**Introduction**

Nest (NestJS) là một framework để xây dựng các ứng dụng server-side bằng Node.js hiệu quả

**Controllers**

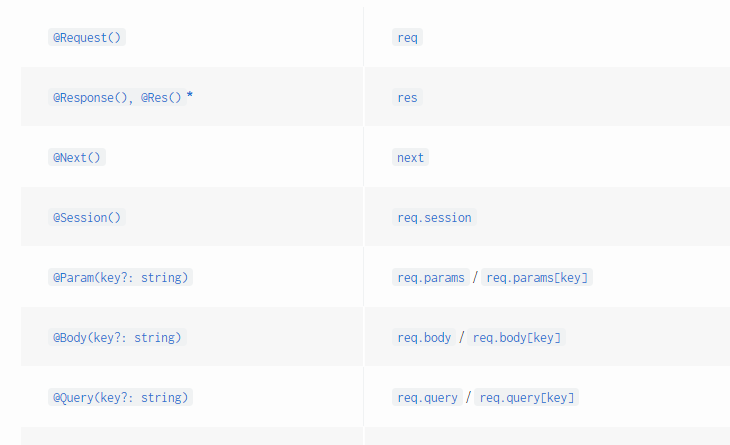
Controllers chịu trách nhiệm xử lý các request đến và response lại client. Giống như các framework khác mỗi controller đảm nhận xử lý các request đến 1 router.

## Routing

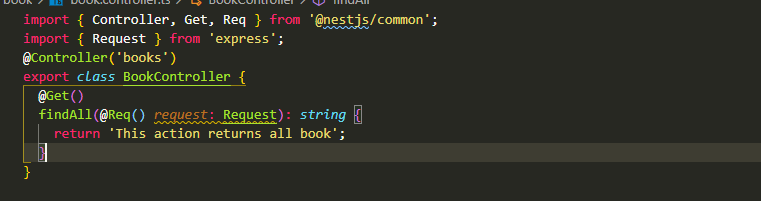
Định nghĩa một controller bằng cách dùng decorator @Controller sau đó truyền vào path prefix, cho phép chúng ta dễ dàng nhóm một nhóm các đường dẫn liên quan và giảm code lặp đi lặp lại

## Request object

Để truy cập vào request để lấy dữ liệu mà client gửi lên sử dụng Request object



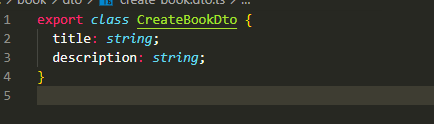
@Get() phía trên method findAll là 1 decorator để nói với Nest tạo 1 trình xử lý cho http request với method GET đến endpoint này

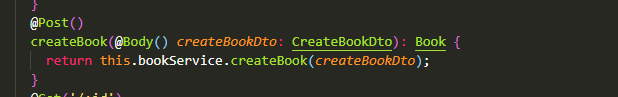


## Resources

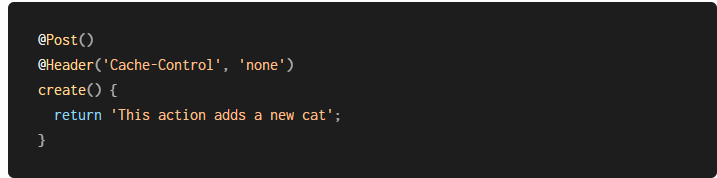
## Request payloads

Lấy dữ liệu trong body bằng cách sử dụng decorator @Body. Tuy nhiên cần định nghĩa 1 DTO



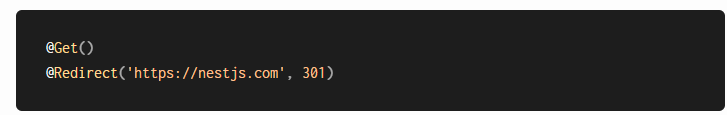


## Request header



## Route parameters

## Redirection



**Providers**

Provider là một khái niệm cơ bản trong Nest. Các services, helper đc xem như là provider

Provider liên kết các class lại với nhau

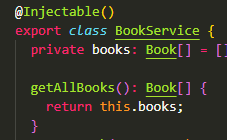
Controller nhận request sau đó giao nhiệm vụ cho Provider xử lý

## **Dependency injection**

* Là một design pattern
* Để cho controller inject vào thông qua constructor



* Để sử dụng cần dung decorator Injectable

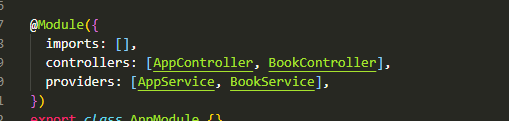


## **Scopes**

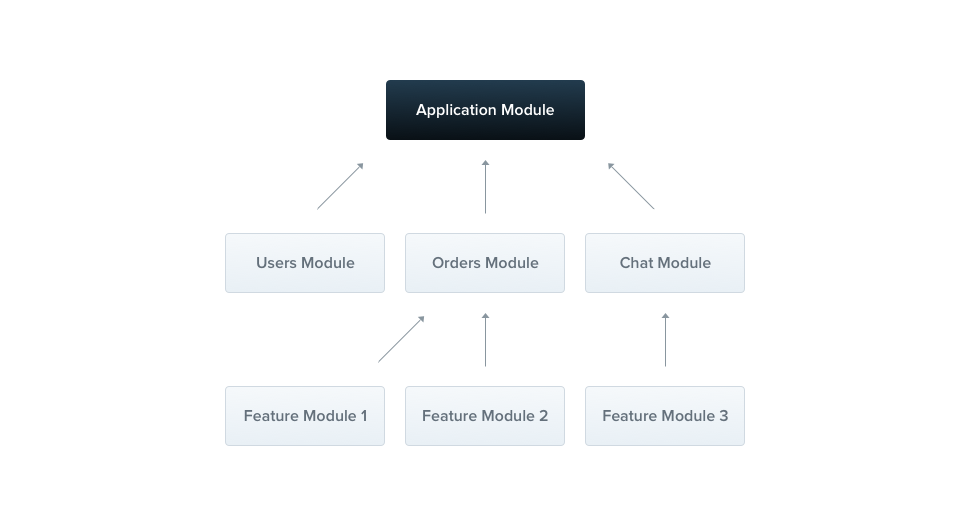
* . Khi một ứng dụng được khởi động, tất cả dependency phải được xử lý, do đó các provider phải được khởi tạo. Khi ứng dụng tắt thì provider bị hủy

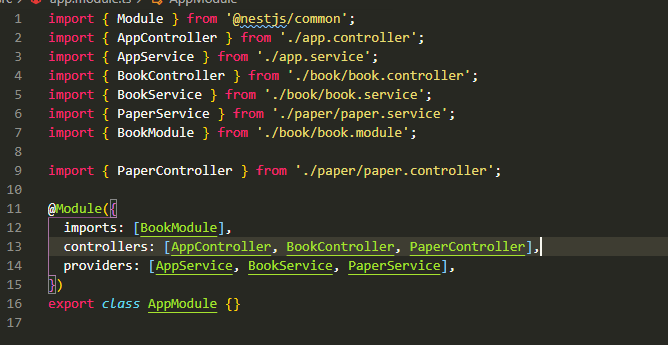
## **Provider registration**

 cần đăng ký provider với Nest để nó có thể thực hiện injection. Chúng ta làm điều này bằng các sửa AppModule module.



# Modules

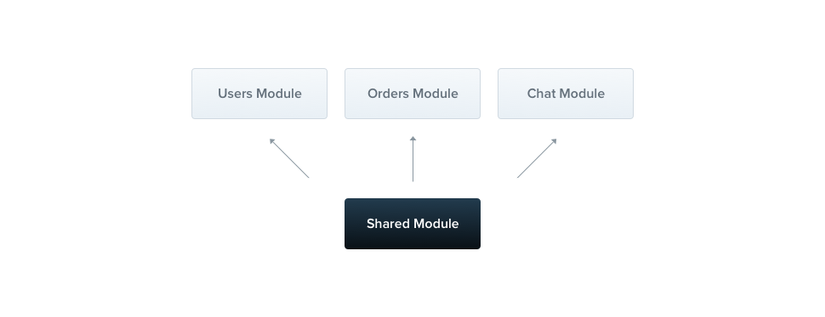
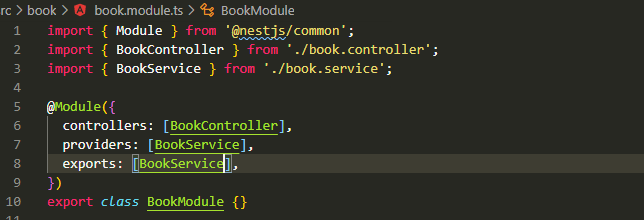
Một module là một class được gắn với một decorator @Module(). Decorator @Module() cung cấp metadata cho **Nest** sử dụng để tổ chức cấu trúc ứng dụng.



* providers - các providers sẽ được khởi tạo bởi Nest injector - Trình khởi tạo và có thể được chia sẽ trên toàn module này.
* controllers - các controller được định nghĩa trong module này, và cũng sẽ được khởi tạo bởi Nest injector.
* imports - các module khác sẽ được import vào module này. Sau khi import vào, module này có thể sử dụng những gì mà các module kia đã exports ra.
* exports - một bộ con các providers củamodule này sẽ mở ra cho các module khác có thể sử dụng khi chúng import module này

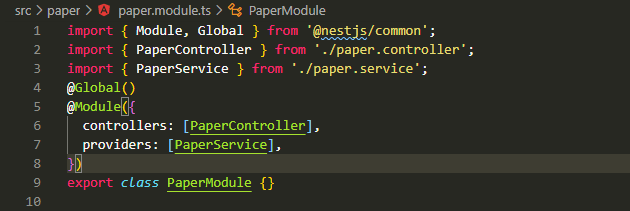
Mặc định module đóng gói các providers. Có nghĩa bên trong module chỉ có thể inject các providers của chính mình hoặc được exports từ các module đã import vào.

## **Shared modules**



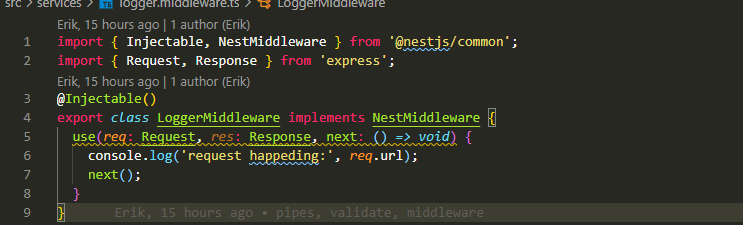
## **Global modules**

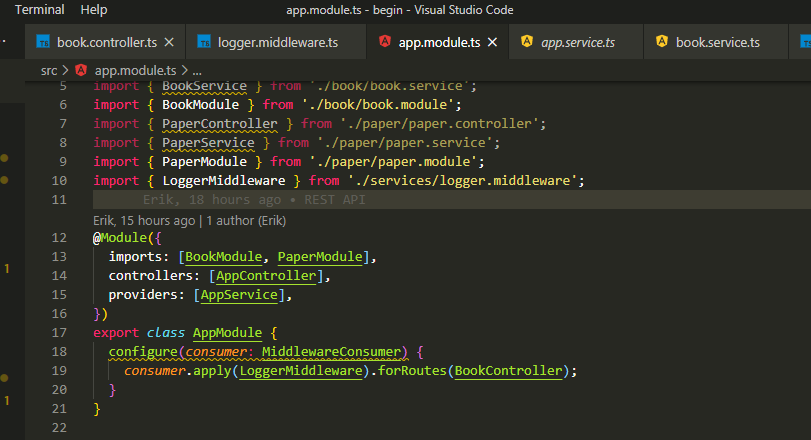
Để cung cấp một bộ các providers mà sẽ có sẳn ở tất cả mọi nơi (ví dụ, database connection, pagination...), hãy tạo một **global** module bằng các sử dụng decorator @Global()

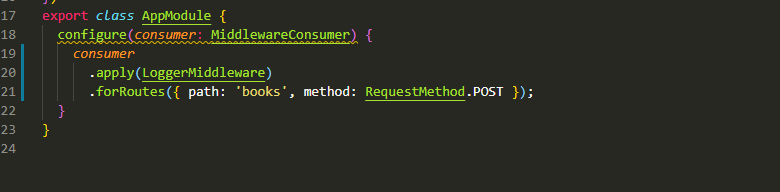


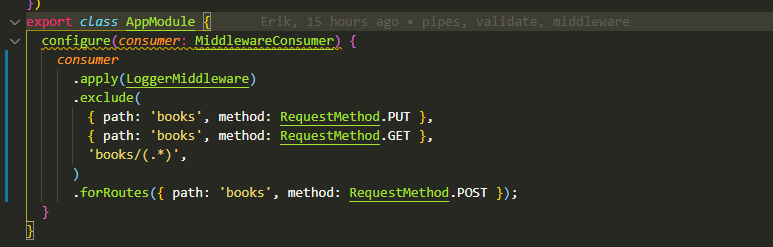
# Middleware

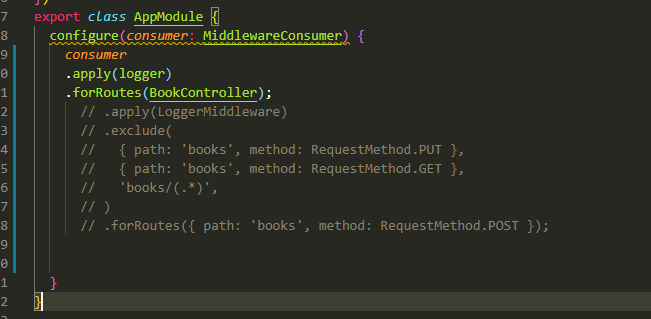
* Là một function được gọi trước khi đi vào route để xử lý
* Đứng giữa req và res
* Tại mỗi middleware có thể access và tùy chỉnh req, res
* Callback next() dùng để chuyển tiếp req sang middleware tiếp theo

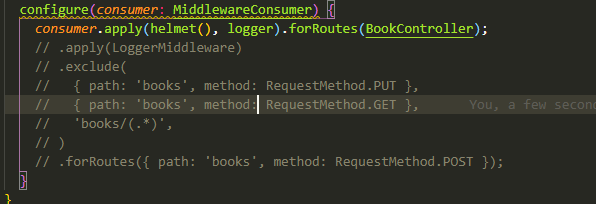


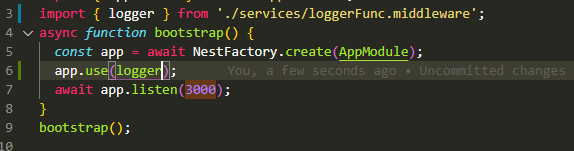






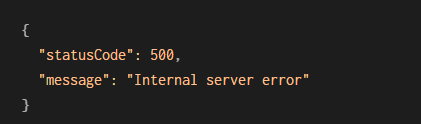






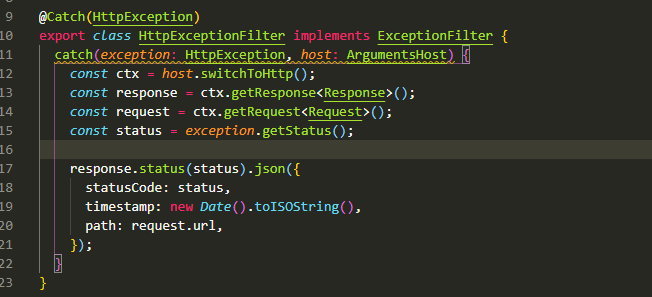
# [Exception filters](https://docs.nestjs.com/exception-filters)

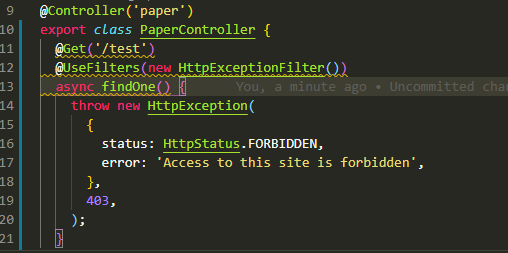
* Nếu có lỗi thuộc về HTTP thì sẽ đc HttpException handle
* Nếu k thuộc HTTP sẽ trả về như sau

****

**Một số builtin**

* BadRequestException
* UnauthorizedException
* NotFoundException
* ForbiddenException

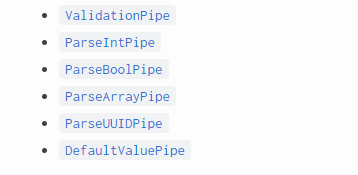
****

****

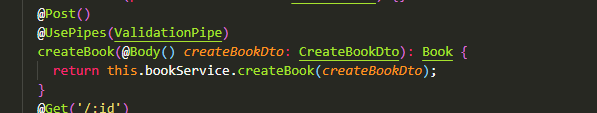
# Pipes

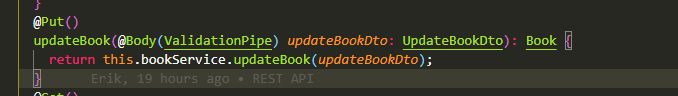
* **Transformation**
* **Validation**

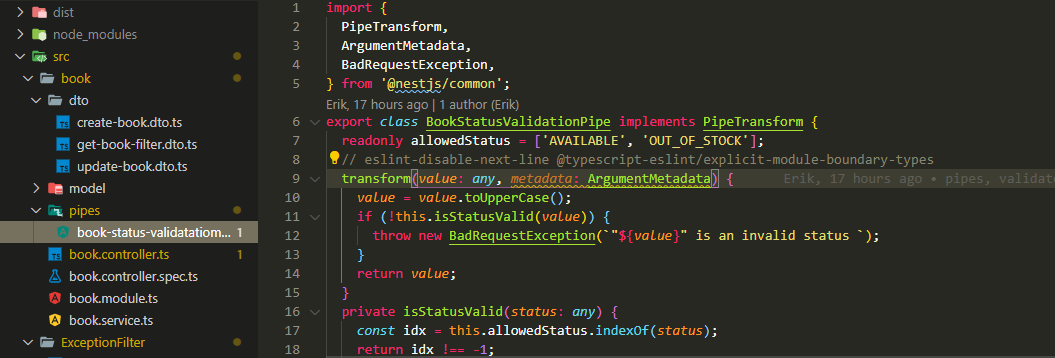
## **Built-in pipes**

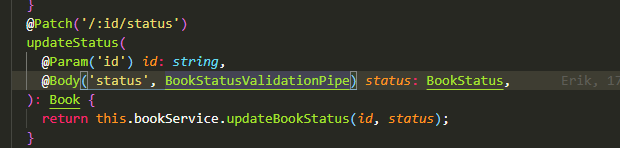
****

## **Binding pipe**

****





****

## **Schema based validation**