Test Objective

Build a small full-stack application with the following requirements:

- **Frontend**: Next.js application
- **Backend**: NestJS API
- **Database**: Use MySQL

Scenario

Create a **Task Management System** where users can:

1. Create an account.
2. Log in.
3. Create, read, update, and delete tasks.
4. Each task should include:
    - Title
    - Description
    - Status (e.g., “To Do”, “In Progress”, “Completed”).
    - Due Date.

Requirements

Backend (NestJS)

1. **Endpoints**:
    - User Authentication:
        - POST /auth/register: Register a user.
        - POST /auth/login: Log in and return a JWT token.
    - Tasks:
        - GET /tasks: Get all tasks for the logged-in user.

- POST /tasks: Create a new task.
- PUT /tasks/:id: Update a task by ID.
- DELETE /tasks/:id: Delete a task by ID.

2. **Authentication**:
    - Use JWT for user authentication.
    - Protect all /tasks endpoints to ensure only logged-in users can access them.
3. **Database**:
    - Use MySQL.
    - Two tables:
        - **Users**: id, email, password (hashed).
        - **Tasks**: id, title, description, status, due_date, user_id (foreign key).
4. **Validation**:
    - Validate request payloads (e.g., using class-validator).

Frontend (Next.js)

1. **Pages**:
    - /register: Form to create an account.
    - /login: Form to log in.
    - /tasks: Dashboard to view tasks.
    - /tasks/new: Form to create a new task.

- /tasks/:id/edit: Form to edit an existing task.

2. **State Management**:

- Use React Context or a state management library (Redux) to store the logged-in user's authentication state.

3. **API Integration**:

- Connect to the NestJS API endpoints.
- Handle error states (e.g., invalid login credentials).

4. **UI Design**:

- Use a minimal UI framework like TailwindCSS or Material-UI.