Assignment: "Project Tracker" — A NestJS + PostgreSQL REST API

Objectives

- Scaffold a NestJS application.
- Connect to PostgreSQL using `@nestjs/typeorm` (TypeORM + `pg` driver)
- Design and implement a **Project Tracker** domain featuring users, projects, and tasks.
- Enforce clean architecture with modules, controllers, services, and DTOs.
- Apply validation, error handling, logging, and basic testing.

Feature Requirements

1. Database Entities & Relationships

- **User**: fields `id`, `username`, `email` (unique), `password` (hashed).
- **Project**: fields `id`, `name`, `description`, `owner` (relation to User).
- **Task**: fields `id`, `title`, `description`, `status`, `project` (relation).

Relations:

- A User owns many Projects.
- A Project has many Tasks.

2. Authentication & Authorization

- Implement **registration** (`POST /auth/register`) and **login** (`POST /auth/login`) using **JWT** (via Passport and JWT strategy).
- Only authenticated users can access Projects and Tasks.
- A user can manage (CRUD) only their own Projects and Tasks.

3. CRUD Operations

- **Users**: registration/login endpoints.
- **Projects**: create, list, retrieve (by id), update, delete.
- **Tasks**: allow CRUD within the context of a specific project (e.g., `/projects/:projId/tasks`).

4. Data Validation & Error Handling

- Use **DTOs** with class-validator decorators (e.g., `@IsString`, `@IsEmail`) for input validation.
- Implement exception filters for clean error messages.
- Return proper HTTP codes (e.g., 201 Created, 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found).

5. Architecture & Modularity

- Organize code into modules: AuthModule, UsersModule, ProjectsModule, TasksModule.
- Leverage dependency injection and proper use of services, controllers, repositories.

6. Logging & Configuration

- Use NestJS's built-in **Logger**
- Manage environment variables using `@nestjs/config` and `.env` files.

7. Testing

- Write unit tests for at least one controller or service using Jest.
