🗓️ Week 3: PostgreSQL + Prisma Integration

🌟 Goal

Understand relational databases (PostgreSQL) and learn how to integrate them into a NestJS backend using Prisma ORM, with a focus on building a multi-model API using relational data and basic authentication.

📓 Topics Covered

1. PostgreSQL Basics

✅ Concepts

- What is a relational database?
- PostgreSQL overview and common use cases
- Basic SQL commands:
    - **DDL**: `CREATE`, `ALTER`, `DROP`
    - **DML**: `INSERT`, `SELECT`, `UPDATE`, `DELETE`
- Constraints: `PRIMARY KEY`, `FOREIGN KEY`, `NOT NULL`, `UNIQUE`
- Data types: `INT`, `VARCHAR`, `BOOLEAN`, `TIMESTAMP`, etc.
- Joins: `INNER JOIN`, `LEFT JOIN`

📒 Study Material

- PostgreSQL Official Docs
- SQLZoo
- SQL Bolt

---

2. Setting up PostgreSQL

- Install PostgreSQL locally or via Docker
- Connect PostgreSQL to PGAdmin
- Create a database and user with proper access roles

- Connect PostgreSQL to a NestJS project via Prisma

---

3. Prisma ORM

✅ Concepts

- What is Prisma?
- Prisma schema: `model`, `datasource`, `generator`
- Setting up Prisma in a NestJS project
- Defining models and generating the client
- Common Prisma CLI commands:
    - `npx prisma init`
    - `npx prisma migrate dev`
    - `npx prisma generate`
- Performing CRUD operations using Prisma Client

📒 Study Material

- Prisma Docs
- Build a REST API with NestJS and Prisma – FreeCodeCamp

🔧 Practice

- Setup Prisma with PostgreSQL database
- Create `User` and `Task` models (schema.prisma)
- Generate migrations and apply to PostgreSQL
- Use Prisma Client to:
    - Create a new user/task
    - Fetch all users/tasks
    - Update and delete entries
- Write service methods using Prisma Client
- Connect services to NestJS controllers
- Implement authentication using `passport` and `JWT`

📊 Projects & Assignments

🏦 Mini Project

Title: Users, Tasks, Projects & Comments API with Auth

Goal: Build a complete REST API where users can manage their own tasks, projects, and comments, backed by PostgreSQL and Prisma ORM, and secured with JWT-based authentication.

🧹 **Models:**

- **User**: `id`, `name`, `email`, `password`
- **Task**: `id`, `title`, `description`, `status`, `userId`, `projectId`
- **Project**: `id`, `name`, `description`, `userId`
- **Comment**: `id`, `content`, `taskId`, `userId`, `createdAt`

🔗 **Relationships:**

- A **User** can have **many Tasks** (One-to-Many)
- A **User** can create **multiple Projects**
- A **Project** can have **many Tasks** (One-to-Many)
- A **Task** can have **many Comments**
- A **User** can write **many Comments**

📌 **Endpoints:**

Auth:

- `POST /auth/register` — Register a new user
- `POST /auth/login` — Login and receive JWT

Users:

- `GET /users` — List all users (admin only)
- `GET /users/:id` — Get one user
- `GET /users/:id/tasks` — Get tasks of a user
- `GET /users/:id/projects` — Get projects created by user
- `GET /users/:id/comments` — Get comments by user

Tasks (require JWT):

- `POST /tasks` — Create a task for logged-in user
- `GET /tasks` — List all tasks for logged-in user
- `GET /tasks/:id` — Get a task by ID
- `PATCH /tasks/:id` — Update task
- `DELETE /tasks/:id` — Delete task
- `GET /tasks/:id/comments` — List all comments for a task

Projects (require JWT):

- `POST /projects` — Create a project
- `GET /projects` — List all projects for logged-in user
- `GET /projects/:id` — Get project details
- `PATCH /projects/:id` — Update project
- `DELETE /projects/:id` — Delete project

Comments (require JWT):

- `POST /comments` — Add a comment to a task
- `PATCH /comments/:id` — Update a comment (author only)
- `DELETE /comments/:id` — Delete a comment (author or admin)

✅ Requirements:

- Use Prisma schema to define models and relationships
- Migrate schema using `prisma migrate`
- Implement DTOs and validation in NestJS
- Secure endpoints using `passport-jwt`
- Use guards and interceptors for route protection
- Return meaningful errors and success responses
- Optional: Add `status` enum to `Task` with values like `pending`, `in_progress`, `done`

📦 Deliverables:

- Complete codebase on GitHub
- Well-structured and modular code
- Clean commit history
- `README.md` with setup instructions and API docs
- Swagger documentation