

TÀI LIỆU KHÓA HỌC “Nest.JS với TypeScript/MongoDB Siêu Dễ”

Tác giả: Hoi Dân IT & Eric

Version: 4.0

Note cập nhật:

- Thêm tóm tắt cho các chương tài liệu.

Chapter 0: Giới Thiệu Về Khóa Học	5
#0.1. Demo Kết Quả Đạt Được Khi Kết Thúc Khóa Học	5
#0.2. Yêu Cầu Của Khóa Học	6
#0.3. Về Khóa Học này	7
#0.4. Về Tác Giả	9
Chapter 1: Tổng quan về NestJS	10
#1. NestJS là gì ?	10
#2. Create A New Project	12
#3. Hello world	13
#4. Kiến trúc của dự án NestJS	15
Chapter 2: NestJS và Typescript	16
#5. Typescript Decorators	16
#6. Sử dụng Decorator với NestJS	18
#7. Mô hình Router với NestJS	20
#8. Controllers	21
Chapter 3: Inversion Of Control	24
#9. Inversion Of Control (IoC)	24
#10.1 Khái Niệm Modules	27
#10.2 Dependencies Injection (DI)	29
#11. Injecting a dependency	31
Chapter 4: NestJS và MVC	32
#12. Ví dụ về controller/service	32
#13. Template View Engine	32
#14. Mô hình MVC	33
#15. Vấn Đề tồn đọng với SSR ?	34
Chapter 5: Connect Database	36
#16. Lựa chọn Database	36
#17. Cài đặt MongoDB	37
#18. Sử dụng MongoDB với NestJS	38
#19. ENV Variables	39
Chapter 6: Restful API	41
#20. Generate resources	41
#21. Create Schema (Model)	42
#22.1 Create A User	43
#22.2 Hash User's Password	44
#22.3 DTO - Data Transfer Object	44
#22.4 Pipe	45
#22.5 Validation	45
#23. Get User by Id	46

#24. Update a User	47
#25. Delete a User	47
Chapter 7: Stateful và Stateless	48
#26. Setup Backend Test	48
#27. Debug NestJS Applications	49
#28. Stateful Application	51
#29. Cơ chế hoạt động của stateful app	52
#30. Ưu, nhược điểm của Stateful ?	53
#31. Mô hình stateless	54
#32. Ưu, nhược điểm của Stateless	55
Chapter 8: JWT - Json Web Token	56
#33. JWT là gì ?	56
#34. Phân loại token sử dụng	57
#35. Giới thiệu về Passport.JS	59
#36. Local Strategies với NestJS	60
#37. Nestjs Guard	62
#38. LocalGuard với Passport	64
#39. Sử dụng JWT	64
#40. Implementing Passport JWT	65
#41. Enable authentication globally	66
#42. Disable Guard	67
#43. Fix bugs và tổng kết	68
Chapter 9: Tư duy phân tích database	70
#44. Giới thiệu đề bài	70
#45. Phân tích model & relationship	71
#46. Thiết kế model	73
Chapter 10: Mongoose Plugins	76
#47. Timestamps plugin	76
#48. Soft-delete plugin	77
#49. Query Builder	78
#50. Setup dự án frontend	79
#51. CORS là gì ?	80
#52.1 Làm sao để bypass CORS ?	82
#52.2 Setup Cors NestJS	83
Chapter 11: Modules Company	84
#53. Generate Modules Company	84
#54. Bài Tập Create Company	85
#55. Update User Type (JWT)	86
#56. Passing Req.user	88

#57. Bài Tập Update Company	89
#58. Bài Tập Delete Company	90
#59. Query with Pagination	91
#60. Giới Thiệu Interceptor	93
#61.1 Transform Response	94
#61.2 Customize Message (Decorator)	96
#62. Version APIs	97
Chapter 12: Modules User	98
#63. Update User Schema	98
#64.1 Bài Tập CRUD Users	99
#64.2 Hướng Dẫn Bài Tập CRUD Users	105
#65. Demo Chức năng login	106
#66. API login	107
#67. Set Cookies	108
#68. Bài Tập API Get Account (F5 Refresh)	109
#69.1 API Refresh Token (Part 1)	110
#69.2. API Refresh Token (Part 2)	111
#70. Bài Tập API Logout	112
#71.1 Test giao diện frontend (Part 1)	113
#71.2 Test giao diện frontend (Part 2)	113
Chapter 13: Modules Job/Resume	114
#72. Bài Tập Tạo Module Jobs	114
#73. Bài tập CRUD Jobs	115
#74. Hướng Dẫn Bài tập CRUD Jobs	120
#75. Giới Thiệu Upload files	121
#76. Validate Upload Files	122
#77. Destination Upload File	123
#78. Bài Tập Update Company with Image	125
#79. Test Giao Diện Frontend Upload File	126
#80. Bài tập Tạo Modules Resumes	128
#81. Bài tập CRUD Resumes	129
#82. Hướng Dẫn Bài tập CRUD Resumes	136
#83.1 Fetch Data with Ref	137
#83.2 Test Giao Diện Frontend	138
Chapter 14: Modules Permission & Roles	139
#84. Bài tập Tạo Modules Permission & Roles	139
#85. Bài tập CRUD Permissions	141
#86. Hướng Dẫn Bài tập CRUD Permissions	146
#87. Bài tập CRUD Roles	146

#88. Hướng Dẫn Bài tập CRUD Roles	150
#89. Fix bugs	150
#90. Nest Lifecycle Event	152
#91. Bài tập Tạo Sample Data	154
#92. Hướng Dẫn Bài Tập Tạo Fake Data	155
#93. Test Giao Diện Frontend	156
#94. Roles Guards	159
Chapter 15: Module Subscribers	160
#95. Bài tập CRUD Subscribers	160
#96. Giới thiệu về gửi email với Node.js	161
#97. Template Email	162
#98. Setup Tài khoản/Server gửi Email	164
#99. Nestjs - Mailer	166
#100. Gửi email với template	168
#101. Fetch Subscribers by skills	170
#102. Test giao diện frontend	171
#103. Cron job	172
#104. Tự động gửi email với cron job	173
#105. Nhận Xét về Dự Án Thực Hành Frontend	173
Chapter 16. Giới Thiệu Kiến Thức Nâng Cao	174
#106. Authorization	174
#107. Helmet	176
#108. Rate Limiting	177
#109. Swagger	178
#110. Healthchecks	181
#111. Docs Whole App Backend	182
#112. Fix Bugs Còn Tồn Động	183
#113. Upgrade NestJS Version	184
#114. Build Docker	187
#115. Nhận Xét Về Khóa Học	188
Lời Kết	189

Chapter 0: Giới Thiệu Về Khóa Học

Giới thiệu về khóa học và tác giả, cũng như demo kết quả đạt được sau khi kết thúc khóa học này.

#0.1. Demo Kết Quả Đạt Được Khi Kết Thúc Khóa Học

Video demo: <https://youtu.be/z0B7I-P7tUs>

Một vài highlight của khóa học:

- **Hiểu rõ, thực hành và áp dụng được ngôn ngữ Typescript cho dự án NestJS**
- Xây dựng tư duy Backend với tính “mở rộng” cao, bằng cách học & sử dụng framework NestJS
- Thực hành dự án viết backend cho website clone itviec.com (website đăng tin/ứng tuyển việc làm)

#0.2. Yêu Cầu Của Khóa Học

Các yêu cầu cần biết trước khi thực hành khóa học:

- **Cần có sự hiểu biết cơ bản về Typescript.** Nếu bạn chưa biết, có thể học nhanh tại đây:
https://www.youtube.com/playlist?list=PLncHg6Kn2JT5emvXmG6kgeGkrQjRqxs_b4
- Cần có kiến thức về MongoDB (sử dụng với Mongoose). Nếu chưa biết, bạn có thể tham khảo khóa học này:
<https://hoidanit.com.vn/course/backend-restful-server-voi-nodejs-va-express-sql-mongodb?id=640b539cfe283eefef939870>

Một vài yêu cầu kiên quyết:

- **Cài đặt môi trường Node.JS :** trong khóa học này, máy tính của mình dùng **version Node.JS 16.20.0**

Để hạn chế lỗi, các bạn nên dùng version trên.(v16.20.0)

Có thể dùng NVM để sử dụng nhiều version của Node.JS trên cùng máy tính
Về NPM, xem tại:

https://www.youtube.com/watch?v=ccjKHLyo4IM&list=PLncHg6Kn2JT6E38Z3kit9Hnif1xC_9Vql&index=40

- **Cài đặt Git:** các thao tác với Git và sử dụng Github, mình không hướng dẫn. Bạn nào chưa biết dùng Git, xem tại đây:
<https://www.youtube.com/watch?v=-BtolPy15fg&list=PLncHg6Kn2JT6nWS9MRjSnt6Z-9Rj0pAlo>
- **Cài đặt Visual Studio Code:** đây là công cụ dùng để code

#0.3. Về Khóa Học này

1. Tại sao mình làm khóa học này?

Mình làm khóa học này để giúp các bạn hiểu và áp dụng Typescript với framework Nestjs

Đồng thời, NestJS là một framework backend được sử dụng rất rộng rãi ngoài kia, rất có ích cho công việc sau này của các bạn.

2. SOS thì làm sao

Với các bạn học viên Udemey, để đảm bảo quyền lợi, các bạn chủ động inbox qua fanpage Hỏi Dân IT để nhận được sự hỗ trợ trong quá trình học tập nhé:

<https://www.facebook.com/askITwithERIC/>

4. Về chuyện leak khóa học và mua lậu

Mình biết rất nhiều bạn khi học khóa học này của mình, là mua lậu qua bên thứ 3. chuyện này là hoàn toàn bình thường, vì thương hiệu “Hoi Dan IT” đang ngày càng khẳng định được vị thế của mình.

Nhiều bạn hỏi mình, sao mình không ‘chặn việc mua lậu’. nói thật, nếu mình làm, là làm được đấy, cơ mà nó sẽ gây ra sự bất tiện cho học viên chân chính (con sâu làm rầu nồi canh). Với lại, ngay cả hệ điều hành windows, còn bị crack nữa là @@

Mình cũng có 1 bài post facebook về chuyện này:

<https://www.facebook.com/askitwitheric/posts/pfbid02gyasktd3semgxat6nevnvwh4c8epzu3i7kpzhr7s7gmmfvcvucz96eb8avnvgnhl>

Với các bạn học viên chân chính, mình tin rằng, những cái các bạn nhận được từ mình khi đã chấp nhận đầu tư, nó sẽ hoàn toàn xứng đáng. vì đơn giản, với cá nhân mình, khách hàng là thượng đế.

VỚI CÁC BẠN MUA LẬU, MÌNH CHỈ MUỐN CHIA SẺ THẾ NÀY:

1. TRÊN ĐỜI NÀY, CHẴNG CÓ GÌ CHẤT LƯỢNG MÀ MIỄN PHÍ CẢ.

VIỆC BẠN MUA LẬU QUA BÊN THỨ 3, LÀ GIÚP BỌN CHÚNG LÀM GIÀU VÀ GÂY THIẾT HẠI CHO TÁC GIẢ.

NẾU NHÌN VỀ TƯƠNG LAI => Càng ngày càng ít tác giả làm khóa học => NGƯỜI BỊ HẠI CUỐI CÙNG VẪN LÀ HỌC VIÊN

2. HÃY HỌC THÓI QUEN TRÂN TRỌNG GIÁ TRỊ LAO ĐỘNG

NÓ LÀ THÓI QUEN, CŨNG NHƯ SẼ LÀ MỘT PHẦN TÍNH CÁCH CỦA BẠN.

ĐỪNG VÌ NGHÈO QUÁ MÀ LÀM MẤT ĐI TÍNH CÁCH CỦA BẢN THÂN.

NẾU KHÓ KHĂN, CỨ INBOX MÌNH, MÌNH HỖ TRỢ. VIỆC GÌ PHẢI LÀM VẬY =))

3. MÌNH ĐÃ TỪNG LÀ SINH VIÊN GIỐNG BẠN, MÌNH HIỂU TẠI SAO CÁC BẠN LÀM VẬY. HÃY BIẾT CHO ĐI. SỐNG ÍCH KỶ, THÌ THEO LUẬT NHÂN QUẢ ĐẤY, CHẴNG CÓ GÌ LÀ NGẪU NHIÊN CẢ

4. NẾU BẠN THẤY KHÓA HỌC HAY, HÃY BIẾT DONATE ĐỂ ỦNG HỘ TÁC GIẢ. LINK DONATE: <https://hoidanit.github.io/official/donate>

Hành động nhỏ nhưng mang ý nghĩa lớn. Hãy vì 1 cộng đồng IT Việt Nam phát triển. Nếu làm như các bạn, có lẽ chúng ta đã không có Iphone, không có Apple như ngày nay rồi @@

#0.4. Về Tác Giả

Mọi thông tin về Tác giả Hỏi Dân IT, các bạn có thể tìm kiếm tại đây:

Website chính thức: <https://hoidanit.com.vn/>

Youtube “Hỏi Dân IT” : <https://www.youtube.com/@hoidanit>

Tiktok “Hỏi Dân IT” : <https://www.tiktok.com/@hoidanit>

Fanpage “Hỏi Dân IT” : <https://www.facebook.com/askITwithERIC/>

Udemy Hỏi Dân IT: <https://www.udemy.com/user/eric-7039/>

Chapter 1: Tổng quan về Nest.JS

Giới thiệu tổng quan về Framework Nest.js và trả lời câu hỏi, tại sao chúng ta lại cần Nest.JS, từ đây viết chương trình Hello World với Framework này.

#1. Nest.JS là gì ?

Tài liệu:

Trang chủ: <https://nestjs.com/>

Tài liệu về Nest.js: <https://docs.nestjs.com/>

Github: <https://github.com/nestjs/nest>

1. what:

- Nest (NestJS) là 1 framework dành cho Server-side applications, chạy với Node.js. (tức nó là backend đấy :v)
- Mặc định sử dụng Typescript, tuy nhiên vẫn có thể code javascript với nestjs (not recommended)

Nestjs khác Nextjs các bạn ha :v

Nest (cái tổ chym), là **framework chuyên làm backend** (sử dụng typescript)

Next(kế tiếp), là **framework tích hợp React phía server, tức là nó chuyên làm frontend.**

Nó có thể làm cả backend nữa, cơ mà, có nhiều nhược điểm đấy :v

2. why: tại sao dùng Nest.js, trong khi đã có các framework như Express, Fastify...

- Nest được xây dựng để đảm bảo tính "hiệu quả" (efficient) và "khả năng mở rộng" (scalable) của app backend
- Nest.js mặc định sử dụng **Express**, tuy nhiên, có thể lựa chọn **Fastify**, tùy thuộc vào nhu cầu của bạn
- **Bằng cách sử dụng Typescript, Nest tuân theo:**
 - + **OOP** (Object Oriented Programming - lập trình hướng đối tượng)
 - + **FP** (Functional Programming), tương tự như function component (khi dùng với React.js)
 - + **FRP** (Functional Reactive Programming) (tương tự rxjs của angular)

=> sử dụng Nest.js khi muốn xây dựng ứng dụng backend có tính mở rộng và đảm bảo hiệu năng (ngoài ra còn có khả năng 'maintain' code).

=> Nest.js được lấy cảm hứng từ Angular (highly testable, scalable, loosely coupled, and easily maintainable applications)

Nếu bạn lựa chọn Angular thay vì React, thì tương tự, lựa chọn Nest.js thay vì code Express thuần.

Nest.js sẽ cung cấp bộ khung framework (architecture - kiến trúc), việc của chúng ta là sử dụng bộ khung đấy để build sản phẩm mình muốn.

#2. Create A New Project

Tài liệu: <https://docs.nestjs.com/#installation>

Về tài liệu cũ hơn (ứng với version cũ hơn), thay đổi đường link url:

<https://docs.nestjs.com/v8/>

Lưu ý: môi trường cài đặt Node.js 16.xgulp'

Cài đặt Nest CLI : **npm i -g @nestjs/cli@9.4.2**

CLI (command line interface), là công cụ cung cấp câu lệnh. Bạn gõ câu lệnh, thay vì code trực tiếp :v

1. Clone từ github

Version Nest.js sử dụng là v9.x

Link github: <https://github.com/haryphamdev/nestjs-basic>

2. Tạo mới từ đầu

Chỉ tạo project mới từ đầu, khi bạn đã có khả năng tự đọc tài liệu và thực hành Nest.js

Sử dụng câu lệnh sau để tạo mới project:

nest new project-name

#3. Hello world

1. Chạy dự án

```
"start": "nest start",  
"dev": "nest start --watch"
```

câu lệnh start dùng để chạy ứng dụng Nest.js (sau khi đã build)

--watch (tương tự như nodemon), dùng để theo dõi sự thay đổi của files -> auto restart lại server

=> Để chạy dự án (môi trường dev), sử dụng câu lệnh: npm run dev

Mặc định, ứng dụng Nest.js chạy trên port 3000
có thể thay đổi tham số port này tại file main.ts

2. Cấu trúc dự án NestJS đã xây dựng

- Thư mục dist: thư mục build của Nest.js. Cần build, vì code Nest.js dùng typescript
-> dịch ra javascript thì Node.js mới thực thi được (server)/hoặc browser thực thi.

- Thư mục test: viết code test

Trong phạm vi của dự án này, bỏ qua thư mục test, và các file có tiền tố spec.ts

- Thư mục src: nơi chứa source code dự án

+ với Nest.js, src được chia theo module (phân tích cụ thể ở video tiếp theo).

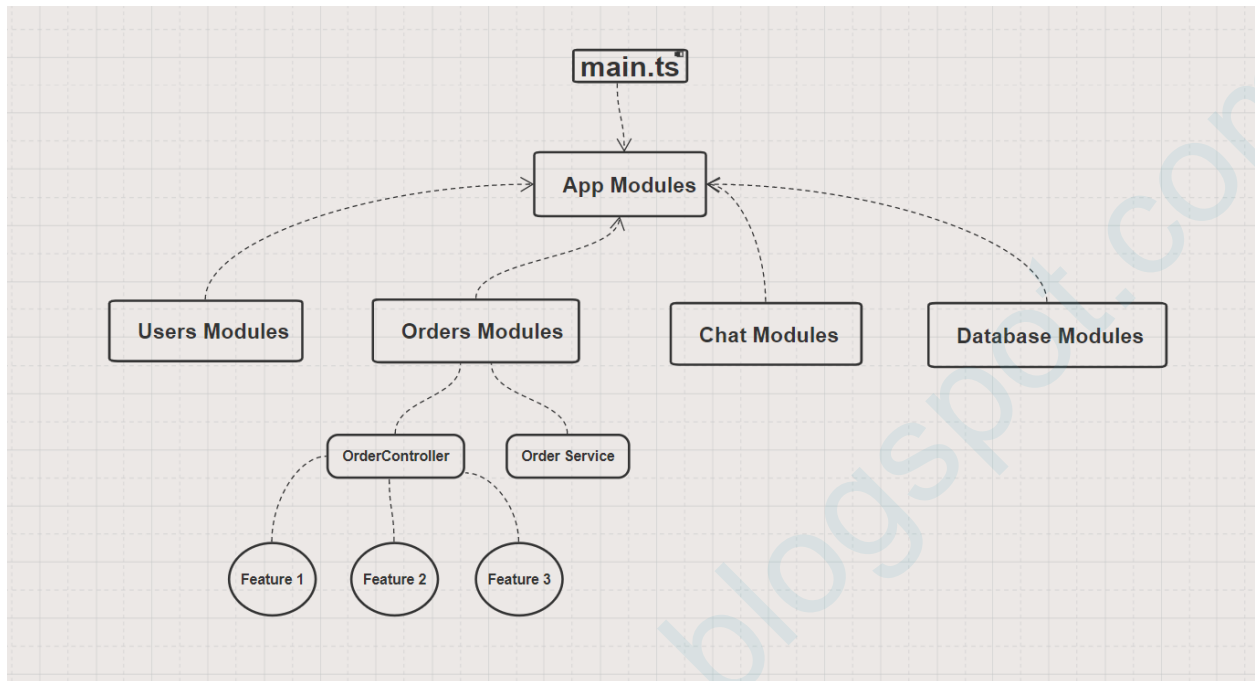
+ hiện tại, dự án có 1 module duy nhất là "module app", bao gồm "app controller, app module và app service"

+ **file main.ts là file chính.** ứng dụng Nest.js sẽ chạy file này đầu tiên (tương tự file server.js trong khóa học restful api)

- Các file phụ:

- + .eslintrc.js : config eslint, giúp check code khi viết theo cú pháp typescript
- + .gitignore : khai báo các file không muốn git quản lý
- + .prettierrc : config prettier, giúp viết code "đẹp hơn" => hiểu đơn giản là format code
- + nest-cli.json : config Nest.js khi chạy CLI/build
- + package.json/package-lock.json : quản lý các thư viện cài đặt cho dự án
- + tsconfig.json : khai báo cách dịch code typescript thành javascript (môi trường development)
- + tsconfig.build.json: khai báo cách dịch code typescript thành javascript (khi build với môi trường production)

#4. Kiến trúc của dự án Nest.JS



Dự án Nest.JS được chia thành các module, phân theo chức năng.

- Cần chia theo module, để có thể phát triển độc lập, đồng thời, tính năng này sẽ ít ảnh hưởng tới tính năng kia (on/off theo ý muốn)
 -
 - File `main.ts` sẽ chạy `app modules`, và `app modules` sẽ bao gồm tất cả các `modules con`.
 -
- Modules con có thể bao gồm (hoặc không) các modules khác

Ví dụ: để kết nối tới database, `Users Modules` có thể bao gồm `Database Modules`

Chapter 2: NestJS và Typescript

Tìm hiểu và học cách sử dụng Decorator của Typescript vào Nest.JS, từ đây nắm vững cách hoạt động Router và Controller khi sử dụng các Decorator có sẵn do Nest.JS cung cấp.

#5. Typescript Decorators

Tài liệu:

<https://www.typescriptlang.org/docs/handbook/decorators.html>

<https://www.digitalocean.com/community/tutorials/how-to-use-decorators-in-typescript>

Lưu ý: video này là cách chúng ta **"hiểu và sử dụng" decorator, không phải 'học cách viết decorator'**.

Tương tự như 'generic', bạn chỉ học, khi và chỉ khi bạn muốn reuse - tái sử dụng code, viết code ở 1 level cao hơn

1. Ví dụ về Decorators khi sử dụng với Nest.JS

@Controller()

@Injectable()

- Nếu bạn đã học java spring boot, Decorators của Nest.js, tương tự như "annotation", sẽ có dấu @ đặc trưng

Ví dụ: khi định nghĩa 1 class với @Controller => Nest sẽ "sử dụng" class này như là controller

2. Decorator là gì (what)

- Decorator là những ký hiệu (chú thích) giúp cho class/property/functions/params có thêm "sức mạnh" mới.

- **Decorator bắt đầu bằng dấu @, sau đấy là "expression".**

(express là function sẽ được thực thi khi được sử dụng với "class sử dụng nó")

bao gồm 5 level:

- + Class declaration itself
- + Properties
- + Accessors (getter/setter)
- + Methods
- + Parameters

ví dụ:

@classDecorator

```
class Person {
```

@propertyDecorator

```
public name: string;
```

@accessorDecorator

```
get fullName() {
```

```
  // ...
```

```
}
```

@methodDecorator

```
printName(@parameterDecorator prefix: string) {
```

```
  // ...
```

```
}
```

```
}
```

3. Tại sao cần Decorator ?

- Decorator giúp bạn viết code ngắn hơn, có khả năng tái sử dụng ở nhiều nơi, đồng thời bổ sung 'sức mạnh' cho các thành phần được 'trang trí' (decorated).

Ví dụ:

```
class User {
```

```
  @Min(0)
```

```
  @Max(10)
```

```
  @IsEmail
```

```
  email: string;
```

```
  @Max(20)
```

```
  password: string;
```

```
}
```

#6. Sử dụng Decorator với Nest.JS

Tài liệu:

<https://www.typescriptlang.org/docs/handbook/decorators.html#introduction>

1. Setup

- Mặc định, Nest đã cấu hình để sử dụng Decorator bằng setup trong file tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES5", // tối thiểu là ES5 trở lên
    "experimentalDecorators": true
  }
}
```

Lưu ý: cho dù tính năng decorator của Typescript đang là “thử nghiệm”, có thể thay đổi trong tương lai, tuy nhiên chúng ta không cần lo lắng. Vì nếu có thay đổi, framework Nestjs sẽ lo chuyện này cho chúng ta :v

2. Phân loại Decorator

- Nest đã viết sẵn (amazing)
- Customize (tự định nghĩa) - sẽ học tại các chương sắp tới

3. Sử dụng Decorator

<https://stackoverflow.com/questions/70449766/typescript-decorators-why-do-some-decorators-require-brackets-and-others-dont>

```
class ExampleClass {  
  @decorate1( )  
  @decorate2  
  method( ) {  
    return "something";  
  }  
}
```

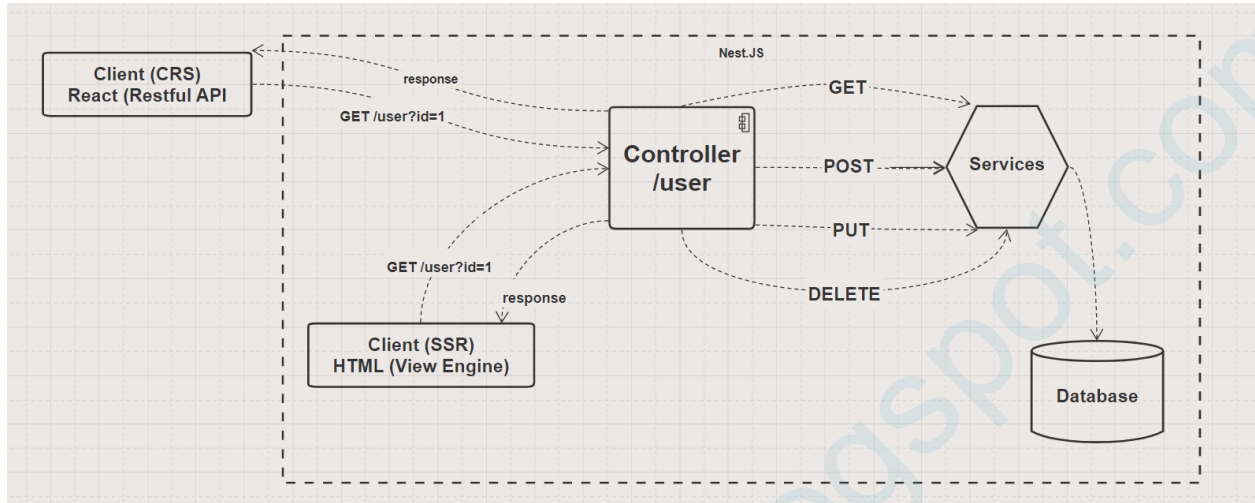
tại sao lại sử dụng: @decorate1() => cần dấu () và @decorate2 => không cần ()
=> phụ thuộc vào cách viết decorator

=> với Nestjs, sử dụng các Decorator viết sẵn, đa phần sẽ cần dấu ()
=> thực thi decorator

Ví dụ: @Controller(), @Injectable()...

#7. Mô hình Router với Nest.JS

Mô hình router của NestJS:



#8. Controllers

Tài liệu: <https://docs.nestjs.com/controllers>

- Remove file .controller.spect.ts (nếu có)

Một Class được gọi là controller, khi nó sử dụng @Controller()

1. Ví dụ về controller

- Tạo thêm userController : lưu ý, không cần hiểu cách khai báo controller vào app modules
- update hello world của appController, không dùng service (service sẽ học ở chương sau)

Ví dụ 1:

```
@Controller('user')
export class UserController {
  @Get()
  findAll(): string {
    return 'This action returns all users';
  }

  @Delete("/by-id")
  findById(): string {
    return 'This action will delete a user by id';
  }
}
```

Ví dụ 2:

```
@Controller()
export class AppController {
  @Get()
  getHello(): string {
    return 'This action hello world';
  }
}
```

2. Cách xử lý của Nest.JS

Bản chất của website, là xoay quanh URL (đường link user truy cập).
ở phía backend, URL chính là routes.

với ví dụ 1:

```
@Controller('user')
export class UserController {
  @Get()
  findAll(): string {
    return 'This action returns all users';
  }

  @Delete("/by-id")
  findById(): string {
    return 'This action will delete a user by id';
  }
}
```

khi khai báo @Controller('user') => Nest sẽ hiểu là "/user"
@Get() => không có tham số đính kèm => ứng với route "/"
=> cộng gộp sẽ ra GET "/user"

@Delete("/by-id") => có tham số đính kèm => ứng với route "/by-id"
=> cộng gộp sẽ ra DELETE "/user/by-id"

=> việc truyền tham số cho @controller("tham-số") sẽ giúp ko phải lặp lại code không cần thiết.

ví dụ, thay vì khai báo

```
@Controller() //không có tham số
@Get('/order') // GET /order
@Put('/order/by-name') // PUT /order/by-name
@Delete('/order/by-id') // DELETE /order/by-id
```

=> khai báo theo nhóm:

```
@Controller('order')
@Get() // GET /order
```

```
@Put('/by-name') // PUT /order/by-name  
@Delete('/by-id') // DELETE /order/by-id
```

3. Tổng kết

@Controller() : khi sử dụng decorator này, nếu bạn không truyền tham số => sinh ra route "/"

@Controller("/prefix"): truyền tham số => sinh ra route "/prefix"

với method GET, POST, PUT, DELETE...

@Get() nếu bạn không truyền tham số => sinh ra GET "/"

@Get("/path") nếu bạn truyền tham số => sinh ra GET "/path"

=> NestJS sẽ tự động "cộng gộp" Controller và method. có nghĩa là

```
@Controller()
```

```
@Get('/user') => sinh ra route "/user"
```

```
@Controller("user1")
```

```
@Get('/filter-by-name') => sinh ra route "/user1/filter-by-name"
```


Chapter 3: Inversion Of Control

Khi Học về OOP (lập trình hướng đối tượng), một trong các cách dùng để tối ưu hóa ứng dụng là áp dụng mô hình **IoC(Inversion of control)** và **DI (Dependency injection)**. Chương này sẽ giúp chúng ta học cách sử dụng các "design pattern" trên vào framework Nest.JS

Lưu ý:

1. nội dung mang tính chất khái quát (giới thiệu), các bạn có thể tìm hiểu thêm từ tài liệu tại các video
2. Khái niệm Inversion of Control, hay Dependencies Injection được sử dụng trong lập trình hướng đối tượng, không phân biệt ngôn ngữ lập trình

#9. Inversion Of Control (IoC)

Tài liệu: <https://www.tutorialsteacher.com/ioc/introduction>

Where ? OOP (object oriented programming)

What ?

- Inversion of Control (IoC), là cách "đảo ngược" sự kiểm soát (tương tự tủ lạnh/điều hòa inverter, có nóng và lạnh :v)
- IoC là khái niệm (principle), ko phải là "cách làm" (làm như thế nào)

Why/When ?

Vấn đề gặp phải:

```
public class A
{
    B b;

    public A()
    {
        b = new B();
    }

    public void Task1() {
        // do something here..
        b.SomeMethod();
        // do something here..
    }
}
```

}

tailieusharefree.blogspot.com

```
public class B {
```

```
    public void SomeMethod() {  
        //doing something..  
    }  
}
```

```
=====
```

class A gọi b.SomeMethod() để hoàn tất task1 (của nó)

class A không thể hoàn thành "task1" nếu nó không có class B => Class A "phụ thuộc" vào class B

class A "kiểm soát" (control) vòng đời của class B (tạo và sử dụng class B)

=> Nhờ có IoC, chúng ta OOP, chúng ta có thể "giảm thiểu" sự phụ thuộc giữa các class với nhau.

Mục đích: có thể tách rời và "test" độc lập, không phụ thuộc vào nhau

How ?

- Sử dụng các "pattern", như Factory, Dependencies Injections...

#10.1 Khái Niệm Modules

Nhắc lại:

IoC (inversion of control) - đảo ngược sự kiểm soát, là nguyên tắc "giảm thiểu" sự phụ thuộc giữa các class với nhau.

thay vì class A control class B, chúng ta sẽ đảo ngược, cho phép A và B có thể hoạt động độc lập

IoC là nguyên tắc (cách thiết kế/bản vẽ), không phải cách làm như thế nào :v

Tài liệu:

<https://angular.io/guide/dependency-injection#understanding-dependency-injection>

1. Modules (Why)

- Khi bạn phát triển ứng dụng nhỏ, chúng ta có thể code "all-in-one" (tất cả code trong 1 file/1 thư mục/1 project).

Với dự án lớn, chúng ta không thể làm vậy, vì

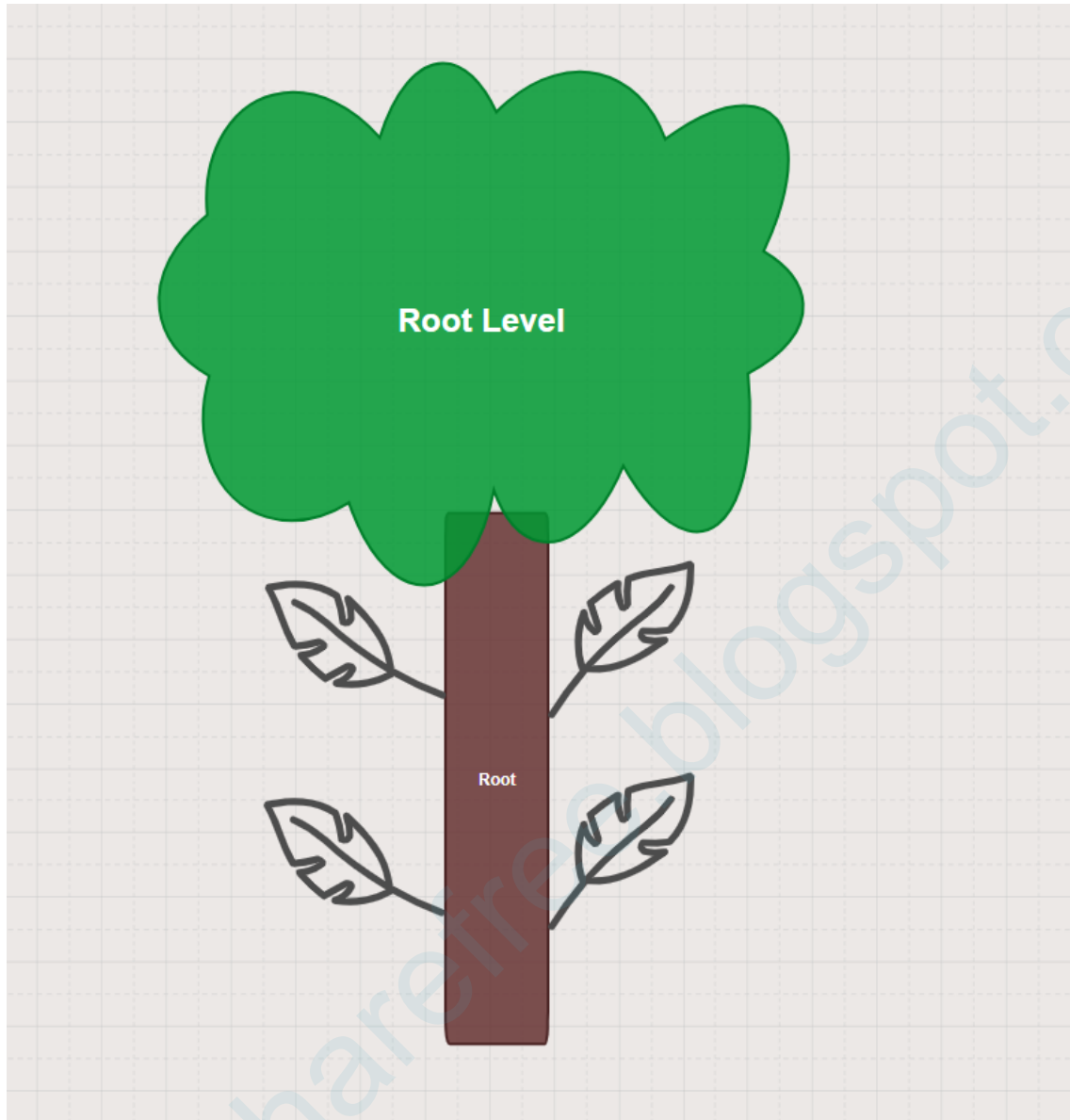
- + không thể đọc code (code quá nhiều)
- + không thể test/maintain

- Giải pháp đặt ra là chia dự án thành các modules. 1 modules có thể là 1 tính năng hoặc 1 nhóm tính năng có liên quan tới nhau.

Ưu điểm lớn nhất của cách làm này:

- + sự phát triển các modules có thể làm độc lập
- + on/off modules này không làm ảnh hưởng tới modules kia (nếu các modules không phụ thuộc vào nhau)

Mình họa = hình cái cây (tree). Các modules chính là các lá của 1 cái cây. Lá này rụng sẽ không làm ảnh hưởng tới cây, hay các lá còn lại.



Ứng dụng trong thực tế:

Ví dụ Facebook có lỗi không gửi được tin nhắn (modules gửi tin nhắn), nhưng website/app facebook vẫn dùng các tính năng khác bình thường.

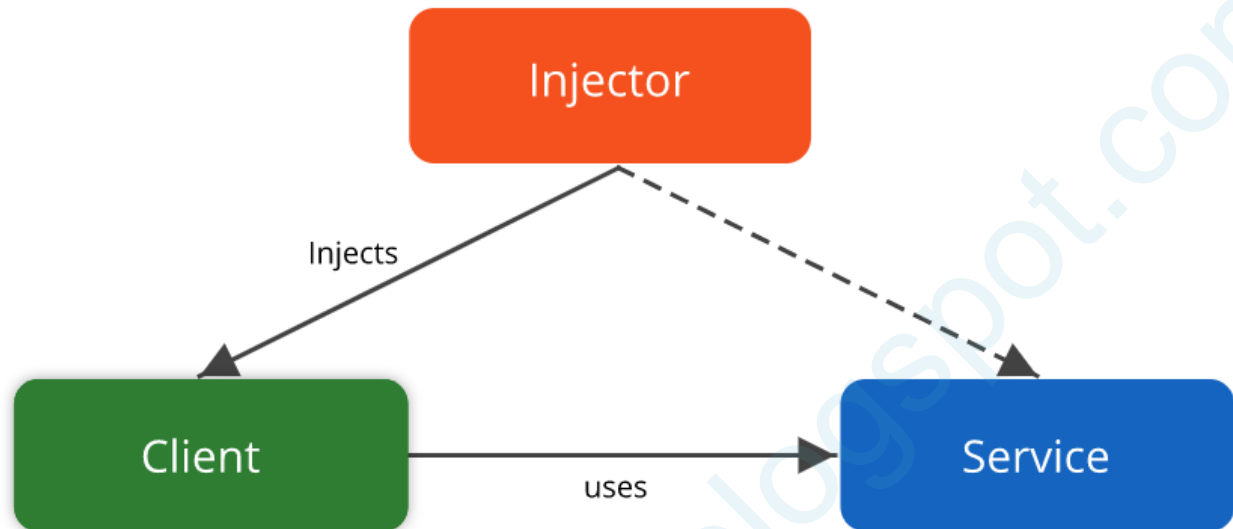
Nếu code all-in-one => lỗi 1 tính năng sẽ gây ra lỗi cả hệ thống không hoạt động

=> DI giúp chúng ta có thể chia code thành các modules độc lập :v

#10.2 Dependencies Injection (DI)

1. Các thành phần của DI (What)

gồm 3 thành phần chính: consumer, provider và injector



ví dụ: class A sử dụng service B

```
class A {
  serviceB.doSomething()
}
```

=> class A gọi là **consumer** (người tiêu thụ/sử dụng)
service B gọi là provider (nhà cung cấp)

còn 1 thành phần, gọi là injector (giúp tạo ra DI, và làm cho class A và B không phụ thuộc vào nhau :v)

2. Ví dụ về phạm vi (scope) và cách sử dụng DI

Ví dụ của angular: <https://angular.io/guide/dependency-injection#providing-dependency>

Sử dụng decorator: @Injectable()

Về scope:

- Nếu inject vào 1 modules cụ thể => chỉ có mình modules đấy sử dụng.
- Nếu inject vào root modules => tất cả có thể dùng :v

#11. Injecting a dependency

Tài liệu: <https://www.tutorialsteacher.com/ioc/dependency-injection>

Có 3 level của DI:

- Constructor Injection
- Property Injection
- Method Injection

Với NestJS (sử dụng javascript), tập trung sử dụng Constructor Injection :v

Tài liệu:

<https://stackoverflow.com/questions/72446772/how-does-the-private-keyword-let-us-b-oth-declare-and-initialize-a-class-instance/72447725>

<https://www.typescriptlang.org/docs/handbook/2/classes.html#parameter-properties>

1. Constructor Injection

Viết đầy đủ theo OOP:

```
@Controller('cats')
export class CatsController {
  private catsService: CatsService;

  constructor(service: CatsService) {
    this.catsService = service;
  }
  /* TODO */
}
```

Viết ngắn gọn, sử dụng keyword "private":

```
@Controller('cats')
export class CatsController {
  constructor(private catsService: CatsService) {}
  /* TODO */
}
```


Chapter 4: NestJS và MVC

NestJS cũng có đủ khả năng và sức mạnh để làm Server theo mô hình MVC truyền thống. Tại đây, chúng ta sẽ tìm hiểu về View Engine, cách viết code theo mô hình Model-View-Controller và các nhược điểm đang còn tồn đọng của mô hình này.

#12. Ví dụ về controller/service

Tài liệu: <https://docs.nestjs.com/recipes/crud-generator>

Mục đích: hiểu các decorator và code Nest.JS "đã làm sẵn".

- Tạo controller
- Tạo Service
- Ôn tập lại DI (dependency injection)

#13. Template View Engine

Tài liệu: <https://docs.nestjs.com/techniques/mvc>

<https://expressjs.com/en/guide/using-template-engines.html>

Cài đặt thư viện:

npm i --save-exact ejs@3.1.9

yarn add ejs@3.1.9

#14. Mô hình MVC

- Không có Model ?

tailieusharefree.blogspot.com

#15. Vấn Đề tồn đọng với SSR ?

1. Server Side Rendering ?

Thực tế, chúng ta có 2 cách để render ra dữ liệu: CSR và SSR.

CSR: Client Side Rendering, có nghĩa client sẽ render ra dữ liệu HTML (ví dụ sử dụng React truyền thống)

SSR: Server Side Rendering, server sẽ render dữ liệu HTML

2. Khi nào sử dụng SSR

Khi bạn muốn client nhận được phản hồi nhanh nhất (vì client có phải làm gì đâu, ngoài việc gửi request. Server sẽ render dữ liệu và gửi response về cho client)

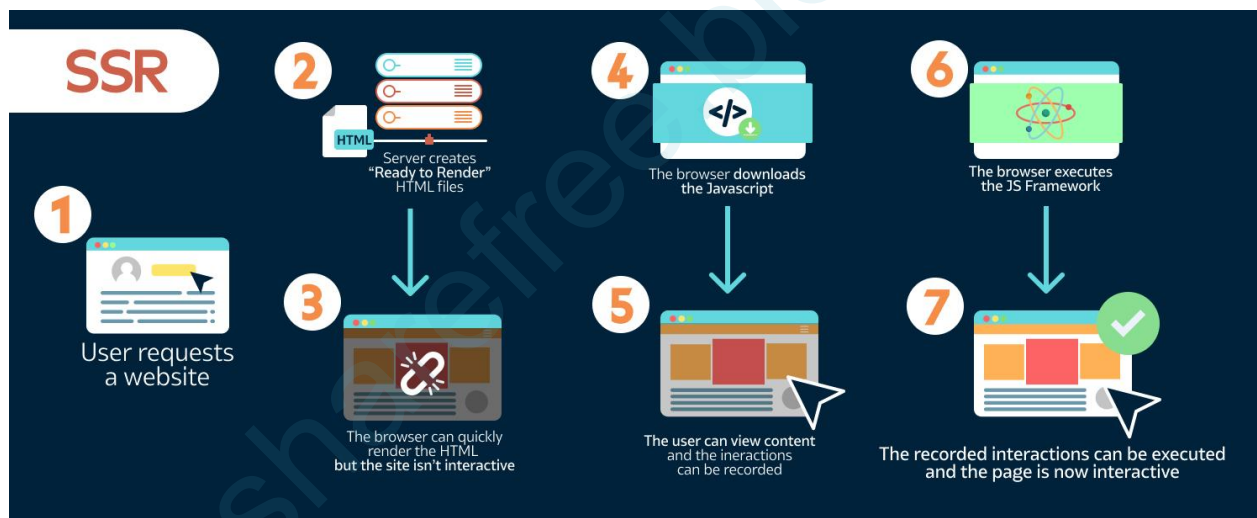
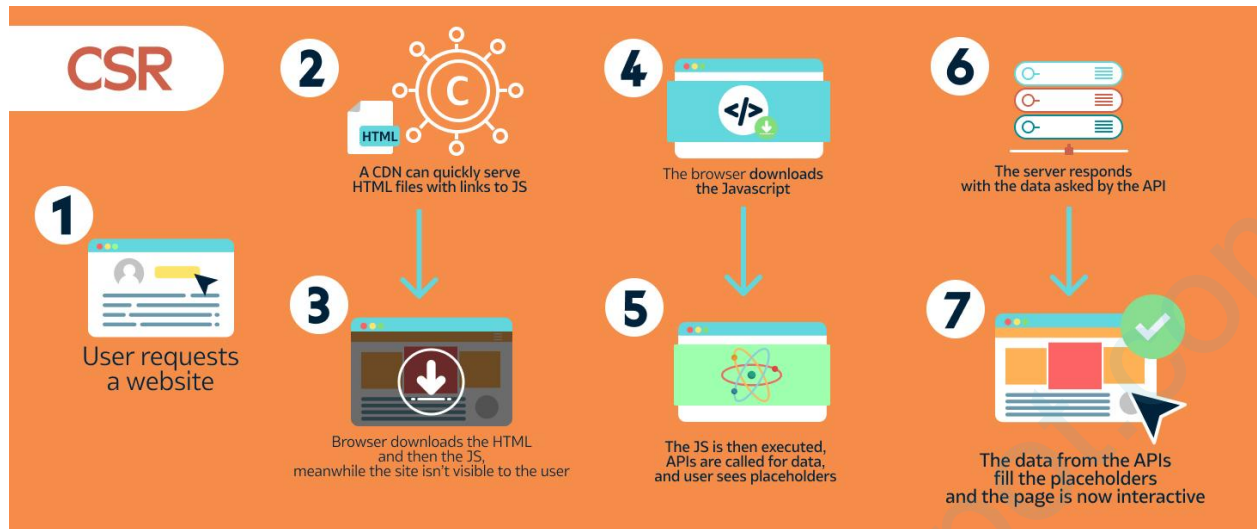
Ở đây, server phải mạnh (nhiều RAM, CPU...) thì mới nhanh. Server yếu thì cũng hơi lâu đấy :v

Ví dụ:

Client truy cập vào hoidanit.com.vn

Server của hoidanit.com.vn sẽ cần render ra trang chủ, ứng với route '/', sau đấy gửi kết quả html (chính là giao diện mà người dùng nhìn thấy)

Nhược điểm: chi phí thông thường sẽ cao :v



Chapter 5: Connect Database

Trong chương này, chúng ta sẽ cùng nhau kết nối backend tới Database bằng cách sử dụng MongoDB. Ngoài ra, sẽ tối ưu hóa cách sử dụng "hàng số", dựa vào cách sử dụng tham số môi trường (.env) với Nest.JS Config.

#16. Lựa chọn Database

Tài liệu: <https://docs.nestjs.com/techniques/database>

Chúng ta code backend thông qua ORM/ODM để chọn nhanh, không tính việc viết 'raw query' truyền thống

1. Các loại database hỗ trợ

- Nest hỗ trợ với bất kỳ loại SQL và NoSQL database nào, giống hệt như việc làm với Express (or Fastify)

Với SQL, hỗ trợ các thư viện nổi tiếng sau: (mysql, sql server, postgres, oracle...)

+ MikroORM: <https://mikro-orm.io/>

+ Sequelize: <https://sequelize.org/>

+ Knex.js: <https://knexjs.org/>

+ TypeORM: <https://typeorm.io/>

+ Prisma: <https://www.prisma.io/>

Với NoSQL, hỗ trợ MongoDB: <https://www.mongodb.com/>

Đặc biệt, NestJS hỗ trợ TypeORM, Sequelize và Mongoose (mongodb) với các package @nestjs/typeorm , @nestjs/sequelize, @nestjs/mongoose

=> giúp việc coding càng trở nên dễ dàng hơn

2. Lựa chọn loại database cho dự án

Với SQL, NestJS "gợi ý" sử dụng TypeORM. Đi làm, việc dùng công nghệ gì/thư viện gì là do cty + người leader quyết định.

Nếu bạn "đang tự học" NestJS, mình cũng gợi ý dùng TypeORM nhé

Yên tâm 1 điều, là khi bạn học được 1 thư viện, các thư viện khác cú pháp nó khác, nhưng mà cách tư duy nó "sam same" (tương tự nhau)

với TypeORM, hỗ trợ: Postgres, Oracle, Microsoft SQL Server, SQLite, MySQL (ko nên dùng MongoDB với typeORM :v)

Với Nest.js và khóa học này, mình sử dụng MongoDB (Mongoose) vì:

- Nodejs vốn dĩ mềm dẻo, rất phù hợp với MongoDB
- SQL sẽ được dùng trong khóa học với Java (tương lai sẽ có)

#17. Cài đặt MongoDB

//todo: add playlist cài đặt và sử dụng mongodb (local + hosting)

Phần mềm cần có : MongoDB Compass

Link download: <https://www.mongodb.com/try/download/compass>

Tạo tài khoản : <https://www.mongodb.com/cloud/atlas/register>

Yêu cầu cần đạt được khi kết thúc video này, là có 1 đường link url kết nối vào MongoDB.

Setup account Mongo Atlas => get URL => check connection với Mongo Compass

Có 2 cách để cài đặt và sử dụng mongodb (mục đích học tập)

1. Thông qua docker

Bạn nào đã học khóa restful API của mình với Express, thì đã không lạ gì với MongoDB và docker nhỉ :v

<https://hoidanit.com.vn/course/backend-restful-server-voi-nodejs-va-express-sql-mongodb?id=640b539cfe283eefef939870>

Sau khi dựng database MongoDB bằng docker image, chúng ta dùng phần mềm MongoDB Compass để có view dữ liệu.

2. Thông qua MongoDB Server

Cài đặt MongoDB Server hoặc tạo tài khoản và sử dụng database Atlas online

#18. Sử dụng MongoDB với NestJS

Để sử dụng ORM, chúng ta sử dụng **mongoose**

Tài liệu: <https://docs.nestjs.com/techniques/mongodb>

Cài đặt thư viện: `npm install --save-exact @nestjs/mongoose@9.2.2 mongoose@7.1.0`

#19. ENV Variables

1. Sử dụng .dotenv

npm i dotenv@16.0.3

Tạo file .env ở level root, và để truy cập 1 biến, chúng ta sử dụng cú pháp:

process.env.VARIABLE_NAME

Với NestJS, chúng ta không sử dụng cách làm này, mặc dù nó vẫn “chạy code” bình thường.

Trong 1 dự án lớn, đôi khi chúng ta cần:

- Nhiều hơn 1 file .env, ví dụ mỗi môi trường sử dụng là 1 file .env

Môi trường test: .env.test

Môi trường development: .env.development

Môi trường uat: .env.uat

Môi trường production: .env.production

- Chúng ta muốn validate dữ liệu của biến trong .env . Cách làm truyền thống, sử dụng trực tiếp .env là không làm được

Rất may mắn, NestJS đã cung cấp giải pháp để giải quyết 2 khóa khăn trên.

2. Config Service

Tài liệu:

<https://docs.nestjs.com/techniques/configuration>

<https://github.com/nestjs/nestjs-config/issues/19>

Cài đặt thư viện:

npm i --save-exact @nestjs/config@2.3.1

yarn add @nestjs/config@2.3.1

Về bản chất, Config Service cũng sử dụng thư viện .env, tuy nhiên, nó đã “code thêm” để giải quyết 2 vấn đề chúng ta nêu ở trên (sử dụng nhiều file .env và validate file .env)

3. Sử dụng Config Service

Tài liệu: <https://docs.nestjs.com/techniques/configuration#using-the-configservice>

- Khai báo config service ở constructor (dùng dependency injection)
constructor(private configService: ConfigService) { }
- Lấy giá trị của .env, theo cú pháp:

```
// get an environment variable  
const myVar = this.configService.get<string>('VARIABLE_NAME');
```

4. Sử dụng với file main.ts

Tài liệu: <https://docs.nestjs.com/techniques/configuration#using-in-the-maints>

```
const configService = app.get(ConfigService);
```

```
const port = configService.get('PORT');
```

5. Sử dụng khi khai báo Module

Tài liệu với MongoDB:

<https://docs.nestjs.com/techniques/mongodb#async-configuration>

Chapter 6: Restful API

Ngoài khả năng có thể tạo server theo mô hình MVC, NestJS cũng có thể làm server Restful APIs. Chúng ta sẽ cùng nhau viết nhanh module Users theo đúng chuẩn Restful, sử dụng các công cụ có sẵn của NestJS, bao gồm generate resources, schema, controller và services.

#20. Generate resources

1. Câu lệnh generate resources

Tài liệu: <https://docs.nestjs.com/recipes/crud-generator>

To avoid generating test files, you can pass the --no-spec flag, as follows:

```
nest g resource users --no-spec
```

2. Cài đặt và sử dụng PostMan

<https://www.postman.com/downloads/>

#21. Create Schema (Model)

Tài liệu: <https://docs.nestjs.com/techniques/mongodb>

Lưu ý: update tên của database

Tạo User Model với các thuộc tính:

- Email
- Password
- Name
- Address
- Phone
- Age

#22.1 Create A User

Tài liệu: <https://docs.nestjs.com/controllers#request-payloads>
<https://docs.nestjs.com/techniques/mongodb#model-injection>

Full example: <https://github.com/nestjs/nest/tree/master/sample/06-mongoose>

Mục tiêu:

Tạo API với endpoint:

POST <http://localhost:8000/users>

Body : {
 Email
 Password
 Name
 Address
 Phone
 Age
}

#22.2 Hash User's Password

Tài liệu:

<https://www.npmjs.com/package/bcryptjs>

npm i --save-exact bcryptjs@2.4.3

npm i --save-dev @types/bcryptjs@2.4.2

yarn add bcryptjs@2.4.3 @types/bcryptjs@2.4.2

#22.3 DTO - Data Transfer Object

<https://docs.nestjs.com/controllers#request-payloads>

Về request object: <https://docs.nestjs.com/controllers#request-object>

DTO là 1 object định nghĩa hình dạng dữ liệu được "transfer" (frontend và backend)

Lưu ý sử dụng class, thay vì type hay interface

#22.4 Pipe

<https://docs.nestjs.com/pipes>

Pipe có 2 tác dụng:

- Transform data : convert string => number ,array ...
- Validate data

#22.5 Validation

<https://docs.nestjs.com/techniques/validation>

<https://docs.nestjs.com/pipes#class-validator>

Cài đặt thư viện:

npm i --save-exact class-validator@0.14.0 class-transformer@0.5.1

yarn add class-validator@0.14.0 class-transformer@0.5.1

Về class-validator: <https://github.com/typestack/class-validator>

Về customize message ?

#23. Get User by Id

Mục tiêu:

Tạo API với endpoint:

GET <http://localhost:8000/users/:id>

Body : Không cần truyền, mà truyền id ở url

Lưu ý về method get => dễ bị trùng url

#24. Update a User

Tài liệu: <https://docs.nestjs.com/techniques/validation#mapped-types>

Lưu ý: omit password

Mục tiêu:

Tạo API với endpoint:

PUT <http://localhost:8000/users/:id>

Body : Không cần truyền, mà truyền id ở url

#25. Delete a User

Mục tiêu:

Tạo API với endpoint:

DELETE <http://localhost:8000/users/:id>

Body : Không cần truyền, mà truyền id ở url

Chapter 7: Stateful và Stateless

Mọi website đều hoạt động dựa trên mô hình Stateful và Stateless. Ở đây, chúng ta cần nắm vững, hiểu rõ và có khả năng phân biệt điểm khác nhau của 2 mô hình này, bởi vì, việc chúng ta lựa chọn mô hình nào, nó sẽ quyết định cách thức tổ chức code cho dự án của chúng ta.

#26. Setup Backend Test

todo: add link backup

Link github: <https://github.com/harypham/nestjs-auth-basic>

1. Các việc cần làm

Sau khi kéo code về, chạy các câu lệnh sau: (version mình sử dụng là node 16.x)

npm i => cài đặt thư viện

setup url Mongodb, file .env

npm run dev => chạy ở chế độ development

2. Test ứng dụng

Lưu ý:

- **với stateful:** sử dụng route /login /user và route '/'

- **với stateless:** sử dụng endpoint

+ login: /stateless/login => truyền body gồm username và password

+ get users /stateless/user => cần truyền bearer token ở header request

#27. Debug NestJS Applications

Tài liệu: https://code.visualstudio.com/docs/editor/debugging#_launch-configurations

Mục tiêu: biết cách đặt breakpoint xem code chạy

=> check giá trị của biến tại các thời điểm khác nhau

1. Chạy nestjs với chế độ debug

Mặc định, nestjs hỗ trợ câu lệnh debugs để đặt break point: start:debug: "nest start --debug --watch",

npm run start:debug

2. Cấu hình vscode

Tài liệu: <https://www.youtube.com/watch?v=QL3KXE1hOgA>

<https://stackoverflow.com/a/63325135>

- **Cần tạo file launch.json** : lưu ý chọn môi trường NodeJS

=> ngay lập tức sẽ có thư mục .vscode (chứa file launch.json - đây là file cấu hình debug)

```
"configurations": [  
  {  
    "type": "node",  
    "request": "launch",  
    "name": "Nest Debug",  
    "runtimeExecutable": "npm",  
    "runtimeArgs": [  
      "run",  
      "start:debug",  
      "--",  
      "--inspect-brk"  
    ],  
    "console": "integratedTerminal",  
    "restart": true,  
    "protocol": "auto",  
    "port": 9229,  
    "autoAttachChildProcesses": true  
  }  
]
```

3. Sử dụng debugs

Run app với debugs mode => node need npm run start:debug

Về các actions: https://code.visualstudio.com/docs/editor/debugging#_debug-actions

Về khóa học debug website (cả frontend/backend) với React/Node.js, các bạn tham khảo:

<https://hoidanit.com.vn/course/debugs-voi-lap-trinh-website-su-dung-react-nodejs?id=640bec8bf7099c369b3bc69a>

#28. Stateful Application

Who you are ? user/guest

What can you do ? (view/create/update/delete)

Mỗi lời gọi request từ client => gửi lên server, cần biết, ai là người thực hiện hành động ấy, và người đấy được phép làm những gì ?

Giữa các lời gọi khác nhau, thông tin user không được lưu lại (**http request**). tức là:

GET /user

GET /profile

với 2 lời gọi này, client không truyền lên dữ liệu user là ai ?

Vậy làm sao để server biết ai là người đang đăng nhập sử dụng hệ thống ?

1. Session

Session là "bộ nhớ" của server, dùng để lưu trữ thông tin của người dùng (phiên đăng nhập)

Do client không truyền lên dữ liệu user => client cần lưu giữ "id" của user, và gửi lên giữa các lời gọi request
(để bảo mật, id thường lưu ở cookies)

Server dựa vào id này, truy vấn vào "session" để biết được người dùng đang đăng nhập là ai => quyết định xử lý request với thông tin đã có

2. Các phương pháp lưu trữ thông tin của client và server

Client (browser): local storage, session, cookies

Server: session (memory/RAM, disk storage/file, database)

#29. Cơ chế hoạt động của stateful app

Giải thích cơ chế hoạt động của stateful với session:

1. cần config session/cookies/passport cho express (file main.ts). như vậy khi app khởi động lên, nó đã biết được sự tồn tại của session.

ở đây, làm product ko lưu session vào Memory.
do dùng với mongodb => lưu session vào mongodb

2. Tạo route login POST /login

với route này, sử dụng 'guard' của nestJS (truyền vào local strategy của passport) => passport xử lý phần còn lại

3. Local Strategy được 'nhúng' khi module khởi động lên, input từ HTML (bao gồm username/password) sẽ tự động chạy vào hàm "validate"

- Nếu username/password không hợp lệ => hiển thị thông báo lỗi

- Nếu username/password hợp lệ => lưu thông tin vào session (request).
session này cũng được lưu "1 bản sao" vào database.
đồng thời, session cookies được lưu tại client (html)

4. Mỗi lần f5, cookies sẽ được gửi lên server.

server dựa vào cookies (lấy ra sessionId) => kết hợp với passport (session serializer), query xuống database bảng session (setup tại bước 10). lấy ra session tương ứng

=> như vậy sẽ maintain được "session của user" mỗi lần refresh page

#30. Ưu, nhược điểm của Stateful ?

Ưu điểm:

1. client không lưu giữ thông tin, ngoại trừ session_id

=> **tính bảo mật cao, ít bị lộ thông tin người dùng**

2. Server có thể "terminated"/chấm dứt/destroy/delete session của user bất cứ khi nào khi cần thiết.

Khi session bị deleted, **user === logout**

Nhược điểm:

- Cần phải có cơ chế save/query session đủ nhanh (save Redis)
(nếu số lượng người dùng truy cập lớn).

- Không thể share sessions giữa các hệ thống khác nhau.

Ví dụ: khi bạn thực hiện chuyển khoản liên ngân hàng. từ vcb => mb , sử dụng ứng dụng của vcb. như vậy, khi bạn chuyển khoản tới mb, mb ko biết bạn là ai, có hợp lệ hay không

...

#31. Mô hình stateless

- Không dùng session

- Client sử dụng: access token/refresh token để định danh thay cho session

- Với mỗi lời gọi request, client gửi kèm token ở header: access token (đã mã hóa/encoded).

Token này chứa thông tin giúp định danh user là ai, và chỉ server mới có thể giải mã (decoded)

- Server sẽ decoded token gửi lên để biết ai là người thực hiện request
=> xử lý request như bình thường

- Để cho an toàn (trường hợp lộ access token, người khác có thể mạo danh bạn), access token thường có thời gian sử dụng ngắn (3 phút, 5 phút, 30 phút...)

Khi request gửi lên server với access_token đã hết hạn
=> thông báo lỗi, và cần sử dụng refresh_token

Sử dụng refresh token để đổi lấy access_token/refresh_token với thời hạn sử dụng mới.

#32. Ưu, nhược điểm của Stateless

Ưu điểm:

- Không dùng session
- Dựa hoàn toàn vào cơ chế tạo token (access_token, refresh_token)
- Backend xử lý đơn giản hơn khi chỉ quan tâm encoded/decoded token
- Có thể “xác thực” giữa các hệ thống khác nhau

Nhược điểm:

- Tìm ẩn rủi ro nếu người dùng để lộ/bị hack token
- 1 token khi đã issued (đã được cấp cho user), không có cách nào để có thể delete token đấy. Có nghĩa là, nếu token đang hợp lệ, bạn không thể delete token đấy

Chapter 8: JWT - Json Web Token

Sử dụng mô hình Stateless, server sẽ không có "session" để kiểm tra user có hợp lệ hay không, nên JWT (Json web token) là một giải pháp rất phổ biến được dùng cho mô hình này. Ở đây, chúng ta sẽ tìm hiểu về JWT, cũng như áp dụng nó để xác thực người dùng trong dự án NestJS

#33. JWT là gì ?

Tài liệu: <https://jwt.io/>

1. What ?

JWT - Json Web Token là 1 chuỗi ký tự đã được mã hóa (tương tự như việc hash password)

Mục đích mã hóa token là để trao đổi giữa các hệ thống với nhau và không làm lộ thông tin nhạy cảm (ví dụ frontend và backend)

Ví dụ về JWT:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyf
Q
.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

JWT gồm 3 thành phần:

- + Header (algorithm & token type) : chứa thuật toán mã hóa, và loại token
- + payload (data): data được truyền đi giữa các bên sử dụng . được sử dụng dưới dạng json (object)
- + verify signature (chữ ký): client ký vào token (đánh dấu). chỉ có server (nắm giữ secret) là có thể giải mã token này.

2. Why ?

- Sử dụng JWT như là 1 cách an toàn để trao đổi thông tin giữa các bên liên quan. (client/server)
- Hỗ trợ thuật toán mạnh mẽ (encoded), và chỉ có secret của server mới có thể giải mã (decoded)

3. When ?

- Có các hệ thống khác nhau, cần có hình thức để xác thực user

#34. Phân loại token sử dụng

1. Access Token

- Được backend issued (backend là người tạo ra token này & lưu dữ secret để decoded token/giải mã token)

- Thông thường được mã hóa dưới dạng JWT.

- Trường hợp hay gặp nhất, là user login . Nếu login thành công => server sẽ trả về cho client bộ access_token/refresh_token

Ứng với mỗi lời gọi request (API). client sẽ cần truyền thêm vào header access_token

Backend sẽ lấy thông tin của user trong token (ví dụ như username, email, id : những thông tin để định danh user là ai)

Lưu ý: không truyền thông tin nhạy cảm trong token (ví dụ như password, otp...)

2. Refresh Token

Để đảm bảo an toàn, token thường có thời hạn sống (expired date)

Vì nếu bị lộ token, người khác có thể mạo danh bạn.

access_token (chứa thông tin user) => có thời hạn sống ngắn

refresh_token (chứa thông tin để tạo ra bộ access_token/refresh_token mới), có thời hạn sống dài: 1 tháng, 3 tháng, 6 tháng, 1 năm...

3. Hình thức lưu trữ token

Frontend:

- **localStorage: access_token** -> thuận tiện việc truy cập/lưu trữ token.
lấy data từ localStorage: localStorage.getItem('item').

Đồng thời, do **access_token có thời hạn sống ngắn**,
nên khi bị lộ (hack localStorage), xác suất có rủi ro cũng giảm thiểu rất nhiều

- sessionStorage: không dùng, vì khi close browser sẽ mất dữ liệu

- cookies: refresh_token.

Cookies có nhiều chế độ để đảm bảo độ an toàn khi truy cập. ví dụ, chỉ cho phép server sử dụng cookies, http= true

client (javascript): không thể lấy cookies = hàm **document.cookie**

Cookies sẽ tự hết hạn theo 1 thời gian nhất định (nếu set expired date) => ko dùng nó cũng tự mất :v

Backend:

Backend có thể lưu trữ token ở memory (RAM), disk (file), hoặc database (hay dùng nhất).

#35. Giới thiệu về Passport.JS

Tài liệu: <https://www.passportjs.org/>

1. What ?

- Passport (hay passportjs) là thư viện giúp việc authentication (xác thực/login) với Node.js trở nên dễ dàng hơn bao giờ hết.

Ờ. dùng thư viện bao giờ không dễ. vậy tại sao cần passport, trong khi chúng ta có thể "tự code" được.

- Thực chất passport là 1 middleware, can thiệp vào req, và res. như vậy sẽ xác nhận được (authenticated) là user đã đăng nhập hay chưa.

- Lưu ý: **authentication vs authorization** (passport là xác thực người dùng đã đăng nhập chưa, ko liên quan gì tới phân quyền người dùng).

2. Why ?

Với việc login local (bạn tự quản lý database cùng với username/password), ok thì tự code được.

Vậy login với bên thứ 3 (third-party) thì sao, ví dụ như Facebook, Google, Apple, Amazon...

chẳng lẽ lại đọc tài liệu, rồi đi ghép api ?

Passport sinh ra để giải quyết vấn đề trên:

- đơn giản hóa việc xác thực người dùng (authentication)
- support hơn 500+ strategies (các loại login khác nhau)
- an toàn hơn so với việc bạn tự code, vì ít nhất thư viện đã được sử dụng rộng rãi + được testing về security

3. Strategies

- Do passport hỗ trợ rất nhiều "kiểu login", nên có rất nhiều strategies được làm ra (phục vụ các mục đích khác nhau)

Trong khóa học này, chúng ta sử dụng các strategies sau:

passport-local: xác thực người dùng thông qua username/password

passport-jwt: xác thực người dùng với jwt (json web token)

#36. Local Strategies với NestJS

Tài liệu: <https://docs.nestjs.com/recipes/passport>

Source code video này:

<https://drive.google.com/file/d/1arjVOi5rkypn91s8SkKn2L15H40eT5IP/view?usp=sharing>

Yêu cầu đặt ra: (mô hình stateless)

- Client login với username/password
- Nếu login thành công, server cần tạo ra access_token (dưới dạng jwt - json web token) gửi về cho client
- Kể từ lúc này, tất cả request gửi lên server, đều cần bearer token ở header (theo chuẩn oauth)
- Nếu request gửi lên ko có token => báo lỗi (protected routes)

1. Xác thực người dùng

Đây là quá trình client gửi lên server username/password. Nhiệm vụ của server là check xem thông tin đăng nhập có hợp lệ hay không ?

Cài đặt thư viện:

```
npm install --save-exact @nestjs/passport@9.0.3 passport@0.6.0  
passport-local@1.0.0
```

cài đặt package check type khi code typescript:

```
npm install --save-dev @types/passport-local
```

Cài 3 thư viện (không phải là cài 1 thư viện duy nhất vì):

- **passport** là thư viện gốc => giúp ra tạo ra middleware (can thiệp vào req và res), và **lưu trữ thông tin người dùng đăng nhập (req.user)**

- **@nestjs/passport** là thư viện viết theo phong cách của nestjs, giúp việc can thiệp vào passport dễ dàng hơn

- **passport-local**: đây là strategy hỗ trợ việc đăng nhập sử dụng username/password

Sau này, khi cần tạo ra jwt, chúng ta sẽ cài thêm strategy: passport-jwt

Về cú pháp Imports, exports sử dụng trong video, các bạn tham khảo ở đây:

<https://docs.nestjs.com/modules>

Với @Modules thì:

providers: the providers that will be instantiated by the Nest injector and that may be shared at least across this module

Providers thông thường là các services, chúng ta dùng để injects vào constructor thông qua DI (dependency injection)

Imports: the list of imported modules that export the providers which are required in this module

Chúng ta sẽ cần import vào modules sử dụng

Exports: the subset of providers that are provided by this module and should be available in other modules which import this module. You can use either the provider itself or just its token (provide value)

Chúng ta cần xuất ra (xuất services/providers) thì cái import ở trên mới có tác dụng

#37. Nestjs Guard

1. Middleware

Tài liệu: <https://expressjs.com/en/guide/using-middleware.html>

Với Node.js, chúng ta có 1 khái niệm là middleware

myfunction (req, res, next){ ...}

request => middleware => response

req => routes => middleware => controller => service => res

Có 2 trường hợp xảy:

- req -> route -> middleware. dữ liệu ko hợp lệ, không thực thi tiếp -> res (thông báo lỗi)
- req -> route -> **middleware. dữ liệu hợp lệ => next()** -> controller -> service -> res

Hiểu đơn giản, middleware, như tên gọi của nó. giúp bạn can thiệp vào giữa req và res :

req -> middleware -> res

2. Guards

Tài liệu: <https://docs.nestjs.com/guards>

Guards (chú bảo vệ :v)

Nó làm nhiệm vụ giống middleware, can thiệp vào req và res . **req -> guards -> res**

But middleware, by its nature, is dumb.

Nếu bạn dùng middleware, bạn chỉ có thể can thiệp vào req và res (req, res, next).

Ví dụ: router('/test', myMiddleware, myController).

với middleware, bạn không thể biết "handler" phía sau là gì, vì lúc nào, bạn cũng làm việc với req và res. phần còn lại là hàm next () đã lo :v

đấy là lý do nó "dumb" => hơi ngầu ngầu ấy :v

Guards thì hoàn toàn ngược lại, nó mạnh mẽ hơn nhờ middleware.
Ngoài khả năng truy cập req, res, nó còn được sử dụng "**ExecutionContext**".

Hiểu 1 cách đơn giản, là không gian thực thi code.

p/s: google để hiểu tổng quan về execution context của javascript.

Guard có nhiệm vụ check true/false

- Nếu true: cho đi tiếp
- Nếu false: trả về phản hồi

3. Ví dụ về viết và sử dụng Guard đơn giản

#38. LocalGuard với Passport

Tài liệu: <https://docs.nestjs.com/recipes/passport#built-in-passport-guards>

Source code video này:

https://drive.google.com/file/d/1ZvOqs8UkGLVI9qqpuJ_9BzzDpZ6r_ofB/view?usp=share_link

#39. Sử dụng JWT

Tài liệu: <https://docs.nestjs.com/recipes/passport#jwt-functionality>

Source code video này:

https://drive.google.com/file/d/1SAJLmVXCou3i55PHH6OJdNmMi5N36QPo/view?usp=share_link

Cài đặt thư viện:

```
npm install --save-exact @nestjs/jwt@10.0.3 passport-jwt@4.0.1
```

```
npm install --save-dev @types/passport-jwt
```

Việc đã làm:

- Xác thực được user đã đăng nhập thành công hay chưa với **passport-local** strategy
- hàm validateUser được sử dụng cho việc trên
- khi login thành công, đã lưu được **thông tin user vào req.user**

Việc chưa làm:

- login thành công chưa trả về jwt (json web token) => sẽ làm trong video này
- chưa bảo vệ route với jwt (sẽ làm ở #40)

Về cú pháp Import/Exports... của modules, tham khảo tại:

<https://docs.nestjs.com/modules>

#40. Implementing Passport JWT

Tài liệu: <https://docs.nestjs.com/recipes/passport#implementing-passport-jwt>

todo: bảo vệ route với jwt

Source code video này:

https://drive.google.com/file/d/1WZJLJxYYCkMasJOEe80TLCoCb-S0oJBP/view?usp=share_link

#41. Enable authentication globally

check jwt cho tất cả api

Tài liệu: <https://docs.nestjs.com/recipes/passport#enable-authentication-globally>

Source code video này:

https://drive.google.com/file/d/1btWabnyYyR_E6jPZTCGJv4QP3KUhlmch/view?usp=sharing

#42. Disable Guard

Source code video này:

https://drive.google.com/file/d/1IfXZboNBgsosy_hAB4C0f5pb9B34Fgiw/view?usp=share_link

Sử dụng @Public()

Về metadata: <https://stackoverflow.com/a/73171823>

Tham khảo:

<https://stackoverflow.com/questions/49429241/nest-js-global-authguard-but-with-exceptions>

Về set global guard in main.ts:

<https://github.com/nestjs/nest/issues/964#issuecomment-480834786>

#43. Fix bugs và tổng kết

Tài liệu: <https://www.npmjs.com/package/ms>

Source code video này:

https://drive.google.com/file/d/1eQFooS_kXQ0hTZXy5y-gNOjAHZpxAp6N/view?usp=share_link

1. Convert String to milliseconds

Cài đặt thư viện:

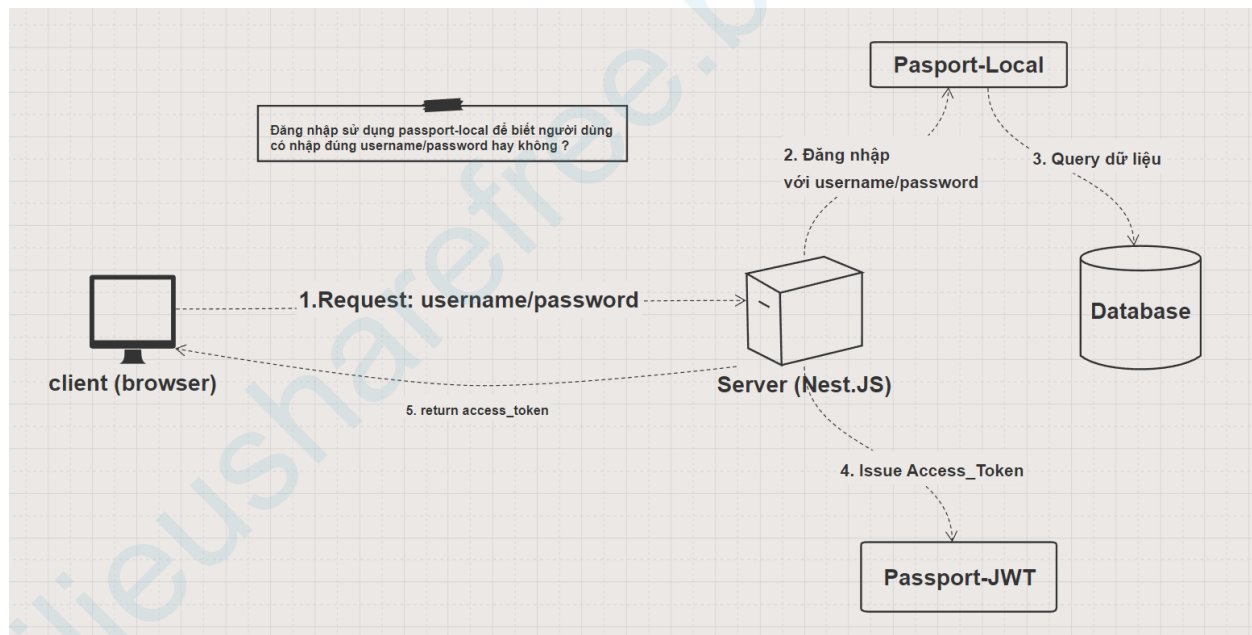
npm i --save-exact ms@2.1.3

npm i --save-dev @types/ms

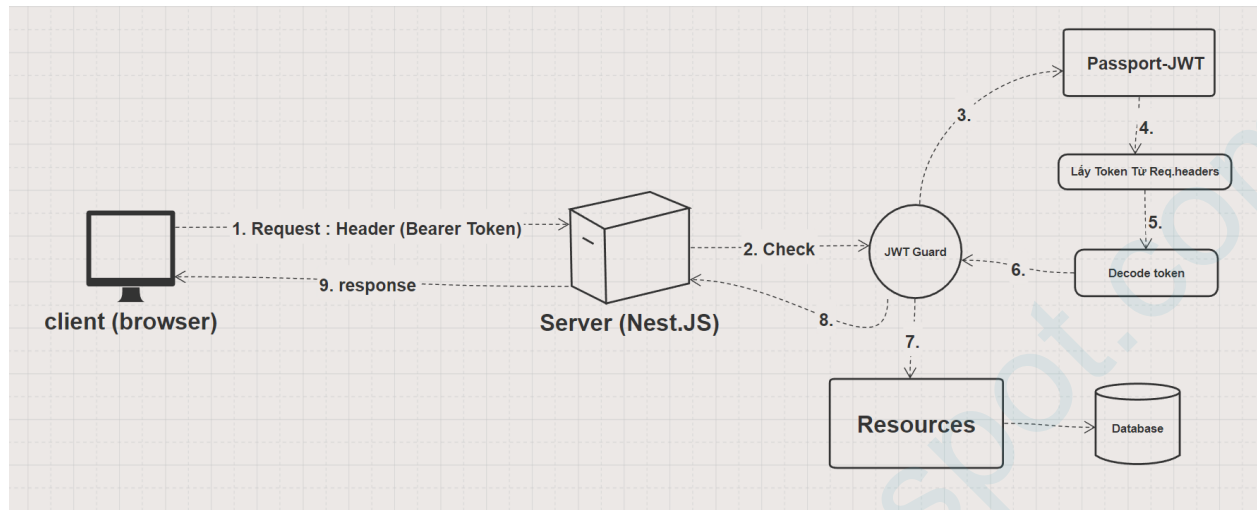
and adding "esModuleInterop": true to compilerOptions of tsconfig.json works for me.

2. Mô hình hoạt động của passport với Local/JWT strategy

Lưu thông login để lấy access_token:



Luồng protected routes với JWT (cần truyền lên JWT ở header request)



Chapter 9: Tư duy phân tích database

Giới thiệu đề tài dự án thực hành cuối khóa, cũng như tư duy phân tích database, cách thiết kế table và mối quan hệ giữa các đối tượng trong dự án thực hành.

#44. Giới thiệu đề bài

Ý tưởng:

<https://itviec.com/>

<https://topdev.vn/>

Đăng tin tuyển dụng, việc làm => kết hợp việc phân quyền

- Ứng viên có thể tìm việc làm theo skills
- Nhà tuyển dụng có thể đăng việc làm
- Đặt lịch (schedule) gửi mail cho subscribers

Phân tích chi tiết chức năng

- Đăng ký, đăng nhập (basic: local)
- Tạo skills để search
- nhà tuyển dụng : thông tin giới thiệu về cty
- ứng viên: người có thể xem bài đăng tuyển dụng và gửi CV
- admin: duyệt CV rồi mới gửi tới nhà tuyển dụng

Nếu 1 ứng viên subscribe 1 skills => gửi mail hàng tuần về những jobs này

#45. Phân tích model & relationship

Warning: với NoSQL, không nên dựa vào relationship, đây không phải là cách làm tối ưu nhất.

Link download file excel:

https://docs.google.com/spreadsheets/d/1KH4FZDKpkQtBmBe_L3y1ZgYebH8BkDya/edit?usp=sharing&ouid=107384225622911964215&rtpof=true&sd=true

1. Đối tượng

Actors: (người sử dụng hệ thống)

- **Ứng viên** (employee)
- **Nhà tuyển dụng**: company
- **admin**

đối tượng khác:

- **CV** của ứng viên
- **jobs** đăng tuyển
- **Skills** để search
- **role**: vai trò của user trong hệ thống (admin, hr...)
- **permission**: quyền hạn sử dụng hệ thống - ám chỉ api của backend

2. Relationship

1 ứng viên có thể gửi đi nhiều CV 1 lúc

1 CV chỉ thuộc 1 ứng viên

1 nhà tuyển dụng có thể đăng nhiều jobs

1 job chỉ thuộc 1 nhà tuyển dụng

1 skills có thể thuộc nhiều jobs

1 jobs có thể có nhiều skills

Về phân quyền:

1 permission (quyền hạn), ám chỉ 1 apis của backend

1 role (vai trò), bao gồm nhiều quyền hạn

1 người dùng sẽ có 1 role duy nhất. nếu muốn merge role => tạo role mới

1 users -> có 1 role

1 role có thể có nhiều user cùng có role này

1 role -> có nhiều permission (apis)

1 permission -> có thể thuộc nhiều role khác nhau

#46. Thiết kế model

1. Users: (admin, ứng viên, nhân viên cty -> phân biệt nhờ role)

- name: string
- email: string <unique>
- password: string
- age: number
- gender: string
- address: string
- company: object {_id, name}
- role: string
- refreshToken: string;
- createdAt: Date
- updatedAt: Date
- isDeleted: boolean
- createdBy: object {_id, email}
- updatedBy: object {_id, email}
- deleteBy: object {_id, email}

2. Companies

- name: string
- address: string
- description: string <markdown>
- createdAt: Date
- updatedAt: Date
- isDeleted: boolean
- createdBy: object {_id, email}
- updatedBy: object {_id, email}
- deleteBy: object {_id, email}

3. Resume

- email: string
- userId: objectId
- url: string
- status: string //PENDING-REVIEWING-APPROVED-REJECTED

history: array object [{ status: string, updatedAt: Date, updatedBy: {_id, email}}]

- createdAt: Date
- updatedAt: Date
- isDeleted: boolean
- createdBy: object {_id, email}
- updatedBy: object {_id, email}
- deleteBy: object {_id, email}

4. Jobs

- name: string
- skill: string
- company: string
- location: string
- salary: number
- quantity: string (số lượng tuyển)
- level: string //INTERN/FRESHER/JUNIOR/SENIOR
- description: string <markdown>
- startDate: Date
- endDate: Date
- isActive: boolean

- createdAt: Date
- updatedAt: Date
- isDeleted: boolean
- createdBy: object {_id, email}
- updatedBy: object {_id, email}
- deleteBy: object {_id, email}

5. Subscribers

- email: string
- skill: string

6. Roles

- name: string <unique>
- description: string
- isActive: boolean
- permissions: array object

- createdAt: Date
- updatedAt: Date
- isDeleted: boolean
- createdBy: object {_id, email}
- updatedBy: object {_id, email}
- deleteBy: object {_id, email}

7. Permissions

- name: string
- path: string
- method: string
- description: string

- createdAt: Date
- updatedAt: Date
- isDeleted: boolean
- createdBy: object {_id, email}
- updatedBy: object {_id, email}
- deleteBy: object {_id, email}

Chapter 10: Mongoose Plugins

Để có thể sử dụng Mongoose (MongoDB) hiệu quả, chúng ta sẽ cùng nhau cài đặt các plugin hỗ trợ việc Query/Filter kết quả, thực hiện "soft-delete", tự động tạo "timestamp" và đặc biệt, khắc phục lỗi CORS với NestJS

#47. Timestamps plugin

1. Timestamps

<https://mongoosejs.com/docs/guide.html#timestamps>

Mục đích:

- khi create => tự động update trường createdAt/updatedAt
- khi update => tự động update trường updatedAt

cần set option: cho schema

<https://stackoverflow.com/a/71743707>

@Schema({timestamps: true }) và khai báo @Prop createdAt/updatedAt

#48. Soft-delete plugin

- Lưu ý:

Với dự án đã có sẵn data, cần update "data cũ" với 2 fields là deletedAt và isDeleted => duplicate data (or error nếu unique)

#soft-delete

<https://www.npmjs.com/package/soft-delete-plugin-mongoose>

Cài đặt:

npm i soft-delete-plugin-mongoose

1. import plugin với nestjs:

<https://docs.nestjs.com/techniques/mongodb#plugins>

```
import { softDeletePlugin } from 'soft-delete-plugin-mongoose';
```

```
MongooseModule.forRootAsync({  
  imports: [ConfigModule],  
  useFactory: async (config: ConfigService) => ({  
    uri: config.get<string>('MONGO_DB_URL'),  
    connectionFactory: (connection) => {  
      connection.plugin(softDeletePlugin);  
      return connection;  
    }  
  })),  
  inject: [ConfigService],  
}),
```

2. Merge type cho Model

<https://github.com/nour-karoui/mongoose-soft-delete>

//2nd way (nestjs way)

```
constructor(@InjectModel('Test') private readonly testModel: SoftDeleteModel<Test>) {}
```

lưu ý: update model thêm 2 fields sau:

deletedAt: null,

isDeleted: false

#49. Query Builder

Tài liệu: <https://www.npmjs.com/package/api-query-params>

Cài đặt: **npm i --save-exact api-query-params@5.4.0**

tailieusharefree.blogspot.com

#50. Setup dự án frontend

Link source code github: <https://github.com/harypham/react-for-nest>

Source code final (cả khóa học) sẽ là nhánh master

Lưu ý: checkout code theo từng modules để thực hành song song với backend

#51. CORS là gì ?

Tài liệu: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

**Demo CORS: checkout source code frontend sang nhánh cors
git checkout cors**

1.What?

Cross-Origin Resource Sharing (CORS): là cơ chế "server" cho phép "origin (domain/port)" từ phía "browser" được truy cập nguồn tài nguyên (APIs)

CORS sẽ xảy ra, khi browser gửi request lên server, và server "chưa cấu hình CORS)

2.Why?

ví dụ nếu không có CORS:
facebook.com

POST facebook.com/change-password

bạn dùng máy tính (browser), nhận được tin nhắn từ người lạ, là đường link : **hacker.com**
Vì tò mò, bạn click vào link trên, và 5 phút sau account FB bị đổi mật khẩu, Why ?

===

Cơ chế của browser:

- Khi bạn login thành công, sẽ lưu "**cookies**". Cookies này sẽ lưu tại máy tính bạn. gồm 2 thành phần:

- + **domain**: tên website sử dụng cookies này
- + **value**: giá trị của cookies

Nếu bạn login google thành công, máy tính bạn sẽ có cookies thứ nhất:

- + **domain: google.com**
- + value: ...

Nếu bạn login facebook thành công, máy tính bạn sẽ có cookies thứ hai:

- + **domain: facebook.com**
- + value: ...

Bạn tắt máy tính & đi ngủ. ngày hôm sau vào google, facebook không cần đăng nhập nữa, Why ?

===

Hiện tại, máy bạn đang có 2 cookies ở trên, khi bạn vào facebook.com, trình duyệt "sẽ tự động gửi" những cookies nào dành cho domain trên.

Như vậy, nhờ có cookies mà bạn "không cần đăng nhập nhiều lần".

Tương tự, khi vào google.com, browser sẽ tự động gửi kèm cookies ứng với domain này.

Check = cách F12, check request header

**=> như vậy, việc gửi kèm cookies này là do browser tự làm.
và cookies "đại diện/tượng trưng" cho user (mà không cần password)**

=====

Khi không có CORS:

- máy tính bạn "đã lưu" cookies của facebook
- bạn vào hacker.com, và trang này gọi request tới facebook.com => browser sẽ tự động gửi kèm cookies của bạn tới facebook.com

Nếu hacker.com thực hiện POST /change-password => change password dựa vào cookies, mà không cần người dùng nhập mật khẩu :v

Thực tế: do facebook.com và hacker.com là 2 domain khác nhau, nên sẽ bị chặn CORS default :v

#52.1 Làm sao để bypass CORS ?

1. update browser

Chỉ sử dụng cách này nếu bạn **"chỉ code Frontend"** và **"không kiểm soát backend"**

settings, extensions

2. Sử dụng Backend

- **Backend do bạn kiểm soát**, lỗi xảy ra khi frontend gọi lên backend => **fix ở backend**
(cách làm này sẽ được hướng dẫn ở video tiếp theo)

- **Backend do bên thứ 3 làm.**

chúng ta chỉ có "mỗi APIs để sử dụng", không có khả năng sửa đổi, cập nhật backend đấy

=> cần viết backend để gọi tới backend

Mô hình: **frontend** = > **backend (của bạn)** => **backend (của đối tác)**

Bạn có biết rằng , thư viện axios, dùng được cả frontend lẫn backend :v

#52.2 Setup Cors NestJS

Tài liệu:

<https://docs.nestjs.com/security/cors>

<https://github.com/expressjs/cors#configuration-options>

```
{  
  "origin": "*",  
  "methods": "GET,HEAD,PUT,PATCH,POST,DELETE",  
  "preflightContinue": false,  
  "optionsSuccessStatus": 204  
}
```

Chapter 11: Modules Company

Bài tập thực hành: tạo module "Company". Ngoài việc thực hiện CRUD, chúng ta sẽ tìm hiểu cách phân trang (paginate data), sử dụng Interceptor để xử lý data trước khi trả về client, customize message response cũng như "đánh version" cho API

#53. Generate Modules Company

- generate: **nest g resource companies --no-spec**
- tạo schema (lưu ý về timestamp và soft-delete)
- tạo DTO + validation

#54. Bài Tập Create Company

Yêu cầu:

Tạo mới endpoint:

(cần truyền token ở header)

POST /companies

Body { name, address, description }

Có validate dữ liệu khi tạo mới

#55. Update User Type (JWT)

1. Create user interface

// users.interface.ts

```
export interface IUser {  
  _id: string;  
  name: string;  
  email: string;  
  role: string;  
}
```

2. Update JWT token (issue token when login)

iss (issuer),
exp (expiration time),
sub (subject),
aud (audience),
iat : timestamp of creation of the JWT

//auth.service.ts

```
async login(user: IUser) {  
  const { _id, name, email, role } = user;  
  const payload = {  
    sub: "token login",  
    iss: "from server",  
    _id,  
    name,  
    email,  
    role  
  };  
  return {  
    access_token: this.jwtService.sign(payload),  
    _id,  
    name,  
    email,  
    role  
  };  
}
```

3. Update decode token

//jwt.strategy.ts

```
async validate(payload: IUser) {  
  const { _id, name, email, role } = payload;  
  return {  
    _id,  
    name,  
    email,  
    role  
  };  
}
```

4. Update message error

//local.strategy.ts

```
async validate(username: string, password: string): Promise<any> {  
  const user = await this.authService.validateUser(username, password);  
  if (!user) {  
    throw new UnauthorizedException("Invalid Username/Password !");  
  }  
  return user; //req.user  
}
```

//jwt-auth.guard.ts

```
handleRequest(err, user, info) {  
  // You can throw an exception based on either "info" or "err" arguments  
  if (err || !user) {  
    throw err || new UnauthorizedException("Token không hợp lệ");  
  }  
  return user;  
}
```

5. Move /login => auth controller

#56. Passing Req.user

Tài liệu:

<https://docs.nestjs.com/custom-decorators#param-decorators>

Mục đích: passing request to service => lấy thông tin users đang đăng nhập

//decorator/customize.ts

```
export const User = createParamDecorator(
  (data: unknown, ctx: ExecutionContext) => {
    const request = ctx.switchToHttp().getRequest();
    return request.user;
  },
);
```

//update controller => passing user

```
@Post()
create(@Body() createCompanyDto: CreateCompanyDto, @User() user: IUser) {
  return this.companiesService.create(createCompanyDto, user);
}
```

//update service

```
create(createCompanyDto: CreateCompanyDto, user: IUser) {
  return this.companyModel.create({
    ...createCompanyDto,
    createdBy: {
      _id: user._id,
      email: user.email
    }
  })
}
```

#57. Bài Tập Update Company

Yêu cầu:

Tạo endpoint:

Patch /companies/id

(lưu ý truyền token JWT ở header)

Body {

name, address, description

}

Có validate dữ liệu, và đồng thời cập nhật trường updatedBy.

#58. Bài Tập Delete Company

limitation of plugin :v

<https://github.com/nour-karoui/mongoose-soft-delete/blob/main/src/soft-delete-plugin.ts>

Yêu cầu:

Tạo endpoint:

Delete **/companies/id**

Không cần truyền lên body (lưu ý truyền token JWT ở header)

đồng thời cập nhật trường deletedBy. (soft-delete)

#59. Query with Pagination

Tài liệu : <https://www.npmjs.com/package/api-query-params>

Input: Frontend truyền lên:

- page: number (trang hiện tại)
- limit: number (số lượng bản ghi muốn lấy)
- query string: điều kiện query (ví dụ tìm theo tên, tìm theo email...)

Output:

```
return {  
  meta: {  
    current: page, //trang hiện tại  
    pageSize: limit, //số lượng bản ghi đã lấy  
    pages: totalPages, //tổng số trang với điều kiện query  
    total: totalItems // tổng số phần tử (số bản ghi)  
  },  
  result //kết quả query  
}
```

Logic xử lý:

- Để phân trang dữ liệu, cần **tìm hiểu về offset và limit**
- Cần lấy ra @Query page/limit
- Count tổng bản ghi với điều kiện filter
- tính offset

@Query /skip/limit

```
const { filter, projection, population } = aqp(rq);  
let { sort } = aqp(rq);  
let offset = (+page - 1) * (+limit);  
let defaultLimit = +limit ? +limit : 10;
```

```
const totalItems = (await this.model.find(filter)).length;  
const totalPages = Math.ceil(totalItems / defaultLimit);
```

```
if (isEmpty(sort)) {  
  // @ts-ignore: Unreachable code error  
  sort = "-updatedAt"  
}
```

```
const result = await this.model.find(filter)  
  .skip(offset)  
  .limit(defaultLimit)  
  // @ts-ignore: Unreachable code error  
  .sort(sort)  
  .populate(population)  
  .exec();
```

```
let { sort } = <{sort: any}>aqp(rq);  
let { sort }: {sort: any} = aqp(rq);  
.sort(sort as any)
```

#60. Giới Thiệu Interceptor

Tài liệu:

<https://docs.nestjs.com/interceptors>

Interceptor là cách can thiệp vào "req và res", tương tự guard - thông minh hơn middleware

Reactive programming:

<https://rxjs.dev/>

Observable > < Promise

#61.1 Transform Response

Tài liệu:

<https://docs.nestjs.com/interceptors#response-mapping>

<https://stackoverflow.com/questions/60189849/how-to-format-response-before-sending-in-nest-js>

<https://docs.nestjs.com/interceptors#binding-interceptors>

Yêu cầu: thống nhất data trả ra của backend (trường hợp thành công), theo định dạng format:

```
{
  message: "",
  statusCode: "", //200 404
  data: ""
}
```

trường hợp lỗi :

<https://docs.nestjs.com/exception-filters>

default:

```
{
  statusCode: "",
  message: ""
}
```

// nếu dùng custom message:

```
{
  "statusCode": "",
  "message": "",
  "error": ""
}
```

//file: transform.interceptor.ts

```
import {
  Injectable,
  NestInterceptor,
  ExecutionContext,
  CallHandler,
} from '@nestjs/common';
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';

export interface Response<T> {
  statusCode: number;
  message?: string;
  data: any;
}

@Injectable()
export class TransformInterceptor<T>
  implements NestInterceptor<T, Response<T>> {
  intercept(
    context: ExecutionContext,
    next: CallHandler,
  ): Observable<Response<T>> {
    return next
      .handle()
      .pipe(
        map((data) => ({
          statusCode: context.switchToHttp().getResponse().statusCode,
          // message: data.message,
          data: data
        }))),
      );
  }
}
```


#61.2 Customize Message (Decorator)

//todo

tailieusharefree.blogspot.com

#62. Version APIs

Tài liệu: <https://docs.nestjs.com/techniques/versioning#uri-versioning-type>

express: `app.use("/api/v1", router)`

Thực tế:

- Backend là server APIs
- Cần phân chia theo version của api

Ví dụ: <https://developers.facebook.com/docs/graph-api/get-started>

Do các hệ thống (lớn) phát triển theo thời gian, đôi khi có "breaking change" giữa các version => backend thường dùng "version"

- Nếu bạn code 1 mình, ko cần version

- Nếu bạn sử dụng dịch vụ của bên thứ 3 (Facebook, Google, Momo), hoặc bạn viết "api cho đối tác dùng" => dùng version cho apis

```
app.enableVersioning({  
  type: VersioningType.URI,  
  // defaultVersion: ['1', '2'],  
  // prefix: 'api/v'  
});
```

sử dụng global-prefix cho backend:

//<https://docs.nestjs.com/faq/global-prefix>

Chapter 12: Modules User

Bài tập thực hành: tạo module User. Ngoài CRUD, chúng ta sẽ tìm hiểu về cookies, cơ chế tạo access token, refresh token cho client.

#63. Update User Schema

1. Update model

Users: (admin, ứng viên, nhân viên cty -> phân biệt nhờ role)

_id

- name: string
- email: string <unique>
- password: string
- age: number
- gender: string
- address: string
- company: object {_id, name}
- role: string
- refreshToken: string;
- createdAt: Date
- updatedAt: Date
- isDeleted: boolean
- createdBy: object {_id, email}
- updatedBy: object {_id, email}
- deleteBy: object {_id, email}

2. Update dto/validation

Validate nested object => company

<https://stackoverflow.com/questions/53786383/validate-nested-objects-using-class-validator-and-nestjs>

#64.1 Bài Tập CRUD Users

Lưu ý:

- đang hardcode ROLE & chưa áp dụng phân quyền ở đây
- về frontend: admin có view create user riêng. client có view register.

Các endpoint cần tạo (6 apis)

1. POST /api/v1/auth/register

- **Không cần truyền lên JWT** (vì chức năng này dành cho client -> đăng ký tài khoản)
- Body : name , email , password , age , gender , address

Ở phía backend, hardcode 'role' === USER & cần hash password trước khi lưu.
Không cần cập nhật createdBy (vì không sử dụng jwt token)

The screenshot shows a REST client interface with a POST request to `http://localhost:8000/api/v1/auth/register`. The request body is set to `x-www-form-urlencoded`. The body contains the following key-value pairs:

Key	Value
name	hỏi dân it
email	eric@gmail.com
password	123456
age	15
gender	male
address	vietnam

Response:

```
{
  "statusCode": 201,
  "message": "Register a new user",
  "data": {
    "_id": ".....", //id của user được tạo
    "createdAt": "....." //thời gian tạo user
  }
}
```

2. POST /api/v1/users

- Cần truyền lên JWT
- Body:
name, email, password
age, gender, address, role
company: object {_id, name}

Ở backend, tự động cập nhật **createdBy**: object {id, email}

=> Data postman sử dụng dạng RAW (json)



Lưu ý về việc import trùng tên User ở service (model và decorator)

Response:

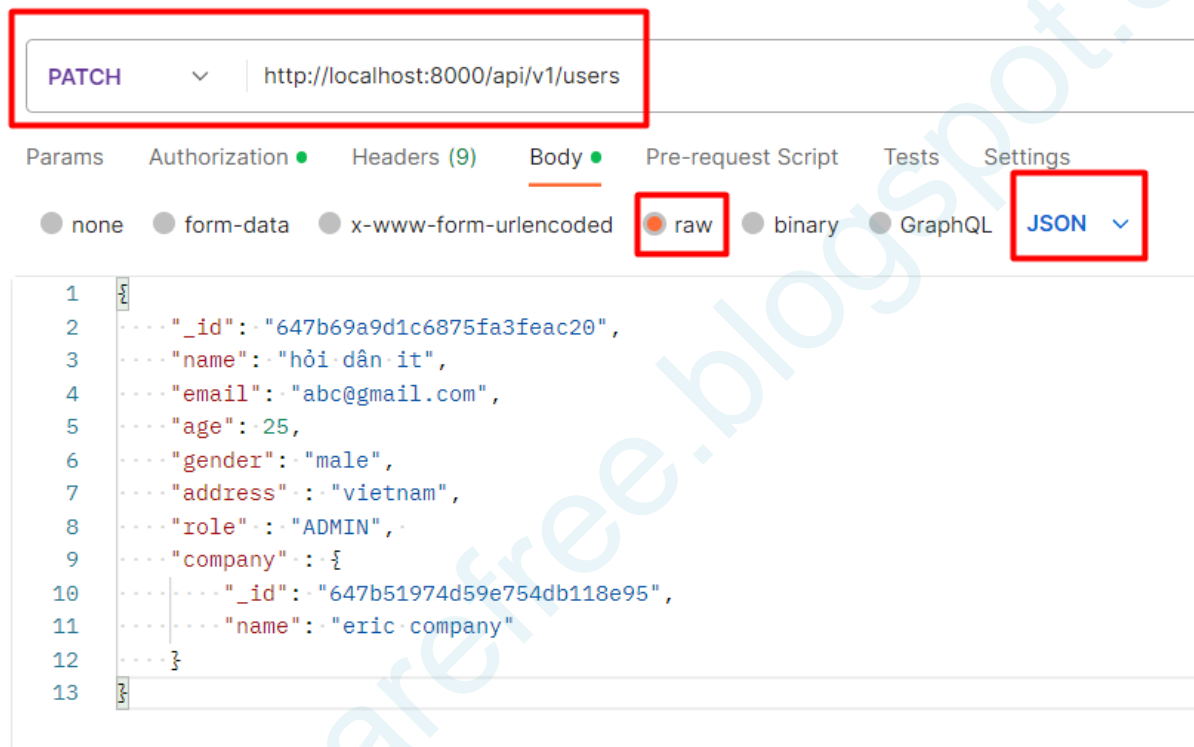
```
{
  "statusCode": 201,
  "message": "Create a new User",
  "data": {
    "_id": ".....",
    "createdAt": "....."
  }
}
```

3. PATCH /api/v1/users

- Cần truyền lên JWT
- Body:
_id , name , email, age, gender, address, role
company: object {_id, name}

Ở backend, tự động cập nhật **updatedBy**: object {id, email}

=> Data postman sử dụng dạng RAW (json)



Response:

```
{
  "statusCode": 200,
  "message": "Update a User",
  "data": {
    "acknowledged": ...,
    "modifiedCount": .....,
    "upsertedId": .....,
    "upsertedCount": .....,
    "matchedCount": .....
  }
}
```

4. DELETE /api/v1/users/:id

- Cần truyền lên JWT

The screenshot shows a REST client interface with a red box highlighting the method and URL. The method is 'DELETE' and the URL is 'http://localhost:8000/api/v1/users/64607ce86b2dad95a0f761e1'. Below the URL bar, there are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Authorization' tab is selected, showing a 'Type' dropdown set to 'Bearer Token' and a 'Token' input field containing a JWT token: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...'. A warning message is displayed: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborator, we recommend using variables. Learn more about variables'.

Ở backend, tự động cập nhật **deletedBy**: object {id, email} và sử dụng soft-delete

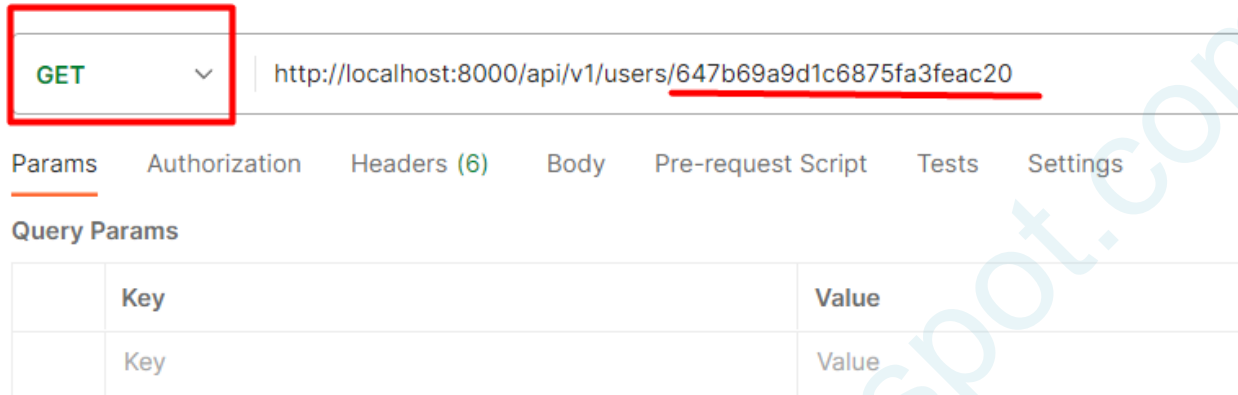
Response:

```
{
  "statusCode": 200,
  "message": "Delete a User",
  "data": {
    "deleted": .....
  }
}
```

5. GET /api/v1/users/:id

Fetch user by ID

- Không cần truyền lên JWT



Key	Value
Key	Value

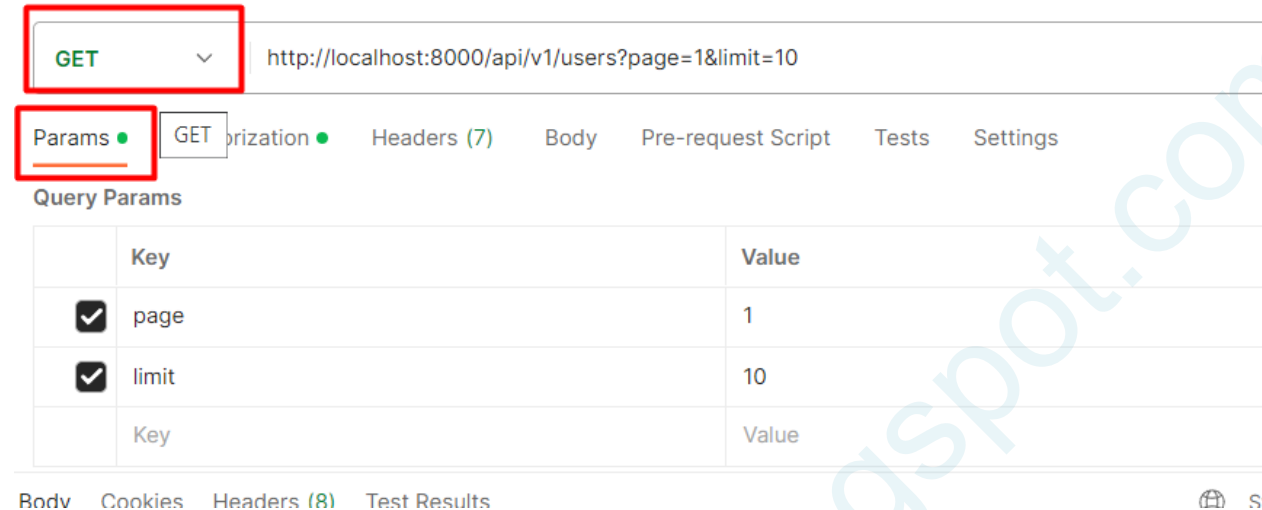
Response: (không trả về password) => trả về full record (without password)

```
{
  "statusCode": 200,
  "message": "Fetch user by id",
  "data": {
    "_id": ".....",
    "name": "hỏi dân it",
    "email": "abc@gmail.com",
    "age": 25,
    "gender": "male",
    "address": "vietnam",
    "company": ".....",
    "role": "ADMIN",
    "createdBy": "....",
    "isDeleted": ".....",
    "deletedAt": ".....",
    "createdAt": ".....",
    "updatedAt": ".....",
    "updatedBy": "...."
  }
}
```


6. GET /api/v1/users

Fetch users with paginate

- Cần truyền lên JWT



The screenshot shows a REST client interface. The method is GET and the URL is `http://localhost:8000/api/v1/users?page=1&limit=10`. The 'Params' tab is selected, showing the following query parameters:

Key	Value
page	1
limit	10

At the bottom, there are tabs for Body, Cookies, Headers (8), and Test Results.

Response: (không trả về password) => trả về full record (without password)

```
{
  "statusCode": 200,
  "message": "Fetch user with paginate",
  "data": {
    "meta": {
      "current": 1,
      "pageSize": 10,
      "pages": .....,
      "total": ....
    },
    "result": [ ..... //users data ]
  }
}
```

#64.2 Hướng Dẫn Bài Tập CRUD Users

//todo

tailieusharefree.blogspot.com

#65. Demo Chức năng login

- yêu cầu:

+ đăng nhập thành công, trả ra access_token và thông tin user

+ lưu refresh_token vào cookies

#66. API login

Yêu cầu:

1. Update API login

POST <http://localhost:8000/api/v1/auth/login>

Response:

```
{
  "statusCode": 201,
  "message": "User Login",
  "data": {
    "access_token": ".....",
    "user": {
      "_id": ".....",
      "name": ".....",
      "email": "....."
    }
  }
}
```

2. Refresh Token

Khi login, cần tạo ra Refresh token

- User login thành công, trước khi return access_token, **cần update thông tin user với refresh token được tạo**
- Refresh token có thể lưu dưới 2 định dạng: jwt hoặc là 1 chuỗi string ngẫu nhiên.
=> sử dụng jwt
- Cần lưu cookies (refresh_token) cho người dùng
<https://docs.nestjs.com/techniques/cookies>

#67. Set Cookies

Tài liệu: <https://docs.nestjs.com/techniques/cookies>

Cài đặt thư viện:

npm i cookie-parser

npm i -D @types/cookie-parser

#68. Bài Tập API Get Account (F5 Refresh)

Bài toán:

- Với mô hình stateful, khi user F5 (refresh website), client sẽ gửi lên session_id (lưu ở cookies), server sẽ check session để biết user nào đang đăng nhập.
- **Với mô hình stateless**, không có sử dụng session, thay vào đây là access_token và refresh_token.
=> khi user F5, cần gọi API của backend, vì client không có khả năng decode (giải mã) access_token để biết được ai là người đang đăng nhập.

Yêu cầu:

Tạo endpoint: **GET /api/v1/auth/account**

- Chỉ cần truyền lên JWT ở header

Response:

```
{
  "statusCode": 200,
  "message": "Get user information",
  "data": {
    "user": {
      "_id": "...",
      "name": "...",
      "email": "...",
      "role": "..."
    }
  }
}
```

#69.1 API Refresh Token (Part 1)

1. Bài toán:

Client sử dụng API, và khi access_token hết hạn => **server sẽ trả ra mã lỗi 401**

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/401>

Code 401 (unauthorized), có nghĩa là không xác thực được người dùng => người dùng không truyền lên access_token, hoặc có truyền lên access_token, nhưng hết hạn.

=> khi nhận code 401, client (frontend) sẽ tự động gọi API refresh_token, sử dụng token này để đổi lấy {access_token, refresh_token} mới.

2. Yêu cầu: tạo endpoint

GET api/v1/auth/refresh

<https://docs.nestjs.com/techniques/cookies>

Response: (trả ra giống hệt như khi login)

```
{
  "statusCode": 200,
  "message": "Get User by refresh token",
  "data": {
    "access_token": "...",
    "user": {
      "_id": "...",
      "name": "...",
      "email": "...",
      "role": "...."
    }
  }
}
```

3. Các bước xử lý

- **Server lấy ra refresh_token từ cookies**
- **Server check (verify) để biết refresh_token có hợp lệ hay không ?**
- Server query database theo refresh_token
=> lấy thông tin user
=> issue access_token mới
- Server trả ra phản hồi (set cookies ứng với refresh_token mới)

#69.2. API Refresh Token (Part 2)

todo

tailieusharefree.blogspot.com

#70. Bài Tập API Logout

Yêu cầu: Tạo endpoint

POST api/v1/auth/logout

Truyền lên JWT ở header

Response :

```
{
  "statusCode": 201,,
  "message": "Logout User",
  "data": "ok"
}
```

Xử lý ở backend:

- Update refresh_token === null (empty)
- Remove refresh_token ở cookies (remove cookies)
- Trả về phản hồi cho client

#71.1 Test giao diện frontend (Part 1)

//todo: fix bugs

#71.2 Test giao diện frontend (Part 2)

Link github: <https://github.com/harypham/react-for-nest>

Checkout sang nhánh code mới để test source code video này, sử dụng câu lệnh:

git checkout test-1

Với backend, khi frontend truyền lên "cookies", thì cần phải cấu hình CORS:

```
//config cors
app.enableCors(
  {
    "origin": true,
    "methods": "GET,HEAD,PUT,PATCH,POST,DELETE",
    "preflightContinue": false,
    "credentials": true
  }
);
```

Sự khác nhau giữa **"origin": true** và **"origin": *** là

***** : cho phép kết nối tới từ bất cứ nơi đâu

true: chỉ cho phép kết nối từ cùng origin với server,

ví dụ: localhost => localhost , không thể từ hoidanit.com => localhost.com

Chapter 13: Modules Job/Resume

Bài tập thực hành tạo module Job/Resume: CRUD job/resume, kết hợp việc sử dụng Multer để upload file cho NestJS, đồng thời sử dụng "ref" để "join data cho MongoDB"

#72. Bài Tập Tạo Module Jobs

`nest g resource jobs --no-spec`

1. Tạo Schema

jobs

- name: string
- skills: string []
- company: object { _id, name }
- location: string
- salary: number
- quantity: number
- level: string
- description : html <string>
- startDate: date
- endDate: date
- isActive: boolean

2. update dto (validate dữ liệu)

//todo

#73. Bài tập CRUD Jobs

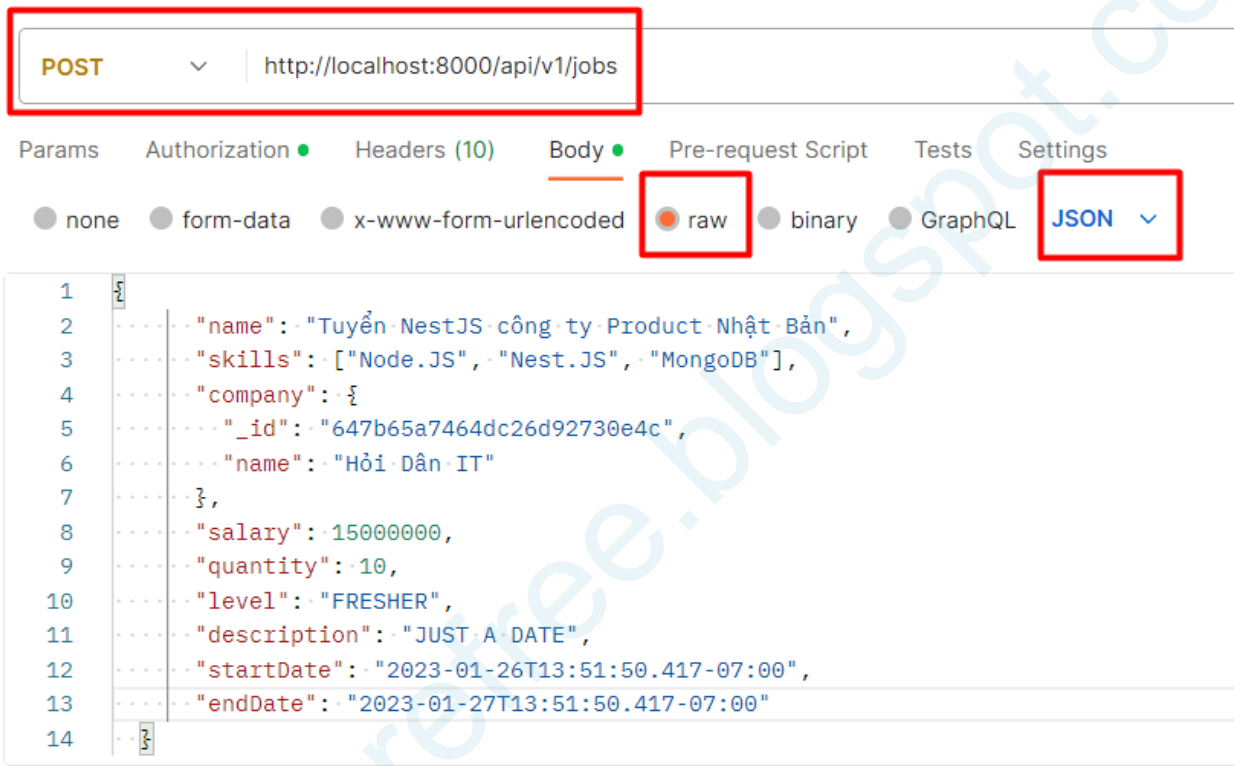
Hoàn thiện các endpoint sau:

1. Create a job

POST /api/v1/jobs

Yêu cầu:

- truyền JWT ở header
- body truyền dạng raw (JSON). Update thông tin createdBy khi tạo mới job



Response:

```
{
  "statusCode": 201,
  "message": "Create a new job",
  "data": {
    "_id": ".....",
    "createdAt": "....."
  }
}
```

Data mẫu khi tạo mới Job:

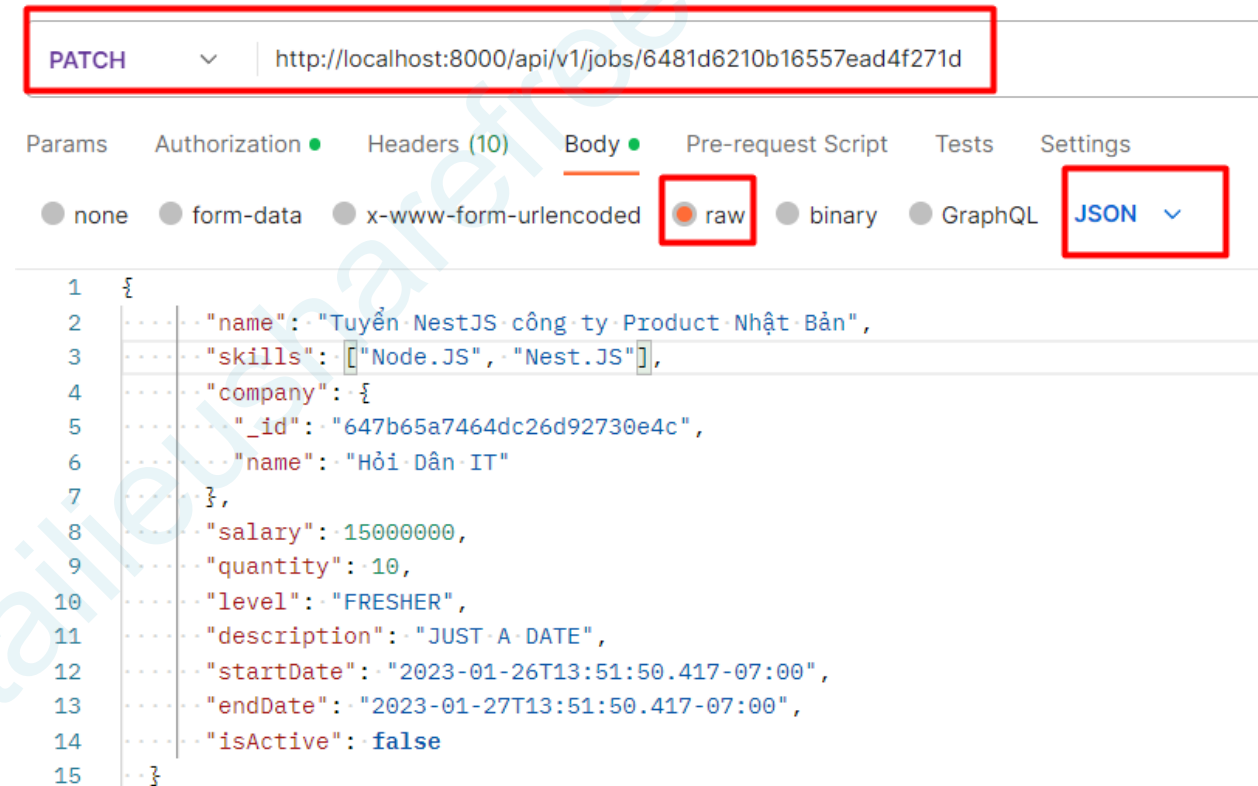
```
{
  "name": "Tuyển NestJS công ty Product Nhật Bản",
  "skills": ["Node.JS", "Nest.JS", "MongoDB"],
  "company": {
    "_id": "647b65a7464dc26d92730e4c",
    "name": "Hỏi Dân IT"
  },
  "salary": 15000000,
  "quantity": 10,
  "level": "FRESHER",
  "description": "JUST A DATE",
  "startDate": "2023-01-26T13:51:50.417-07:00",
  "endDate": "2023-01-27T13:51:50.417-07:00",
  "isActive": true
}
```

2. Update a job

PATCH /api/v1/jobs/id

Yêu cầu:

- truyền JWT ở header. Truyền động ID ở url
- body dạng raw (JSON). Update thông tin updatedBy khi tạo update job



Response:

```
{
  "statusCode": 200,
  "message": "Update a job",
  "data": {
    "acknowledged": .....,
    "modifiedCount": .....,
    "upsertedId": .....,
    "upsertedCount": .....,
    "matchedCount": .....
  }
}
```

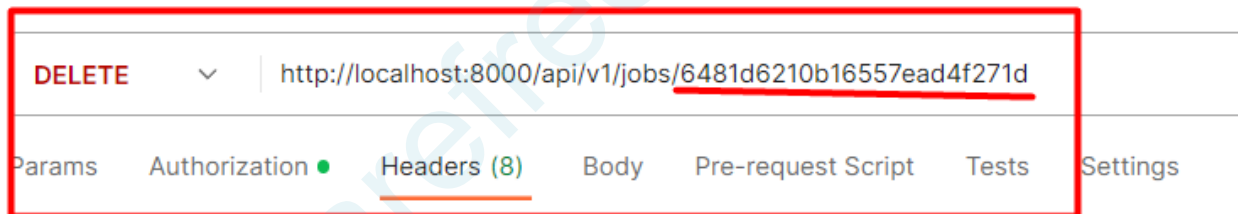
3. Delete a job

DELETE /api/v1/jobs/id

Yêu cầu:

- truyền JWT ở header. Truyền động ID ở url và sử dụng soft-delete

Update thông tin deletedBy khi tạo delete job



Response:

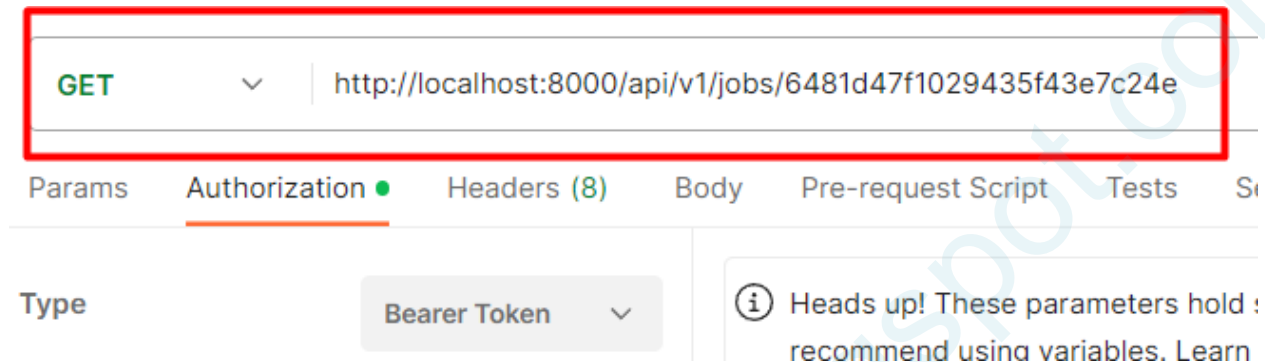
```
{
  "statusCode": 200,
  "message": "Delete a job",
  "data": {
    "deleted": .....
  }
}
```

4. Fetch job by id

GET /api/v1/jobs/id

Yêu cầu:

- truyền JWT ở header. Truyền động ID ở url



Response:

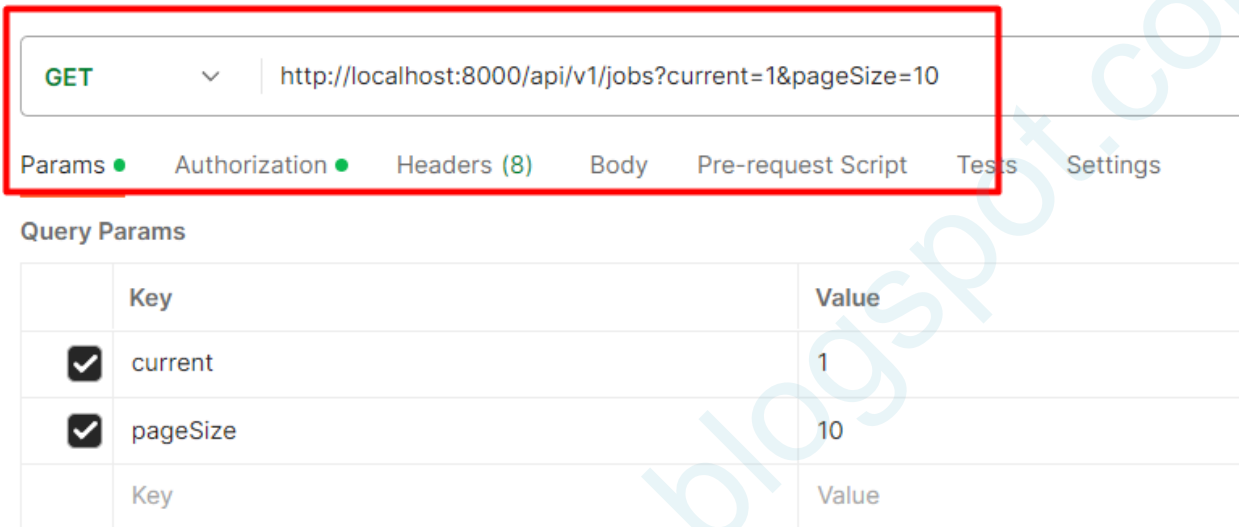
```
{
  "statusCode": 200,
  "message": "Fetch a job by id",
  "data": {
    "_id": ".....",
    "name": ".....",
    "skills": [ ..... ],
    "company": { "_id": "...." "name": "...."},
    "salary": .....,
    "quantity": .....,
    "level": ".....",
    "description": ".....",
    "startDate": "...",
    "endDate": "...",
    "isActive": ...,
    "createdBy": {
      "_id": ".....",
      "email": "....."
    },
    "isDeleted": ...,
    "deletedAt": ...,
    "createdAt": "....",
    "updatedAt": "...",
  }
}
```

5. Fetch job with paginate

GET /api/v1/jobs

Yêu cầu:

- truyền JWT ở header. Phân trang kết quả



	Key	Value
<input checked="" type="checkbox"/>	current	1
<input checked="" type="checkbox"/>	pageSize	10
	Key	Value

Response:

```
{
  "statusCode": 200,
  "message": "Fetch jobs with pagination",
  "data": {
    "meta": {
      "current": 1,
      "pageSize": 10,
      "pages": .....,
      "total": .....
    },
    "result": [..... ]
  }
}
```


#74. Hướng Dẫn Bài tập CRUD Jobs

//todo

tailieusharefree.blogspot.com

#75. Giới Thiệu Upload files

Tài liệu: <https://docs.nestjs.com/techniques/file-upload>

1. Nguyên tắc khi upload file

- Sử dụng POST (create new data)
- Sử dụng form-data (xử lý file), data truyền dưới dạng **multipart/form-data**
- Sử dụng thư viện multer (hiệu năng cao) :
<https://www.npmjs.com/package/multer>

Trong khóa restful api, để đơn giản hóa, mình sử dụng thư viện :

<https://www.npmjs.com/package/express-fileupload>

2.Setup

Tạo service upload file: (vì sẽ sử dụng service này ở nhiều nơi)

nest g resource files --no-spec

Cài đặt thư viện: npm i -D @types/multer

(không cần cài đặt thư viện multer, vì Nestjs đã hỗ trợ sẵn multer, chỉ cần cài đặt package types/multer để hỗ trợ cú pháp khi code typescript)

3.Xử lý file

Tài liệu: <https://docs.nestjs.com/techniques/file-upload#basic-example>

File upload thực chất là 1 middleware

Req => middleware => Res

```
@Post('upload')
@UseInterceptors(FileInterceptor('file')) //tên field sử dụng trong form-data
uploadFile(@UploadedFile( ) file: Express.Multer.File) {
  console.log(file);
}
```

FileInterceptor('file' , options) => có 2 tham số đầu vào

<https://github.com/expressjs/multer#multeropts>

#76. Validate Upload Files

Tài liệu: <https://docs.nestjs.com/techniques/file-upload#file-validation>

1. Validate File

Sử dụng cách truyền thống, tương như sử dụng với Express và Multer

<https://stackoverflow.com/questions/49096068/upload-file-using-nestjs-and-multer>

```
@UploadedFile(  
  new ParseFilePipeBuilder()  
    .addFileTypeValidator({  
      fileType: 'jpeg',  
    })  
    .addMaxSizeValidator({  
      maxSize: 1000  
    })  
    .build({  
      errorHttpStatusCode: HttpStatus.UNPROCESSABLE_ENTITY  
    })  
)  
file: Express.Multer.File
```

https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types

2. Giới thiệu về Upload Multiples

Trường hợp 1: upload array files (cùng key)

Tài liệu: <https://docs.nestjs.com/techniques/file-upload#array-of-files>

Trường hợp 2: upload multiple files (khác key)

<https://docs.nestjs.com/techniques/file-upload#multiple-files>

#77. Destination Upload File

Tài liệu: <https://docs.nestjs.com/techniques/file-upload>
<https://github.com/nestjs/nest/issues/2136>

1. Yêu cầu

- Cấu hình thư mục upload file (**destination**)
- **Thay đổi tên file** khi upload thành công (tránh trường hợp bị ghi đè/mất file do tên file upload bị trùng nhau)

2. Các bước thực hiện

- Config default options: **tại files modules**
<https://docs.nestjs.com/techniques/file-upload#default-options>
- Tạo class config, tham khảo:
`@Injectable()`
`export class MulterConfigService implements MulterOptionsFactory {`
 getRootPath = () => {
 return process.cwd();
 };

 ensureExists(targetDirectory: string) {
 fs.mkdir(targetDirectory, { recursive: true }, (error) => {
 if (!error) {
 console.log('Directory successfully created, or it already exists.'); return;
 }
 switch (error.code) {
 case 'EEXIST':
 // Error:
 // Requested location already exists, but it's not a directory.
 break;
 case 'ENOTDIR':
 // Error:
 // The parent hierarchy contains a file with the same name as the dir
 // you're trying to create.
 break;
 default:
 // Some other error like permission denied.
 console.error(error);
 break;
 }
 });
 }
}

```
createMulterOptions(): MulterModuleOptions {
  return {
    storage: diskStorage({
      destination: (req, file, cb) => {
        const folder = req?.headers?.folder_type ?? "default";
        this.ensureExists(`public/images/${folder}`);
        cb(null, join(this.getRootPath(), `public/images/${folder}`))
      },
      filename: (req, file, cb) => {
        //get image extension
        let extName = path.extname(file.originalname);

        //get image's name (without extension)
        let baseName = path.basename(file.originalname, extName);

        let finalName = `${baseName}-${Date.now()}${extName}`
        cb(null, finalName)
      }
    })
  };
}
```

#78. Bài Tập Update Company with Image

Yêu cầu:

- Tạo thêm field : **logo: string** đối với model Company
- Sửa 2 api là create và update company, cập nhật thêm field **logo: string**

Logic xử lý:

1. Khi **tạo mới/update** công ty, frontend upload ảnh

=> gọi api upload

=> backend sẽ upload (lưu trữ) file ảnh trên server, frontend nhận lại tên ảnh đã upload thành công

2. Frontend **submit tạo mới/update**, chỉ cần truyền **tên ảnh** đã upload thành công (**định dạng string**)

3. Backend **khi tạo mới/update công ty, chỉ lưu vào database tên ảnh** (field logo: string), còn **không cần xử lý logic upload file**, vì logic này đã được xử lý trước đó thông qua api upload file

#79. Test Giao Diện Frontend Upload File

//Part 1: fix bugs

```
@Post('upload')
@ResponseMessage("Upload Single File")
@UseInterceptors( FileInterceptor('fileUpload'))
```

=> bỏ decorator @Public

rename @UseInterceptors(FileInterceptor('fileUpload'))

- folder_type === "company"

- **@Public API fetch all company** => homepage

- **@Public API fetch all jobs** => homepage

- **@Public API fetch job by id** => view detail job

- **@Public API fetch company by id** => view detail company

- **khi tạo mới/update 1 jobs**, thông tin company sẽ bao gồm:

```
"company": {
  "_id": "647b65a7464dc26d92730e4c",
  "name": "Hỏi Dân IT"
  "logo: string
},
```

- **update thêm fields location** khi update/create jobs

- **Tạo fake data jobs/company (with logo)**

Link data fake:

<https://drive.google.com/drive/folders/1xqQETcMaksHvA9JSXXZNv0Bye76ZudO8?usp=sharing>

Part 2: run Frontend

Checkout sang branch **test-2**:

git checkout test-2

Cài đặt thư viện cần thiết:

npm i

#80. Bài tập Tạo Modules Resumes

nest g resource resumes --no-spec

Model resume:

- email: string
- userId: objectId
- url: string
- status: string// PENDING-REVIEWING-APPROVED-REJECTED
- companyId: objectId
- jobId: objectId
- history: [
 - { status: string, updatedAt: Date, updatedBy: { _id, email} }]
- createdAt: Date
- updatedAt: Date
- deletedAt: Date
- isDeleted: boolean
- createdBy: { _id, email },
- updatedBy: { _id, email },
- deletedBy: { _id, email }

//todo: update Create/update DTO

//Bổ sung api get company by ID => phục vụ frontend

#81. Bài tập CRUD Resumes

Cần tạo 6 endpoints sau:

1. Create a Resume

POST /api/v1/resumes

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8000/api/v1/resumes
- Body Type:** x-www-form-urlencoded
- Body Content:**

	Key	Value
<input checked="" type="checkbox"/>	url	abc
<input checked="" type="checkbox"/>	companyId	647b51974d59e754db118e95
<input checked="" type="checkbox"/>	jobId	647b51974d59e754db118e95
	Key	Value

Yêu cầu:

- Cần jwt ở header (người dùng đã đăng nhập).
- Body truyền lên:
url: string (url của cv)
companyId: string
jobId: string

Backend xử lý:

lưu các thông tin:

- email (lấy từ req.user/jwt)

- userId (lấy từ req.user)

- status: "PENDING"

- history: [

```
{
  status: "PENDING",
  updatedAt: new Date,
  updatedBy: {
    _id: user._id,
    email: user.email
  }
}
```

```
}
```

```
]
```

```
]
```

- createdBy: { _id, email} //lấy từ req.user

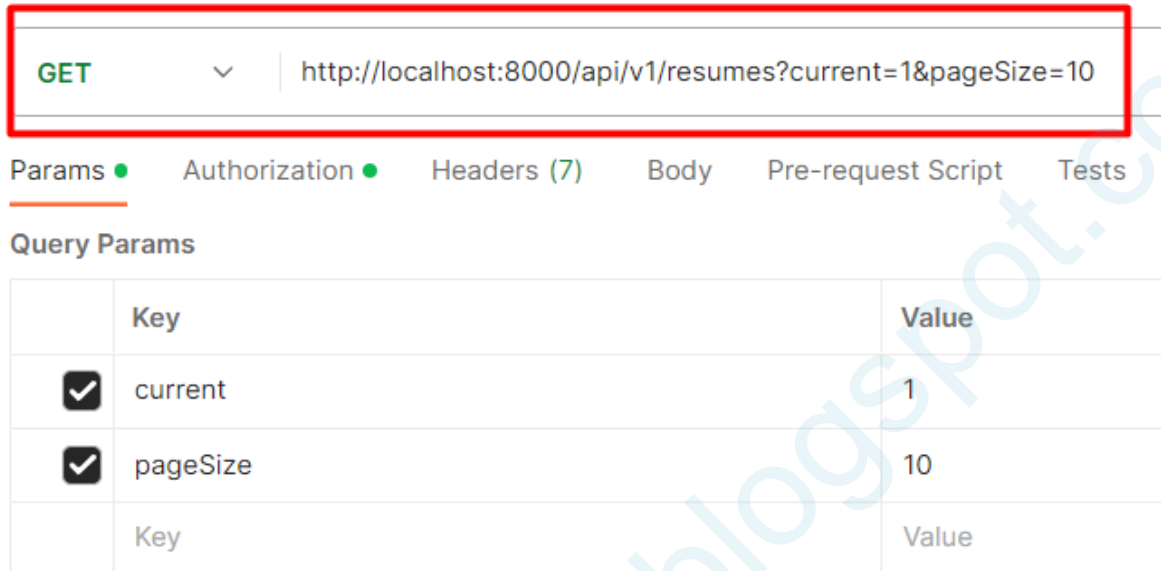
Response:

```
{
  "statusCode": 201,
  "message": "Create a new resume",
  "data": {
    "_id": ".....",
    "createdAt": "....."
  }
}
```

2. Fetch all resume (with paginate)

GET /api/v1/resumes

Yêu cầu: truyền lên JWT ở header, đồng thời query string ở url



GET

Params ● Authorization ● Headers (7) Body Pre-request Script Tests

Query Params

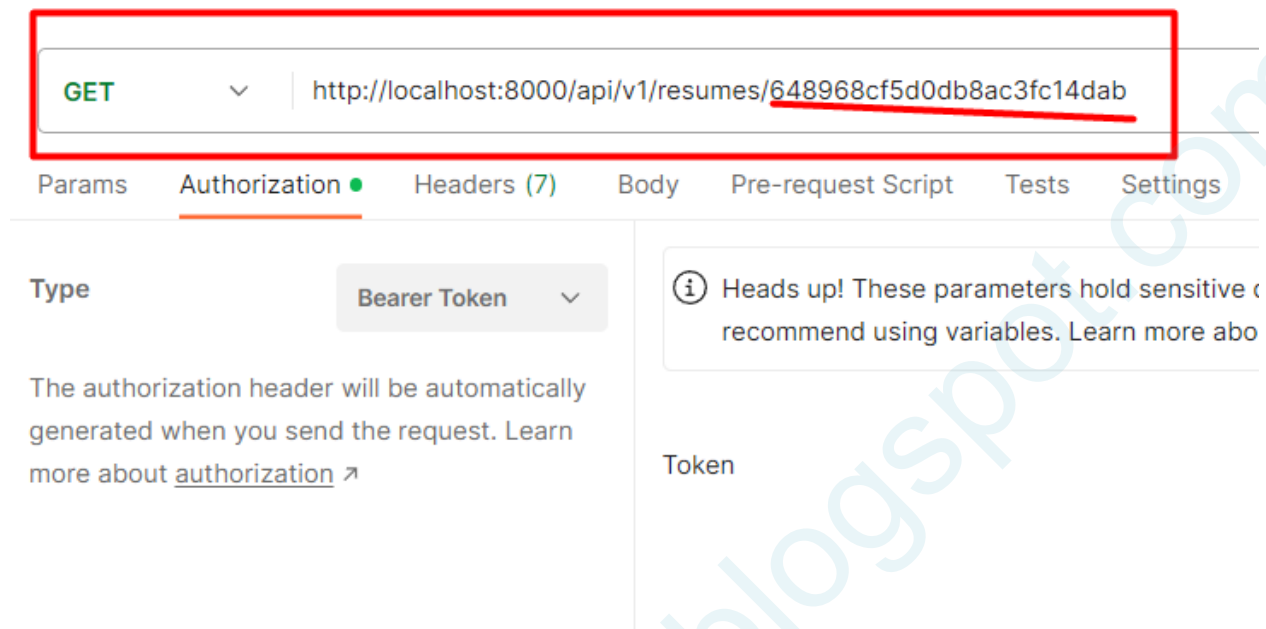
	Key	Value
<input checked="" type="checkbox"/>	current	1
<input checked="" type="checkbox"/>	pageSize	10
	Key	Value

Response:

```
{
  "statusCode": 200,
  "message": "Fetch all resumes with paginate",
  "data": {
    "meta": {
      "current": 1,
      "pageSize": 10,
      "pages": ...,
      "total": ....
    },
    "result": [
      {...}
    ]
  }
}
```

3. Fetch resume by id

GET /api/v1/resumes/id



Yêu cầu:

- Truyền động id ở url
- Truyền JWT ở header

Response:

```
{
  "statusCode": 200,
  "message": "Fetch a resume by id",
  "data": {
    //data resume here
  }
}
```

4. Change status a Resume (update a resume)

PATH /api/v1/resumes/id

The screenshot shows a REST client interface with a red box highlighting the method and URL. The method is 'PATCH' and the URL is 'http://localhost:8000/api/v1/resumes/648968cf5d0db8ac3fc14dab'. Below the URL bar, there are tabs for 'Params', 'Authorization', 'Headers (9)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Body' tab is selected, and the content type is set to 'x-www-form-urlencoded'. A table below shows the body data with a checked checkbox for 'status' and a value of 'bbbbbb'.

	Key	Value
<input checked="" type="checkbox"/>	status	bbbbbb
	Key	Value

Yêu cầu:

- Header truyền JWT
- body truyền thêm **status**

Backend xử lý:

Update các thuộc tính: **status, updatedBy, history**

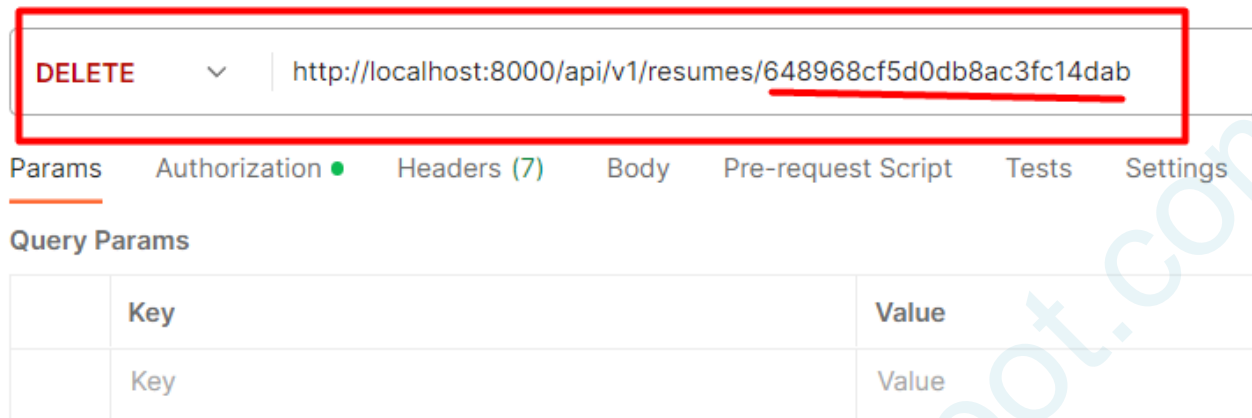
history là array => push thêm object mới vào array đã có

Response:

```
{
  "statusCode": 200,
  "message": "Update status resume",
  "data": {
    "Acknowledged": ...,
    "modifiedCount": ...,
    "upsertedId": ...,
    "upsertedCount": ...,
    "matchedCount": ...
  }
}
```

5. Delete a resume

DELETE /api/v1/resumes/id



DELETE

Params Authorization ☒ Headers (7) Body Pre-request Script Tests Settings

Query Params

	Key	Value
	Key	Value

Yêu cầu:

- Truyền JWT ở header
- Truyền động id trên url
- Sử dụng soft-delete

Response:

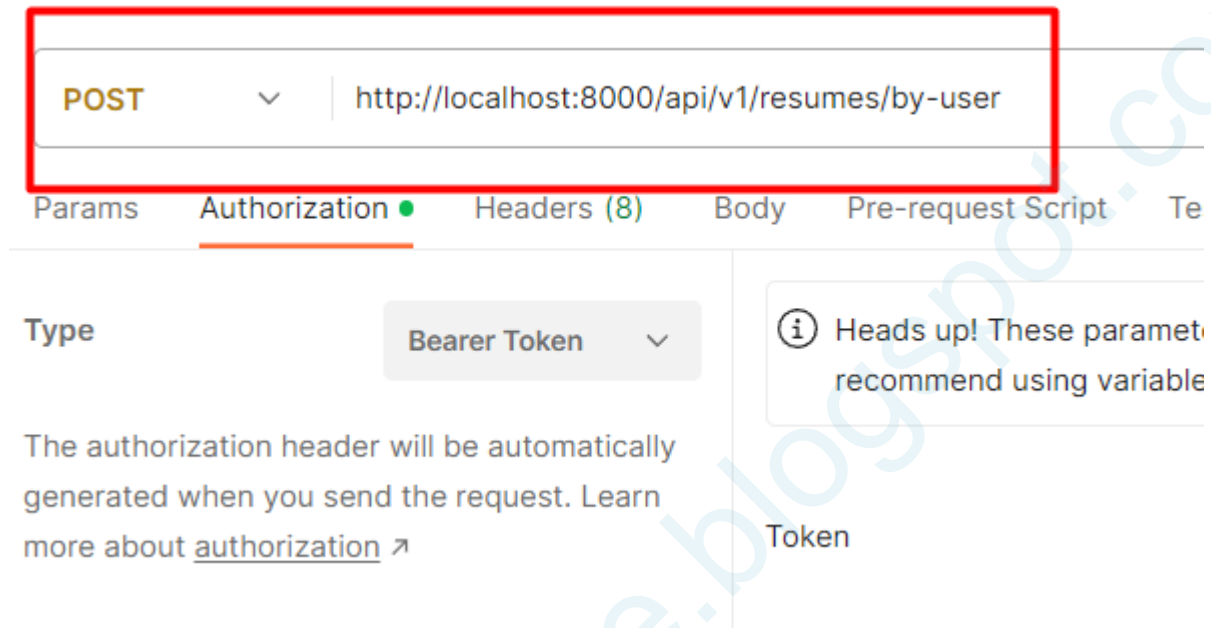
```
{
  "statusCode": 200,
  "message": "Delete a resume by id",
  "data": {
    "deleted": ....
  }
}
```

6. get CV by user

=> **frontend**, sau khi apply a job, có thể xem lại lịch sử

POST /api/v1/resumes/by-user

không dùng get => trùng với id



Yêu cầu:

- Truyền lên JWT ở header

Response:

```
{
  "statusCode": 201,
  "message": "Get Resumes by User",
  "data": [
    //data resumes
  ]
}
```


#82. Hướng Dẫn Bài tập CRUD Resumes

//todo

tailieusharefree.blogspot.com

#83.1 Fetch Data with Ref

- fetch resume by Id => with Ref

<https://www.npmjs.com/package/api-query-params>

<https://docs.nestjs.com/techniques/mongodb>

- fetch resume with paginate

http://localhost:8000/api/v1/resumes?current=1&pageSize=10&populate=companyId,jobId&fields=companyId._id, companyId.name, companyId.logo, jobId._id, jobId.name

#83.2 Test Giao Diện Frontend

Để test giao diện, checkout sang branch **test-3**

git checkout test-3

npm i

Chapter 14: Modules Permission & Roles

Bài tập thực hành tạo module Permission/roles : CRUD Permissions/role, kết hợp áp dụng NestJS Life Cycle để tạo fake data cho dự án Backend.

#84. Bài tập Tạo Modules Permission & Roles

Sử dụng command:

nest g resource permissions --no-spec

nest g resource roles --no-spec

Mô hình phân quyền:

1 user => có 1 role . 1 role => có n permissions

=> 1 user có n permissions khi sử dụng hệ thống

Sau này muốn thay đổi permission của user, chỉ cần thay đổi role là xong
(update role hiện tại, hoặc tạo role mới => gán user vào)

1.Model Permission (Quyền hạn sử dụng hệ thống)

Mỗi permission chính là 1 api ở backend

- name : string
- apiPath : string
- method: string
- module: string //thuộc modules nào ?
- createdAt
- updatedAt
- deletedAt
- isDeleted
- createdBy : {_id, email}
- updatedBy: {_id, email}
- deletedBy: {_id, email}

2. Model Role (Vai trò trong hệ thống)

- name: string
- description: string
- isActive: boolean
- **permissions**: [] // array of object id. Ví dụ: ["648968cf5d0db8ac3fc14dab", "648968cf5d0db8ac3fc14dab"...]
- createdAt
- updatedAt
- deletedAt
- isDeleted
- createdBy : { _id, email }
- updatedBy: { _id, email }
- deletedBy: { _id, email }

=> lưu ý cách khai báo schema

3. Create/Update DTO

//todo

#85. Bài tập CRUD Permissions

Tạo các endpoint sau:

1. Create a Permission

POST api/v1/permissions

The screenshot shows a REST client interface with a POST request to `http://localhost:8000/api/v1/permissions`. The request body is set to `x-www-form-urlencoded` and contains the following data:

Key	Value
<input checked="" type="checkbox"/> name	fetch user
<input checked="" type="checkbox"/> apiPath	/users
<input checked="" type="checkbox"/> method	POST
<input checked="" type="checkbox"/> module	BLA BLA

Yêu cầu:

- Truyền JWT ở header
- Body truyền lên các tham số: **name, apiPath, method, module**

Xử lý ở backend:

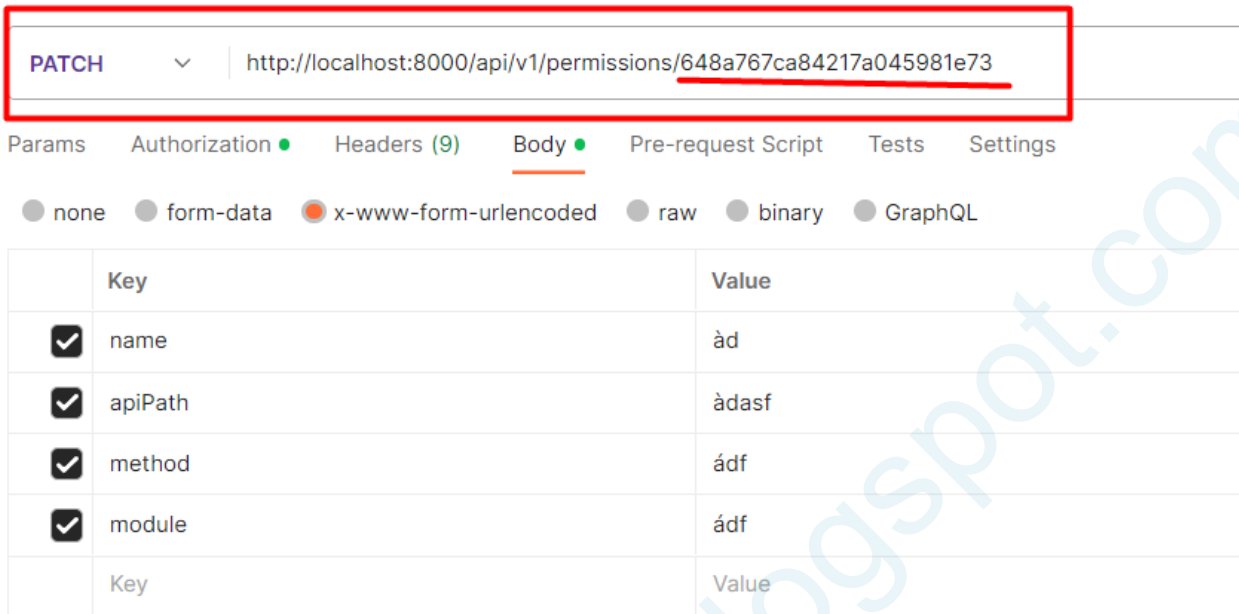
- Trước khi lưu, cần check xem "apiPath" + "method" đã tồn tại chưa? Nếu đã tồn tại thì thông báo lỗi. Chỉ tạo mới permission có "apiPath" + "method" chưa tồn tại
- Lưu thêm thông tin **createdBy**

Response:

```
{
  "statusCode": 201,
  "message": "Create a new permission",
  "data": {
    "_id": ".....",
    "createdAt": "...."
  }
}
```

2. Update a Permission

PATCH /api/v1/permissions/:id



The screenshot shows a REST client interface with a red box highlighting the method and URL. The URL is `http://localhost:8000/api/v1/permissions/648a767ca84217a045981e73`. The method is `PATCH`. The request body is `x-www-form-urlencoded` with the following fields:

Key	Value
<input checked="" type="checkbox"/> name	àd
<input checked="" type="checkbox"/> apiPath	àdasf
<input checked="" type="checkbox"/> method	ádf
<input checked="" type="checkbox"/> module	ádf

Yêu cầu:

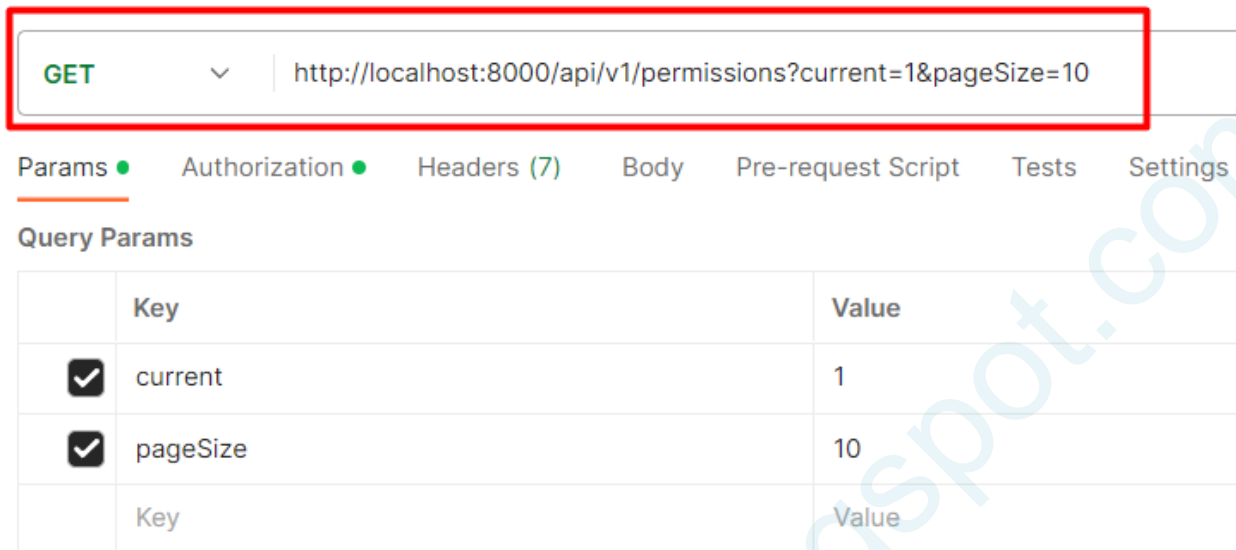
- Truyền lên JWT ở header
- Truyền động ID trên url

Response:

```
{
  "statusCode": 200,
  "message": "Update a permission",
  "data": {
    "acknowledged": .....,
    "modifiedCount": ...,
    "upsertedId": ...,
    "upsertedCount": .....,
    "matchedCount": ....
  }
}
```

3. Fetch Permission with paginate

GET /api/v1/permissions



GET

Params ● Authorization ● Headers (7) Body Pre-request Script Tests Settings

Query Params

	Key	Value
<input checked="" type="checkbox"/>	current	1
<input checked="" type="checkbox"/>	pageSize	10
	Key	Value

Yêu cầu:

- Truyền JWT ở header
- Truyền động params để phân trang

Response:

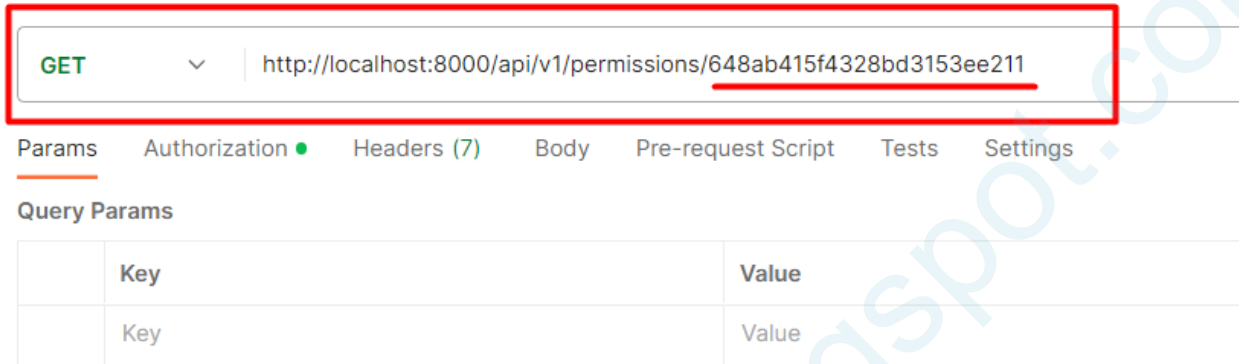
```
{
  "statusCode": 200,
  "message": "Fetch permissions with paginate",
  "data": {
    "meta": {
      "current": 1,
      "pageSize": 10,
      "pages": .....,
      "total": .....
    },
    "result": [...]
  }
}
```


4. Fetch Permission by ID

GET /api/v1/permissions/:id

Yêu cầu:

- Truyền JWT ở header
- Truyền động id ở header



GET http://localhost:8000/api/v1/permissions/648ab415f4328bd3153ee211

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

Key	Value
Key	Value

Response:

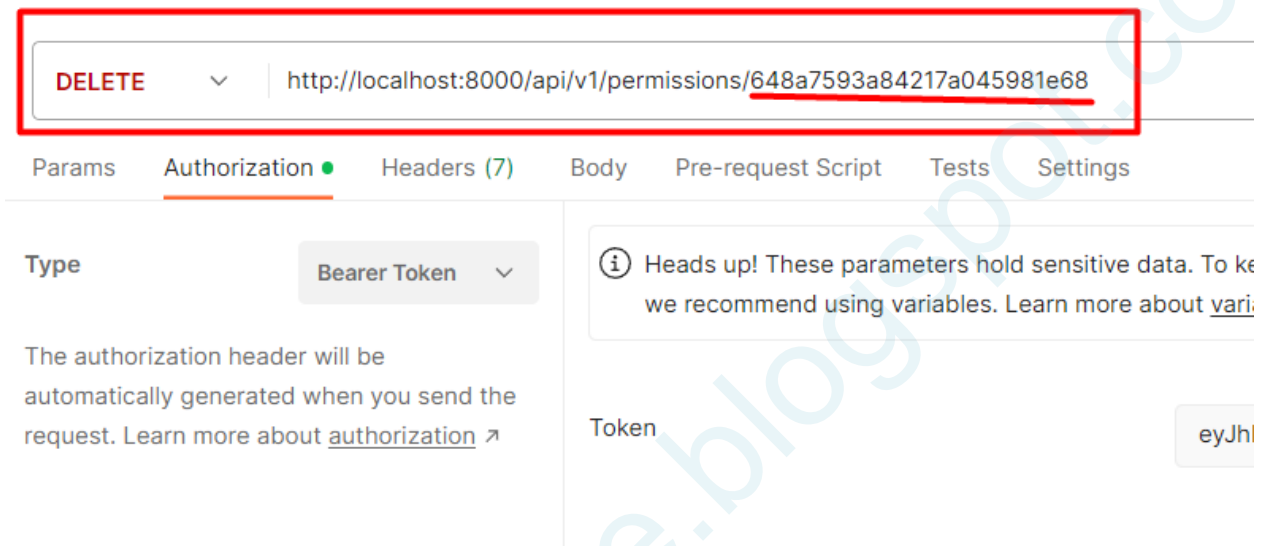
```
{
  "statusCode": 200,
  "message": "Fetch a permission by id",
  "data": {
    ...
  }
}
```

5. Delete a permission

DELETE /api/v1/permissions/:id

Yêu cầu:

- Truyền JWT ở header
- Truyền động id trên url
- **Sử dụng Soft delete**



Response:

```
{
  "statusCode": 200,
  "message": "Delete a permission",
  "data": {
    "deleted": ...
  }
}
```

#86. Hướng Dẫn Bài tập CRUD Permissions

//todo

#87. Bài tập CRUD Roles

1. Create Role

POST /api/v1/roles

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8000/api/v1/roles
- Body Type:** raw (selected)
- Body Content:**

```
1 {
2   "name": "Group Admin",
3   "description": "Admin thi full quyền v",
4   "isActive": true,
5   "permissions": ["648a79fd8bfe16764b81207d", "648a79f98bfe16764b81207a", "648a79f58bfe16764b812077"]
6 }
```

Yêu cầu:

- Truyền JWT ở header
- Truyền body dưới dạng raw/JSON, **bao gồm các trường name, description, isActive, permissions**

Backend xử lý: không lưu trùng tên của group. **Check "name"** trước khi lưu

Response:

```
{
  "statusCode": 201,
  "message": "Create a new role",
  "data": {
    "_id": ".....",
    "createdAt": "....."
  }
}
```

2. Update a Role

PATCH /api/v1/roles/:id

The screenshot shows a REST client interface with the following details:

- Method:** PATCH
- URL:** http://localhost:8000/api/v1/roles/648a7cfee52334e2323a7760
- Body Type:** raw (selected)
- Content Type:** JSON (selected)
- Body Content:**

```
1 {
2   "name": "Group Admin 1",
3   "description": "Admin thi full quyền v",
4   "isActive": true,
5   "permissions": ["648a79fd8bfe16764b81207d", "648a79f98bfe16764b81207a", "648a79f58bfe16764b812077"]
6 }
```

Yêu cầu:

- Truyền JWT ở header
- Truyền data ở body, dưới dạng raw/json

Backend cần check "name" trước khi cho update, đảm bảo "name" là khác nhau

Response:

```
{
  "statusCode": 200,
  "message": "Update a role",
  "data": {
    "acknowledged": .....,
    "modifiedCount": .....,
    "upsertedId": .....,
    "upsertedCount": ...
    "matchedCount": ...
  }
}
```

3. Fetch Role with paginate

GET /api/v1/role

Yêu cầu:

- Truyền jwt ở header
- Truyền động params trên url

GET ▼ http://localhost:8000/api/v1/roles?current=1&pageSize=10 ...

Params ● Authorization ● Headers (7) Body Pre-request Script Tests Settings

Query Params

	Key	Value
<input checked="" type="checkbox"/>	current	1
<input checked="" type="checkbox"/>	pageSize	10

4. Fetch Role by ID

GET /api/v1/role/:id

Yêu cầu:

- Truyền jwt ở header
- Truyền động id trên url

GET ▼ http://localhost:8000/api/v1/roles/648a7cf5e52334e2323a7760

Params Authorization ● Headers (7) Body Pre-request Script Tests Settings

Query Params

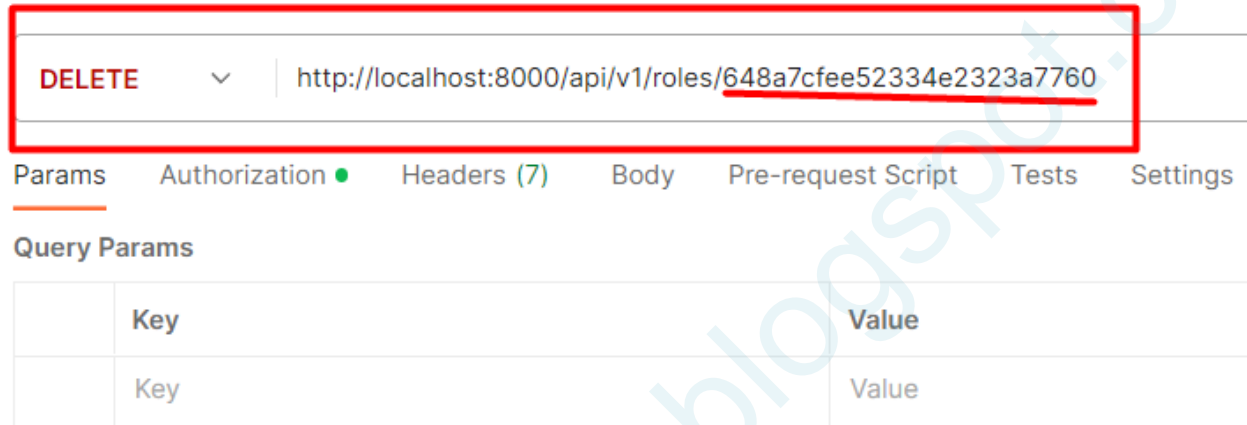
	Key	Value
	Key	Value

5. Delete a Role

DELETE /api/v1/role/:id

Yêu cầu:

- Truyền jwt ở header
- Truyền động id trên url
- **Sử dụng soft delete**



DELETE ⌵ | http://localhost:8000/api/v1/roles/648a7cfee52334e2323a7760

Params Authorization ● Headers (7) Body Pre-request Script Tests Settings

Query Params

	Key	Value
	Key	Value

#88. Hướng Dẫn Bài tập CRUD Roles

//todo

#89. Fix bugs

1. lưu ý khi viết fetch Role by Id

=> trả thêm field module

```
async findOne(id: string) {
  if (!mongoose.Types.ObjectId.isValid(id)) {
    throw new BadRequestException("not found role")
  }

  return (await this.roleModel.findById(id)).populate({
    path: "permissions",
    select: { _id: 1, apiPath: 1, name: 1, method: 1, module: 1 }
  });
}
```

2. khi update role => ko check name

3. api fetch resume - by - user => populate companyId, jobId

```
async findByUsers(user: IUser) {
  return await this.resumeModel.find({
    userId: user._id,
  })
  .sort("-createdAt")
  .populate([
    {
      path: "companyId",
      select: { name: 1 }
    },
    {
      path: "jobId",
      select: { name: 1 }
    }
  ])
}
```

4. fix api delete user/role

= không delete admin

= không delete role admin

5. Fix user role

=> chuyển qua dùng ref

6. api fetch user by id/ fetch with paginate

=> role : {_id, name}

7. api login

- trả thêm permission của người dùng

#90. Nest Lifecycle Event

Tài liệu: <https://docs.nestjs.com/fundamentals/lifecycle-events>

1. Lifecycle là gì ?

Vòng đời (lifecycle) là quá trình ứng dụng Nest.js chạy.

Ví dụ với con người, lifecycle gồm có: sinh ra => lớn lên => die

Tương tự, với **Nest.js**, gồm **3 phases** chính:

Phase 1: **initializing** (khởi tạo). ví dụ như khởi tạo connections, DI (dependencies injection), modules...

Phase 2: **running** (chính là lúc ứng dụng đã chạy thành công) => enjoy

Phase 3: **terminating**: trong quá trình chạy, nếu có "bugs" xảy ra, khiến server bị lỗi => die

Để cho nó không die nữa thì easy => restart lại server. quá trình trên lại lặp lại từ phase 1 tới phase 3

2. Tại sao cần lifecycle

- Sử dụng lifecycle để can thiệp vào quá trình chạy ứng dụng, ví dụ:
 - + khi ứng dụng bị die => do something :v
 - + khi ứng dụng chưa chạy lên => check logic, init fake data ...

3. Sử dụng với Nest.JS

Mục đích: tạo init data (dữ liệu thật, không phải dữ liệu fake) cho database

Trong thực tế, có 3 cách tạo data cho database:

- Cách 1: sử dụng script của database, tức là mở database => tự thêm dữ liệu vào
Với công ty, nếu database do team khác quản lý => dev không tự làm được điều này :v
- Cách 2: cài đặt/sử dụng thư viện hỗ trợ **migration/seeder** (rất tiếc là có rất ít thư viện làm điều này với Mongoose/mongodb)
- Cách 3: tạo data init từ code :v => dev có thể làm :v

Ý tưởng:

- Mỗi lần (everytime) ứng dụng Nest.js khởi tạo, sẽ gọi function để check:
- + Để biết có dữ liệu hay chưa => dùng hàm find, hoặc hàm count
- + Nếu có dữ liệu init rồi => không làm gì cả
- + Nếu chưa có dữ liệu => tạo init data

Cách làm:

Tài liệu: <https://docs.nestjs.com/fundamentals/lifecycle-events#usage>

nest g resource databases --no-spec

=> không cần generate crud

Viết hàm onModuleInit bên trong service

#91. Bài tập Tạo Sample Data

Xóa 3 collections users, roles và permissions trong database

- Mặc định 2 role: admin và user
- mặc định 2 users: admin và user
- mặc định permissions

Thứ tự tạo:

- Tạo permissions
 - Tạo role (với permissions) được tạo ở trên
 - Tạo user với role được tạo ở trên
-
- Cần setup tham số trong .env

#92. Hướng Dẫn Bài Tập Tạo Fake Data

Sử dụng logger:

<https://docs.nestjs.com/techniques/logger#using-the-logger-for-application-logging>

Download source code module databases:

https://drive.google.com/file/d/1B9zqP-qHt1dqcc0ZvBrAOfG_bY7m-oZO/view?usp=sharing

#93. Test Giao Diện Frontend

PART 1: FIX BUGS

<https://stackoverflow.com/questions/14504385/why-cant-you-modify-the-data-returned-by-a-mongoose-query-ex-findbyid>

Mục tiêu cần đạt được sau khi kết thúc video này:

Các apis sau:

- api/v1/auth/login
- api/v1/auth/account
- api/v1/auth/refresh

=> Cần trả thêm thêm thông tin về role và permissions

Khi register => tạo fake role user

Api login/refresh_token/fetch account, trả thêm permission []

Các phần kiểm tra (dựa vào phần fix tại #89):

1. lưu ý khi viết fetch Role by Id

=> trả thêm field module

```
async findOne(id: string) {
  if (!mongoose.Types.ObjectId.isValid(id)) {
    throw new BadRequestException("not found role")
  }

  return (await this.roleModel.findById(id)).populate({
    path: "permissions",
    select: { _id: 1, apiPath: 1, name: 1, method: 1, module: 1 }
  });
}
```

2. khi update role => ko check name

3. api fetch resume - by - user => populate companyId, jobId

```
async findByUsers(user: IUser) {
  return await this.resumeModel.find({
    userId: user._id,
  })
}
```

```
.sort("-createdAt")
  .populate([
    {
      path: "companyId",
      select: { name: 1 }
    },
    {
      path: "jobId",
      select: { name: 1 }
    }
  ])
}
```

4. fix api delete user/role

= không delete admin

= không delete role admin

5. Fix user role

=> chuyển qua dùng ref

6. api fetch user by id/ fetch with paginate

=> role : { _id, name }

7. api login

- trả thêm permission của người dùng

PART 2: TEST GIAO DIỆN

Checkout code sang nhánh mới **test-4**
git checkout test-4

Cài đặt thư viện cần thiết: **npm i**

#94. Roles Guards

1. Lũồng JWT hiện tại

Khi User gửi 1 request sử dụng jwt ở header, tới 'protected endpoint' (tức là những api không sử dụng @public):

Bước 1: JWT auth guard sẽ được chạy (config trong file main.ts)

app.useGlobalGuards(new JwtAuthGuard(reflector));

Bước 2: Passport JWT strategy chạy (file jwt.strategy.ts)

Nếu token hết hạn/không hợp lệ => thông báo lỗi.

Trường hợp token hợp lệ, decode (giải mã) token => gán vào req.user

2. Các bước cần sửa

- Sau khi giải mã token jwt, cần gán thêm permissions vào req.user (tương tự như api login và api refresh token)

- Khi JWT auth guard được chạy, cần check permission tại đây.

Cần so sánh **req.route** và **permission**

Tài liệu: <https://expressjs.com/en/5x/api.html#req.route>

Chapter 15: Module Subscribers

Bài tập thực hành, tạo module Subscribers: CRUD subscribers, đồng thời tìm hiểu cách gửi email với Nest.js sử dụng "template xây dựng sẵn", kết hợp với "cron jobs" để gửi email tự động.

#95. Bài tập CRUD Subscribers

Yêu cầu:

1. Tạo module Subscribers:

nest g resource subscribers --no-spec

2. Fix lỗi permission ?

Sử dụng source code frontend tại branch test-4

- Fix lỗi permissions
- Tạo permissions cho module subscribers

3. Tạo DTO

//todo

4. Tạo CRUD endpoint

//todo

#96. Giới thiệu về gửi email với Node.js

Để gửi email, cần 2 thứ:

- Một là server để gửi email.

ví dụ: bạn ngồi tại cty A, sử dụng email của cty A, gửi email tới địa chỉ hoidanit@gmail.com

Email sẽ được vận chuyển từ Server Email của cty A tới Google (gmail).

Hiểu đơn giản, **server để gửi Email chính là "bưu điện"** (post office)

- Hai là nội dung email, chính là "lá thư" gửi tới người nhận (letter)

Ở đây, bưu điện (server) sẽ là nơi gửi lá thư (email) tới người nhận thông qua đường truyền internet.

Người nhận muốn đọc email => login vào server (chứa email) => thấy email được gửi tới

Cách Gửi email với Nodejs

Sử dụng thư viện Node Mailer : <https://nodemailer.com/about/>

Quá trình gửi email gồm 3 bước:

1. cài đặt thư viện: npm install nodemailer

2. config transporter

```
let transporter = nodemailer.createTransport(transport[, defaults])
```

ở đây, chúng ta cần setup server để gửi email (email này sẽ được xuất phát từ đâu)

3. sending email

```
transporter.sendMail(data[, callback])
```

data gửi đi sẽ bao gồm:

- nội dung email : text, images...
- **người nhận** => đây chính là đích đến của email

#97. Template Email

1. Tại sao cần template email ?

- Nếu bạn chỉ gửi email thông thường, không quan tâm tới format (cấu trúc, định dạng...) thì không cần "template"

- Template chính là cấu trúc định dạng cho 1 email, giúp email được gửi đi theo định dạng nhất định

- **Việc sử dụng "template" giúp email gửi đi được chuyên nghiệp hơn**

Ví dụ, nếu bạn cần gửi đi 1 loạt email, chỉ "khác nhau" tên người nhận
=> nên tạo ra template để gửi

Hiểu đơn giản, template, giống hệt như việc bạn đi làm giấy tờ "tại cơ quan hành chính"

Cơ quan đưa bạn "form mẫu" (template). bạn điền thông tin
=> end game (email đc gửi đi)

2. Viết template bằng công cụ gì ?

Template là "form mẫu", và việc của bạn là "điền thêm data" vào chỗ trống.

Cách 1: sử dụng function của javascript.

- Cách làm này cũng tạo ra được template, tuy nhiên, không thể tạo ra được email "đẹp".
ví dụ như:

+ format table

+ highlight text...

Cách 2 (được sử dụng phổ biến): sử dụng template với view engine, tức là viết dưới định dạng HTML

3. Ví dụ về template engine html

Thực chất nội dung body là 1 trang html

Việc sử dụng view engine giúp bạn có thể viết được "logic code" bên trong html.

ví dụ với handlebars:

<https://www.npmjs.com/package/handlebars>

```
var data = {  
  "name": "Alan", "hometown": "Somewhere, TX",  
  "kids": [  
    {"name": "Jimmy", "age": "12"},  
    {"name": "Sally", "age": "4"}  
  ]  
};
```

```
var source = "<p>Hello, my name is {{name}}. I am from {{hometown}}. I have " +  
  "{{kids.length}} kids:</p>" +  
  "<ul>{{#kids}}<li>{{name}} is {{age}}</li>{{/kids}}</ul>";
```

```
var template = Handlebars.compile(source);
```

```
var result = template(data);
```

```
// Would render:
```

```
// <p>Hello, my name is Alan. I am from Somewhere, TX. I have 2 kids:</p>
```

```
// <ul>
```

```
// <li>Jimmy is 12</li>
```

```
// <li>Sally is 4</li>
```

```
// </ul>
```

#98. Setup Tài khoản/Server gửi Email

- Trong thực tế, **đa phần bạn sẽ sử dụng server email do bạn xây dựng**
- Để cho đơn giản và tiết kiệm chi phí, chúng ta sử dụng Gmail để gửi email
- **Sở đồ gửi email với NestJS và Gmail**

1. Server backend (NestJS) => **cung cấp user/pass + nội dung email** cho Gmail
2. Gmail tiếp nhận thông tin, đọc thông tin email để biết người gửi/người nhận
3. Gmail gửi email tới đích đến

Như vậy, chúng ta cần cung cấp user/password của tài khoản gmail cho Google Email Server

1. Về app password

Tài liệu: <https://support.google.com/accounts/answer/185833>

App password là 1 chuỗi string gồm 16 ký tự, dùng để login (sử dụng) tài khoản Google của bạn.

App password khác với password bạn hay dùng để đăng nhập tài khoản google.

Why ?

Ngoài password, chúng ta sử dụng app password để có thể truy cập Google Account.

Cách làm này dùng để giao tiếp giữa server => server

(vì server không có giao diện để login). và ngoài ra, không cần để lộ password.

vì khi mất password => game over

Ví dụ: server facebook gọi api (cung cấp email/app password) => gọi tới server google để truy cập tài khoản.

Như vậy, app password là cách server/app có quyền để truy cập vào tài khoản google của bạn (mà không cần mật khẩu)

2. Yêu cầu của app password

- Tương tự như password, nên lưu trữ vào file .env và không nên public để hạn chế rủi ro
- Tài khoản của bạn phải bật **2-step Verification**

Tài liệu: <https://support.google.com/accounts/answer/185839>

3. Tạo app password

Tài liệu: <https://support.google.com/accounts/answer/185833>

Yêu cầu : cần tạo được app password. nếu không tạo được thì không thể làm các video tiếp theo -.-

#99. Nestjs - Mailer

Tài liệu: <https://nest-modules.github.io/mailer/docs/mailer.html>

1. Cài đặt thư viện

```
npm install --save-exact handlebars@4.7.7 @nestjs-modules/mailer@1.8.1  
nodemailer@6.9.3
```

```
npm install --save-dev @types/nodemailer
```

2. Tạo Module

```
nest g resource mail --no-spec
```

=> không cần generate CRUD

- Lấy config từ file .env

```
//mail.controller.ts
```

```
@Get()  
@Public()  
@ResponseMessage("Test email")  
async handleTestEmail() {  
  await this.mailerService.sendMail({  
    to: "haryphamdev@gmail.com",  
    from: "Support Team" <support@example.com>', // override default from  
    subject: 'Welcome to Nice App! Confirm your Email',  
    html: '<b>welcome bla bla</b>', // HTML body content  
  });  
}
```

//mail.module.ts

```
@Module({
  imports: [
    MailerModule.forRootAsync({
      useFactory: async (configService: ConfigService) => ({
        transport: {
          host: 'smtp.gmail.com',
          secure: false,
          auth: {
            user: 'user@example.com',
            pass: 'topsecret',
          },
        },
      }),
      // template: {
      //   dir: join(__dirname, 'templates'),
      //   adapter: new HandlebarsAdapter(),
      //   options: {
      //     strict: true,
      //   },
      // },
      inject: [ConfigService],
    }),
  ],
  controllers: [MailController],
  providers: [MailService]
})
export class MailModule { }
```

=> chưa dùng template

#100. Gửi email với template

```
import { HandlebarsAdapter } from '@nestjs-modules/mailer/dist/adapters/handlebars.adapter';
```

1. Cấu hình CLI dịch file template

//nest-cli.json

```
{
  "collection": "@nestjs/schematics",
  "sourceRoot": "src",
  "compilerOptions": {
    "assets": ["mail/templates/**/*"], // or "**/*.hbs" all files ending with .hbs
    "watchAssets": true // copy assets in watch mode
  }
}
```

2. Sử dụng template

Download file template trong video:

<https://drive.google.com/file/d/1hVR3w3VTnz3Map66QYC1qSH41RgUax30/view?usp=sharing>

Preview Template before send email

//set preview = true

=> lưu ý: config env để on/off tính năng preview

Cách tạo ra template theo ý muốn:

- Code HTML và style CSS trong cùng 1 file (không code riêng lẻ HTML và css) => sử dụng css với tag <style> </style>
- Convert HTML/CSS => inline CSS
- Mặc định Gmail sẽ bỏ qua CSS ở phần header => để đảm bảo an toàn hơn cho người dùng (tránh nhúng link javascript ở header)
- Nên sử dụng layout table (không dùng css flex)

3. Test gửi email với template

- Fill mặc định thông tin vào template
- Fill động thông tin vào template
- Sử dụng vòng lặp for để render list job

Tổng kết:

Tính năng tạo template và gửi mail, có thể áp dụng cho :

- + gửi mã code khi register/login
- + gửi link confirm khi quên mật khẩu

#101. Fetch Subscribers by skills

1. Truyền động tham số vào template

//todo

2. Fetch Subscribers by skills

```
const subscribers = await this.subscriberModel.find({ });
for (const subs of subscribers) {
  const subsSkills = subs.skills;
  const jobWithMatchingSkills = await this.jobModel.find({ skills: { $in: subsSkills } });
  //todo
  //build template
}
```

Hàm format VND:

```
salary: `${item.salary}`.replace(/\B(?=(\d{3})+(?!\d))/g, ',') + " đ",
```

#102. Test giao diện frontend

git checkout master

tạo thêm api:

+ fetch subscriber by email => lấy thông tin subscribe skills của user đăng nhập
//add decorator skip permission

fix bugs update ko truyền full data => bị mất dữ liệu
//bugs update dto

```
app.useGlobalPipes(new ValidationPipe({  
  whitelist: true,  
  // forbidNonWhitelisted: true  
}));
```

<https://docs.nestjs.com/techniques/validation#using-the-built-in-validationpipe>

+ upsert subscriber => update function update => set upsert = true

<https://stackoverflow.com/questions/9661081/mongoose-update-upsert>

#103. Cron job

Tài liệu:

<https://en.wikipedia.org/wiki/Cron>

<https://docs.nestjs.com/techniques/task-scheduling>

1. Cài đặt thư viện

npm i --save-exact @nestjs/schedule@3.0.1

npm install --save-dev @types/cron

2. Sử dụng cron job

```
// * * * * *  
// | | | | |  
// | | | | | day of week  
// | | | | months  
// | | | day of month  
// | | hours  
// | minutes  
// seconds (optional)
```

0 10 0 5 * *

run at 12h10 a.m on the fifth day of the month.

#104. Tự động gửi email với cron job

//todo

#105. Nhận Xét về Dự Án Thực Hành Frontend

= Phần chưa làm

HR chỉ xem đc job của cty ấy. admin full quyền.

trước khi fetch list job => cần thêm điều kiện tìm theo company id

tương tự, trước khi fetch list resume

Chapter 16. Giới Thiệu Kiến Thức Nâng Cao

Giới thiệu các kiến thức nâng cao với NestJS, giúp tối ưu server backend. Ngoài ra, chúng ta sẽ tìm hiểu về cách upgrade version NestJS và cách build dự án backend với Docker.

#106. Authorization

Authentication: xác định người dùng là ai

=> chức năng login để biết là guest/users hệ thống

=> ở đây chúng ta sử dụng cơ chế JWT

Authorization: sau khi đã đăng nhập thành công (authenticated), chúng ta cần xác định, người dùng có thể làm gì ?

=> role và permission

Tài liệu: <https://docs.nestjs.com/security/authorization>

Về Cách làm hiện tại:

- Chia role và permission "tại database"

=> việc khai báo "role" và permission, nếu làm ko chuẩn (không chính xác), sẽ không chạy :v

Cách làm tối ưu hơn, sử dụng :

<https://docs.nestjs.com/security/authorization#integrating-casl>

Trang chủ: <https://casl.js.org/v6/en/>

Với CASL, hỗ trợ việc phân quyền sử dụng:

- Prisma (SQL)

- Mongoose (NoSQL)

- React/Angular/Vue (phân quyền ở frontend)

- CASL hỗ trợ việc phân quyền ở backend, và cũng hỗ trợ phân quyền ở databas (tương tự như cách chúng ta đã làm)

Tại sao mình không hướng dẫn CASL:

- Cú pháp viết không phù hợp với beginners
- CASL chỉ phù hợp khi bạn dùng backend để phân quyền người dùng.

Trong thực tế, nếu bạn dùng SSO, thì việc phân quyền sẽ làm luôn tại SSO.

#107. Helmet

<https://docs.nestjs.com/security/helmet>

npm i --save helmet

```
import helmet from 'helmet';  
// somewhere in your initialization file  
app.use(helmet());
```

=> Test lại upload file (rải CV)

#108. Rate Limiting

Tài liệu: <https://docs.nestjs.com/security/rate-limiting>

npm i --save @nestjs/throttler@4.1.0

tailieusharefree.blogspot.com

#109. Swagger

Tài liệu:

<https://docs.nestjs.com/openapi/introduction>

npm i --save-exact @nestjs/swagger@7.0.4

1. Todo

- add CLI => bind CLI (với DTO)

<https://docs.nestjs.com/openapi/cli-plugin#using-the-cli-plugin>

- add tags (cho controller)

- add bearer token

<https://stackoverflow.com/a/76223916>

```
.addBearerAuth(  
  {  
    type: 'http',  
    scheme: 'Bearer',  
    bearerFormat: 'JWT',  
    in: 'header',  
  },  
  'token',  
)  
.addSecurityRequirements('token')
```

- update login:

<https://stackoverflow.com/questions/57457231/how-nestjs-uses-nestjs-swagger-to-generate-documentation-on-passport-strategies>

tạo thêm dto:

```
//create-user.dto
```

```
export class UserLoginDto {
```

```
  @IsString()
```

```
  @IsNotEmpty()
```

```
  @ApiProperty({ example: 'ajanuw', description: '账号' })
```

```
  readonly username: string;
```

```
  @IsString()
```

```
  @IsNotEmpty()
```

```
  @ApiProperty({
```

```
    example: '123456',
```

```
    description: '密码',
```

```
  })
```

```
  readonly password: string;
```

```
}
```

```
//auth.controller
```

```
update controller:
```

```
@ApiBody({ type: UserLoginDto, })
```

2. Limitation

- Chưa add theo version

<https://github.com/nestjs/swagger/issues/1810>

- Chưa auto add to controller

- Chưa handle type MongoDB

Code tham khảo phần cấu hình swagger:

```
//config swagger
const config = new DocumentBuilder()
  .setTitle('NestJS Series APIs Document')
  .setDescription('All Modules APIs')
  .setVersion('1.0')
  .addBearerAuth(
    {
      type: 'http',
      scheme: 'Bearer',
      bearerFormat: 'JWT',
      in: 'header',
    },
    'token',
  )
  .addSecurityRequirements('token')
  .build();
const document = SwaggerModule.createDocument(app, config);
SwaggerModule.setup('swagger', app, document, {
  swaggerOptions: {
    persistAuthorization: true,
  }
});
```

#110. Healthchecks

Tài liệu: <https://docs.nestjs.com/recipes/terminus>

npm install --save-exact @nestjs/terminus@10.0.1

1. Tạo health module

nest g module health

nest g controller health

2. Check database

//controller

@Controller('health')

export class HealthController {

 constructor(

 private health: HealthCheckService,

 private db: MongooseHealthIndicator,

) {}

 @Get()

 @Public()

 @HealthCheck()

 check() {

 return this.health.check([
 () => this.db.pingCheck('database'),
]);

 }

}

3. Devops

prometheus vs grafana

#111. Docs Whole App Backend

Tài liệu: <https://compodoc.app/>

<https://docs.nestjs.com/recipes/documentation>

npm i -D @compodoc/compodoc

- add to git ignore

#112. Fix Bugs Còn Tồn Động

1. Upload DTO => whitelist: true

2. upload file invalid => still uploaded

```
createMulterOptions(): MulterModuleOptions {
  return {
    storage: diskStorage({
      destination: (req, file, cb) => {
        const folder = req?.headers?.folder_type ?? "default";
        this.ensureExists(`public/images/${folder}`);
        cb(null, join(this.getRootPath(), `public/images/${folder}`))
      },
      filename: (req, file, cb) => {
        //get image extension
        let extName = path.extname(file.originalname);

        //get image's name (without extension)
        let baseName = path.basename(file.originalname, extName);

        let finalName = `${baseName}-${Date.now()}${extName}`
        cb(null, finalName)
      },
    }),
    fileFilter: (req, file, cb) => {
      const allowedFileTypes = ['jpg', 'jpeg', 'png', 'gif', 'pdf', 'doc', 'docx'];
      const fileExtension = file.originalname.split('.').pop().toLowerCase();
      const isValidFileType = allowedFileTypes.includes(fileExtension);

      if (!isValidFileType) {
        cb(new HttpException('Invalid file type', HttpStatus.UNPROCESSABLE_ENTITY), null);
      } else
        cb(null, true);
    },
    limits: {
      fileSize: 1024 * 1024 * 1 // 1MB
    }
  };
}
```


#113. Upgrade NestJS Version

Tài liệu:

<https://docs.nestjs.com/migration-guide>

1. Update version thư viện

Với npm, chúng ta có options để update với npm:

<https://docs.npmjs.com/cli/v9/commands/npm-update>

Cài đặt thư viện check version mới của package:

<https://www.npmjs.com/package/npm-check-updates>

Cài đặt global: **npm install -g npm-check-updates**

or dùng với npx: **npx npm-check-updates**

Sử dụng câu lệnh sau để check version update: **npm**

Lưu ý: sử dụng version control để kiểm soát file package.json

Vì nếu update failed => cài đặt lại version cũ

Để check update "tất cả" packages, sử dụng: **ncu -u**

Trường hợp update "tất cả" gây ra lỗi => chỉ update các package cần thiết

Câu lệnh: ncu -u sẽ làm thay đổi package.json

=> cần chạy câu lệnh **npm i** để cài đặt

2. Một vài lỗi thường gặp

```
npm ERR! code ERESOLVE
npm ERR! ERESOLVE could not resolve
npm ERR!
npm ERR! While resolving: nest-basic-hoidanit@0.0.1
npm ERR! Found: @nestjs/core@9.4.0
npm ERR! node_modules/@nestjs/core
npm ERR!   @nestjs/core@"10.0.3" from the root project
npm ERR!   node_modules/@nestjs-modules/mailer
npm ERR!     @nestjs-modules/mailer@"1.8.1" from the root project
npm ERR!   7 more (@nestjs/mongoose, @nestjs/platform-express, ...)
npm ERR!
npm ERR! Could not resolve dependency:
npm ERR! @nestjs/core@"10.0.3" from the root project
npm ERR!
npm ERR! Conflicting peer dependency: @nestjs/platform-express@10.0.3
npm ERR! node_modules/@nestjs/platform-express
npm ERR!   peerOptional @nestjs/platform-express@"^10.0.0" from
npm ERR!   @nestjs/core@10.0.3
npm ERR!   node_modules/@nestjs/core
npm ERR!     @nestjs/core@"10.0.3" from the root project
npm ERR!
npm ERR! Fix the upstream dependency conflict, or retry
npm ERR! this command with --force, or --legacy-peer-deps
npm ERR! to accept an incorrect (and potentially broken) dependency resolution.
npm ERR! See C:\Users\tuan.pv\AppData\Local\npm-cache\eresolve-report.txt for a full
report.
```

=> sử dụng --legacy-peer-deps

npm i --legacy-peer-deps

=====

> nest start --watch

```
node:internal/modules/cjs/loader:936
  throw err;
  ^
```

Error: Cannot find module 'webpack'

Require stack:

Nguyên nhân lỗi có thể là do conflict version của @nestjs/cli bên trong devDependencies và CLI cài đặt global.

Để check, kiểm tra version cài đặt global: `ncu -g`

Gỡ version global: `npm uninstall -g @nestjs/cli`

Cài đặt (upgrade) version global:

`npm i -g @nestjs/cli@10.0.3` => trùng version với devDependencies

Kiểm tra version cài đặt global: `npm list -g`

Trường hợp bị lỗi => gỡ version global

===

Lưu ý về mail template => lưu ý về đặt tên thư mục => webpack sẽ không dịch code assets

#114. Build Docker

Yêu cầu: máy tính đã cài đặt docker và đã có kiến thức về docker

=> đây là kiến thức đi làm sẽ dùng.

render.com

- Không deploy lên host miễn phí, vì tốn time và thực tế không làm vậy

- Build docker image test local

Download file docker sử dụng trong video:

https://drive.google.com/file/d/1fwc_a4HhDgtq-HfSUBI6821bCoKArlIm/view?usp=sharing

#115. Nhận Xét Về Khóa Học

Học gì tiếp đây :v

Đừng quên review khóa học để nhận được ưu đãi cho khóa học kế tiếp các bạn nhé :v

tailieusharefree.blogspot.com

Lời Kết

Như vậy là chúng ta đã cùng nhau trải qua hơn 130+ videos về sử dụng framework NestJS dành cho backend Node.JS (typescript)

Tất cả các kiến thức mình chia sẻ, đều được lấy từ kinh nghiệm đi làm của mình và...

trang tài liệu NestJS: <https://docs.nestjs.com/>

Dĩ nhiên rằng, trong quá trình quá trình thực hiện khóa học này, mình sẽ không thể tránh khỏi những sai sót (vì nếu không có sai sót thì mình làm member của team NestJS rồi :v).

Vì vậy, nếu thấy sai sót, các bạn cứ thoải mái đóng góp qua Fanpage Hỏi Dân IT nhé.

<https://www.facebook.com/askITwithERIC>

Nếu bạn thấy khóa học này hữu ích, đừng quên Review đánh giá trên Udemy để nhận được ưu đãi (giảm giá) cho các khóa tiếp theo nhé ^^

Hẹn gặp lại các bạn ở các khóa học tiếp theo

Hỏi Dân IT (Eric)