🗓️ Week 2: NestJS + TypeScript

🎯 Goal

Understand the fundamentals of TypeScript and NestJS, and use them to build a scalable and modular REST API with validation and testing.

📓 Topics Covered

1. TypeScript Basics

✅ Concepts

- Type annotations: `string`, `number`, `boolean`, `any`, `unknown`, `void`, `never`
- Type Inference and Type Aliases
- Interfaces vs Types: When and how to use
- Union & Intersection Types
- Enums and Literal Types
- Functions with typed parameters and return types
- Classes, Inheritance, Access Modifiers (`public`, `private`, `protected`)
- Generics: Functions and Classes
- Modules and ES6 Imports/Exports
- Working with `tsconfig.json`

📚 Study Material

- TypeScript Handbook

- [TypeScript Crash Course - Traversy Media]()
- [TypeScript for Beginners - Academind]()

🛠️ Practice

- Convert JavaScript functions to TypeScript (5 examples)
- Define and implement an interface for a User object

---

2. NestJS Fundamentals

✅ Concepts

- What is NestJS and why use it?
- Project setup using Nest CLI (`npm i -g @nestjs/cli`, `nest new project-name`)
- Understanding the core architecture:
  - Modules
  - Controllers
  - Services
- How Dependency Injection works in NestJS
- Routing and Request Methods (GET, POST, DELETE, PATCH, PUT)
- Data Transfer Objects (DTOs) using TypeScript
- Validation using `class-validator` and `class-transformer`
- Middleware basics: Logging
- Basic Pipes: Transformation and Validation pipes

📚 Study Material

- NestJS Docs

- nestJs Basics

- Nestjs Tutorial
- NestJS Crash Course - Code with Mosh
- Build a REST API with NestJS - FreeCodeCamp

🛠️ Practice

- Create a new NestJS project
- Build a `Users` module with controller and service
- Use DTOs for request validation
- Setup request logging middleware
- Test endpoints (you can use Postman)

💪 Projects & Assignments

✅ Practice Tasks

- Create a simple `User` DTO with validation (name, email, age)
- Create a reusable validation pipe
- Add custom error messages using class-validator decorators
- Simulate a user database with an in-memory array
- Explore dynamic routes with `@Param()` and `@Query()` decorators

📦 Mini Project

Title: User/Product Management API (In-Memory)

Goals:

- Build a full-featured CRUD REST API using NestJS
- Apply modular design and DTO-based validation
- Practice building, testing, and documenting endpoints

Requirements:

- Create a `users` or `products` module
- CRUD Endpoints:
    - `GET /users` or `GET /products` — list all entries
    - `GET /users/:id` — get one entry by ID
    - `POST /users` — create a new entry with DTO validation
    - `PATCH /users/:id` — update specific fields
    - `DELETE /users/:id` — remove entry
- Use in-memory array for data persistence
- Use `class-validator` decorators for validation
- Use `@Param`, `@Body`, `@Query` appropriately
- Add basic error handling for not-found and bad input
- Include logging middleware (optional)
- Add Swagger documentation (optional)

Deliverables:

- Full codebase pushed to GitHub
- Clean and meaningful commit messages
- A `README.md` explaining how to run the app and API endpoints
- Optional: Postman/Thunder Client collection for API testing

---

🚀 Advanced Task

Title: Modular Task Management API

Goal: Learn how to structure larger features and use advanced DTO logic.

Requirements:

- Create a `Tasks` module separate from `Users`
- Endpoints:
  - `GET /tasks`
  - `POST /tasks`
  - `PATCH /tasks/:id`
  - `DELETE /tasks/:id`
- Fields: `title`, `description`, `status` (enum: `pending`, `in_progress`, `done`)
- Use DTOs for create and update operations
- Add custom validation for `status`
- Use service methods for all logic, controller only for routing
- Modular folder structure

- Add Logger middleware
- Push code to GitHub with `README.md`

📝 End of Week Checklist

- TypeScript basics practiced and understood
- Able to create and explain DTOs in NestJS
- Successfully built and tested a modular REST API
- Project pushed to GitHub with a proper README
- Mini project reviewed and merged