Overview
This project is a basic task management system intended to be used by people from a managerial perspective. That is, the system should be able to handle tasks with set duration such as facilities booking, or simply serve as a general to-do list.

User Interface
The application user interface should be intuitive and uncluttered. The ultimate goal is that users should be able to perform actions with minimal clicks and not need to worry about managing unnecessary settings.

It will be implemented in React.js.

Ideally, there will be 2 main views: a to-do list view and a scheduler view. The to-do list view will consist of clickable cards, organised by category. Users will be able to assign each task a category for better organisation, but their default category upon creation would be "Uncategorised". A good implementation of this could be a whiteboard-style screen, scrollable downwards. The user could then create new categories and then drag and drop the cards into their respective categories. This would provide a fast way of categorising tasks, though it could get difficult if many categories are created. Such an implementation could also be difficult to implement in the allocated time.

A single task card will consist of a name (text box), an optional description (text box), an optional priority (dropdown), and an optional duration (datetime picker). These can be easily editable after creation by clicking on the cards and accessing the task setting screen.

The scheduler view will consist of a calendar with a daily view by default, but can be changed to a monthly view. The scheduler view will retrieve all tasks with the duration specified and display them as bars covering the specified duration. If the duration is specified, the date is a mandatory field but the time is not. If the time field is left blank, the task will be displayed as an "All-day" task, covering the entire day. The bars can also be clicked, bringing up the same task setting screen as in the to-do list view. Upon modifying a task's settings, the changes should be reflected immediately. The scheduler could be implemented using the KendoReact scheduler component. In the interests of time, this component could be left out on final submission.

Back-end
The back-end will be implemented in Ruby on Rails, using a SQLite3 database. The database will contain 3 main tables, though the whole database will contain more to follow normalisation forms.

Table name	Main fields
Users	UserID (UUID), Username, password (salted and hashed ideally), whiteboardIDs
Whiteboards	WhiteboardID, OwnerUserID, TaskIDs
Tasks	TaskID, Name, Description, Priority, Duration

Input sanitization must be performed to maintain security (parametrization to prevent SQL injection etc). This will be done to best effort as there will not be time for security testing.

The following are routes that will be set up:

1) /
    - This will host the main page, and the user can toggle between the to-do list views and scheduler views here.
2) /login
    - Login page. Users will specify username and password, which will be checked against the database. Upon successful login, an auth-cookie will be generated. Details of the auth-cookie are TBD.
3) /register
    - Users can register a new account here, specifying a username and password. **Usernames must be unique**.
4) /logout
    - Logout page. Cookies maintained on the server will be deleted, though I'm not sure of the best practices here.
5) **GET** /api/tasks/{taskID}
    - Retrieve all details related to a task and return in JSON format.
6) **POST** /api/tasks/create
    - auth-cookie must be specified. WhiteboardID must be specified in the body. Depending on the implementation, user details (ID) must also be supplied in the body. All details will be sent in JSON format. A new task will be created in the database, and the whiteboards table will be updated to reflect the new task.
7) **PUT** /api/tasks/update/{taskID}
    - Updates task details on a whiteboard. Task details will be specified, as well as all fields specified in route 6 in the body of the request. The task table will be updated.

8) **DELETE** /api/tasks/delete/{taskID}
    - Deletes a task from a whiteboard. The body of the request will be equivalent to route 6. The task and whiteboard tables will both be updated.
9) **GET** /api/whiteboards/{whiteboardID}
    - Gets a whiteboard and all its tasks for display, returned in JSON format.
10) **POST** /api/whiteboards/create
    - auth-cookie must be specified. Depending on the implementation, user details (ID) must also be supplied in the body. All details will be sent in JSON format. A new whiteboard will be created in the database, and the whiteboards table will be updated.
11) **PUT** /api/whiteboards/update/{whiteboardID}
    - Implementation details TBD.
12) **DELETE** /api/whiteboards/delete/{whiteboardID}
    - Whiteboard will be deleted. Whiteboards table will be updated, as well as all tasks in the whiteboard.

Regarding authentication, I don't think it is feasible to get a certificate for HTTPS, therefore, username and password information will be sent in plaintext. Obviously this isn't very secure, and certainly security can be vastly improved for this application, but this is more of a proof-of-concept.