lam sao de biet toi dang di dung huong vi toi dang lam mot du an viet mot web dua tren phan mem ggdrive va toi dang o buoc viet unit test bang nest js dung typeorm de test case cho cac truong hop xay ra dua tren cau query ( toi da thiet ke xong database ) va sep toi bat buoc toi phai viet lai thu vien dependency injection thay vi dung thu vien co san cua nest js va tiep do la viet lai mo hinh mvc va ap dung no cho unit test cua toi

**ChatGPT đã nói:**

👌 mình hiểu rồi, để mình giúp bạn “xâu chuỗi” lại lộ trình nhé, vì nghe bạn mô tả thì bạn đang phải:

1. **Không được dùng DI sẵn có của NestJS** → bạn phải tự viết lại một DI container tối giản (như mình với bạn đang viết).
2. **Không được dùng MVC sẵn của NestJS** → bạn phải dựng lại MVC model:
   * Model = Entities (database + TypeORM).
   * View = tạm bỏ qua (nếu chỉ viết API backend).
   * Controller = định nghĩa route + gọi service.
   * Service = xử lý logic nghiệp vụ.
   * Repository = giao tiếp DB.
3. **Viết Unit Test** cho service, controller bằng Jest + TypeORM (in-memory / SQLite) với chính cái DI framework bạn đã viết lại.

**✅ Cách để biết bạn đang đi đúng hướng**

* Nếu bạn đã:
  + **Thiết kế database** xong (entities ok).
  + **Có DI Container** cơ bản (đăng ký class, resolve dependencies, có scope như singleton, transient, request).
  + **Có Decorator** cho @Injectable, @Controller, @Get, @Post…
  + **Có Unit Test** check resolve, check lifecycle của DI.

👉 Thì bạn **đang đi đúng hướng**, vì đây là bước mô phỏng lại NestJS framework ở mức core.

**🔑 Những phần bạn cần viết tiếp**

1. **Bổ sung Decorator cho Controller/Routes**: @Put, @Delete, @Patch, @Param, @Body, @Query.  
   → Điều này giúp bạn viết unit test controller dễ dàng.
2. **Router Loader**: viết 1 hàm đọc metadata từ @Controller + @Get… để map ra danh sách routes → từ đó test được input/output.  
   (Ở NestJS thật thì Express/Fastify đảm nhận, nhưng bạn chỉ cần in ra console hoặc simulate request trong test).
3. **Repository Layer**: vì bạn đang dùng TypeORM, bạn nên giữ Repository của TypeORM để khỏi viết lại quá nhiều.
4. **Unit Test Level**:
   * **Service Test**: test business logic độc lập (mock repository).
   * **Controller Test**: test route + call service (mock service hoặc dùng DI container để inject service thật).
   * **Integration-lite**: test controller + service + repository với SQLite in-memory.

**✅ Bạn đang ở bước nào?**

* Theo những gì bạn nói, bạn đã có:
  + Database xong.
  + DI container cơ bản (có @Injectable, Scope, Container.resolve).
  + Bắt đầu có Decorator cho controller (@Controller, @Get, @Post).
  + Unit test cơ bản cho DI.

👉 Tức là bạn đang ở **bước giữa**: đã có DI core, cần mở rộng decorator cho controller + chuẩn bị test service/controller thật.

**📌 Gợi ý hướng đi tiếp**

* **Bước 1**: Hoàn thiện decorators cho controller (@Put, @Delete, @Param, @Body…)
* **Bước 2**: Viết RouteExplorer để duyệt metadata từ controller → ánh xạ route (trong unit test bạn không cần Express, chỉ cần giả lập gọi).
* **Bước 3**: Viết unit test cho controller thật (gọi method, check trả về, check injection service).
* **Bước 4**: Kết nối TypeORM SQLite in-memory, viết test repository.
* **Bước 5**: Viết integration test kết hợp controller + service + repo.

👉 Mình hỏi thêm: bạn muốn mình viết tiếp **RouteExplorer** (bộ đọc metadata từ @Controller và các decorator như @Get, @Post) để có thể simulate gọi API trong unit test luôn không? Vì đây là bước cần để test controller thực sự.