

**1. Server-Side (Real-Time Task Notifications):**

* **Socket.IO Initialization**:
  + In your socket.js file, the Socket.IO server is initialized. This creates a WebSocket server that listens for events and can emit messages to connected clients. The server is linked to the HTTP server to handle both HTTP and WebSocket requests.
  + This Socket.IO server is then exported and used in your main server (index.js).
* **Task Creation & Emitting Event**:
  + When a new task is created, the server saves the task in MongoDB via the TaskList model. After successfully saving the task, the server emits a **newTask** event using io.getIO().emit("newTask", newTask);.
  + The **newTask** event sends the newly created task object to all connected clients.

**2. Client-Side (Receiving Real-Time Task Updates):**

* **Connecting to Socket.IO**:
  + On the client side, the application connects to the Socket.IO server, typically inside a component like App.js or the relevant place where you set up the WebSocket connection.
  + The client listens for the **newTask** event, which will be emitted by the server whenever a new task is created.
* **Handling the newTask Event**:
  + Once the client receives the **newTask** event, it triggers a callback function that processes the received task. This is usually handled in a state update or a similar mechanism to display the task in the UI without needing to refresh the page.
* **Example in createTask.js**:
  + Inside the handleSubmit function of createTask.js (the component responsible for creating tasks), you likely call the server to create the task, and once the task is successfully created (and the task is saved to MongoDB), the server emits the **newTask** event.
  + When the **newTask** event is emitted from the server, it is caught by the socket listener on the client-side. The listener function processes this event, typically updating the UI by adding the new task to the list of tasks.

**Flow of Events:**

1. **User Creates Task**: The user submits the form to create a task via the handleSubmit function in createTask.js.
2. **Server-Side Task Creation**: The server processes the task creation (saving to MongoDB) and emits the **newTask** event to all connected clients.
3. **Client-Side Event Listening**: The client listens for the **newTask** event through the established WebSocket connection. When the event is received, the client updates the state (or UI) with the newly created task.
4. **UI Update**: The newly created task appears in real-time on the client without needing to refresh the page, providing a smooth user experience.