Objective:

The purpose of this test is to assess the candidate's proficiency in building a full-stack web application using React (frontend), Node.js (backend), and TypeScript (on both frontend and backend). The candidate will be required to create a basic Kanban board, implementing key features such as creating, updating, deleting tasks, and persisting the data on the server.

Test Requirements:

The candidate should complete the following tasks:

1. Frontend: React + TypeScript

- Build a simple Kanban board interface using React and TypeScript.
- The interface should have the following columns:
  - **To Do**
  - **In Progress**
  - **Done**
- Each column will display tasks with basic details like task title, description, and status.
- The user should be able to:
  - **Add** a new task to the "To Do" column.
  - **Drag and drop** tasks between the columns.
  - **Edit** a task’s title and description.
  - **Delete** a task.

2. Backend: Node.js + Express + TypeScript

- Build a REST API using Node.js, Express, and TypeScript to handle the following:
  - **Create a task**: Store a new task with title, description, and status.
  - **Retrieve all tasks**: Get all tasks to populate the Kanban board.
  - **Update a task**: Modify an existing task's title, description, or status (e.g., moving a task from "To Do" to "In Progress").
  - **Delete a task**: Remove a task from the board.
- Use in-memory storage for simplicity (e.g., an array), but provide clear separation between the model, controller, and routes.

3. State Management & Persistence

- Use React's state to manage the tasks on the frontend.
- The frontend should fetch the initial task data from the backend API.
- Ensure that when tasks are added, updated, or deleted on the frontend, the changes are reflected in the backend and vice versa (persistence).

4. Styling

- Use a modern UI framework like **Radix UI or Mantine UI** for styling.
- Ensure the Kanban board is responsive and works well on different screen sizes.

Bonus Points:

- Use **React DnD** or any similar library to implement the drag-and-drop functionality between the columns.
- Write basic **unit tests** for React components (using **Jest** or **React Testing Library**).
- Use **ESLint** and **Prettier** for consistent code formatting.
- Write basic tests for the backend using **Jest** or **Mocha**.
- Provide a **docker-compose** file to set up the development environment (frontend, backend, and a database if desired).
- Use **TypeORM** or a similar ORM for backend models, even if it's in-memory or with SQLite.

Deliverables:

1. **Front-end code** (React + TypeScript) with functionality to create, edit, delete, and move tasks on the Kanban board.
2. **Backend code** (Node.js + Express + TypeScript) with REST API endpoints for task management.
3. Detailed **README** file explaining:
    - How to set up and run the project locally.
    - Any assumptions or decisions made.
    - How to run any tests included.
4. Optional but encouraged: Dockerfile and/or docker-compose setup for both frontend and backend.

Evaluation Criteria:

1. **Code Quality**:
   - Proper use of TypeScript on both frontend and backend.
   - Well-structured, modular code.
   - Adherence to best practices for API design and React component design.
2. **Functionality**:
   - The application should work as described.
   - All core features (add, update, delete, drag-and-drop) should be functional.
3. **UI/UX**:
   - Clean, responsive design.
   - User-friendly interactions (drag-and-drop, editing, etc.).
4. **Documentation**:
   - Clear instructions on setting up the project and running the application locally.
   - Explanations of any non-obvious implementation choices.
5. **Bonus Features** (if implemented):
   - Use of state management libraries or patterns like Zustand (if required).
   - Unit tests and other automated tests.
   - Dockerization of the project for easier setup.

Time Estimate:

Candidates should aim to complete the task within 5-6 hours. However, they can take additional time to refine their code, implement bonus features, and write documentation.

This technical test is designed to simulate a typical full-stack project, assessing the candidate's ability to deliver a complete solution from both the frontend and backend perspectives.

You can use skeletons or scaffolding tools to setup the project initially, we recommend Typescript + React + Vite as stack.