NestJS REST API Project Structure

Full REST API with validation, OpenAPI documentation, TypeORM database, and proper module separation.

#nestjs #typescript #nodejs #api #rest #swagger #typeorm

PNG PDF Copy Prompt

Project Directory

```
myproject/
  src/ Application sou…
    main.ts Swagger setup, …
    app.module.ts Root module
    config/ Configuration
      configuration.ts Typed config
      database.config.ts
      swagger.config.ts
    common/ Shared utilities
      decorators/
        public.decorator.ts
      filters/
        http-exception.filter.ts Global error ha…
      interceptors/
        transform.interceptor.ts Response wrappi…
      guards/
        auth.guard.ts
      dto/
        pagination.dto.ts
    modules/ Feature modules
      users/
        users.module.ts
        users.controller.ts
        users.service.ts
        dto/
          create-user.dto.ts class-validator
          update-user.dto.ts
        entities/
          user.entity.ts TypeORM entity
      auth/
        auth.module.ts
        auth.controller.ts
        auth.service.ts
        strategies/
          jwt.strategy.ts
          local.strategy.ts
    database/ TypeORM setup
      database.module.ts
      migrations/ TypeORM migrati…
        .gitkeep
  test/
    app.e2e-spec.ts
    jest-e2e.json
  nest-cli.json
  tsconfig.json
  tsconfig.build.json
  package.json
  .env.example
  .eslintrc.js
  .prettierrc
  .gitignore
  README.md
```

Why This Structure?

This structure separates concerns with feature modules in `modules/`, shared code in `common/`, and configuration in `config/`. Swagger provides auto-generated API docs, class-validator ensures type-safe validation, and TypeORM handles database operations.

Key Directories

`src/modules/` - Feature modules (users, auth, etc.) with their own DTOs and entities

`src/common/` - Shared guards, filters, interceptors, decorators

`src/config/` - Typed configuration using @nestjs/config

`src/database/` - TypeORM connection and migrations

Getting Started

1. `npm install`
2. `cp .env.example .env` (set DATABASE_URL)
3. `npm run migration:run`
4. `npm run start:dev`
5. Visit http://localhost:3000/api for Swagger UI

Module Anatomy

`*.module.ts` - Module definition, imports providers

`*.controller.ts` - HTTP routes, uses decorators

`*.service.ts` - Business logic, injects repository

`dto/` - Data Transfer Objects with validation

`entities/` - TypeORM database entities

When To Use This

- Production APIs requiring documentation
- Projects with database persistence needs
- Team-based development
- APIs needing authentication/authorization
- Microservices with well-defined contracts

Trade-offs

`More boilerplate` - Each feature needs module, controller, service, DTOs

`Learning curve` - NestJS decorators and DI take time to master

`TypeORM sync` - Use migrations in production, never synchronize: true