

# TÀI LIỆU KHÓA HỌC “Java Spring Restful API”

Tác giả: Hôi Dân IT & Eric

Version: 4.0

Note cập nhật:

- **Update video #140 Giải Thích Về Lỗi RestResponse cannot be cast to class String (v4)**
- **Update video #141 Fix Lỗi khi chạy lombok với Spring Tool Suite (v4)**
- Cập nhật full source code của khóa học (video #2.2)
- Update link download Mysql Workbench (nếu cài đặt lỗi - v3)

<b>Chapter 1: Bắt buộc xem</b>	<b>5</b>
#1. Hướng dẫn sử dụng khóa học này hiệu quả	5
#2.1 Tài liệu của khóa học	7
#2.2 Hướng Dẫn Sử Dụng Source Code của Khóa Học	7
#3. Demo kết quả đạt được	8
#4. Yêu cầu để học được khóa học này	11
#5. Về Tác giả	12
<b>Chapter 2: Setup Environment</b>	<b>14</b>
#6. Cài Java version 17	14
#7. Cài đặt Visual Studio Code	15
#8. Cấu hình Visual Studio Code	16
#8.1 Lưu ý quan trọng về "Dấu Cách"	16
#9. Tại sao mình dùng VScode ?	17
#10. Cài đặt và sử dụng Git	18
#11. Cài đặt Google Chrome	19
#12. Cài đặt Postman	20
#13. Cài đặt MySQL Workbench	20
<b>Chapter 3: Hello World với Spring REST</b>	<b>21</b>
#14. Setup Dự Án Thực Hành	21
#15. Quá trình tạo dự án Spring	23
#16. Cách đẩy dự án lên Git	24
#17. Cấu trúc dự án thực hành	25
#18. Spring Build Tool	26
#19. My Goal ?	27
<b>Chapter 4: CRUD User với Restful API</b>	<b>28</b>
#20. Tổng quan các kiến thức sẽ học	28
#21. JSON	30
#22. API là gì	31
#23. Status code	32
#24. Test API với Postman	32
#25. Viết Api đầu tiên	33
#26. @RequestBody	34
#27. Java JSON Data Binding	34
#28. @PathVariable	34
#29. Bài tập Get User	35
#30. Chữa Bài Tập Get User	36
#31. Bài tập Update User	36
#32. Chữa Bài Tập Update User	36
#33. Tổng Kết Về RESTful (Basic)	37
#34. Spring Data Rest Project (Bonus)	38
<b>Chapter 5: Response Entity</b>	<b>39</b>
#35. Tại sao cần Response Entity ?	39

#36. HTTP Status Code hay dùng	40
#37. Update Status Code & Body cho APIs	41
#38. Giải thích code (Bonus)	42
<b>Chapter 6: Xử lý Exception</b>	<b>43</b>
#39. Throw Exception	43
#40. Phạm Vi của Exception	44
#41. @ControllerAdvice	45
#42. Format Response before Sending (Part 1)	47
#43. Format Response before Sending (Part 2)	50
<b>Chapter 7: Spring Security với Json Web Token</b>	<b>51</b>
#44. Mô hình Stateful và Stateless	51
#45. Chúng ta đang đứng ở đâu ?	52
#46. Cơ chế xác thực của Stateless	53
#47. JSON Web Token (JWT)	54
#48. Cơ chế mặc định của Spring Security	55
#49. OAuth Flow	56
#50. Spring và OAuth	58
#51. Login Flow	61
#52. loadUserByUsername	63
#53. Debug Code (Bonus)	64
#54. Cơ chế tạo JWT Token	65
#55. Tạo Key (Part 1)	67
#56. Tạo Key (Part 2)	70
#57. Bảo Vệ Endpoint (API) với JWT	71
#58. Xử lý JWT Exception	73
#59. Tổng kết về JWT	75
<b>Chapter 8: Phân tích dự án thực hành</b>	<b>77</b>
#60. Giới thiệu dự án thực hành	77
#61. Phân tích Model cho Databases	78
#62. Setup dự án thực hành Frontend	79
#63. CORS là gì ?	80
#64. Cách fix CORS	82
#65. Spring và CORS	83
<b>Chapter 9: Modules Company</b>	<b>85</b>
#66. Model Company	85
#67. Bài Tập Create Company	87
#68. Chữa Bài Tập Create Company	89
#69. Before Save Entity với JPA (Part 1)	89
#70. Before Save Entity với JPA (Part 2)	90
#71. Bài Tập Get/Update/Delete Company	91
#72. Chữa Bài Tập Get/ Update/Delete Company	93
#73. Query với Pagination	94
#74. Giới thiệu về Specification	96

#75. Query với Filter (Part 1)	98
#76. Query với Filter (Part 2)	99
#77. Customize Message với Annotation (Bonus)	100
#78. Versioning API	101
<b>Chapter 10: Modules User</b>	<b>102</b>
#79. Update User Model	102
#80. Bài tập CRUD User	103
#81. Chữa Bài tập CRUD User	109
#82. API Login	109
#83. Set Cookies (Part 1)	111
#84. Set Cookies (Part 2)	113
#85. API Get Account (F5 - Refresh)	114
#86. Giải thích cơ chế JWT và Spring Security (Bonus)	115
#87. API Refresh Token (Part 1)	117
#88. API Refresh Token (Part 2)	119
#89. Bài tập API Logout	120
#90. Nguyên tắc check code frontend	121
#91. Test giao diện frontend	124
<b>Chapter 11: Modules Job/Resume</b>	<b>125</b>
#92. Code Refactoring	125
#93. Model Relationship (Associations)	125
#94. Bài Tập Update Model User/Company	127
#95. Model Job	128
#96. Bài Tập CRUD Job	130
#97. Chữa Bài tập CRUD Job	132
#98. Về Upload File	132
#99. Read File From Path	133
#100. Upload File (Part 1)	135
#101. Upload File (Part 2)	136
#102. Download a File (Bonus)	138
#103. Model Resume	139
#104. Bài Tập CRUD Resume	140
#105. Chữa Bài tập CRUD Resume	141
#106. Test giao diện Frontend (Part 1)	141
#107. Test giao diện Frontend (Part 2)	142
<b>Chapter 12: Modules Permission &amp; Role</b>	<b>143</b>
#108. Model Permission & Roles	143
#109. Bài tập CRUD Permissions & Roles	144
#110. Chữa Bài tập CRUD Permissions & Roles	145
#111. Update User với Permissions & Roles	145
#112. Tạo Fake Data với SQL	146
#113. Test Giao Diện Frontend (Part 1)	147
#114. Tạo Sample Data (Part 1)	148

#115. Tạo Sample Data (Part 2)	149
#116. Test Giao Diện Frontend (Part 2)	150
#117. Cách xử lý phân quyền tại Frontend (Bonus)	151
#118. Interceptor	152
#119. Test Giao Diện Frontend (Part 3)	154
<b>Chapter 13: Modules Subscribers</b>	<b>156</b>
#120. Model Subscribers	156
#121. Bài tập CRUD Subscribers	157
#122. Chữa bài tập CRUD Subscribers	158
#123. Cấu hình Send Email với Spring	158
#124. Hello Word với Spring Email	159
#125. Send Email với Template (Part 1)	160
#126. Send Email với Template (Part 2)	163
#127. Send Email với Template (Part 3).	164
#128. Cron Job (Part 1)	166
#129. Cron Job (Part 2)	167
#130. Nhận xét về dự án Frontend	168
<b>Chapter 14: Tổng Kết</b>	<b>170</b>
#131. Xử lý Global Exception	170
#132. Swagger	171
#133. Logging	173
#134. Build Dự Án với Docker	174
#135. Nhận xét về cách code dự án Spring	175
#136. Cách tự tạo dự án Spring Restful của bạn	176
#137. Nhận xét về dự án thực hành	178
#138. What's next ? Học gì tiếp theo	179
#139. Suy Nghĩ Về Chuyện Thực Tập & Làm Fresher (Bonus)	180
#140. Giải Thích Về Lỗi RestResponse cannot be cast to class String	181
#141. Cấu Hình Lombok cho STS (Nếu Gặp Lỗi)	182

## Chapter 1: Bắt buộc xem

Hướng dẫn sử dụng khóa học hiệu quả

### #1. Hướng dẫn sử dụng khóa học này hiệu quả

Bạn vui lòng "xem video lần lượt" theo trình tự. Vì khóa học như 1 dòng chảy, video sau sẽ kế thừa lại kết quả của video trước đó.

#### 1. Dành cho học viên "có ít thời gian"

Nếu bạn vội, cần học nhanh, hoặc "bạn đã biết rồi", thì "vẫn xem video, cơ mà không cần code theo".

Lưu ý: vẫn xem qua tài liệu khóa học để biết "video hướng dẫn gì".

Đã "Không xem video", thì cần "đọc giáo án".

Có như vậy mới biết khóa học nó làm cái gì.

#### 2. Dành cho học viên "thông thường"

**Nguyên tắc:**

- Xem video lần lượt
- Xem video kết hợp với giáo án. Bạn không cần take note, vì những điều quan trọng đã có trong giáo án

- **Bạn vui lòng code theo video.**

Nếu bạn "code theo ý bạn", vui lòng "không hỏi khi có bugs".

Câu chuyện này giống như việc bạn đi khám bệnh, nhưng không tin lời bác sĩ

=> Nếu bạn giỏi, bạn làm luôn bác sĩ, còn đi khám bệnh làm gì.

- **Bạn có thể "code theo ý bạn muốn", sau khi "đã kết thúc khóa học"**

- Nếu bạn có thắc mắc (hoặc có ý tưởng/nhận thấy bugs), take note lại, bạn hỏi, rồi mình giải đáp.

Chứ không phải là "tự ý làm theo điều các bạn muốn".

Vì đa phần, các bugs trong khóa học mình đã fix hết rồi.

Nên là yên tâm để học theo bạn nhé.

### 3. Về cách code.

Bạn vui lòng code theo video, từ cách đặt tên biến, hàm. Vì mình đã tuân theo "convention tối thiểu" khi bạn đi làm đấy

### 4. Về bài tập thực hành

Đối với bài tập thực hành, bạn cứ code theo cách bạn hiểu, và kết hợp với "search Google, stackoverflow..."

**KHÔNG DÙNG CHATGPT.** Đây giống kiểu chưa học "phép tính", mà đã "dùng máy tính".

**Nên nhớ 1 điều, trước 2023, không có chat gpt, thì mình học như thế nào ?**

Khi bạn đã đi làm, bạn có quyền dùng cái gì bạn thích, còn với beginner, hãy biết say NO với CHAT GPT.

tương tự bạn dạy con bạn:

học lớp cấp 1: không chịu học tính nhẩm => đưa luôn máy tính cho nó. Rồi máy tính ra, là không biết làm phép tính

còn với học sinh cấp 2, 3 : dùng máy tính tùy thích

### 5. Về source của cả khóa học

**Source code của cả khóa học KHÔNG ĐƯỢC CUNG CẤP.**

Học viên cần code theo video để có source code.

Chỉ có videos mình nói "cho source code", source code sẽ có trong tài liệu khóa học.

### 6. Về cách support học viên

về cách support (có 3 cách):

1. bạn đặt câu hỏi trực tiếp trên udemy
2. đăng lên group fb dành cho học viên udemy
3. inbox trực tiếp page này bạn nhé

## **#2.1 Tài liệu của khóa học**

Link download tài liệu khóa học được cập nhật theo thời gian, các bạn lưu ý về version của tài liệu để biết được tài liệu là cũ hay mới nhé.

## **#2.2 Hướng Dẫn Sử Dụng Source Code của Khóa Học**

Đây là khóa học thực hành, mình “KHÔNG khuyến khích” sử dụng source code.

Vì nếu các bạn “không tự code”, kiến thức không bao giờ (never) là của các bạn.

Đi làm, ai code hộ bạn ?

Link project final, xem [tại đây](#)

//Có thể xem full source code, hoặc xem code theo từng video (check theo commit)



### #3. Demo kết quả đạt được

Link video demo: <https://youtu.be/NemtTpGNsps>

#### 1. Công nghệ sử dụng

**Ý tưởng:** tương tự khóa Nest.js, điểm khác biệt là dùng Java thay vì Typescript (Nodejs):  
<https://hoidanit.vn/khoa-hoc/nestjs-voi-typescript-mongodb-sieu-de-64686ec6fb456bbb90663dd6.html>

**Backend:** Java Spring

- **Spring Boot** : cấu hình và chạy dự án Spring một cách nhanh chóng
- **Spring Security**: xác thực (authentication) và phân quyền người dùng (authorization) với JWT (json web token) sử dụng cơ chế oauth2 (không viết custom Filter)
- **Spring JPA** : xử lý và thao tác với cơ sở dữ liệu database với ORM

**Frontend** : React Vite (typescript) **được cung cấp sẵn. Không học code React trong khóa học này**

**Database:** **MySQL** (phần mềm MySQL Workbench)

Build Tool: **Gradle - Kotlin**

**Các kỹ năng khác:**

- Viết code theo mô hình Dependency Injection
- Thực hành kỹ năng Debug với Spring (đặc biệt là Spring Security)
- Quản lý API với Swagger
- Gửi email theo template, cron job để tự động gửi email
- Sử dụng MySQL với ORM (Spring JPA), hỗ trợ filter với Specification (Paging/Sorting)

## 2. Học viên nào có thể học ?

Học viên cần trang bị các kiến thức sau trước khi theo học:

- Biết cú pháp của Java và có tư duy lập trình hướng đối tượng
- Đã có kiến thức cơ bản về Spring, bao gồm:
  - Viết code theo mô hình Controller - Service - Repository
  - Sử dụng Spring JPA để query dữ liệu
  - Sử dụng mô hình MVC với Spring

Học viên bắt đầu từ số 0, hoặc chưa từng code Spring, theo học không phù hợp, tham khảo: <https://hoidanit.vn/khoa-hoc/java-spring-mvc-ultimate-for-beginners-65ce0b770c05f4450fbd86ac.html>

## 3. Về source của cả khóa học

Source code của cả khóa học **KHÔNG ĐƯỢC CUNG CẤP**.

Học viên cần code theo video để có source code, hoặc:

Mỗi video thực hành luôn có source code đính kèm trong tài liệu khóa học

### Tại sao mình làm vậy:

hơn 3 năm làm khóa học, mình nhận thấy 99% học viên xin source code sẽ "không code" và "không hiểu" dự án làm gì, và truyền đạt kiến thức gì  
=> kết quả học tập không cao

và, đừng có lấy lý do "bận" nên không có thời gian học. Nếu "bận", thì bạn "học làm gì", thuê người khác cho nhanh.

Nên nhớ, không có đường tắt khi "học kiến thức". Vì nếu nó "nhanh đến", thì nó cũng "nhanh quên" thôi @@

### Yên tâm 1 điều là:

- Nếu bạn code theo video, 100% sẽ có được thành quả
- Source code được update theo thời gian. 1 khi bugs được học viên report  
=> sẽ được fix ngay lập tức

#### 4. Triển khai dự án

Dự án được chạy tại localhost và không triển khai lên hosting, vì:

- rất ít hosting FREE hỗ trợ java + mysql
- hosting FREE không lưu trữ ảnh upload

Tuy nhiên, trong khóa học có hướng dẫn build với Docker

=> nếu bạn muốn triển khai thực tế, mua vps, cài docker và triển khai

Tham khảo (hướng triển khai với docker): <https://hoidanit.vn/khoa-hoc/ultimate-guide-to-deploy-react-nodejs-640bee82f7099c369b3bc6a4.html>

#### 5. Về testing

**Khóa học này không hướng dẫn viết testcase**

#### 6. Về cách support khi gặp bug

có 3 cách:

1. bạn đặt câu hỏi trực tiếp trên udemy
2. đăng lên group Facebook dành cho học viên udemy
3. inbox trực tiếp Facebook Page: <https://www.facebook.com/askITwithERIC/>

#### **#4. Yêu cầu để học được khóa học này**

- Bạn cần biết cú pháp java và java 8 (lamda) + tư duy lập trình hướng đối tượng  
[https://www.youtube.com/playlist?list=PLncHg6Kn2JT5EVkhKoJmzOytHY39Mrf\\_o](https://www.youtube.com/playlist?list=PLncHg6Kn2JT5EVkhKoJmzOytHY39Mrf_o)

- Cần có kiến thức cơ bản về Spring : Spring JPA, Spring Security  
<https://hoidanit.vn/khoa-hoc/java-spring-mvc-ultimate-for-beginners-65ce0b770c05f4450fbd86ac.html>

- Về source code của cả khóa học => không share  
Source code (nếu có) sẽ được đính kèm theo từng video trong tài liệu khóa học

## **#5. Về Tác giả**

### **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.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/>

## 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 “Hỏi Dân 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/pfbid02gyasktd3semgxat6nevnvwh4c8epzu3i7kpzhr7s7gmmfcvucyz96eb8avnvgnhl>

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 KỸ, 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.vn/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 @@

## Chapter 2: Setup Environment

*Cài đặt môi trường thực hiện dự án*

### #6. Cài Java version 17

Môi trường Java trong dự án này sử dụng version 17. Bạn vui lòng sử dụng version này để hạn chế tối đa lỗi có thể xảy ra.

Link tải Java 17 và IDE sử dụng trong video (windows):

[https://drive.google.com/drive/folders/11\\_KIENr8SJebcQMGMFJSDgr1Sli0Hv0J?usp=sharing](https://drive.google.com/drive/folders/11_KIENr8SJebcQMGMFJSDgr1Sli0Hv0J?usp=sharing)

- Lưu ý: cài đặt java 17

#### 1. JDK (java development kit)

<https://www.oracle.com/java/technologies/downloads/archive/>

<https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>

download và cài đặt java 17 (8, 11, 17, 21 -> LST: long term support)

- kiểm tra version: **java --version**

- kiểm tra java path (if needed)

[https://www.w3schools.com/java/java\\_getstarted.asp](https://www.w3schools.com/java/java_getstarted.asp)

#### 2. Trường hợp dùng nhiều version Java

Nếu như bạn **bắt buộc phải dùng nhiều version của Java** trên cùng một máy tính, **có thể (xác suất nhỏ) là xảy ra lỗi** trong quá trình thực hiện dự án.

Ngược lại, nếu không “bắt buộc” phải dùng nhiều version Java, bạn nên dùng một version thôi. Như vậy nó sẽ đảm bảo môi trường thực thi code của bạn và mình là giống hệt nhau, hạn chế tối đa lỗi có thể xảy ra.

Với trường hợp dùng nhiều version Java, mình sẽ đề cập lại (cũng như hướng giải quyết) tại chapter sau (khi setup dự án thực hành)

## **#7. Cài đặt Visual Studio Code**

Công cụ code trong dự án sử dụng VSCode, 1 IDE hoàn toàn miễn phí

Link download:

<https://code.visualstudio.com/download>



## #8. Cấu hình Visual Studio Code

### 1. Format Code

Setup Format on Save

Mục đích: Mỗi lần nhấn Ctrl + S , code sẽ được auto format trông cho đẹp/dễ nhìn

### 2. Cài đặt Extensions

**Lưu ý:** off các extension như eslint, prettier ... để tránh xung đột

**Fact:** đi làm, người ta cấu hình eslint, prettier..thông qua code, vì mỗi 1 dự án (1 khách hàng 1 yêu cầu), cài global qua extension thì cái nào cũng giống cái nào

Đồng thời, với rule trên sẽ đảm bảo mọi thành viên trong team sẽ có cấu hình giống nhau

**Các extensions cài đặt thêm:**

- code spell checker : hỗ trợ check chính tả khi đặt tên tiếng anh
- extension pack for java : hỗ trợ code/debug java
- spring boot Extension Pack : hỗ trợ code java spring

#### #8.1 Lưu ý quan trọng về "Dấu Cách"

- **Trailing Spaces** : cái này dùng để check file có thừa khoảng trắng hay không, ae cài đặt thêm nhé. Dùng extension để check file, ví như file **application.properties** (các bạn copy thừa khoảng trắng là nó bị toang)

<https://marketplace.visualstudio.com/items?itemName=shardulm94.trailing-spaces>

Cấu hình chỉ ignore (check) một vài file cụ thể:

<https://github.com/shardulm94/vscode-trailingspaces?tab=readme-ov-file#ignore-syntax>

```
"trailing-spaces.syntaxIgnore": ["html", "java", "javascript", "markdown", "sql",  
"typescript", "typescriptreact", "xml", "xsl", "yaml"]
```

### **#9. Tại sao mình dùng VScode ?**

Có 2 IDE nổi tiếng nhất để code java

- Eclipse/Spring Tool Suite (miễn phí)
- IntelliJ IDEA (trả phí), dùng miễn phí bản Community

**IDE thực chất là công cụ code, giúp gợi ý code và phát hiện lỗi**

=> dùng công cụ code nào mà bạn "thoải mái nhất"

**Mình chọn VScode vì:**

- miễn phí (nếu bạn code Frontend thì chắc chắn bạn sẽ thích)
- có gợi ý code và phát hiện lỗi
- theme dark :v
- hỗ trợ git out-of-the-box
- support mạnh mẽ cho "frontend" => tức là 1 IDE cho cả frontend/backend

**Trong khóa học này, chỉ cần bạn code giống mình, dùng IDE nào không quan trọng, điều quan trọng, chính là cách chúng ta code ra làm sao :v**

**Recommend: dùng VScode, để đảm bảo bạn và mình giống nhau 100%, từ coding cho tới debug :v**

Yên tâm 1 điều là: điều quan trọng nhất chính là khả năng bạn "tư duy" (mindset), bạn có thể dùng những điều học được, để áp dụng sang IDE bạn thích.

Fact: mình đã từng rơi vào công ty (khá to), cơ mà không mua license bản quyền phần mềm (trong khi không cho dùng crack)

=> bắt buộc phải dùng các công cụ free @@

## **#10. Cài đặt và sử dụng Git**

- Nếu bạn chưa biết gì về Git, xem nhanh tại đây:

<https://www.youtube.com/playlist?list=PLncHg6Kn2JT6nWS9MRjSnt6Z-9Rj0pAlo>

**- Sử dụng Git theo nguyên tắc:**

### **1. Học xong video nào, commit đẩy lên Github/Gitlab**

=> tạo cơ hội để thực hành câu lệnh của Git, ví dụ:

git add

git commit

git push...

### **2. Git là công cụ "mặc định bạn phải biết" khi đi làm phần mềm**

=> điều 1 ở trên giúp bạn thực hành

### **3. Thói quen học xong video nào, đẩy code lên Git, giúp bạn tạo ra bản "backup" cho project của bạn**

Ví dụ máy tính bạn bị hỏng đột xuất/bị mất

=> vẫn còn code, chỉ cần pull về code tiếp mà không phải code từ đầu.

Nên nhớ, khóa học này mình không share source code => chỉ bạn giúp được bạn

### **4. Trong trường hợp bạn bị bug**

=> bạn có thể gửi link github/gitlab cho mình xem => support fix bug

=> **Mục đích sử dụng git ở đây là : backup code + thực hành công cụ đi làm mà bạn "phải biết" nếu muốn đi thực tập/đi làm.**

## **#11. Cài đặt Google Chrome**

Lưu ý: sử dụng version tiếng anh

=> change language

Mục tiêu:

- Sử dụng Google Chrome để chạy ứng dụng web
- Ngôn ngữ hiển thị là Tiếng Anh

## **#12. Cài đặt Postman**

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

Mục đích: dùng Postman để test backend Java

## **#13. Cài đặt MySQL Workbench**

Link tài liệu hướng dẫn:

- Windows: <https://dev.mysql.com/doc/refman/5.7/en/windows-installation.html>
- MacOS: <https://dev.mysql.com/doc/refman/5.7/en/macos-installation.html>

Link hướng dẫn dành cho MacOS: <https://www.youtube.com/watch?v=2cvH0HRjZF8>

Link tải file cài đặt:

<https://dev.mysql.com/downloads/installer/>

Download file cài đặt sử dụng trong video (windows):

[https://drive.google.com/file/d/1RvOJDROSEqrxD\\_w\\_OLwxJ2c4h6Y6kRI6/view](https://drive.google.com/file/d/1RvOJDROSEqrxD_w_OLwxJ2c4h6Y6kRI6/view)

Lưu ý: version MySQL mình cài đặt (link drive) là 8.0.37

Khi cài đặt, các bạn chọn full options (cài đặt full), nó hơi khác với video (mình quay video là version 8.0.35).

Trong quá trình cài đặt (nếu có lỗi xảy ra), ví dụ như không initial database..., các bạn fix bằng cách chủ động tải version mới nhất về xem có bị lỗi không nhé.

## **Chapter 3: Hello World với Spring REST**

*Làm quen với Java Spring Framework sử dụng REST*

### **#14. Setup Dự Án Thực Hành**

**Bắt buộc clone project thực hành từ git về để thực hiện cho khóa học này. KHÔNG TỰ Ý TẠO PROJECT để code.**

Lý do: việc clone code sẽ đảm bảo code của bạn và mình là giống nhau. Hạn chế tối đa lỗi có thể xảy ra (trong quá trình thực hiện khóa học)

#### **1. Cài đặt dự án**

Trước khi tiến hành, cần đảm bảo máy tính bạn đã cài đặt VSCode và bạn đã biết cách sử dụng git.

##### **Bước 1:** Clone dự án

<https://gitlab.com/public-starter-projects/1/000-java/02-java-spring-restful/01-java-spring-jobhunter-starter>

##### **Bước 2:** Chạy dự án

Mở dự án tải về bằng VSCode

Chờ một xíu để VSCode cài đặt thư viện cần thiết

Chạy dự án.

Mặc định, project sẽ được chạy tại: <http://localhost:8080/>

#### **2. Trường hợp trong máy có nhiều version của java**

Nếu tại bước 2, bạn không chạy được dự án với VSCode, làm sao các bước sau:

**Nguyên nhân lỗi:** VSCode không biết chọn version Java nào để chạy dự án (do tồn tại nhiều version java trong máy tính)

### **Cách khắc phục:**

**Bước 1:** Tải và cài đặt một trong các công cụ sau (chỉ cần cài 1 công cụ thôi)

- Eclipse/Spring Tool Suite (miễn phí)
- IntelliJ IDEA (trả phí), dùng miễn phí bản Community

**Recommend:** sử dụng Spring Tool Suite (miễn phí, tải về dùng luôn)

Link tải: <https://spring.io/tools>

Video cài đặt:

[https://www.youtube.com/watch?v=HXqceeN9X74&list=PLncHg6Kn2JT5EVkhKoJmzOytHY39Mrf\\_o&index=16](https://www.youtube.com/watch?v=HXqceeN9X74&list=PLncHg6Kn2JT5EVkhKoJmzOytHY39Mrf_o&index=16)

**Bước 2:** Mở dự án Spring bằng công cụ IDE tại bước 1

Điểm khác biệt giữa việc mở dự án = VSCode và mở = IDE như Eclipse, Spring Tool Suite... là các IDE này thông minh hơn, sinh ra để code Java

VSCode supports chính Javascript.

Chạy dự án = IDE tại bước 1, thông thường sẽ chạy thành công (không cần setup gì đặc biệt)

**Bước 3:** Mở lại dự án với VSCode

Magic ở đây, là chúng ta cần cấu hình version Java thực hiện cho dự án (IDE làm tại bước 2)

Lần này, chạy dự án với VSCode sẽ thành công :v

## #15. Quá trình tạo dự án Spring

Lưu ý: Video này xem cho biết, không làm theo video này.

Về quá trình tự code dự án Spring như thế nào. Cuối khóa học mình sẽ có video hướng dẫn chi tiết cụ thể

The screenshot shows the start.spring.io web application generator interface. The browser address bar shows 'start.spring.io'. The interface is divided into two main sections: 'Project' and 'Dependencies'.

**Project Section:**

- Project:** Radio buttons for Gradle - Groovy, **Gradle - Kotlin** (selected), and Maven.
- Language:** Radio buttons for **Java** (selected), Kotlin, and Groovy.
- Spring Boot:** Radio buttons for 3.3.0 (SNAPSHOT), 3.3.0 (M3), 3.2.5 (SNAPSHOT), **3.2.4** (selected), 3.1.11 (SNAPSHOT), and 3.1.10.
- Project Metadata:**
  - Group: vn.hoidanit
  - Artifact: jobhunter
  - Name: jobhunter
  - Description: Spring Restful by Hoi Dan IT
  - Package name: vn.hoidanit.jobhunter
  - Packaging: **Jar** (selected), War
  - Java: 22, 21, **17** (selected)

**Dependencies Section:**

- ADD DEPENDENCIES... CTRL + B** button.
- Spring Boot DevTools** (DEVELOPER TOOLS): Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
- Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- Thymeleaf** (TEMPLATE ENGINES): A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.
- Spring Security** (SECURITY): Highly customizable authentication and access-control framework for Spring applications.
- Spring Data JPA** (SQL): Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
- MySQL Driver** (SQL): MySQL JDBC driver.
- Validation** (JIO): Bean Validation with Hibernate validator.
- Spring Boot Actuator** (OPS): Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.



## #16. Cách đẩy dự án lên Git

### 1. Check code kết nối tới database mysql

Đảm bảo rằng bạn đã cài đặt MySQL Workbench, và có tài khoản đăng nhập vào database.

//**build.gradle.kts**

Enable 2 dependencies là jpa và mysql

### 2. Quản lý mã nguồn với git

**Lưu ý: học xong video nào, tạo commit, đẩy code lên git ứng với video đấy.**

Tên message viết có ý nghĩa, để người khác còn đọc & hiểu bạn đang làm cái gì

Mục đích: ví dụ bạn cần fix bug, cơ mà không biết lỗi từ khi nào, dùng git để check

Hoặc, khi cần nhờ mình fix bug, gửi link git

**Tránh tình trạng không dùng git, máy tính hư, mất hết code đã học**

**Bước 1:** Truy cập Github/Gitlab/Bitbucket... để tạo repository

Recommend dùng Gitlab (nếu bạn có nhiều dự án và muốn gom nhóm để quản lý), còn không thì dùng Github cho nó thân thiện

**Bước 2:** Tiến hành commit

git add .

git commit -m "message-co-y-nghia"

//trong lần đầu tiên đẩy commit, do bạn clone code của mình (đã cấu hình git remote url), nên cần ghi đè remote url sang repository của bạn

git remote set-url origin new.git.url/here

<https://stackoverflow.com/a/2432799>

**Bước 3:** Test new commit

//todo

## **#17. Cấu trúc dự án thực hành**

- Những file có tiền tố "gradle" là công cụ build dự án, hiểu một cách đơn giản:

- + quản lý các thư viện cài đặt
- + dịch code/build/run

+ bao gồm:

build.gradle.kts

gradlew (gradlew.bat)

settings.gradle.kts

- Thư mục viết code: src/main

## #18. Spring Build Tool

### 1. Gradle

Giúp bạn:

- Dịch code java -> .class
- Copy phần resource vào thư mục build
- Tạo file jar
- Run test
- And more...

Về Gradle: <https://www.youtube.com/watch?v=R6Z-Sxb837I>

Sử dụng gradle:

- Run dự án Spring Boot:  
**gradle bootRun**  
**/gradle bootRun**
- Build file jar: gradlew bootJar

### 2. So sánh Gradle và các công cụ khác

Tham khảo:

<https://stackoverflow.com/questions/45335874/gradle-what-is-the-benefit-if-i-switch-from-groovy-to-kotlin>

Maven vs Gradle

<https://gradle.org/maven-vs-gradle/>

Về lý do tại sao tồn tại song song thư mục **bin** và thư mục **build**: (open issue)

<https://github.com/redhat-developer/vscode-java/issues/2338>

## **#19. My Goal ?**

Tại sao là My goal , không phải Your goal ?

Link dự án Jhipster: <https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter>

Môi trường cài đặt:

Java : 17.0.9

Node.js : 20.11.1

`./gradlew -x webapp`

`npm start`

Dự Án Spring MVC:

<https://hoidanit.vn/khoa-hoc/java-spring-mvc-ultimate-for-beginners-65ce0b770c05f4450fbd86ac.html>

Những điều mình đã làm tại dự án Spring MVC:

- Sử dụng Spring JPA
- Full mô hình MVC (với view là html)
- Sử dụng Spring Security

## **Chapter 4: CRUD User với Restful API**

*Sử dụng Spring REST làm tính năng CRUD*

### **#20. Tổng quan các kiến thức sẽ học**

#### **1. Nội dung sẽ học**

- Phát triển REST APIs với Spring ( @RestController )
- Tìm hiểu các khái niệm liên quan tới REST APIs, JSON và giao thức HTTP
- Sử dụng Postman để test API

#### **2. Vấn đề tồn đọng**

##### **Dưới góc nhìn của Developer (DEV)**

Ví dụ dự án laptopshop, khóa học Spring MVC:

<https://hoidanit.vn/khoa-hoc/java-spring-mvc-ultimate-for-beginners-65ce0b770c05f4450fbd86ac.html>

##### **Dưới góc nhìn của business (phân tích nghiệp vụ):**

Ví dụ: Xây dựng ứng dụng Weather App

Chắc chắn thông tin (dữ liệu) về Thời tiết, bạn sẽ không có, cần phụ thuộc vào bên thứ 3 cung cấp dịch vụ  
(ví dụ Trung tâm Dự báo Khí tượng thủy văn Quốc gia)

Kiến trúc dự án:

My Weather App (Client) -----> Weather Service (Server)

##### **- Làm sao để kết nối giữa client và server ?**

Sử dụng REST API thông qua giao thức http  
REST : REpresentational State Transfer

##### **- Ngôn ngữ lập trình nào được sử dụng ?**

REST không phụ thuộc vào ngôn ngữ lập trình

-> viết Client/Server bằng ngôn ngữ bạn muốn: Java, C#, PHP, Javascript, Python...

**- Sử dụng định dạng data nào (data format) ?**

JSON và XML được dùng

JSON sử dụng phổ biến nhất

JSON : JavaScript Object Notation

## #21. JSON

Tham khảo: [https://www.w3schools.com/js/js\\_json\\_intro.asp](https://www.w3schools.com/js/js_json_intro.asp)

### 1. JSON là gì

- JavaScript Object Notation
- Là 1 định dạng để lưu trữ và trao đổi dữ liệu (mà không mã hóa gì hết / plain text)
- Được sử dụng ở nhiều ngôn ngữ khác nhau: Java, C#, Javascript ...

### 2. Cú pháp

Ví dụ về JSON:

```
{  
  "id": 1,  
  "name": "Hỏi Dân IT",  
  "age": 25,  
  "active": true  
}
```

- Sử dụng dấu { //data } để định nghĩa JSON
- Các thuộc tính được định nghĩa theo quy luật:  
name: value (ngăn cách với nhau bởi dấu hai chấm)

Thuộc tính name luôn được bọc bởi " " (double quotes)

### 3. JSON values

- number
- string
- boolean
- nested JSON object
- Array
- null

//todo : json với nested object

## **#22. API là gì**

Tham khảo: <https://jsonplaceholder.typicode.com/>

### **1. HTTP là gì**

Tham khảo: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>

HTTP Method -----> CRUD Operation

POST ----- Create a new entity

GET ----- Read a list of entities or single entity

PUT ----- Update an existing entity

DELETE ----- Delete an existing entity

### **2. Cấu trúc HTTP Request**

- Request line : method + URL
- Header variables
- Message body: json

### **3. API**

API, hiểu đơn giản, là một đường link URL tại backend

Frontend sẽ gọi tới đường link URL này để lấy/sử dụng dữ liệu.



### **#23. Status code**

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

### **#24. Test API với Postman**

Tham khảo:

<https://jsonplaceholder.typicode.com/>

Lợi thế của postman so với browser

//todo

## **#25. Viết Api đầu tiên**

Source code video này (#25):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Lưu ý: trong chương học này chưa xử lý Exception. Về Exception sẽ được xử lý tại chương học tiếp theo

Cách ẩn file với VSCode:

<https://stackoverflow.com/a/30142299>

### **Yêu cầu:**

#### **Bước 1:** Tạo Entity User

name: string

email: string

password: string

#### **Bước 2:**

Tạo url backend, hardcode data để tạo user

Chỉ cần tạo được user trong database là thành công.

Viết code theo DI:

<https://docs.spring.io/spring-data/jpa/reference/jpa/getting-started.html>

Controller -> Service -> Repository

## **#26. @RequestBody**

Source code video này (#26):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tài liệu:

<https://docs.spring.io/spring-framework/reference/6.0/web/webmvc/mvc-controller/ann-methods/requestbody.html>

Yêu cầu:

Từ Postman, gửi thông tin User

Controller nhận thông tin, tạo mới User

Controller trả lại thông tin user được tạo

## **#27. Java JSON Data Binding**

Data binding là quá trình convert JSON data thành Java POJO (Java Object) hoặc ngược lại.

- Spring sử dụng Jackson Project (behind the scenes)

<https://github.com/FasterXML/jackson>

## **#28. @PathVariable**

Source code video này (#28):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tài liệu: <https://www.baeldung.com/spring-pathvariable>

**Sửa lại URL:**

POST (CREATE): <http://localhost:8080/user>

Làm chức năng xóa user by id:

DELETE (Xóa): [http://localhost:8080/user/\\${user-id}](http://localhost:8080/user/${user-id})

## **#29. Bài tập Get User**

**Yêu cầu 1: fetch user by id**

**Tạo URL:** GET : [http://localhost:8080/user/\\${user-id}](http://localhost:8080/user/${user-id})

Output: Trả về đối tượng user (nếu tồn tại)

Lưu ý: chưa cần xử lý exception (ví dụ như id không tồn tại). Về xử lý exception sẽ được làm tại chapter tiếp theo.

Hiện tại đang test trường hợp hoàn hảo (dữ liệu hợp lệ 100%)

**Yêu cầu 2: fetch all user (chưa cần phân trang - pagination)**

**Tạo URL:** GET : <http://localhost:8080/user>

Output : Trả về tất cả users có trong database

Gợi ý: sử dụng kiểu dữ liệu List<User> . Java Spring sẽ tự động convert từ List sang Array khi trả về kết quả

### **#30. Chữa Bài Tập Get User**

Source code video này (#30):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### **#31. Bài tập Update User**

Yêu cầu: Cập nhật thông tin user (name, email, password)

Tạo URL:

PUT : <http://localhost:8080/user>

Body:

```
{  
  id, name, email, password  
}
```

Output: Trả về đối tượng user (nếu tồn tại)

### **#32. Chữa Bài Tập Update User**

Source code video này (#32):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

## **#33. Tổng Kết Về RESTful (Basic)**

### **1. Chuẩn RESTful**

#### **Ứng với tác vụ CRUD:**

##### **GET (READ) - Đọc thông tin**

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET>

- Được dùng để lấy data
- Không truyền data ở body
- Có thể truyền data ở url (ví dụ delete by id - @PathVariable)

##### **POST (CREATE) - Tạo mới thông tin**

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST>

- Được dùng để truyền data lên Server
- Có thể truyền data ở body request (@RequestBody)

##### **PUT/PATCH (UPDATE) - Cập nhật thông tin**

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH>

- Được dùng để truyền data lên Server
- Có thể truyền data ở body request (@RequestBody)

##### **DELETE (DELETE) - Xóa thông tin**

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/DELETE>

- Được dùng để xóa data
- Có thể truyền data ở url (ví dụ delete by id - @PathVariable)

## #34. Spring Data Rest Project (Bonus)

Tài liệu: <https://spring.io/projects/spring-data-rest>

### 1.Spring Data Rest

- **Tự động tạo endpoint** (api) ứng với domain (model) mà không cần viết code.  
Ví dụ, bạn có Entity là User, nó sẽ tự động tạo :  
GET /users  
POST /users ...
- Có thể tùy chỉnh (customize) để phục vụ API của JPA, như paging, sorting...

### 2. So sánh Spring Data Rest và @RestController

Nhược điểm của đũa này, là ưu điểm của đũa kia :v

**Spring Data Rest:** code ngắn, phù hợp để phát triển ứng dụng một cách nhanh nhất có thể (vì code rất ít). Vì vậy, phù hợp để giải quyết tác vụ CRUD đơn giản.

**@RestController** : code dài hơn. 100% kiểm soát code của bạn. Bạn thích viết ngắn viết dài tùy thích :v

Việc lựa chọn cái nào, là phụ thuộc vào bài toán bạn gặp phải :v

## Chapter 5: Response Entity

*Xử lý phản hồi của API với Response Entity*

### #35. Tại sao cần Response Entity ?

Mô hình đang thực hiện: **client** -> **server**

Cụ thể:

**client** (browser/postman) **gửi request (yêu cầu)** tới **server** (java) **thông qua RESTful APIs**

**Server** (java) **gửi response (phản hồi)** về cho **client**

Hiện tại, phản hồi (response) đang ở dạng text, hoặc JSON object

Để các hệ thống nói chuyện với nhau 1 cách đầy đủ nhất, 1 lời phản hồi (response) sẽ gồm:

- **Thông tin header** (ví dụ bạn muốn dùng cookies chẳng hạn)
- **Thông tin status** (http status) : mã phản hồi
- **Thông tin body (nếu có)** : data phản hồi, thông thường dưới dạng JSON

#### 1. Spring ResponseEntity

Tài liệu: <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/ResponseEntity.html>

- Kế thừa `HttpEntity`, với bổ sung `http status`

Minh họa một vài cách dùng thông dụng:

- trả về `http status`

**`ResponseEntity.status(HttpStatus.Ok).body(null);`**

- trả về `Status` và `headers`:

**`ResponseEntity.status(HttpStatus.Ok).headers(Instance_of_HttpHeaders).build();`**

- trả về `Status`, `headers` và `body`:

**`ResponseEntity.status(HttpStatus.Ok).headers(Instance_of_HttpHeaders).body(Instance_of_object_send_back_to_client);`**



## **#36. HTTP Status Code hay dùng**

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

### **Mã lỗi ám chỉ request thành công:**

**200** - request succeeded (hay dùng cho method GET/PUT/DELETE)

**201** - request created a resource (hay dùng cho method POST)

**204** - no content to return (dùng khi bạn muốn thông báo không có data ở phản hồi)

**202** - Accepted (the request has been accepted for processing, but the processing has not been completed): dùng khi bạn chạy job/background task và muốn gửi phản hồi cho client

### **Mã lỗi ám chỉ request thất bại (lỗi do client):**

**400** - Bad request (lỗi exception, validate...)

**401** - Unauthorized (unauthenticated): bạn chưa đăng nhập, có nghĩa rằng bạn cần login thành công thì mới có quyền sử dụng endpoint (API)

**403** - Forbidden (unauthorized): bạn đã đăng nhập thành công, tuy nhiên, bạn không có quyền hạn (authorization) để thực hiện tác vụ này

**404** - Resource not found : lỗi huyền thoại cmnr :v

**405** - Method not supported : check cho đúng method khi sử dụng với endpoint

**415** - Media not supported: bạn cần truyền đúng định dạng format mà server/client mong muốn. Ví dụ, bạn không thể dùng JSON để gửi file lên server (cần dùng formData)

### **Mã lỗi ám chỉ request thất bại (lỗi do server):**

**500** - Internal Server error: lỗi xảy ra bên trong Server, cần đọc logs để biết lỗi gì (exception, bugs...)

**503** - Service Unavailable : server ngỏm (không chạy) nên không có sẵn để sử dụng

**504** - Gateway Timeout : server (có thể) không ngỏm, cơ mà không phản hồi trong thời gian quy định (quá lâu để phản hồi)

### **#37. Update Status Code & Body cho APIs**

Source code video này (#37):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tài liệu: <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/ResponseEntity.html>

//responseEntity<void> nếu set body = null

//lưu ý: endpoint nên viết số nhiều

Ví dụ => user => users

//update cho tất cả CRUD endpoint

### **#38. Giải thích code (Bonus)**

//tham khảo jhipster

[https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/web/rest/DepartmentResource.java?ref\\_type=heads](https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/web/rest/DepartmentResource.java?ref_type=heads)

//builder pattern

<https://springframework.guru/gang-of-four-design-patterns/builder-pattern/>

## Chapter 6: Xử lý Exception

Xử lý ngoại lệ trong quá trình sử dụng API

### #39. Throw Exception

Source code video này (#39):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

[https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/web/rest/DepartmentResource.java?ref\\_type=heads](https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/web/rest/DepartmentResource.java?ref_type=heads)

//todo: // fetch user by id

Yêu cầu:

xử lý exception khi truy cập API:

<http://localhost:8080/users/5a>

Ở đây: 5a là string (invalid)

#### Bước 1: Định nghĩa exception

Tham khảo: <https://stackoverflow.com/questions/8423700/how-to-create-a-custom-exception-type-in-java>

#### Bước 2: Xử lý exception

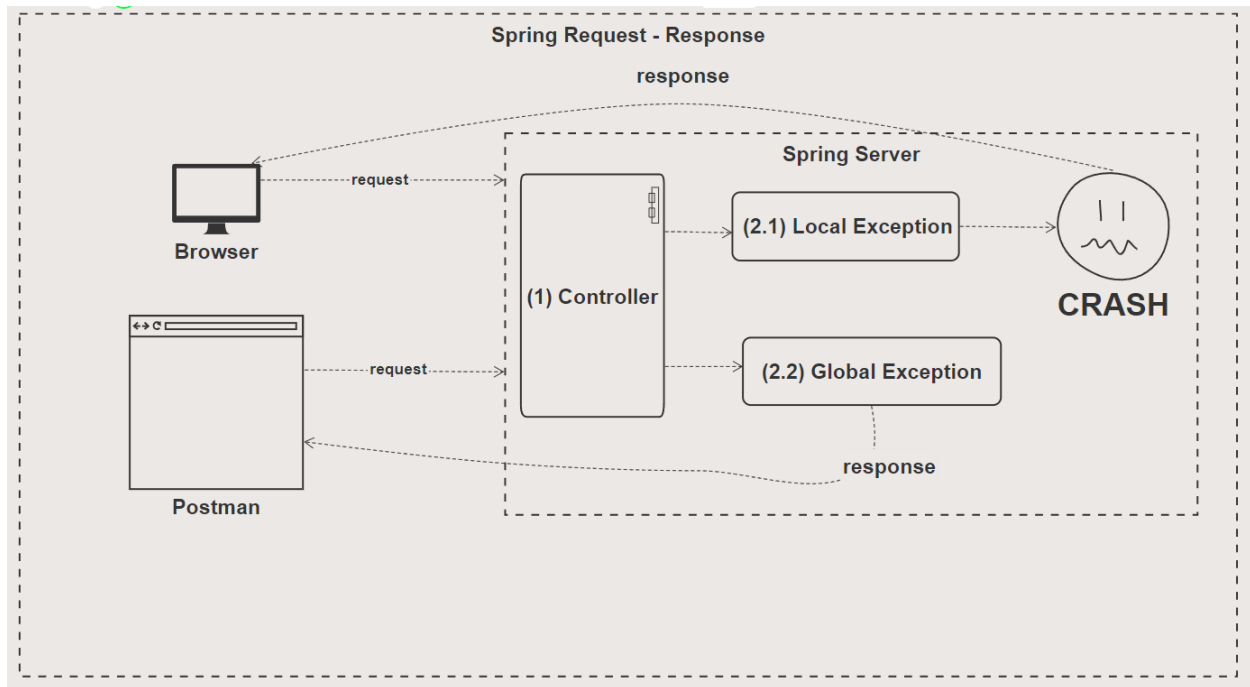
<https://springframework.guru/exception-handling-in-spring-boot-rest-api/>

@ExceptionHandler Annotation

```
@ExceptionHandler(value = BlogAlreadyExistsException.class)
public ResponseEntity
handleBlogAlreadyExistsException(BlogAlreadyExistsException
blogAlreadyExistsException) {
    return new ResponseEntity("Blog already exists", HttpStatus.CONFLICT);
}
```

## #40. Phạm Vi của Exception

//todo



### 1. Giới thiệu về AOP

<https://docs.spring.io/spring-framework/reference/core/aop.html>

Tham khảo: [https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/tree/master/src/main/java/com/mycompany/myapp?ref\\_type=heads](https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/tree/master/src/main/java/com/mycompany/myapp?ref_type=heads)

## #41. @ControllerAdvice

### 1. Spring @ControllerAdvice

Tài liệu: <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/ControllerAdvice.html>

- Được giới thiệu từ version Spring Framework **3.2**
- Xử lý @ExceptionHandler, @InitBinder, or @ModelAttribute được chia sẻ tại tất cả controller trong ứng dụng MVC
- Cũng có thể làm tương tự cho RESTful, tuy nhiên cần **@ResponseBody**

@ResponseBody giúp convert Response trả về dưới dạng JSON

### Ví dụ 1: Xử lý với MVC

#### @ControllerAdvice

```
public class GlobalExceptionHandler {
    @ExceptionHandler(value = Exception.class)
    public ModelAndView defaultErrorHandler(HttpServletRequest req, Exception e) throws
    Exception {
        ModelAndView mav = new ModelAndView();
        mav.addObject("exception", e);
        mav.addObject("url", req.getRequestURL());
        mav.setViewName("error");
        return mav;
    }
}
```

### Ví dụ 2: Xử lý với RESTful

```
// Handle custom exceptions across the application
@ExceptionHandler(CustomException.class)
@ResponseBody
public ResponseEntity<?> handleCustomException(CustomException ex, WebRequest
request) {
    ErrorDetails errorDetails = new ErrorDetails(
        HttpStatus.BAD_REQUEST.value(),
        ex.getMessage(),
        request.getDescription(false));
    return new ResponseEntity<>(errorDetails, HttpStatus.BAD_REQUEST);
}
```

Có 1 cách nữa để sử dụng với RESTFul, là sử dụng `@RestControllerAdvice`

`@RestControllerAdvice` được giới thiệu từ version 4.3

<https://docs.spring.io/spring-framework/docs/4.3.9.RELEASE/javadoc-api/org/springframework/web/bind/annotation/RestControllerAdvice.html>

So sánh `@RestControllerAdvice` và `@ControllerAdvice`:

<https://stackoverflow.com/a/43124517>

`@RestControllerAdvice` = `@ControllerAdvice` + `@ResponseBody`

Source code video này (#41):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

## #42. Format Response before Sending (Part 1)

Source code video này (#42):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tài liệu: <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/mvc/method/annotation/ResponseBodyAdvice.html>

### 1. Ví dụ về API trong thực tế

[https://developers.momo.vn/v2/#/docs/query\\_status](https://developers.momo.vn/v2/#/docs/query_status)

Một kết quả trả ra của API, luôn cần có:

- **Status** code : 200, 201, ...
- **Message** : miêu tả nếu có
- **Data** : nếu có

Mỗi 1 công ty có 1 quy định khác nhau về format của phản hồi, tuy nhiên, tối thiểu API cần cung cấp 3 thông tin trên

Mục tiêu: Trả về response theo format định sẵn

Trường hợp thành công:

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

Ví dụ:

```
{
  "statusCode": 201,
  "message": "User Login",
  "data": {
    "access_token": "",
    "user": {}
  }
}
```



**Trường hợp lỗi :**

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

Ví dụ: {

```
  "message": "Username/password không hợp lệ!",  
  "error": "Unauthorized",  
  "statusCode": 401  
}
```

## 2.Format Response

**Cách 1:** sử dụng local với ResponseEntity

<https://stackoverflow.com/a/44840010>

**Cách 2:** sử dụng global với Controller Advice

<https://stackoverflow.com/a/52104852>

**Bước 1:** Định nghĩa format object

```
public class RestResponse<T> {  
  private String error;  
  
  // message có thể là string, hoặc arrayList  
  private Object message;  
  private T data;  
}
```

## **Bước 2:** Sử dụng ControllerAdvice

<https://stackoverflow.com/a/51551663>

@ControllerAdvice

```
public class CustomResponseBody implements ResponseBodyAdvice<Object> {
```

```
    @Override
```

```
    public boolean supports(
```

```
        MethodParameter returnType,
```

```
        Class converterType
```

```
    ) {
```

```
        return true;
```

```
    }
```

```
    @Override
```

```
    public Object beforeBodyWrite(
```

```
        Object body,
```

```
        MethodParameter returnType,
```

```
        MediaType selectedContentType,
```

```
        Class selectedConverterType,
```

```
        ServerHttpRequest request,
```

```
        ServerHttpResponse response
```

```
    ) {
```

```
        // Check if the status code represents an error
```

```
        if (status >= 400) {
```

```
            //case error
```

```
        } else {
```

```
            //case success
```

```
        }
```

```
    }
```

```
}
```

### **#43. Format Response before Sending (Part 2)**

Source code video này (#42):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

## Chapter 7: Spring Security với Json Web Token

Bảo vệ APIs với cơ chế JWT của mô hình Stateless

### #44. Mô hình Stateful và Stateless

Tương tự khi so sánh Monolithic và Microservice

#### 1.Keyword hay gặp

**Lưu ý 1:** Không nên dịch nghĩa, thay vào đây là nên “cảm nhận” ý nghĩa của nó.

**Lưu ý 2:** các dịch nghĩa bên dưới mang tính chất tương đối. Bạn có thể google để xem “cách định nghĩa” chính xác nhất.

**Stateful** = state application + **full** : chứa đầy state của application

Lưu trữ thông tin bên trong ứng dụng, ví dụ như thông tin người dùng đăng nhập

**Session** : phiên đăng nhập. Được dùng trong mô hình Stateful, cách mà ứng dụng lưu trữ data giữa các lời gọi request

**Stateless** = state application + **less** : không chứa state của application

Không lưu trữ thông tin trong ứng dụng (nothing at all)

Trong tiếng anh: useful vs useless (homeless :v)

**Monolithic:** (architecture) all-in-one : bạn code tất cả mọi thứ trong 1 dự án. Ví dụ như mô hình MVC truyền thống

**Microservice:** (architecture): single-unit : bạn chia code thành các “thành phần riêng lẻ” (module/service), mục đích là giảm sự phụ thuộc giữa các thành phần, và tăng tính mở rộng

#### 2. Stateful và Stateless

Stateful = Monolithic (khóa học MVC)

Stateless = Microservice (khóa học này - Restful API)

**Không có khái niệm Monolithic hay Microservice tốt hơn**, cái nào cũng có ưu và nhược điểm của nó. Việc bạn cần làm, là lựa chọn công nghệ sao cho nó “phục vụ tốt với mục đích của bạn”

## #45. Chúng ta đang đứng ở đâu ?

**Câu hỏi 1:** Tại sao chúng ta chọn Stateless thay vì Stateful (khóa MVC - Monolithic) ?

- Đơn giản, vì code khổ quá, muốn tìm cách nào đó mà “đời thẳng dev” nó nhàn hơn :v

Sử dụng RESTful API, thông thường (hay dùng nhất) là áp dụng cho mô hình Stateless.

Có nghĩa rằng, bạn sẽ chia tách code thành Frontend và Backend riêng lẻ (không phụ thuộc vào công nghệ)

Ví dụ:

**Frontend:** bạn code với Angular/Vue/React.

Hoặc thậm chí chỉ là HTML, CSS, Javascript (AJAX)

**Backend:** sử dụng bất cứ ngôn ngữ backend nào bạn muốn, như Javascript, Java, PHP...

Javascript : Express, Nestjs...

Java: Java Spring

PHP: Laravel

C# : .Net

Python: Django

Ruby: Ruby on Rail ...

RESTful API là cầu nối giữa frontend và backend. Điều này lý giải:

- **Frontend** sẽ không thể thao tác với database (CRUD dữ liệu), mà cần làm gián tiếp qua API
- **Backend** sẽ không cần code giao diện, chỉ thực hiện nhiệm vụ thao tác với dữ liệu (lưu trong database) và tạo ra API cho frontend dùng.

Chúng ta đang làm nhiệm vụ này (tạo ra RESTful API)

## #46. Cơ chế xác thực của Stateless

Trong mô hình Stateless, không tồn tại khái niệm "Session", thay vào đấy là "Token"

### 1. Cơ chế xác thực dựa vào Session (Stateful)

**Bước 1:** login với username/password

Nếu login thành công, server sẽ tạo lưu thông tin tại:

- Client thông qua cookies (lưu SESSION\_ID)
- Server trong memory (RAM) hoặc database (lưu full thông tin của user login)

**Bước 2:** Mỗi lần người dùng F5 (refresh) website gửi 1 request từ client lên server, các bước làm tại Server:

- Client sẽ gửi kèm SESSION\_ID (thông qua cookies)
- Server sẽ kiểm tra SESSION\_ID **có đang tồn tại hay không** ? Nếu có, tiếp tục việc truy cập như thông thường (else, logout)

**Mô hình Stateful với Session** chỉ áp dụng hiệu quả, khi và chỉ khi bạn kiểm soát cả frontend và backend

### 2. Cơ chế xác thực dựa vào Token (Stateless)

Server với Server, mobile app, desktop app... (những cái không có cookies)

Token: là 1 chuỗi ký tự đã được mã hóa (chỉ Server mới có thể hiểu)

Ví dụ: adfasdfasdfasdfxyadfyajsdfasdfad

**Bước 1:** login với username/password

Nếu login thành công, server sẽ tạo **token**, lưu tại đâu, client tự quyết định.

**Server không lưu bất cứ thông tin gì về việc user login**

**Bước 2:** Mỗi lần người dùng F5 (refresh) website gửi 1 request từ client lên server, sẽ cần **gửi kèm token** đã có tại bước 1.

- Server sẽ giải mã token để biết được user có hợp lệ hay không ?

## **#47. JSON Web Token (JWT)**

Tài liệu:

<https://jwt.io/>

### **1. JWT là gì ?**

<https://jwt.io/introduction>

- Là một chuỗi ký tự được “mã hóa” (thông qua thuật toán) và có tính bảo mật cao.
- Được sử dụng để trao đổi thông tin giữa các hệ thống với nhau (server - server, client - server)

### **2. Cấu trúc của JWT**

Gồm:

- Header
- Payload (đây là cái chúng ta quan tâm nhất)
- Signature

## #48. Cơ chế mặc định của Spring Security

Source code video này (#48):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### 1. Cấu hình Password Encoder

Tham khảo:

[https://gitlab.com/public-starter-projects/1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/config/SecurityConfiguration.java?ref\\_type=heads](https://gitlab.com/public-starter-projects/1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/config/SecurityConfiguration.java?ref_type=heads)

```
//todo : hoàn thiện tính năng CREATE user với hash Password  
passwordEncoder.encode()
```

**Yêu cầu bắt buộc:**

**Bước 1:** Xóa hết user đang có trong database

**Bước 2:** Tạo 3 user với thông tin email như sau (mật khẩu mặc định: 123456)

[admin@gmail.com](mailto:admin@gmail.com)

[user@gmail.com](mailto:user@gmail.com)

[hoidanit@gmail.com](mailto:hoidanit@gmail.com)

KHÔNG NÊN đặt mật khẩu phức tạp, vì đây là môi trường test. Sau khi học hết chapter này, việc tạo tài khoản như nào, đặt mật khẩu ra làm sao, bạn làm tùy ý. Nhưng, cần giữ 3 tài khoản ở trên để thực hành hết khóa học

**Còn khi đang học, vui lòng XEM & LÀM THEO HƯỚNG DẪN** (đỡ phải suy nghĩ)

### 2. Enable Security

```
//todo: cấu hình base http basic
```



## #49. OAuth Flow

### 1. Lưu ý về cách code JWT với Spring

KHÔNG LÀM NHƯ CÁC VÍ DỤ BÊN DƯỚI: ĐÂY LÀ CÁCH CODE CHO CHẠY ĐƯỢC, KHÔNG PHẢI LÀ CÁCH CÔNG TY LÀM, VÌ NÓ TIỀM ẨN RỦI RO VỀ SECURITY

<https://www.geeksforgeeks.org/spring-boot-3-0-jwt-authentication-with-spring-security-using-mysql-database/>

<https://www.youtube.com/watch?v=KxqlJblhzfl>

Giải thích quy trình code cho chạy được:

**Bước 1:** Cài đặt các thư viện hỗ trợ JWT (encode/decode)

**Bước 2:** Viết Filter (tạm gọi là jwtFilter) để xử lý cho JWT

Filter này sẽ được chạy trước khi Spring Security xử lý (chạy các Filter khác)

jwtFilter sẽ có nhiệm vụ decode/verify (giải mã và xác thực tính hợp lệ) của JWT mà client truyền lên

### 2. OAuth là gì ?

Tham khảo:

<https://www.youtube.com/watch?v=ZV5yTm4pT8g>

<https://www.youtube.com/watch?v=ZDuRmhLSLOY>

OAuth (Open Authorization) là một chuẩn (standard) dùng để xác thực thông tin người dùng thông qua token. (từ 2012)

Tham khảo (v2.0): <https://datatracker.ietf.org/doc/html/rfc6749>

Được ứng dụng rộng rãi trong mô hình stateless (microservice) khi một ứng dụng liên quan tới nhiều dịch vụ (service)

**Các vai trò (role) khi sử dụng OAuth:**

- **Resource Owner** : người sở hữu nguồn tài nguyên, thông thường chính là User - người sở hữu tài khoản của họ. Ví dụ, bạn là chủ sở hữu của tài khoản Facebook do bạn tạo ra
- **Resource Server**: nơi hosting dữ liệu của người dùng. Ví dụ, tài khoản Facebook của bạn sẽ được lưu trữ tại server nào đó do Facebook Quản lý
- **Client**: Ứng dụng muốn truy cập/sử dụng thông tin của người dùng
- **Authorization Server**: nơi chịu trách nhiệm tạo ra access-token để cho **Client** sử dụng **Resource Server** (sau khi đã được **Resource Owner** cho phép)

Ví dụ:

<https://www.codeproject.com/Articles/1171546/OAuth-Authorization-flows-explained-with-examples>

## #50. Spring và OAuth

### 1. Lịch sử ra đời và sử dụng với Spring

- Trước 2022: Spring Security OAuth

<https://spring.io/blog/2022/06/01/spring-security-oauth-reaches-end-of-life>

<https://github.com/spring-attic/spring-security-oauth>

- Sau 2022 : có 2 dự án (project) nổi bật hỗ trợ OAuth là:

**Spring Security (hỗ trợ OAuth 2.0)**

<https://docs.spring.io/spring-security/reference/6.1/servlet/oauth2/index.html>

<https://github.com/spring-projects/spring-security>

**Spring Authorization Server (hỗ trợ OAuth 2.1)**

<https://spring.io/projects/spring-authorization-server>

<https://github.com/spring-projects/spring-authorization-server>

Câu hỏi đặt ra là, nên lựa chọn dự án OAuth nào cho dự án của chúng ta ?

Tại sao Spring Team không gộp làm 1 dự án, mà lại chia thành 2 dự án riêng lẻ ?

**Spring Security:**

- Gắn liền với hệ sinh thái của Spring, gần như là default (cấu hình mặc định) khi liên quan tới việc Authentication/Authorization
- Một sự thay đổi nhỏ thôi, cũng có thể là breaking-change, khi có quá nhiều ứng dụng phụ thuộc vào nó. Cơ mà, tích hợp luôn OAuth thì quá tuyệt vời (all-in-one)

**Spring Authorization Server:**

- Ra đời muộn hơn Spring Security (2022) :  
<https://spring.io/blog/2022/11/22/spring-authorization-server-1-0-is-now-ga>
- Up to date với Security, có nghĩa là cập nhật những cái mới nhất, ví dụ OAuth 2.1
- Là một dự án độc lập (base dự trên Spring Security)

## 2. Mô hình áp dụng

<https://docs.spring.io/spring-security/reference/servlet/oauth2/index.html#oauth2-resource-server>

Cài đặt dependencies:

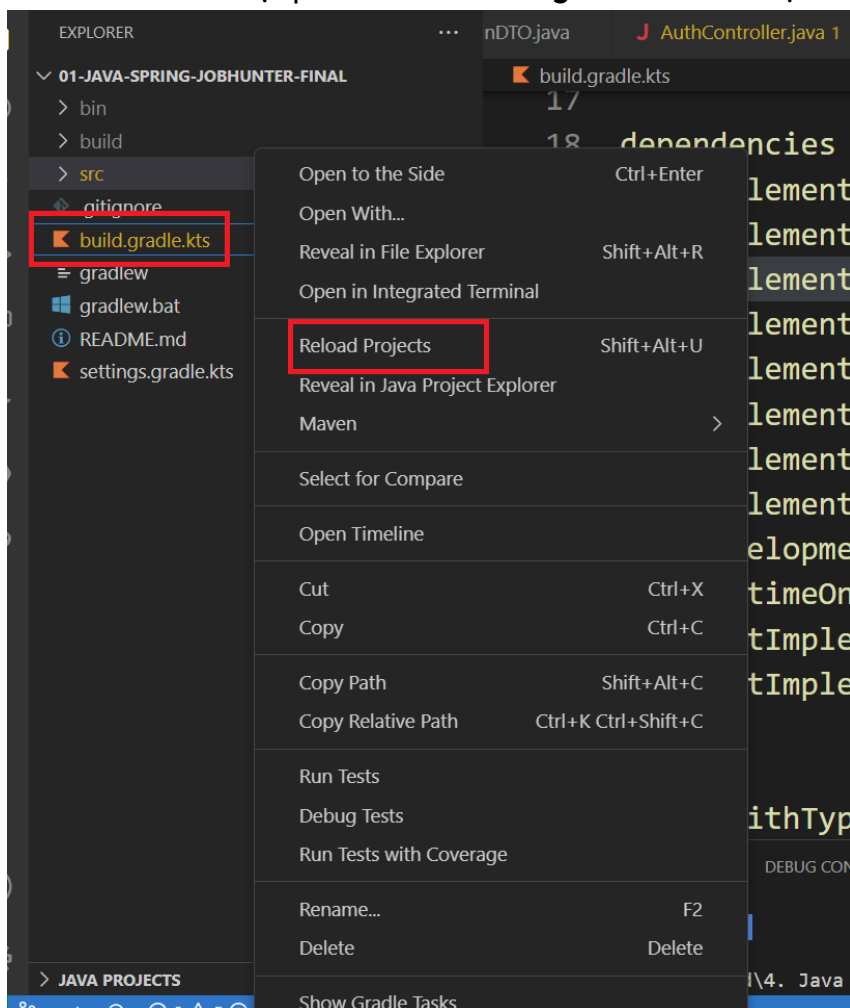
```
implementation("org.springframework.boot:spring-boot-starter-oauth2-resource-server")
```

Fix lỗi không cài đặt dependency:

**Bước 1:** thêm dependency vào file build của gradle (lưu ý là trong video mình có yêu cầu thay đổi, vì lúc quay video mình có bug)

```
implementation("org.springframework.boot:spring-boot-starter-oauth2-resource-server")
```

**Bước 2:** nhấn chuột phải vào file **build.gradle.kts** => chọn **Reload Projects**



Với OAuth2, chúng ta cần:

- **Auth Server** (Authentication/Authorization): nơi xác định người dùng có tồn tại hay không ? Nếu có, người dùng có quyền hạn gì sau khi đã xác thực/đăng nhập thành công)
- **Resource Server** : nơi chứa nguồn tài nguyên (data) mà người dùng muốn truy cập

Áp dụng vào bài toán của chúng ta: (cách làm đơn giản nhất)

**Client** : React

**Server** : Java Spring (chỉ có 1 server) - all in one

=> server này cần làm các nhiệm vụ sau:

- Xác thực người dùng (authentication): chức năng login.
- Xác định quyền hạn của người dùng (authorization) : permission và role
- Sau khi đã xác thực và đảm bảo người dùng “phù hợp”, server cho phép client truy cập nguồn tài nguyên (thông qua api)

## #51. Login Flow

### 1. Mô hình MVC

<https://docs.spring.io/spring-security/reference/servlet/authentication/architecture.html>

Khi bạn nhấn nút login, sẽ gửi request lên server và trigger filter.

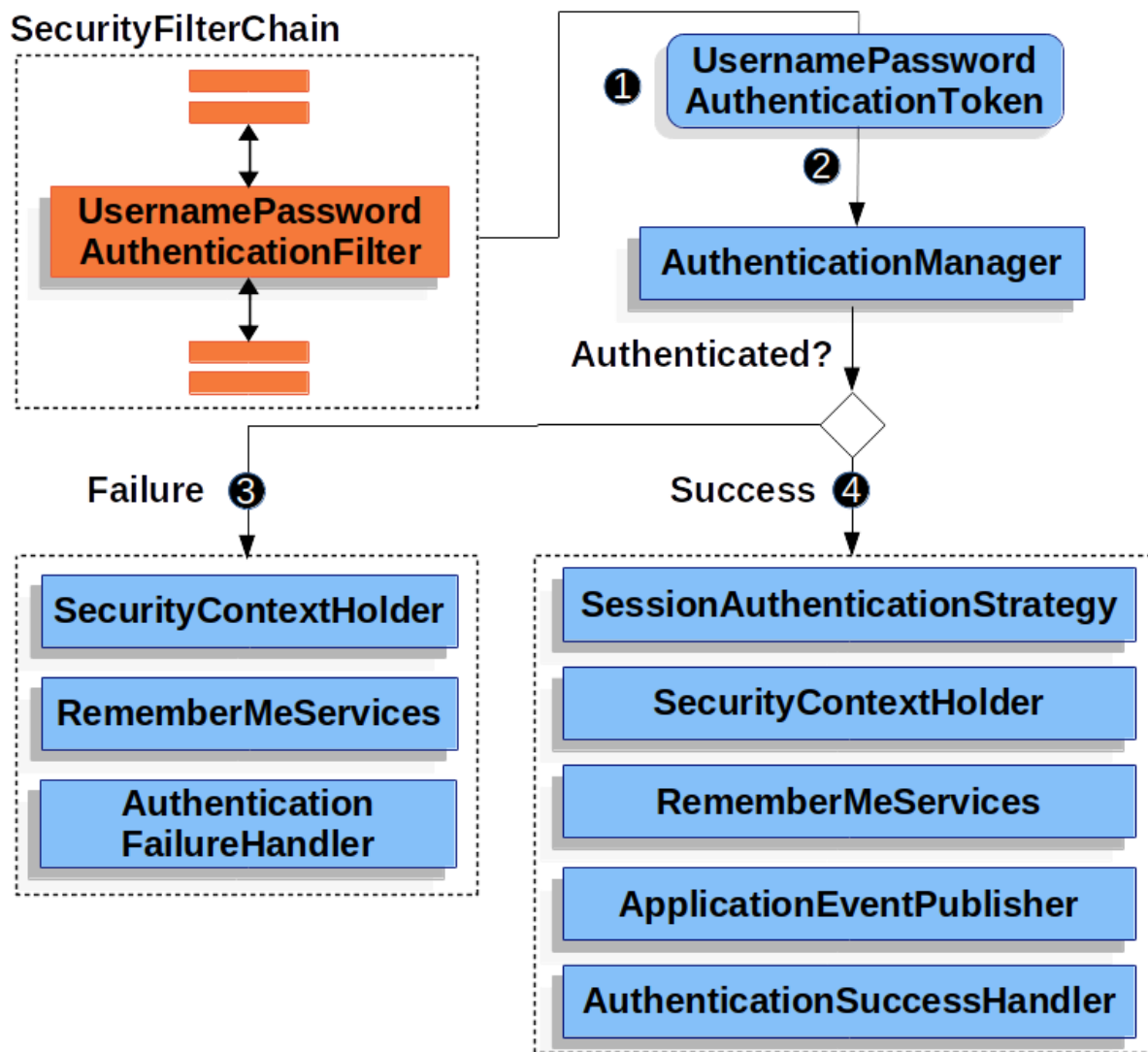
Filter được gọi ở đây là **UsernamePasswordAuthenticationFilter**.

**AuthenticationManager**

**ProviderManager**

**DaoAuthenticationProvider**

**UserDetails loadUserByUsername()**



### 2. Đối với Mô Hình Stateless

Bạn không có form login để trigger filter, thay vào đấy là API endpoint.

**Bước 1:** bạn submit api với username/password

**Bước 2:** Spring sẽ không trigger Filter nào hết, mà sẽ chạy thẳng vào controller, nơi bạn định nghĩa endpoint (vì cơ chế JWT là cơ chế không được xây dựng sẵn của Spring)

=> chúng ta cần viết logic để xử lý đăng nhập người dùng (kế thừa tương tự logic của **UsernamePasswordAuthenticationFilter** )

Why ?

Vì khi làm theo luồng của Spring Security, chúng không cần bận tâm về quá trình so sánh mật khẩu user và nạp thông tin user vào Security Context (để tái sử dụng - tương tự như Session)

//Nạp input gồm username/password vào Security

```
UsernamePasswordAuthenticationToken authenticationToken  
= new UsernamePasswordAuthenticationToken(username, password)
```

//xác thực người dùng => cần viết hàm loadUserByUsername

```
Authentication authentication =  
authenticationManagerBuilder.getObject().authenticate(authenticationToken);
```

//nạp thông tin (nếu xử lý thành công) vào SecurityContext

```
SecurityContextHolder.getContext().setAuthentication(authentication);  
String jwt = this.createToken(authentication);
```

**Tham khảo:**

[https://gitlab.com/public-starter-projects/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/web/rest/AuthenticateController.java?ref\\_type=heads#L54](https://gitlab.com/public-starter-projects/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/web/rest/AuthenticateController.java?ref_type=heads#L54)

## #52. loadUserByUsername

Source code video này (#52):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### Bước 1: Tạo DTO class

username: String

password: String

### Bước 2: Tạo endpoint: /login

//Nạp input gồm username/password vào Security

```
UsernamePasswordAuthenticationToken authenticationToken  
= new UsernamePasswordAuthenticationToken(username, password)
```

//xác thực người dùng => cần viết hàm loadUserByUsername

```
Authentication authentication =  
authenticationManagerBuilder.getObject().authenticate(authenticationToken);
```

//nạp thông tin (nếu xử lý thành công) vào SecurityContext

```
SecurityContextHolder.getContext().setAuthentication(authentication);
```

### Bước 3: Viết Service UserDetails loadUserByUsername

//lưu ý: xử lý custom exception (nếu sai mật khẩu/email)

Tham khảo: [https://gitlab.com/public-starter-projects/1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blame/master/src/main/java/com/mycompany/myapp/security/DomainUserDetailsService.java?ref\\_type=heads#L20](https://gitlab.com/public-starter-projects/1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blame/master/src/main/java/com/mycompany/myapp/security/DomainUserDetailsService.java?ref_type=heads#L20)

Về cách đặt Bean Name:

<https://stackoverflow.com/a/53758500>

```
return new User(  
    user.getEmail(),  
    user.getPassword(),  
    Collections.singletonList(new SimpleGrantedAuthority("ROLE_USER")));
```



## #53. Debug Code (Bonus)

Source code video này (#53):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### 1. Xử lý Exception

- Validate dữ liệu username/password (không được để trống)
- Validate trường hợp username/password đã truyền lên, nhưng không hợp lệ, bao gồm email không tồn tại/hợp lệ và sai password

Advance: handle tất cả exceptions:

[https://gitlab.com/public-starter-projects/1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/web/rest/errors/ExceptionHandler.java?ref\\_type=heads](https://gitlab.com/public-starter-projects/1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/web/rest/errors/ExceptionHandler.java?ref_type=heads)

//hiện tại, chưa xử lý exception của security (sẽ được xử lý cùng với việc validate jwt)

<https://www.baeldung.com/spring-security-exceptionhandler>

### 2. Luồng Debug

Kỹ năng debug: <https://code.visualstudio.com/docs/editor/debugging>

AuthenticationManager

ProviderManager (hàm authenticate)

=> quét ra DaoAuthenticationProvider

UserDetail

## #54. Cơ chế tạo JWT Token

### 1. Cài đặt thư viện

Chúng ta đã cài đặt:

implementation("org.springframework.boot:spring-boot-starter-oauth2-resource-server")

Công cụ trên đã hỗ trợ đủ cơ chế encode/decode JWT (mà không cần cài đặt thêm gì hết)

<https://github.com/spring-projects/spring-security/blob/main/oauth2/oauth2-jose/spring-security-oauth2-jose.gradle>

api 'com.nimbusds:nimbus-jose-jwt'

=> thư viện cần học cách sử dụng là: **nimbus-jose-jwt**

<https://connect2id.com/products/nimbus-jose-jwt>

### 2. Cách thuật ngữ hay dùng

**JOSE**: JSON Object Signing and Encryption

**JWS**: JSON Web Signatures (quá trình validate data - client gửi data lên server, server cần validate)

**JWE**: JSON Web Encryption (quá trình mã hóa data - trước khi gửi data về cho client, server cần mã hóa)

**JWK**: JSON Web Key

**JWA**: JSON Web Algorithms

**HMAC** : Hash-based Message Authentication Code (sử dụng SHA, key secret và message để hash tạo ra output)

Về JWT, tham khảo ebook:

[https://drive.google.com/drive/folders/1XqqX\\_00eWaGlFvWLkHAwvxabdk2WovNw?usp=sharing](https://drive.google.com/drive/folders/1XqqX_00eWaGlFvWLkHAwvxabdk2WovNw?usp=sharing)

### 3. Cơ chế tạo ra JWT

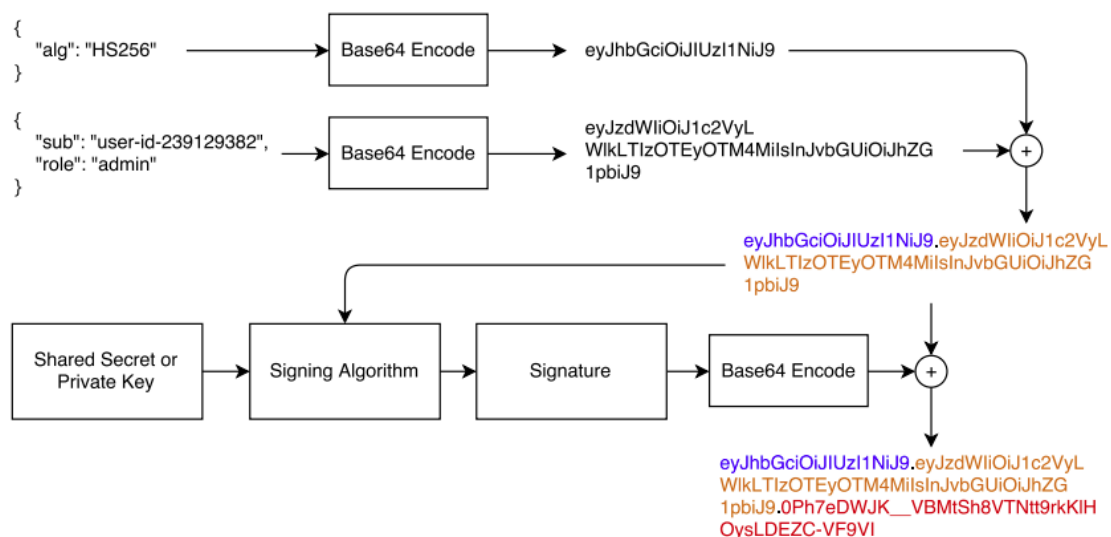
JWT được cấu tạo từ 3 thành phần: **header.payload.signature**

**header** => encode dưới dạng base64

**payload** (thông tin data chứa trong token) => encode dưới dạng base64

**Signature** (chữ ký) => được tạo nên từ thuật toán mã hóa + (header + payload) + key (mật khẩu) dưới dạng base64

Như vậy, signature gồm 3 thành phần: **thuật toán mã hóa** , **data của (header + payload)** và **key** (có thể là mật khẩu/ hoặc sử dụng private/public key)



có 2 cách phổ biến

a shared secret (hs256) (cách dùng trong khóa học này)

a private/public key pair (rs256)

## #55. Tạo Key (Part 1)

Source code video này (đã bao gồm code của video #56) (#55):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

**Signature** (chữ ký) => được tạo nên từ thuật toán mã hóa + (header + payload) + key (mật khẩu) dưới dạng base64

### 1. JWK (Json Web Key)

Key được tạo ra với một mục đích, cho dù JWT bị lộ ra, nếu không có “key”, bạn không thể giải mã được token

Có 2 hình thức tạo key phổ biến: một là dùng mật khẩu (hash) và hai là dùng public/private key

#### Với mô hình public/private key pair: (thuật toán RSA)

Ví dụ bạn dùng JWT để giao tiếp giữa Server A (chủ) và Server B (khách)

Server A sẽ giữ private key

Server B chỉ được phép giữ public key. Và dùng public key này để giải mã (decrypt) token được A cung cấp. Với public key, B không thể “tự tạo được token”

#### Với mô hình dùng mật khẩu (hash):

Ví dụ bạn dùng JWT để giao tiếp giữa Server A (chủ) và Server B (khách)

Server A sẽ giữ mật khẩu (key)

Server B cũng cần biết mật khẩu trên (shared key) . Và key này để giải mã (decrypt) token được A cung cấp. B có thể “tự tạo được token”

## 2. Tạo Key

**Bước 1:** cấu hình .env variable

(sử dụng yaml)

security:

authentication:

jwt:

# This token must be encoded using Base64 and be at least 256 bits long  
(you can type ``openssl rand -base64 64`` on your command line to generate a 512 bits one)

base64-secret:

```
ODcxODJlOGI4YTdhZWNIYzE4NmRhNzQwYzQ4ZmIxOTQzODMzZDcxOWQ2OGVINTk5MjY0MTcxNTc3YTcyZmIzN2I0YmI4NmI1ZDU2NDU4YzUxZjI0MDMwM2E5MDI4YWU5YTlyMWQ5ZDNhODBhMmNhZjFkM2VIZWRhN2Y4ZWVlOTk=
```

//update file **application.properties**

Lưu ý: ae copy paste thì cần để giá trị trên cùng 1 dòng (làm giống hệt như video), còn trong tài liệu là nó tự động xuống dòng. File application.properties tính cả dấu cách, dấu enter xuống dòng đấy

**hoidanit.jwt.base64-**

**secret=**qoAEABDke07+AVLepXB4aCMtsT0wMAqR5x2VFyldsnx6e75YQkJH2UcZKTjEyoNgG71SBCXfq5N6NVZxWOfsHQ==

**hoidanit.jwt.token-validity-in-seconds=86400**

**Bước 2:** Sử dụng .env

@Value("\${hoidanit.jwt.base64-secret}")

private String jwtKey;

<https://stackoverflow.com/a/30528430>

**Bước 3:** Tạo key

//update file security config:

```
public static final MacAlgorithm JWT_ALGORITHM = MacAlgorithm.HS512;

private SecretKey getSecretKey() {
    byte[] keyBytes = Base64.from(jwtKey).decode();
    return new SecretKeySpec(keyBytes, 0, keyBytes.length, JWT_ALGORITHM.getName());
}

@Bean
public JwtEncoder jwtEncoder() {
    return new NimbusJwtEncoder(new ImmutableSecret<>(getSecretKey()));
}
```

## #56. Tạo Key (Part 2)

Source code video này (đã bao gồm code của video #56) (#55):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tham khảo:

[https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/web/rest/AuthenticatedController.java?ref\\_type=heads#L81](https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/web/rest/AuthenticatedController.java?ref_type=heads#L81)

### Bước 4: Tạo Access Token (JWT)

```
public String createToken(Authentication authentication) {

    Instant now = Instant.now();
    Instant validity = now.plus(this.tokenValidityInSeconds, ChronoUnit.SECONDS);

    // @formatter:off
    JwtClaimsSet claims = JwtClaimsSet.builder()
        .issuedAt(now)
        .expiresAt(validity)
        .subject(authentication.getName())
        .claim(AUTHORITIES_KEY, authorities)
        .build();

    JwsHeader jwsHeader = JwsHeader.with(JWT_ALGORITHM).build();
    return this.jwtEncoder.encode(JwtEncoderParameters.from(jwsHeader,
        claims)).getTokenValue();
}
```

## #57. Bảo Vệ Endpoint (API) với JWT

Source code video này (#57):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tài liệu:

<https://docs.spring.io/spring-security/reference/servlet/oauth2/resource-server/jwt.html#oauth2resourceserver-jwt-sansboot>

### Bước 1: khai báo sử dụng JWT

@Bean

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
    http  
        .authorizeHttpRequests(authorize -> authorize  
            .anyRequest().authenticated()  
        )  
        .oauth2ResourceServer((oauth2) -> oauth2.jwt(Customizer.withDefaults()));  
    return http.build();  
}
```

Mục đích của bước 1, là sử dụng **BearerTokenAuthenticationFilter** (filter sẽ tự động extract - lấy token từ header của request gửi lên server - amazing :v )

Sau khi khai báo như trên, **nếu chạy project sẽ bị "toang"**, vì sử dụng JWT, ngoài encoder (mã hóa), chúng ta cần decoder (giải mã)

### Bước 2: cấu hình decoder

<https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/config/annotation/web/configurers/oauth2/server/resource/OAuth2ResourceServerConfigurer.html>

Có 3 cách:

supply a Jwk Set Uri via OAuth2ResourceServerConfigurer.JwtConfigurer.jwkSetUri  
or

supply a JwtDecoder instance via  
OAuth2ResourceServerConfigurer.JwtConfigurer.decoder,

Or expose a JwtDecoder bean



//@FunctionalInterface

@Bean

```
public JwtDecoder jwtDecoder() {  
    NimbusJwtDecoder jwtDecoder = NimbusJwtDecoder.withSecretKey(  
        getSecretKey()).macAlgorithm(SecurityUtil.JWT_ALGORITHM).build();  
    return token -> {  
        try {  
            return jwtDecoder.decode(token);  
        } catch (Exception e) {  
            System.out.println(">>> JWT error: " + e.getMessage());  
            throw e;  
        }  
    };  
}
```

### **Bước 3:** test API

Lưu ý: cần allow /login permitAll

Cần login để lấy access\_token

Mỗi khi gọi API, cần truyền lên token này

## #58. Xử lý JWT Exception

Source code video này (#58):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### 1. Xử lý exception

```
//default exception
.exceptionHandling(
    exceptions -> exceptions
        .authenticationEntryPoint(new BearerTokenAuthenticationEntryPoint()) //401
        .accessDeniedHandler(new BearerTokenAccessDeniedHandler()) //403
```

//custom

<https://devlach.com/blog/java/spring-security-custom-authentication-failure>

//class CustomAuthenticationEntryPoint.java (download trong source code đính kèm)

//hỗ trợ tiếng việt

```
response.setContentType("application/json;charset=UTF-8");
```

//không làm như này nhé:

```
server.servlet.encoding.charset=UTF-8
```

```
server.servlet.encoding.force=true
```

## 2. Lưu data vào Security Context (req.user)

**//khi login**

```
Authentication authentication =  
authenticationManagerBuilder.getObject().authenticate(authenticationToken);  
SecurityContextHolder.getContext().setAuthentication(authentication);
```

**//khi decode thành công**

```
@Bean  
public JwtAuthenticationConverter jwtAuthenticationConverter() {  
    JwtGrantedAuthoritiesConverter grantedAuthoritiesConverter = new  
JwtGrantedAuthoritiesConverter();  
    grantedAuthoritiesConverter.setAuthorityPrefix("");  
    grantedAuthoritiesConverter.setAuthoritiesClaimName(AUTHORITIES_KEY);  
  
    JwtAuthenticationConverter jwtAuthenticationConverter = new  
JwtAuthenticationConverter();  
  
    jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter(grantedAuthoritiesConv  
erter);  
    return jwtAuthenticationConverter;  
}
```

## #59. Tổng kết về JWT

### 1. Quá trình tạo ra/mã hóa Token (encode)

**Khi nào tạo ra token:** người dùng login thành công, server cần trả ra token gửi cho client.

Tất cả các request sau này gửi lên server truy cập API, client cần truyền lên token này để định danh (xác thực)

**Quá trình encode gồm các bước:**

JWT = **header.payload.signature**

Phần **header** bao gồm thông tin về thuật toán mã hóa => khai báo thuật toán

Phần **payload** là data truyền theo token (được dùng để định danh người dùng)

Phần **signature** là chữ ký, được tạo ra bằng cách :  
JWK (key) + thuật toán + ký vào (header + payload)

=> với Java Spring, cần cấu hình **JwtEncoder**, khai báo **Key + thuật toán**

@Bean

```
public JwtEncoder jwtEncoder() { ... }
```

//todo: giải thích code

### 2. Quá trình giải mã Token (decode)

Client muốn truy cập & sử dụng API, cần truyền lên Token (đã có từ bước login) tại header của Request, thông thường là dạng **Bearer you-token-here**

**Bước 1:** client gửi kèm JWT token ở header request

**Bước 2:** Tại phía Server của Spring (sau khi đã cấu hình oauth2-resource-server), sẽ kích hoạt filter **BearerTokenAuthenticationFilter**

Filter này sẽ “tự động tách” Bear Token (bạn không cần phải làm thủ công, thư viện đã làm sẵn rồi)

Token sẽ được xử lý tiếp, thông qua:

**JwtDecoder** : giải mã token (check tính hợp lệ của token)

Check như thế nào :

JWT = **header.payload.signature**

Decoder sẽ tách header, payload, đồng thời lấy Key + thuật toán để băm ngược ra **signature**

Nếu 2 signature là trùng nhau => token hợp lệ.

Quá trình này tương tự việc so sánh mật khẩu khi login (băm mật khẩu thành hash để so sánh với database, nếu trùng nhau, tức là nhập đúng thông tin)

**JwtAuthenticationConverter** : convert data chứa trong token, lưu vào Spring Security Context để reuse

## **Chapter 8: Phân tích dự án thực hành**

*Phân tích và thiết kế dự án thực hành của khóa học*

### **#60. Giới thiệu dự án thực hành**

Source code video này (#60):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

//fix bug trường hợp không truyền lên JWT token

//handle lỗi khi không truyền lên token

Lưu ý về trường hợp token hết hạn (set cho vài nghìn năm :v)

### **1. Về dự án thực hành**

Ý tưởng:

<https://hoidanit.vn/khoa-hoc/nestjs-voi-typescript-mongodb-sieu-de-64686ec6fb456bbb90663dd6.html>

**Đề tài :** website việc làm (ý tưởng itviec.com)

Các đối tượng tham gia:

- Users (có phân quyền chi tiết): HR, Admin, normal user
- Permission/Role
- Company (công ty đăng tin tuyển dụng)
- Job/Resume
- Subscriber (đăng ký nhận tin tuyển dụng qua email)

## **#61. Phân tích Model cho Databases**

File excel sử dụng trong video này (#61):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

//phân tích chi tiết model

//phân tích mối quan hệ

// Lưu ý:

Về cách tư duy và thiết kế database như nào, ràng buộc mối quan hệ ra làm sao sẽ không được phân tích trong khóa học này, vì nội dung đấy đã được làm trong khóa học Spring MVC rồi (chapter 9 và chapter 10)

<https://hoidanit.vn/khoa-hoc/java-spring-mvc-ultimate-for-beginners-65ce0b770c05f4450fbd86ac.html>

## #62. Setup dự án thực hành Frontend

### 1. Cài đặt Node.js

Nodejs là môi trường chạy code javascript, tương tự JDK để chạy code Java

**Version node.js mình sử dụng là 16.20.0:**

<https://nodejs.org/download/release/v16.20.0/>

Cài chính xác version để hạn chế tối đa lỗi có thể xảy ra. (nvm)

Kiểm tra đã cài đặt thành công với câu lệnh: **node -v**

### 2. Cài đặt dự án Frontend

Link gitlab: <https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/03-react-vite-jobhunter>

**Bước 1:** clone dự án

**Bước 2:** checkout nhánh code cors với câu lệnh  
**git checkout cors**

**Bước 3:** cài đặt các thư viện cần thiết với câu lệnh  
**npm i**

**Bước 4:** update file .env.development và .env.production  
//update url backend

**Bước 5:** chạy dự án  
//chạy tại dev  
//chạy tại production



## #63. CORS là gì ?

### 1. Minh họa lỗi

Mở source code frontend, gõ câu lệnh sau:

**Bước 1:** chuyển code qua nhánh cors  
**git checkout cors**

//check code mới nhất: **git pull**

**Bước 2:** cài đặt thư viện cần thiết  
**npm i**

**Bước 3:** chạy dự án ở chế độ dev  
//lưu ý: backend chạy tại localhost:8080

**npm run dev**

Chờ 1 xíu để browser lưu cache, truy cập: <http://localhost:3000/>

**F12, check tab network**

(failed)net::ERR\_CONNECTION\_REFUSED => chưa chạy backend

Nếu không thấy lỗi **CORS error** cũng chẳng sao :v

### 2. CORS là gì

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

Cross-Origin Resource Sharing (CORS) là câu chuyện Server cho phép tên miền nào có quyền load tài nguyên của nó

**Server:** localhost:8080

**Client:** localhost:3000 => khác nhau ở port nên bị chặn CORS

Như khóa Spring MVC, dùng SSR, cả server và client đều là localhost:8080 nên không bị

### Tại sao lại cần CORS ?

Tất cả lý do cơ chế cookies của browser. Cookies lưu thông tin người dùng (dùng định danh người dùng, tương tự session\_id)

Với code frontend, nếu gọi API từ client tới server, browser sẽ tự động đính kèm cookies ứng với server đấy.

### Thử tưởng tượng case sau:

**Bạn dùng browser**, bạn sử dụng facebook.com

=> browser sẽ lưu cookies của bạn ứng với server facebook.com

(mục đích mỗi lần bạn refresher, hoặc vài tuần sau bạn quay lại facebook mà không cần đăng nhập/định danh bạn là ai)

Bạn chat với 1 người lạ trên facebook, gửi cho bạn 1 đường link **hot-girl.xyz**

Bạn nhấn vào link, và truy cập hot-girl.xyz

**Nếu không có cơ chế CORS**, và trang web hot-girl.xyz là do hacker dựng lên, nó sẽ code như sau:

**Từ hot-girl.xyz gọi API của facebook.com**, ví dụ: **facebook.com/getPhoneNumber**

**=> lấy thông tin số điện thoại sau lưng bạn :v**

**API trên sẽ gọi được (mà không cần đăng nhập vào facebook)**, vì mặc định, browser sẽ gửi kèm cookies của facebook.com (khi bạn gọi tới server đấy)

=> để tránh tình trạng trên, mặc định, các browser hiện đại sẽ có cơ chế CORS (default).

Có nghĩa là, dùng browser, không thể từ hot-girl.xyz gọi tới facebook.com (vì khác tên miền)

Tương tự với case của chúng ta: từ localhost:3000 không thể gọi tới localhost:8000

## #64. Cách fix CORS

Cơ chế CORS sinh ra để giúp lướt web an toàn hơn, và cung cấp thêm sức mạnh cho server, khi có thể quyết định “tên miền nào” có thể truy cập nguồn tài nguyên của nó

=> muốn fix CORS, nên fix ở Server (cách thực tế sử dụng)

Mô hình áp dụng: client - server thông qua Restful API, đang bị CORS, có các cách fix sau:

Cách hiệu quả nhất (được dùng trong khóa học), là cấu hình CORS tại backend, như vậy frontend chỉ dùng và không cần phải sửa đổi gì

Tuy nhiên, khi đi làm trong thực tế, nếu server (API) không do bạn viết, hoặc đối tác cung cấp không thể sửa đổi thì làm thế nào ?

**Cách 1:** disabled security của browser

<https://stackoverflow.com/questions/3102819/disable-same-origin-policy-in-chrome>

=> không khuyến khích dùng cách này

**Cách 2:** không sử dụng browser trực tiếp, mà dùng server - server  
(khuyến khích dùng cách này, vì control 100%)

CORS chỉ xảy ra giữa browser và server. Còn server gọi server sẽ không có CORS

Nếu client => gọi Server A bị CORS (server này bạn ko kiểm soát), thì:

Client => gọi Server B (server này bạn kiểm soát, sẽ setup CORS) => gọi Server A

**Cách 3:** dùng proxy cho code frontend (again, bạn cần tìm hiểu công cụ sử dụng có dùng proxy hay không)

Client => gọi proxy => gọi server A (thứ bạn không kiểm soát). Như vậy, nó sẽ gần giống như cách 2 ở trên.

Bạn nào dùng nginx (reverse proxy sẽ hiểu :v )

## #65. Spring và CORS

Source code video này (#65):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tài liệu:

<https://docs.spring.io/spring-security/reference/servlet/integrations/cors.html>

Phạm vi cấu hình CORS: local hoặc global

<https://docs.spring.io/spring-framework/reference/web/webmvc-cors.html#mvc-cors-intro>

Tham khảo Jhipster:

Cấu hình filter:

[https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/config/WebConfigurer.java?ref\\_type=heads#L84](https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/config/WebConfigurer.java?ref_type=heads#L84)

Cấu hình env:

[https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/resources/config/application-dev.yml?ref\\_type=heads#L68](https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/resources/config/application-dev.yml?ref_type=heads#L68)

### 1.Cấu hình CORS

Mặc định, spring security hỗ trợ sẵn `org.springframework.web.filter.CorsFilter`

### **Tạo bean:**

@Configuration

public class CorsConfig {

    @Bean

    public CorsConfigurationSource corsConfigurationSource() {

        CorsConfiguration configuration = new CorsConfiguration();

        configuration.setAllowedOrigins(Arrays.asList("http://example.com"));

        configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS")); // Allowed methods

        configuration.setAllowedHeaders(Arrays.asList("Authorization", "Content-Type", "Accept"));

        configuration.setAllowCredentials(true);

        configuration.setMaxAge(3600L);

        // How long the response from a pre-flight request can be cached by clients

        UrlBasedCorsConfigurationSource source = new

        UrlBasedCorsConfigurationSource();

        source.registerCorsConfiguration("/\*\*", configuration); // Apply this configuration to all paths

        return source;

    }

}

## Chapter 9: Modules Company

*Thực hiện CRUD với model Company*

### #66. Model Company

Source code video này (#66):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

#### 1. Tạo model Company

**companies**

```
id      long
name    String
description  String => Text
address  String
logo     String
createdAt  Date
updatedAt  Date
createdBy  String
updatedBy  String
```

```
@Column(columnDefinition = "MEDIUMTEXT")
```

```
private String description;
```

<https://stackoverflow.com/a/13932834>

```
//Java Datetime
```

<https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/->

[/blob/master/src/main/java/com/mycompany/myapp/domain/JobHistory.java?ref\\_type=heads#L25](/blob/master/src/main/java/com/mycompany/myapp/domain/JobHistory.java?ref_type=heads#L25)

<https://docs.oracle.com/javase/8/docs/api/java/time/Instant.html>

```
private Instant createdAt;
```

**Sử dụng lombok**

<https://projectlombok.org/#>

<https://projectlombok.org/setup/gradle>

```
plugins {
```

```
    id("io.freefair.lombok") version "8.6"
```

```
}
```

```
//format date global
```

[https://gitlab.com/public-starter-projects/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/config/DateTimeFormatConfiguration.java?ref\\_type=heads](https://gitlab.com/public-starter-projects/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/config/DateTimeFormatConfiguration.java?ref_type=heads)

American vs. European Date Formats:

American (MM/dd/yyyy): 12/31/2023 (December 31, 2023)

European (dd/MM/yyyy): 31/12/2023 (31 December 2023)

24-Hour Clock (HH:mm:ss): "23:45:00" (11:45 PM)

12-Hour Clock (hh:mm:ss a): "11:45:00 PM"

ISO 8601 Standard:

Date and Time Combined (yyyy-MM-dd'T'HH:mm:ss'Z'): "2023-12-31T15:45:30Z"

UTC Offset: "2023-12-31T10:15:30+01:00" (Time in UTC+1 zone)

Named Time Zones: "2023-12-31T09:15:30 Europe/Paris"

## #67. Bài Tập Create Company

Fix bug video #66: bổ sung @Entity cho class Company

**Yêu cầu:**

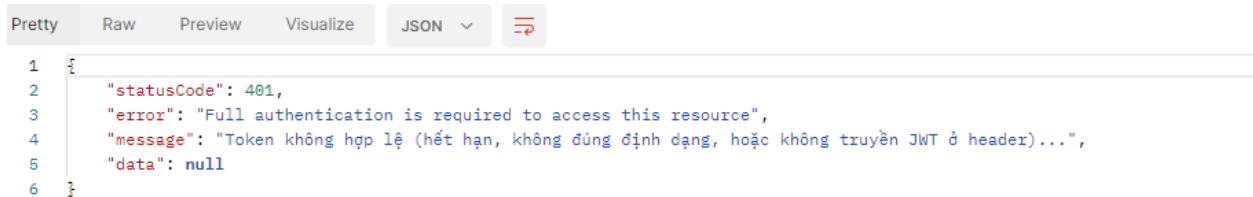
**POST** <http://localhost:8080/companies>

Input: thông tin công ty cần tạo (dưới dạng JSON)

Output: thông tin công ty đã tạo

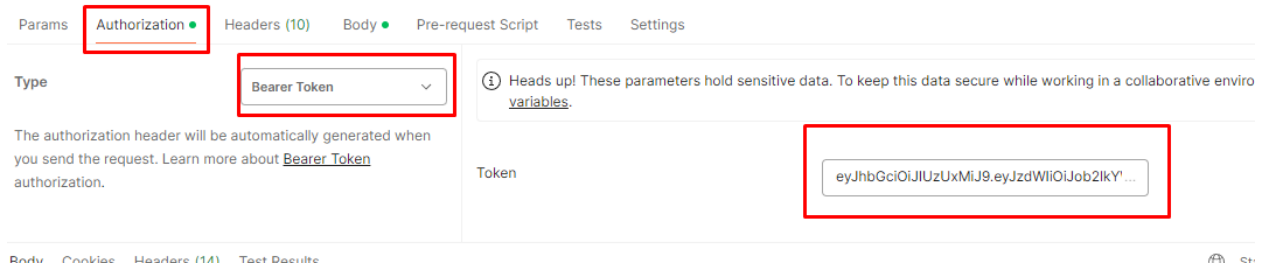
Cần login để lấy JWT trước khi gửi request => truyền Bearer Token ở header request

Đây là lỗi không truyền bearer Token



```
1 {
2   "statusCode": 401,
3   "error": "Full authentication is required to access this resource",
4   "message": "Token không hợp lệ (hết hạn, không đúng định dạng, hoặc không truyền JWT ở header)...",
5   "data": null
6 }
```

Truyền ở đây (lấy token tại api login)



**Ví dụ về body request: (JSON)**

```
{
  "name": "hoidanit's company",
  "description": "my first company",
  "address": "hanoi",
  "logo": "abc.png"
}
```

**Request nếu gọi thành công:**



Trả ra thông tin của Company được tạo, status = 201 (tương tự như trong ảnh)

The screenshot shows a Postman interface for a REST client. At the top, the method is set to **POST** and the URL is `http://localhost:8080/companies`. Below the URL bar, the **Body** tab is selected, and the **raw** radio button is chosen. The content type is set to **JSON**. The request body is a JSON object:

```
1 {
2   "name": "hoidanit's company",
3   "description": "my first company",
4   "address": "hanoi",
5   "logo": "abc.png"
6 }
```

Below the request body, the **Test Results** tab is selected, showing the response status: **Status: 201 Created**. The response time is 93 ms and the size is 668 B. The response body is a JSON object:

```
1 {
2   "statusCode": 201,
3   "error": null,
4   "message": "CALL API SUCCESS",
5   "data": {
6     "id": 1,
7     "name": "hoidanit's company",
8     "description": "my first company",
9     "address": "hanoi",
10    "logo": "abc.png",
11    "createdAt": null,
12    "updatedAt": null,
13    "createdBy": null,
14    "updatedBy": null
15  }
16 }
```

### **#68. Chữa Bài Tập Create Company**

Source code video này (#68):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### **#69. Before Save Entity với JPA (Part 1)**

Tài liệu: <https://stackoverflow.com/questions/45064420/business-logic-before-to-save-an-entity-in-spring-jpa>

[https://docs.jboss.org/hibernate/orm/6.5/userguide/html\\_single/Hibernate\\_User\\_Guide.html#events-jpa-callbacks](https://docs.jboss.org/hibernate/orm/6.5/userguide/html_single/Hibernate_User_Guide.html#events-jpa-callbacks)

<https://www.baeldung.com/jpa-entity-lifecycle-events>

<https://www.timestamp-converter.com/>

## **#70. Before Save Entity với JPA (Part 2)**

Source code video này (#70):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### **1. Format timezone**

<https://www.digitalocean.com/community/tutorials/java-8-date-localdate-localdatetime-instant>

<https://stackoverflow.com/questions/71854271/java-instant-datetime-how-to-get-the-datetime-with-timezone-offset-only-without>

### **2. Spring Security Context**

[https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/security/SecurityUtils.java?ref\\_type=heads](https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/security/SecurityUtils.java?ref_type=heads)

## #71. Bài Tập Get/Update/Delete Company

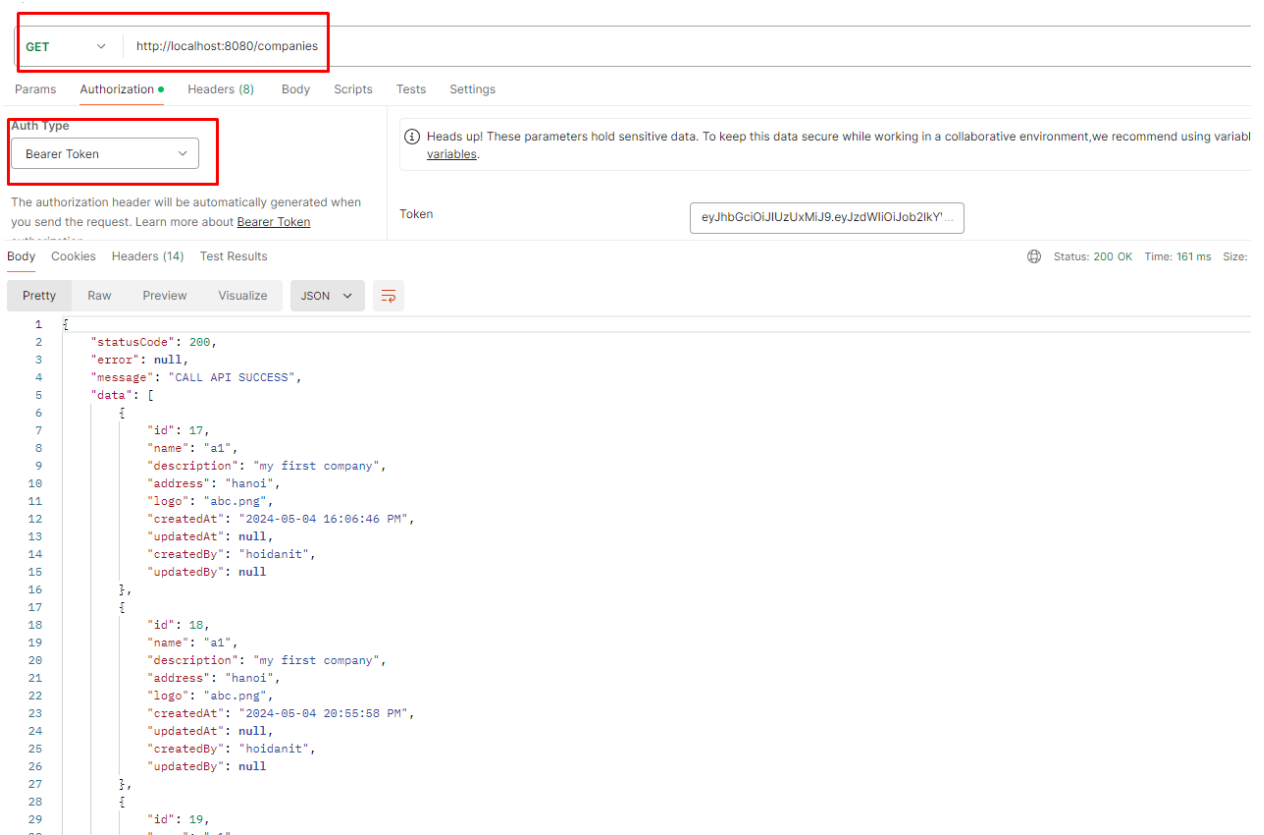
**Yêu cầu 1: Lấy danh sách công ty**

**GET** <http://localhost:8080/companies>

**Cần truyền lên JWT ở header, và không truyền data ở body**

**Gợi ý: ResponseEntity<List<Company>>**

**Output: lấy tất cả công ty, chưa cần phân trang (pagination)**



## Yêu cầu 2: Cập nhật công ty

PUT <http://localhost:8080/companies>

Cần truyền JWT ở header, và truyền data update ở body (có id)

Gợi ý: ResponseEntity<Company> trả ra response và sử dụng @PreUpdate tại domain để tự động cập nhật **updatedAt** và **updatedBy**

The screenshot displays a REST client interface with a PUT request to `http://localhost:8080/companies`. The request body is in raw JSON format, containing an update for a company with ID 18. The response is also in raw JSON format, showing a successful status (200) and the updated company details, including timestamps and the user who performed the update.

**Request:**

```
1 {
2   "id": 18,
3   "name": "a1",
4   "description": "my first company",
5   "address": "hanoi1",
6   "logo": "abc.png"
7 }
```

**Response:**

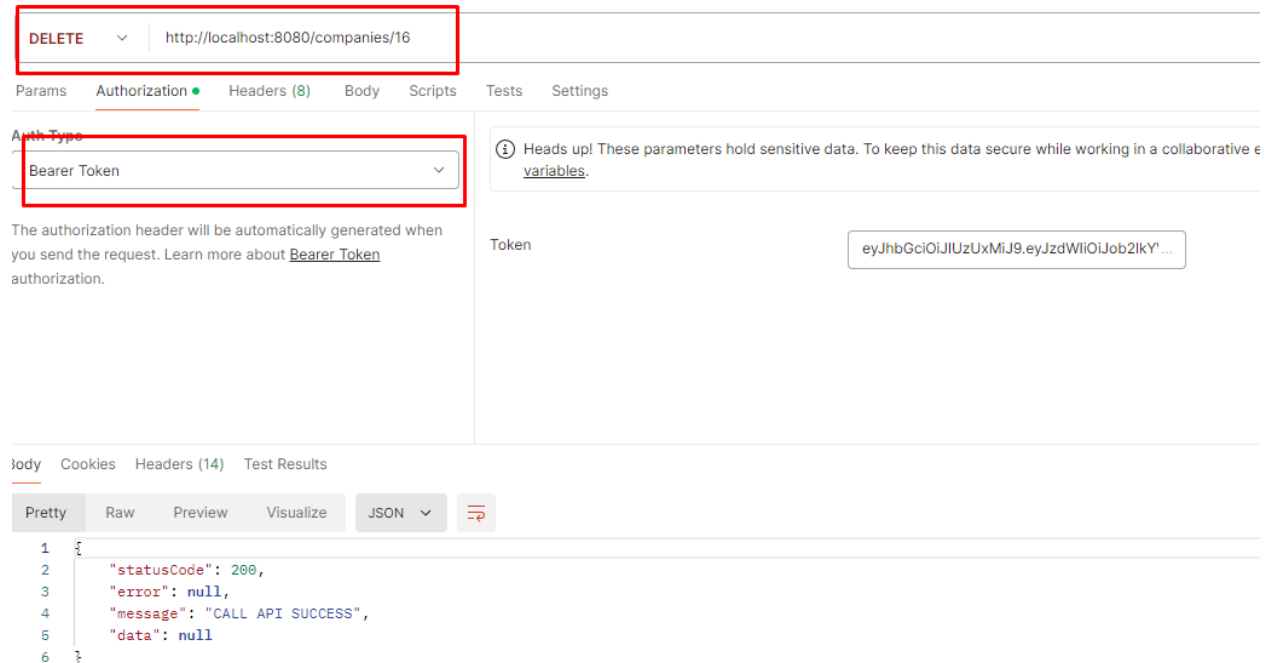
```
1 {
2   "statusCode": 200,
3   "error": null,
4   "message": "CALL API SUCCESS",
5   "data": {
6     "id": 18,
7     "name": "a1",
8     "description": "my first company",
9     "address": "hanoi1",
10    "logo": "abc.png",
11    "createdAt": "2024-05-04 20:55:58 PM",
12    "updatedAt": "2024-05-05T02:51:22.963643900Z",
13    "createdBy": "hoidanit",
14    "updatedBy": "hoidanit@gmail.com"
15  }
16 }
```

### Yêu cầu 3: Xóa công ty

**DELETE** <http://localhost:8080/companies/id-company>

Cần truyền JWT ở header, và truyền id của công ty cần xóa trên đường link url

Gợi ý: ResponseEntity<Void> để không cần trả ra phản hồi



### #72. Chữa Bài Tập Get/ Update/Delete Company

Source code video này (#72):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### #73. Query với Pagination

Source code video này (#73):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Vấn đề : Lấy danh sách công ty

GET <http://localhost:8080/companies>

Cần truyền lên JWT ở header, và không truyền data ở body

=> Nếu data nhiều, thời gian fetch sẽ lâu, và người dùng cũng không xem hết data

=> không hiệu quả

Ví dụ: <https://shopee.vn/Th%E1%BB%9Di-Trang-Nam-cat.11035567>

#### 1. Cách xử lý phân trang

Về nguyên tắc: sử dụng **offset** và **limit** của SQL

<https://www.sqltutorial.org/sql-limit/>

=> frontend sẽ truyền lên **page** và **limit** => backend sẽ tự tính offset

Với Repository:

**JpaRepository** kế thừa **PagingAndSortingRepository**

**Page<T> findAll(Pageable pageable);**

=> cần truyền lên đối tượng Pageable là sẽ được phân trang

#### 2. Cách thực hiện

**Bước 1:** Frontend truyền lên 2 tham số:

**current** //page hiện tại

**pageSize** //số lượng bản ghi muốn lấy

@RequestParam("**current**") Optional<String> currentOptional,

@RequestParam("**pageSize**") Optional<String> pageSizeOptional

## **Bước 2:** Tạo đối tượng Pageable

//page tính từ 0

Pageable pageable = PageRequest.of(page - 1, limit);

//truyền sang repository => done

## **Bước 3:** format output

//cần tạo DTO để return data cho client

### **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: array-data //kết quả query  
}
```



## #74. Giới thiệu về Specification

//review phần filter của company và fix bug pageNumber + 1

### 1. Giới thiệu về Specification

<https://docs.spring.io/spring-data/jpa/reference/jpa/specifications.html>

**Bước 1:** repository kế thừa **JpaSpecificationExecutor<T>**

Nhờ việc kế thừa này, chúng ta có method findAll (truyền thêm Specification)  
`List<T> findAll(Specification<T> spec);`

`Page<T> findAll(Specification<T> spec, Pageable pageable);`

**Bước 2:** build specification (cú pháp của predicate và CriteriaBuilder)

<https://docs.oracle.com/javaee%2F7%2Fapi%2F%2F/javax/persistence/criteria/CriteriaBuilder.html>

```
public class CustomerSpecs {  
  
    public static Specification<Customer> isLongTermCustomer() {  
        return (root, query, builder) -> { //đây là lamda, ghi đề hàm toPredicate  
            LocalDate date = LocalDate.now().minusYears(2);  
            return builder.lessThan(root.get(Customer_.createdAt), date);  
        };  
    }  
  
}
```

**Bước 3:** Áp dụng tại service/controller, ví dụ:

```
List<Customer> customers = customerRepository.findAll(isLongTermCustomer());
```

## 2. Vấn đề tồn đọng

Cần viết thủ công từng predicate (Specification) ứng với từng tiêu chí filter

Ví dụ: filter theo name, address, price ...

Không tái sử dụng được logic giữa các Entity có điểm tương đồng, ví dụ entity User và Employee đều có chung thuộc tính name => vẫn viết 2 specification

### **Giải pháp 1: sử dụng generic để tái sử dụng**

<https://stackoverflow.com/questions/71904939/jpa-specification-generic-type-class>

<https://stackoverflow.com/questions/61241257/how-to-use-single-jpa-specification-class-and-methods-for-multiple-entities>

Ưu điểm: có thể tái sử dụng code, và bạn control 100% (từ a tới z)

Nhược điểm: viết = cộm (có nghĩa là bạn tự code)

### **Giải pháp 2: sử dụng thư viện**

Ví dụ:

<https://github.com/querydsl/querydsl>

<https://docs.spring.io/spring-data/jpa/reference/repositories/core-extensions.html>

Ưu điểm: có thể tối ưu hóa query cho bạn (out-of-the-box)

Nhược điểm: bạn cần tốn thời gian để học cú pháp của thư viện đấy

**Giải pháp đề ra: sử dụng "code thuần" hay thư viện không quan trọng**

### **Điều quan trọng nhất là luyện khả năng tư duy logic**

Bạn không cần phải học thuộc cú pháp (vì không làm nhiều, sao nhớ được. Đồng thời, chắc gì khi đi làm, công ty bạn join, nó sẽ làm như cách bạn biết :v )

## #75. Query với Filter (Part 1)

//source code video này sẽ được gộp vào #76

### 1. Cài đặt thư viện

<https://mvnrepository.com/artifact/com.turkraft.springfilter/jpa/3.1.7>

<https://github.com/turkraft/springfilter>

// <https://mvnrepository.com/artifact/com.turkraft.springfilter/jpa/3.1.7>  
**implementation("com.turkraft.springfilter:jpa:3.1.7")**

### 2. Sử dụng thư viện (lưu ý: chưa làm pagination và sorting)

<https://github.com/turkraft/springfilter?tab=readme-ov-file#jpa-integration>

**Bước 1:** cập nhật repository để có thể sử dụng Specification

@Repository

public interface UserRepository extends

JpaRepository<User, Long>,

**JpaSpecificationExecutor**<User>

{

}

**Bước 2:** update controller

@GetMapping(value = "/search")

Page<Entity> search(@Filter **Specification**<Entity> spec, Pageable page) {

return repository.findAll(spec, page);

}

**Bước 3:** test với frontend

//todo

## **#76. Query với Filter (Part 2)**

Source code video này (#76):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

//sử dụng pagination và sorting

<https://www.baeldung.com/spring-data-sorting>

Tương tự Spring REST project:

<https://docs.spring.io/spring-data/rest/reference/paging-and-sorting.html>

đối với Spring, hỗ trợ out-of-the-box đối tượng Pageable, nếu truyền lên url:

page

size

sort

Ví dụ: page=2&size=2&sort=price,asc

//cấu hình pagination:

<https://stackoverflow.com/questions/39884860/how-to-configure-spring-boot-pagination-starting-from-page-1-not-0>

<https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html#application-properties.data.spring.data.web.pageable.default-page-size>

## #77. Customize Message với Annotation (Bonus)

Source code video này (#77):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tham khảo:

<https://stackoverflow.com/questions/12260037/how-to-create-custom-annotation-in-java>

<https://www.digitalocean.com/community/tutorials/java-annotations>

//java reflection: <https://www.oracle.com/technical-resources/articles/java/javareflection.html>

**Bước 1:** tạo ra annotation

<https://www.geeksforgeeks.org/java-retention-annotations/>

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ApiMessage {
    String value();
}
```

**Bước 2:** sử dụng annotation

```
@GetMapping("/users")
@ApiMessage("fetch all users")
public ResponseEntity<ResultPaginationDTO> getAllUser()
```

**Bước 3:** Lấy giá trị của annotation

//phụ thuộc vào nơi bạn lấy giá trị của annotation, sẽ có các cách lấy giá trị khác nhau  
<https://stackoverflow.com/a/47922587>

```
//MethodParameter returnType
ApiMessage message = returnType.getMethodAnnotation(ApiMessage.class);
```

## **#78. Versioning API**

Source code video này (#78):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

**Yêu cầu:**

Update api, bắt đầu với tiền tố: /api/v1/

Ví dụ: <https://developers.facebook.com/tools/explorer/>

Tham khảo:

Cách làm thủ công (đánh version): <https://stackoverflow.com/a/39066141>

Thư viện: <https://github.com/lkqm/spring-api-versioning>

## **Chapter 10: Modules User**

*Thực hiện CRUD với model User*

### **#79. Update User Model**

Source code video này (#79):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

//cập nhật các thông tin sau

```
id      long
name    String
email   String
password String
age     int
gender  String // MALE/FEMALE
address String
refreshTokenString
createdAt Date
updatedAt Date
createdBy String
updatedBy String
```

### **1. Giới thiệu về Enum**

<https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html>

**//sử dụng enum với Entity**

<https://stackoverflow.com/questions/67825729/using-enums-in-a-spring-entity>

<https://dev.to/noelopez/spring-rest-working-with-enums-ma>

**//enum so với constant**

Việc sử dụng enum sẽ ép kiểu dữ liệu chặt hơn so với constant => an toàn hơn cho dự án của bạn

<https://stackoverflow.com/questions/11575376/why-use-enums-instead-of-constants-which-is-better-in-terms-of-software-design>

## #80. Bài tập CRUD User

Lưu ý: update tất cả endpoint (api) trong postman với tiền tố /api/v1

### 1. Trước khi bắt đầu

```
//AuthController.java
@RestController
@RequestMapping("/api/v1")
public class AuthController {

//SecurityConfiguration.java
.requestMatchers("/", "/api/v1/login").permitAll()
```

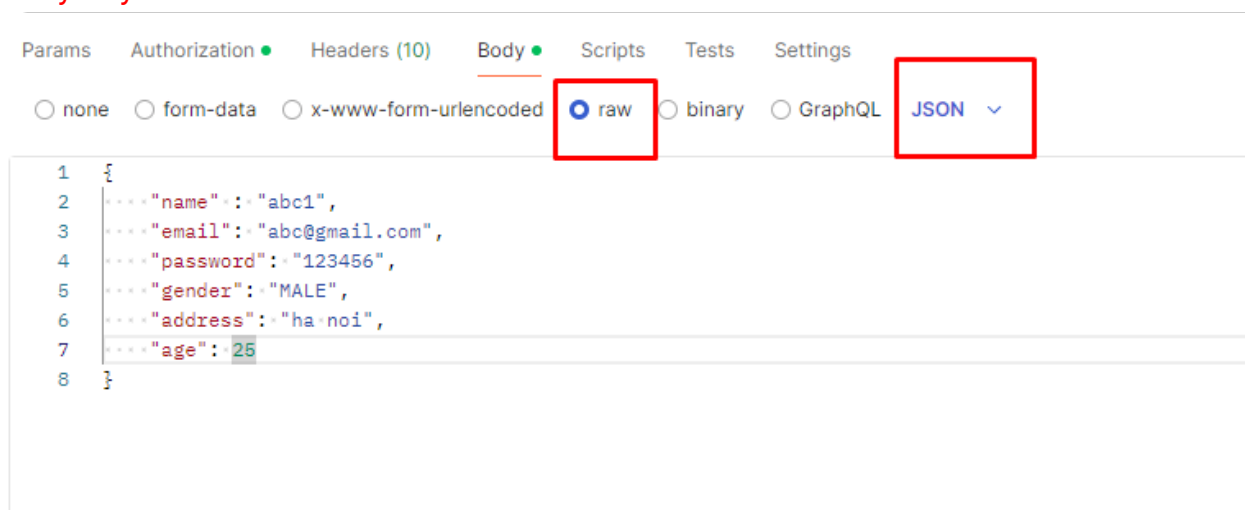
### 2. Yêu cầu cần làm

Yêu cầu 1: tạo mới user

**POST** <http://localhost:8080/api/v1/users>

Lưu ý: cần truyền lên JWT ở header

Body truyền raw data





Ví dụ về data gửi lên server:

```
{
  "name": "abc1",
  "email": "abc@gmail.com",
  "password": "123456",
  "gender": "MALE",
  "address": "ha noi",
  "age": 25
}
```

**Trước khi trả về phản hồi, check các điều kiện sau:**

- Cần kiểm tra email đã tồn tại trong hệ thống chưa. Nếu đã tồn tại, thông báo lỗi  
Gợi ý: sử dụng **existsByEmail** của repository
- Chỉ trả ra thông tin cần thiết, không trả ra tất cả thông tin của user (ví dụ như password)  
Gợi ý: tạo thêm **DTO class** để mapping object

The screenshot displays a REST client interface with a POST request to `http://localhost:8080/api/v1/users`. The request body is a JSON object representing a user. The response is also in JSON format, showing a successful status (201), no error, a message, and the created user data along with a timestamp.

```
POST http://localhost:8080/api/v1/users

{
  "name": "abc1",
  "email": "abc@gmail.com2",
  "password": "123456",
  "gender": "MALE",
  "address": "ha noi",
  "age": 25
}
```

```
{
  "statusCode": 201,
  "error": null,
  "message": "Create a new user",
  "data": {
    "id": 33,
    "name": "abc1",
    "email": "abc@gmail.com2",
    "gender": "MALE",
    "address": "ha noi",
    "age": 25,
    "createdAt": "2024-05-10T01:52:18.844109900Z"
  }
}
```

## Yêu cầu 2: update user

**PUT** <http://localhost:8080/api/v1/users>

Lưu ý: cần truyền lên JWT ở header

Body truyền raw data (truyền thêm id)

```
{
  "id": 33,
  "name": "eric",
  "gender": "FEMALE",
  "age": 20,
  "address": "hcm"
}
```

//ném ra lỗi nếu như id truyền lên không tồn tại

//gợi ý: làm tương tự như CREATE user, cũng viết DTO để trả về kết quả

The screenshot displays a REST client interface with a PUT request to `http://localhost:8080/api/v1/users`. The request body is a JSON object with fields: `id` (33), `name` ("eric"), `gender` ("FEMALE"), `age` (20), and `address` ("hcm"). The response body, shown in the 'Body' tab, is a JSON object with fields: `statusCode` (200), `error` (null), `message` ("Update a user"), `data` (an object with the same user details as the request), and `updatedAt` ("2024-05-10T02:37:29.110444200Z").

```
PUT http://localhost:8080/api/v1/users

{
  "id": 33,
  "name": "eric",
  "gender": "FEMALE",
  "age": 20,
  "address": "hcm"
}
```

```
{
  "statusCode": 200,
  "error": null,
  "message": "Update a user",
  "data": {
    "id": 33,
    "name": "eric",
    "gender": "FEMALE",
    "address": "hcm",
    "age": 20,
    "updatedAt": "2024-05-10T02:37:29.110444200Z"
  }
}
```

### Yêu cầu 3: fetch user by id

**GET** <http://localhost:8080/api/v1/users/id-user>

Lưu ý: cần truyền lên JWT ở header

Body không truyền data, truyền kèm id ở header request

//ném ra lỗi nếu như id truyền lên không tồn tại

//trả ra thông tin user (không có kèm theo password)

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

- Method:** GET
- URL:** <http://localhost:8080/api/v1/users/31>
- Auth Type:** Bearer Token
- Token:** eyJhb...
- Body:** Pretty, Raw, Preview, Visualize (JSON selected)
- Response Body (JSON):**

```
1 {
2   "statusCode": 200,
3   "error": null,
4   "message": "fetch user by id",
5   "data": {
6     "id": 31,
7     "email": "abc@gmail.com",
8     "name": "abc1",
9     "gender": "MALE",
10    "address": "ha noi",
11    "age": 25,
12    "updatedAt": null,
13    "createdAt": null
14  }
15 }
```

#### Yêu cầu 4: fetch all user với filter và pagination

**GET** <http://localhost:8080/api/v1/users>

Lưu ý: cần truyền lên JWT ở header

Header truyền thêm:

**page:** trang muốn lấy

**size:** số lượng phần tử tối đa lấy tại một trang

**sort:** sắp xếp

**filter:** điều kiện query

Ví dụ:

[http://localhost:8080/api/v1/users?page=1&size=20&sort=name,asc&filter=email ~  
'@gmail.com' and name ~ 'it'](http://localhost:8080/api/v1/users?page=1&size=20&sort=name,asc&filter=email~'@gmail.com'andname~'it')

#### Yêu cầu : data trả về không chứa thông tin mật khẩu, refresh token của người dùng

//gợi ý: mapping data trước khi trả về res (sử dụng DTO)

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

- Method:** GET
- URL:** <http://localhost:8080/api/v1/users?page=1&size=20&sort=name,asc>
- Query Params:**

Key	Value	Description
page	1	
size	20	
sort	name,asc	
filter	email ~ '@gmail.com' and name ~ 'it'	
- Response Body (JSON):**

```
1 {
2   "statusCode": 200,
3   "error": null,
4   "message": "fetch all users",
5   "data": {
6     "meta": {
7       "page": 1,
8       "pageSize": 20,
9       "pages": 1,
10      "total": 13
11    },
12    "result": [
13      {
14        "id": 22,
15        "email": "abc@gmail.com",
16        "name": "abc1",
17        "gender": null,
18        "address": null,
19        "age": 0,
20        "updatedAt": null,
21        "createdAt": null
22      },
23      {
24        "id": 23,
25        "email": "abc@gmail.com",
26        "name": "abc1",
27        "gender": null,
```

### Yêu cầu 5: Xóa user

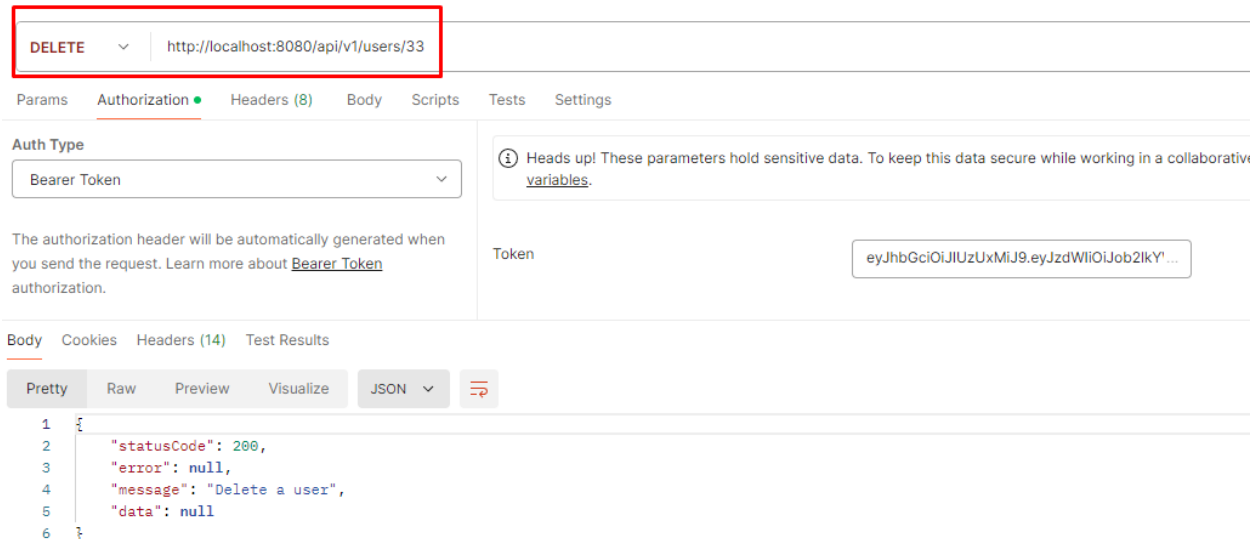
**DELETE** <http://localhost:8080/api/v1/users/id-user>

Lưu ý: cần truyền lên JWT ở header

Body không truyền data, truyền kèm id ở header request

// ném ra lỗi nếu như id truyền lên không tồn tại

// ResponseEntity<Void> để không cần trả ra data (data = null)



### Yêu cầu 6: xử lý exception

//handle exception not found (nếu url api không tồn tại)

//handle exception filter (nếu truyền không đúng thông tin của filter)

## #81. Chữa Bài tập CRUD User

Source code video này (#81):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

//handle exception not found (nếu url api không tồn tại)

## #82. API Login

Source code video này (#83):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### 1. Yêu cầu của api login

Nếu người dùng đăng nhập thành công, backend sẽ làm 2 việc:

- Trả về phản hồi cho api, bao gồm access\_token và thông tin của user, theo format sau:

```
{
  accessToken: "JSON WEB TOKEN",
  user: {
    email: "hoidanit@gmail.com",
    name: "Hỏi Dân IT",
    id: "25"
  }
}
```

- Ngoài ra, **refresh\_token** sẽ lưu vào cookies

### 2. Cách làm (thêm thông tin user cho api login)

Hiện tại, data user login thành công, lưu tại Authentication (**Principal**)

=> viết custom User hoặc call API để lấy thêm thông tin user

### Bước 1: update ResLoginDTO.java

//sử dụng inner class

[https://www.w3schools.com/java/java\\_inner\\_classes.asp](https://www.w3schools.com/java/java_inner_classes.asp)

```
@Setter
@Getter
public class ResLoginDTO {
    private String accessToken;
    private UserLogin user;

    @Getter
    @Setter
    @AllArgsConstructor
    static public class UserLogin {
        private String email;
        private String name;
    }
}
```

**Bước 2:** set giá trị của object với static

```
ResLoginDTO.UserLogin u = new ResLoginDTO.UserLogin("email", "name");
```

## #83. Set Cookies (Part 1)

//source code video này được gộp vào #84

### 1. Giới thiệu về cookies

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

Với browser, để lưu trữ data của user, có 3 cách hay dùng nhất:

**Local Storage** : thông tin được lưu trữ mãi mãi, không bao giờ mất

**Session Storage**: bạn đóng browser là thông tin lưu trữ sẽ clear

**Cookies**: chỉ mất khi và chỉ khi “bị hết hạn” (không liên quan gì tới việc đóng browser)

### // các thuộc tính thường dùng

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie>

Expires : thời gian hết hạn, định dạng Date

Max-Age: thời gian hết hạn, định dạng number (seconds) (độ ưu tiên cao hơn Expires)

HttpOnly : boolean (thường set = true để không cho client JS truy cập/sử dụng cookie)

### // cơ chế thường dùng trong mô hình stateless (liên quan tới cookies)

Người dùng sau khi login thành công, sẽ được server trả về:

**access\_token** : token để định danh người dùng (thời gian sống ngắn : 5 phút, 10 phút...)

**refresh\_token** : nếu access token bị hết hạn, sử dụng refresh token để renew (thời gian sống lâu hơn nhiều, thường là 1 ngày, 30 ngày ...)

=> **access token được lưu tại Local Storage** (frontend dễ dàng truy cập và sử dụng) .

Đặt thời gian sống ngắn để giảm thiểu rủi ro

**Refresh token được lưu tại cookies** với mục đích là Server sử dụng (vì cookie luôn được gửi kèm với mỗi lời gọi request) => lưu ở cookies sẽ an toàn hơn (do thời gian sống của token lâu hơn access token)



### //cơ chế hoạt động của cookies

Có bao giờ bạn thắc mắc tại sao Facebook, Google, Tiktok... 1 tuần, 1 tháng , 1 năm... bạn không cần đăng nhập lại (trên cùng 1 máy tính) mỗi lần sử dụng ?

#### **Câu trả lời là cookies.**

Trong lần đăng nhập đầu tiên thành công, Server (FB, Tiktok...) sẽ trả về thông tin của người dùng và lưu vào cookies.

Cookies này thường có thời hạn sống khá lâu (vài tháng tới cả năm)

=> sau vài tháng bạn vào lại, cookies sẽ đc sử dụng (refresh token) để đổi lấy access token mới => bạn không cần login lại :v

## 2. Luồng logic

//update thời gian của access và refresh token (chia thành 2 biến riêng lẻ)  
hoidanit.jwt.token-validity-in-seconds=8640000

//tạo refresh token (chứa user email)

Subject (sub) : đối tượng mà token này hướng tới (định danh user)

=> sử dụng email là ok

Claim (key: value): meta data => lưu gì cũng đc

```
user: {  
  email: "hoidanit@gmail.com",  
  name: "Hỏi Dân IT",  
  id: "25"  
}
```

//update user với refresh token

//set cookies to client (có thể test = postman)

## **#84. Set Cookies (Part 2)**

Source code video này (#84):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tài liệu:

<https://reflectoring.io/spring-boot-cookies/>

<https://docs.spring.io/spring-framework/docs/current/javadoc-api/org.springframework.http.ResponseCookie.html>

//update user với refresh token

//set cookies to client (có thể test = postman)

// các thuộc tính thường dùng

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie>

Path: url set cookie

Domain: nếu không set, thì chỉ chính same domain (không tính sub domain)

## #85. API Get Account (F5 - Refresh)

Source code video này (#85):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### 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 đấy 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": {
    "id": "...",
    "name": "...",
    "email": "...",
  }
}
```

## #86. Giải thích cơ chế JWT và Spring Security (Bonus)

Source code video này (#86):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Do sử dụng mô hình stateless (tách riêng frontend và backend), nên không thể sử dụng session, chúng ta sử dụng token để định danh người dùng.

Để truy cập endpoint (APIs) tại backend, đối với mỗi lời gọi (request), frontend cần truyền thêm Token (gọi là access token). Token này sẽ được gán ở header mỗi request (Bearer Token)

Access Token này có thời gian sống ngắn (thực tế được dùng từ 30 phút tới 1 ngày) để đảm bảo an toàn, đồng thời, được viết dưới dạng JWT (JSON Web Token)

### 1. API Login

POST <http://localhost:8080/api/v1/auth/login>

Mục đích: client (frontend) **lấy được access token** để truy cập API backend, đồng thời có được refresh token (sử dụng khi access token hết hạn)

### Cơ chế xử lý tại backend:

Xử lý đăng nhập với Spring Security, và nếu đăng nhập thành công, tạo ra access token, bao gồm 2 thành phần:

**subject** (sub): đối tượng của Token (đảm bảo tính duy nhất), sử dụng email là hợp lý  
**claim** : thông tin miêu tả về token

### Ví dụ về access token (chứa user)

eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJob2lkYW5pdEBnbWFpbC5jb20iLCJleHAiOiE3MjQwNTkzNT  
EslmlhdCI6MTcxNTQxOTM1MSwidXNlciI6eyJpZCI6MSwiZW1haWwiOiJob2lkYW5pdEBnbWFp  
bC5jb20iLCJuYW1lIjoisOG7j2kgRMOibiBJVCJ9fQ.0V3Ss4AWnHf8f6HtSEWKLchHSJAmHW0ef3  
WUtDj2fn0vGYEjbyqIO2nor0lq06DTcAOAthT9BBvQ3gD9M0jThw

### Ví dụ về access token (chứa user và permission)

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJob2lkYW5pdEBnbWFpbC5jb20iLCJwZXJtaXNzaW9uIjpblJJPTEVfVVNFUI9DUkVBVEUiLCJST0xFOX1VTRVJfVVBEQVRFI0slmV4cCI6MjU3OTQ3NjQ0MCwiaWF0IjoxNzE1NDc2NDQwLCJ1c2Vyljp7ImkljoxLCJlbWFpbCI6ImhvaWRhbmI0QGdtYWlsLmNvbSIslm5hbWUiOiJl4buPaSBew6JuIEUln19.Wa0Cc0zx9AgGpWYpZ3Ucjk5PyENCNDAsdjpoZnOY06jRnk4hpFf9cFyZfrvWWKZI839JX7_gSZ56TTXMrV9dFw
```

### 2. Với tất cả các API khác (cần truyền access token ở header)

Khi gửi request gồm access token, hàm decode sẽ được chạy đầu tiên để giải mã token

```
//SecurityConfiguration.java  
return jwtDecoder.decode(token);
```

Kế tiếp, sẽ tiến hành  **nạp "authority"** (quyền hạn) vào security context. Làm được điều này, do có:

```
@Bean  
public JwtAuthenticationConverter jwtAuthenticationConverter() {  
    grantedAuthoritiesConverter.setAuthoritiesClaimName("permission");  
  
    return jwtAuthenticationConverter;  
}
```

```
//tiền hành debug  
// nạp quyền hạn (authority) từ jwt vào security context  
JwtGrantedAuthoritiesConverter
```

```
//filter  
BearerTokenAuthenticationFilter
```

## #87. API Refresh Token (Part 1)

//source code video này gộp vào #88

### 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

**Response: (trả ra giống hệt như khi login)**

```
{
  "statusCode": 200,
  "message": "Get User by refresh token",
  "data": {
    "accessToken": "...",
    "user": {
      "id": "...",
      "name": "...",
      "email": "...",
    }
  }
}
```

### 3. Các bước xử lý

Tài liệu:

<https://reflectoring.io/spring-boot-cookies/>

[https://docs.spring.io/spring-framework/docs/current/javadoc-api/org.springframework/http/ResponseCookie.html](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org.springframework.http.ResponseCookie.html)

**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)

## **#88. API Refresh Token (Part 2)**

Source code video này (#88):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

//todo

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)



## #89. Bài tập API Logout

Source code video này (#89):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Yêu cầu: Tạo endpoint

### POST api/v1/auth/logout

Truyền lên JWT ở header

**Response :**

```
{
  "statusCode": 200,
  "message": "Logout User",
  "data": null
}
```

Xử lý ở backend:

- Nhờ có spring security, lấy ra email của người dùng
- Update refresh\_token === null (empty) (tìm theo email)
- Remove refresh\_token ở cookies (remove cookies)  
<https://reflectoring.io/spring-boot-cookies/>
- Trả về phản hồi cho client : ResponseEntity<Void>

## #90. Nguyên tắc check code frontend

Source code video này (#90):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### 1. Rule tuân theo

Code frontend chỉ lỗi, khi và chỉ khi bạn không chạy lên được dự án frontend.

Tất cả các trường hợp còn lại, nếu đã chạy thành công dự án frontend, tuy nhiên, check giao diện không được kết quả như video hướng dẫn, cần check response của api, tức là lỗi của backend java.

**Lỗi ở đây, là backend java đang viết API trả ra định dạng “không phải là thứ frontend mong muốn”.**

API chạy được, nhưng cần đảm bảo trả ra đúng định dạng (format) mà frontend mong muốn

### 2. Danh sách API cần check trong module này

**Fix bug 1 :** <https://stackoverflow.com/questions/12583638/when-is-the-jsonproperty-property-used-and-what-is-it-used-for>

```
public class ResLoginDTO {  
    @JsonProperty("access_token")  
    private String accessToken;
```

```
//todo: fix bug còn lại
```

## 1.API login

POST <http://localhost:8080/api/v1/auth/login>

```
{
  "statusCode": 200,
  "error": null,
  "message": "CALL API SUCCESS",
  "data": {
    "access_token": "jwt token",
    "user": {
      "id": 1,
      "email": "hoidanit@gmail.com",
      "name": "Hỏi Dân IT"
    }
  }
}
```

## 2. API Get Account

GET <http://localhost:8080/api/v1/auth/account>

```
{
  "statusCode": 200,
  "error": null,
  "message": "fetch account",
  "data": {
    "user": {
      "id": 1,
      "email": "hoidanit@gmail.com",
      "name": "Hỏi Dân IT"
    }
  }
}
```

### 3. API get refresh token

GET <http://localhost:8080/api/v1/auth/refresh>

```
{
  "statusCode": 200,
  "error": null,
  "message": "Get User by refresh token",
  "data": {
    "access_token": "jwt token",
    "user": {
      "id": 1,
      "email": "hoidanit@gmail.com",
      "name": "Hỏi Dân IT"
    }
  }
}
```

### 4. Các API về CRUD

=> check video bài tập CRUD

## #91. Test giao diện frontend

Version node.js mình sử dụng là **16.20.0**:

<https://nodejs.org/download/release/v16.20.0/>

Cài chính xác version để hạn chế tối đa lỗi có thể xảy ra. (nvm)

Kiểm tra đã cài đặt thành công với câu lệnh: **node -v**

Link gitlab dự án frontend: <https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/03-react-vite-jobhunter>

### Bước 1:

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

**git checkout test-1**

Lưu ý về check file env, đảm bảo backend chạy chính xác port

### Bước 2: cài đặt thư viện cần thiết

**npm i**

### Bước 3: build dự án

**npm run build**

### Bước 4: chạy dự án tại chế độ production

**npm run preview**

//test dự án = tab ẩn danh (hoặc cần clear hết cookies/localStorage ứng với port 3000)

### Các tính năng test:

- Login thành công, trả về token  
Access token lưu tại Local Storage  
Refresh token lưu tại Cookies
- CRUD user User/Company <http://localhost:3000/admin>
- Demo tính năng khi access token (test = cách setup tại backend) ( **x-no-retry** )  
//bổ sung header để không bị cors

## **Chapter 11: Modules Job/Resume**

*Bài tập thực hành tạo module Job/Resume: CRUD job/resume, kết hợp việc sử dụng upload/download file với Java Spring*

### **#92. Code Refactoring**

Source code video này (#92):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tổ chức lại thư mục **domain** và **service**

Chia thành request/response

### **#93. Model Relationship (Associations)**

Source code video này (#93):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tài liệu:

[https://docs.jboss.org/hibernate/orm/6.4/introduction/html\\_single/Hibernate\\_Introduction.html#associations](https://docs.jboss.org/hibernate/orm/6.4/introduction/html_single/Hibernate_Introduction.html#associations)

<https://www.baeldung.com/jpa-hibernate-associations>

#### **1. Phân loại**

Gồm 3 loại quan hệ chính:

- **One to One** : quan hệ 1 - 1
- **One to Many** hay **Many to One** : quan hệ 1 - nhiều ( 1 - N )
- **Many to many** : quan hệ nhiều - nhiều ( N - N )

Chỉ chiều hướng mối quan hệ (ràng buộc cha/con), chúng ta có 2 keywords:

**unidirectional** : quan hệ 1 chiều

**bidirectional**: quan hệ 2 chiều

**Quan hệ giữa User - Company:**

1 user chỉ thuộc 1 công ty ( 1 - 1 )

1 công ty bao gồm nhiều user ( 1 - N )

=> mối quan hệ giữa User và Company là N - 1

**2. Áp dụng cho Model**

1 user chỉ thuộc 1 công ty => table User sẽ lưu **company\_id**

N user cùng thuộc 1 công ty => ManyToOne

**@ManyToOne**

@JoinColumn(name = "company\_id")

private Company company;

1 công ty có nhiều user => OneToMany

**@OneToMany**( mappedBy = "company")

List<User> users;

**3. Fetch Lazy or Eager ?**

<https://www.baeldung.com/hibernate-lazy-eager-loading>

<https://stackoverflow.com/questions/2990799/difference-between-fetchtype-lazy-and-eager-in-java-persistence-api>

## #94. Bài Tập Update Model User/Company

Source code video này (#94):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

**Yêu cầu 1:** Tạo user, kèm thêm thông tin của công ty

**POST** <http://localhost:8080/api/v1/users>

```
{
  "name" : "Hỏi Dân IT",
  "email": "hoidanit@gmail.com2",
  "password": "123456",
  "gender": "MALE",
  "address": "ha noi",
  "age": 25,
  "company": {
    "id": 17
  }
}
```

//logic xử lý tại backend, nếu truyền thêm company, cần check xem company đấy có tồn tại hay không (check theo id)

- Nếu công ty tồn tại => set data company cho user
- Nếu công ty không tồn tại => set data = null

**Yêu cầu 2:** thêm thông tin company khi thao tác với user

//khi update user

//khi fetch user by id

//khi fetch all user

**Yêu cầu 3:** khi xóa công ty, cần xóa user thuộc công ty, rồi mới xóa công ty

//trong thực tế, có thể thêm 1 fields là active = true/false (chứ không xóa hết data)

**Yêu cầu 4:** khi fetch công ty, sẽ ignore (bỏ đi thuộc tính user)

//tránh lặp vô hạn



## #95. Model Job

Source code video này (#95):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### 1. Tạo model Job

#### Table jobs:

id	long
name	String
location	String
salary	double
quantity	int
level	String FRESHER/JUNIOR... => sử dụng enum (tương tự <b>gender</b> )
description	String longText
startDate	Date
endDate	Date
isActive	boolean
createdAt	Date
updatedAt	Date
createdBy	String
updatedBy	String
<b>company_id</b>	long (mapping relationship)
<b>skills</b>	Array (mapping relationship)

//level : INTERN, FRESHER, MIDDLE, SENIOR

#### Table skills:

id	long
name	String
createdAt	Date
updatedAt	Date
createdBy	String
updatedBy	String

## 2. Định nghĩa relationship

### Quan hệ giữa Company - Job:

1 công ty có thể đăng tuyển nhiều jobs ( 1 - N )

1 job chỉ thuộc 1 công ty ( 1 - 1 )

=> mối quan hệ giữa Company và Job là 1 - N

//company

@**OneToMany**(mappedBy = "company", fetch = FetchType.LAZY)

@JsonIgnore

List<Job> jobs;

//job

@**ManyToOne**

@JoinColumn(name = "company\_id")

private Company company;

### Quan hệ giữa Job - Skill:

1 job có nhiều kỹ năng (skill) ( 1 - N )

1 skill có thể thuộc nhiều job khác nhau ( 1 - N )

=> mối quan hệ giữa Job và Skill là N-N

//job

@**ManyToMany**(fetch = FetchType.LAZY)

@JsonIgnore

@JoinTable(name = "job\_skill", joinColumns = @JoinColumn(name = "job\_id"),

inverseJoinColumns = @JoinColumn(name = "skill\_id"))

private List<Skill> skills;

//skill

@**ManyToMany**(fetch = FetchType.LAZY, mappedBy = "skills")

@JsonIgnore

private List<Job> jobs;

## #96. Bài Tập CRUD Job

```
//rename isActive => active (table job)
@ManyToMany(fetch = FetchType.LAZY)
@JsonIgnoreProperties(value = { "jobs" })
private List<Skill> skills;
```

### 1. CRUD Skills

**//Create a skill** : tên skill là **duy nhất và không được để trống**

Body: truyền name

```
{ "name" : "JAVA" }
```

**//update a skill** : tên skill là **duy nhất và không được để trống**

Body: truyền name và id

Với tính năng xóa skill, các bạn làm sau cùng nhé.

**Làm thêm/sửa/get skills (bước 1)**

**Làm thêm/sửa/xóa/get job (bước 2)**

Rồi hãy làm xóa skill (vì khi đấy đã đủ data và có ràng buộc về quan hệ)

**//delete a skill** (không cần làm, vì khi delete job (owner side) sẽ tự động xóa trong job\_skill

//bạn nào muốn làm, thì cần test 2 trường hợp:

Trường hợp 1: chưa có dữ liệu của job

Trường hợp 2: đã có dữ liệu của job (tức là table job\_skill có dữ liệu của skill muốn xóa)

=> video tiếp theo mình có hướng dẫn làm chức năng xóa skill này :v

**//fetch all skills** with pagination

## 2. CRUD Job

//create a job

Tạo DTO để get list skills và trả ra phản hồi tương ứng

//cần check skills trước khi lưu

//data mẫu

```
{
  "name": "Dev Java",
  "location": "Hà Nội",
  "salary": "10000000",
  "quantity": 10,
  "level": "JUNIOR",
  "description": "BLA BLA",
  "startDate": "2024-05-04T13:55:58.454607Z",
  "endDate": "2024-05-04T13:55:58.454607Z",
  "active": true,
  "skills": [
    {
      "id": 500
    },
    {
      "id": 3
    }
  ]
}
```

//update a job

//delete a job

//fetch all job with pagination

//get a job by id

## #97. Chữa Bài tập CRUD Job

Source code video này (#97):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Với tính năng xóa skill, các bạn làm sau cùng nhé.

**Làm thêm/sửa/get skills (bước 1)**

**Làm thêm/sửa/xóa/get job (bước 2)**

Rồi hãy làm xóa skill (vì khi đấy đã đủ data và có ràng buộc về quan hệ)

## #98. Về Upload File

Tham khảo:

<https://spring.io/guides/gs/uploading-files>

### 1.Có bao nhiêu cách để upload file (Storage)

- Sử dụng dịch vụ : AWS S3
- Lưu vào database (Blob)
- **Lưu file vào server**

### 2.Các khái niệm hay dùng:

#### MultipartFile

<https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/multipart/MultipartFile.html>

//tượng trưng cho file gửi từ client lên server

//thay vì gửi text, chúng ta sử dụng binary

#### Path

<https://docs.oracle.com/javase/8/docs/api/java/nio/file/Path.html>

//tượng trưng cho địa chỉ lưu trữ file trên máy tính của bạn

## #99. Read File From Path

Source code video này (#99):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### 1. Cấu hình path

Phân loại path: <https://learn.microsoft.com/en-us/dotnet/standard/io/file-path-formats#traditional-dos-paths>

Lưu ý: về phần cấu hình, lưu ý check dấu cách (empty space)

//cấu hình base path (set up env), sử dụng absolute path  
**application.properties**

Sử dụng câu lệnh để in ra path: **pwd**

//với windows, lưu ý chuyển \ thành /

D:\1.Udemy\Backend\4. Java Spring Restful\upload

=> được chuyển thành

D:/1.Udemy/Backend/4. Java Spring Restful/upload

**Ví dụ:**

hoidanit.upload-file.base-path=D:/1.Udemy/Backend/4. Java Spring Restful/upload/

### 2. Test access from localhost

Link download test file:

[https://drive.google.com/drive/folders/1XqqX\\_00eWaGlfvWLkHAwvxabdk2WovNw?usp=drive\\_link](https://drive.google.com/drive/folders/1XqqX_00eWaGlfvWLkHAwvxabdk2WovNw?usp=drive_link)

//cấu hình Spring

Đặt tên link là: **localhost:8080/storage/file\_name**

//lưu ý về security

//**StaticResourcesWebConfiguration.java**

@Configuration

```
public class StaticResourcesWebConfiguration  
    implements WebMvcConfigurer {
```

```
    @Value("${hoidanit.upload-file.base-path}")  
    private String basePath;
```

```
    @Override
```

```
    public void addResourceHandlers(ResourceHandlerRegistry registry) {  
        registry.addResourceHandler("/storage/**")  
            .addResourceLocations(basePath);  
    }
```

```
}
```

### 3. Security concern

- Chỉ cho phép đọc file tại 1 nơi cố định
- Chỉ cho phép đọc file type nhất định

## #100. Upload File (Part 1)

Tạo API **upload single file** :

**POST** <http://localhost:8080/api/v1/files>

**Bước 1:** Frontend gửi lên file và tên folder lưu trữ (**Sử dụng form-data thay vì json**)

**file:** file cần upload

**folder:** tên thư mục upload, phân theo từng tính năng

**Bước 2:** Backend lấy thông tin gửi lên

@PostMapping("/files")

public String upload(

    @RequestParam("file") MultipartFile file,

    @RequestParam("folder") String folder

)

Tham khảo:

<https://spring.io/guides/gs/uploading-files>

<https://www.bezkoder.com/spring-boot-file-upload/>

**Bước 3:** create a folder if not exists

<https://stackoverflow.com/a/53252034>

```
public void createUploadFolder(String folder) throws URISyntaxException {
    URI uri = new URI(folder);
    Path path = Paths.get(uri);
    File tmpDir = new File(path.toString());
    if (!tmpDir.isDirectory()) {
        try {
            Files.createDirectory(tmpDir.toPath());
            System.out.println(">>> CREATE NEW DIRECTORY SUCCESSFUL, PATH = " + folder);
        } catch (IOException e) {
            e.printStackTrace();
        }
    } else {
        System.out.println(">>> SKIP MAKING DIRECTORY, ALREADY EXISTS");
    }
}
```



**Bước 4:** save file to folder

```
public void store(MultipartFile file, String folder) throws URISyntaxException,
IOException {
    // create unique filename
    String finalName = System.currentTimeMillis() + "-" + file.getOriginalFilename();

    URI uri = new URI(basePath + folder + "/" + finalName);
    Path path = Paths.get(uri);
    try (InputStream inputStream = file.getInputStream()) {
        Files.copy(inputStream, path,
            StandardCopyOption.REPLACE_EXISTING);
    }
}
```

**#101. Upload File (Part 2)**

Source code video này (#101):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

**Bước 5:** trả về phản hồi với file name

**fileName:** uploaded-file-name

## **Bước 6:** validate file upload

File is empty

File extensions

File size (max = 5MB)

```
String fileName = file.getOriginalFilename();
List<String> allowedExtensions = Arrays.asList("pdf", "jpg", "jpeg", "png", "doc", "docx");
List<String> allowedMimeTypes = Arrays.asList(
    "application/pdf",
    "image/jpeg",
    "image/png",
    "application/msword",
    "application/vnd.openxmlformats-officedocument.wordprocessingml.document"
);

// Validate extension
boolean isValidExtension = allowedExtensions.stream().anyMatch(ext ->
fileName.toLowerCase().endsWith("." + ext));
if (!isValidExtension) {
    return ResponseEntity.badRequest().body("Invalid file type based on extension.");
}

// Validate MIME type
String contentType = file.getContentType();
if (!allowedMimeTypes.contains(contentType)) {
    return ResponseEntity.badRequest().body("Invalid file type based on MIME type.");
}

// Check file size
long maxSize = 5 * 1024 * 1024; // 5 MB in bytes
if (file.getSize() > maxSize) {
    //todo
}
```

## **#102. Download a File (Bonus)**

Source code video này (#102):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tham khảo: <https://stackoverflow.com/questions/35680932/download-a-file-from-spring-boot-rest-service>

### **#103. Model Resume**

Source code video này (#103):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

```
id      long
email   String
url      String
status: enum// PENDING-REVIEWING-APPROVED-REJECTED
createdAt   Date
updatedAt   Date
createdBy    String
updatedBy    String
user_id
job_id
```

//

Xác định mối quan hệ:

User - resume : 1 - N

Job - resume : 1 - N

## #104. Bài Tập CRUD Resume

//create a resume

```
{
  "email" : "hoidanit@gmail.com",
  "url": "mycv.pdf",
  "status": "PENDING",
  "user": {
    "id": 1
  },
  "job": {
    "id": 4
  }
}
```

//update a resume (chỉ update status)

//delete a resume by id

//fetch a resume by id

//fetch all resume with pagination

### **#105. Chữa Bài tập CRUD Resume**

Source code video này (#105):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### **#106. Test giao diện Frontend (Part 1)**

Source code video này (#106):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

//fetch company by id

//allow public access

@PreAuthorize("permitAll()") ???

//check api first

//quy định thư mục upload

company: chứa logo công ty

resume: chứa cv ứng viên gửi lên

user: chứa avatar user

## #107. Test giao diện Frontend (Part 2)

Version node.js mình sử dụng là **16.20.0**:

<https://nodejs.org/download/release/v16.20.0/>

Cài chính xác version để hạn chế tối đa lỗi có thể xảy ra. (npm)

Kiểm tra đã cài đặt thành công với câu lệnh: **node -v**

Link gitlab dự án frontend: <https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/03-react-vite-jobhunter>

**git checkout test-2-final**

//drops jobs, skills, resumes trước khi test

đăng nhập với tài khoản admin: hoidanit@gmail.com

== giao diện admin

**Bước 1:** tạo 1 công ty

//lưu ý về upload file ở đây

=> nếu failed. cần check response của api upload file

**Bước 2:** tạo skills

//crud skills

**Bước 3:** tạo 1 job

//lưu ý, đang select company => đúng ra cần lấy thông tin của user => user tạo job cho cty của user

**Bước 4:** apply a job

**Bước 5:** review resume

update status

download the resume (chưa làm)

## Chapter 12: Modules Permission & Role

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

### #108. Model Permission & Roles

Source code video này (#108):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

#### permissions

id	long
name	String
apiPath	String
method	String
module	String
createdAt	Date
updatedAt	Date
createdBy	String
updatedBy	String

#### roles

id	long
name	String
description	String
active	boolean
createdAt	Date
createdBy	String
updatedAt	Date
updatedBy	String
permissions	Array

//xác định mối quan hệ giữa Permissions (quyền hạn) và Roles (vai trò)

1 Roles (Vai trò) có nhiều Permissions (quyền hạn)

1 quyền hạn có thể thuộc nhiều role

=> N Roles - N Permissions => cần join\_table (tương tự jobs-skills)



## #109. Bài tập CRUD Permissions & Roles

//Lưu ý: cần truyền Bearer Token ở header mỗi request

Thứ tự test api:

- **Create a permission**

```
{
  "name": "create a user",
  "apiPath": "/users",
  "method": "POST",
  "module": "USERS"
}
```
- **Update a permission**  
//thêm id
- **Get permission with pagination**
- **Create a role** : truyền thêm permissions  
//check name  
// cần check permission truyền lên (tương tự như job và skill)

```
{
  "name": "admin1",
  "description": "vai trò của admin",
  "active": true,
  "permissions": [
    {"id": 1},
    {"id": 2}
  ]
}
```
- **Update a role** : check id, name  
//thêm id
- **Get role with pagination**
- **Delete a role** (auto delete permssion\_role)
- **Delete a permission** (cần xóa permission\_role, rồi mới xóa permission)

## **#110. Chữa Bài tập CRUD Permissions & Roles**

Source code video này (#110):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

## **#111. Update User với Permissions & Roles**

Source code video này (#111):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### **1. Cập nhật mối quan hệ giữa User và Role**

//user.java

**@ManyToMany**

@JoinColumn(name = "role\_id")

private Role role;

//role.java

**@OneToMany**(mappedBy = "role", fetch = FetchType.LAZY)

@JsonIgnore

List<User> users;

### **2. Update API Users**

//update CRUD user với Role

//update login thành công (refresh, get account), trả thêm role/permission cho frontend

### **3. Bổ sung**

//bỏ điều kiện update role by name (vì có thể update permissions, ko update name)

//viết api fetch resume by user: callFetchResumeByUser

<https://github.com/turkraft/springfilter/issues/363>

<https://github.com/turkraft/springfilter/issues/233#issuecomment-1590045915>

## **#112. Tạo Fake Data với SQL**

Source code video này (#112):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### **1. Bổ sung**

viết api register : /api/v1/auth/register

//fetch role by id

//update api update permission

### **2. Xóa tất cả table**

<https://stackoverflow.com/a/3476803>

```
SET FOREIGN_KEY_CHECKS = 0;
drop table if exists companies;
drop table if exists jobs;
drop table if exists job_skill;
drop table if exists permission_role;
drop table if exists permissions;
drop table if exists resumes;
drop table if exists roles;
drop table if exists skills;
drop table if exists users;
SET FOREIGN_KEY_CHECKS = 1;
```

### **3. Tạo fake data**

**Link download file sql (và logo):**

[https://drive.google.com/drive/folders/15TiCWqWurS30jY7vlnljFSOcKepOccGy?usp=drive\\_link](https://drive.google.com/drive/folders/15TiCWqWurS30jY7vlnljFSOcKepOccGy?usp=drive_link)

Thứ tự tạo fake data:

- Tạo users
- Tạo companies
- Tạo skills
- Tạo jobs
- Tạo job\_skill

## #113. Test Giao Diện Frontend (Part 1)

### 4. Test giao diện Frontend

Version node.js mình sử dụng là **16.20.0**:

<https://nodejs.org/download/release/v16.20.0/>

Cài chính xác version để hạn chế tối đa lỗi có thể xảy ra. (nvm)

Kiểm tra đã cài đặt thành công với câu lệnh: **node -v**

Link gitlab dự án frontend: <https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/03-react-vite-jobhunter>

**git checkout test-3-final**

Với file `.env.production` và `.env.development`, các bạn sửa tham số:  
`VITE_ACL_ENABLE=false`

//test crud permissions & roles

- Tạo permission
- Tạo role
- Gán permission vào role
- Tạo user với Role (đã tạo ở trên)

## #114. Tạo Sample Data (Part 1)

Tài liệu: <https://docs.spring.io/spring-boot/how-to/data-initialization.html>

//quản lý version database với **liquibase**

<https://www.baeldung.com/liquibase-refactor-schema-of-java-app>

//các cách tạo fake data với **faker**

<https://www.baeldung.com/java-faker>

//tạo câu lệnh query database

//viết code để tạo database (dựa vào life cycle)

## #115. Tạo Sample Data (Part 2)

Source code video này (#115):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tài liệu:

<https://stackoverflow.com/a/41644743>

<https://docs.spring.io/spring-boot/api/java/org/springframework/boot/CommandLineRunner.html>

Ý tưởng: sử dụng interface **CommandLineRunner**

### Bước 1:

//DatabaseInitializer.java

public class DatabaseInitializer implements CommandLineRunner

### Bước 2: init data

//coding (tải tại source code cung cấp)

Admin thì full quyền: admin@gmail.com

//tạo permissions

//tạo roles, gán full permission cho role

//tạo users, gán role cho user

### Bước 3: Xóa table

SET FOREIGN\_KEY\_CHECKS = 0;

drop table if exists permission\_role;

drop table if exists permissions;

drop table if exists resumes;

drop table if exists roles;

drop table if exists users;

SET FOREIGN\_KEY\_CHECKS = 1;

### Bước 4: Test data

## #116. Test Giao Diện Frontend (Part 2)

Source code video này (#116):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

//update security

<https://stackoverflow.com/a/74633151>

### 1. Test giao diện Frontend

Version node.js mình sử dụng là **16.20.0**:

<https://nodejs.org/download/release/v16.20.0/>

Cài chính xác version để hạn chế tối đa lỗi có thể xảy ra. (npm)

Kiểm tra đã cài đặt thành công với câu lệnh: **node -v**

Link gitlab dự án frontend: <https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/03-react-vite-jobhunter>

git checkout test-3-final

Với file .env.production và .env.development, các bạn sửa tham số:  
**VITE\_ACL\_ENABLE=true**

//bonus

tính năng search

update filter resume. HR chỉ xem được cv của cty HR

//

Test phân quyền

## #117. Cách xử lý phân quyền tại Frontend (Bonus)

Để có thể hide/show giao diện ứng với phân quyền (quyền hạn), **frontend sẽ cần biết Role (vai trò) và Permission (quyền hạn) của người dùng đăng nhập.**

=> khi login/refresh token/getAccount (F5), backend cần trả ra role và permission cho frontend

Tại giao diện frontend, thực chất là viết if/else để render giao diện On/off với tham số: **VITE\_ACL\_ENABLE=true/false**

Các keywords để xử lý phân quyền phía frontend:

- Lưu trữ quyền hạn mà backend trả về (role/permission) ứng với người dùng đăng nhập. Với source code cung cấp, data được lưu tại **redux**
- Khai báo list permission tại Frontend, **file permissions.ts** (hoặc viết api fetch tất cả permissions, lưu vào localStorage/IndexedDB)
- Viết component **access.tsx**

Component trên là component cha, bọc ngoài các component con “cần check quyền hạn”

```
<Access
  permission={ALL_PERMISSIONS.COMPANIES.UPDATE}
  hideChildren
  >
    <EditOutlined
      onClick={() => {
        setOpenModal(true);
        setDataInit(entity);
      }}
    />
  </Access >
```



## #118. Interceptor

Source code video này (#118):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tham khảo:

<https://stackoverflow.com/a/40291647>

<https://www.baeldung.com/spring-mvc-handlerinterceptor>

<https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html>

//get pattern

<https://stackoverflow.com/a/57242149>

### 1. Cách làm

Mặc định, với 1 lời gọi request từ client gửi lên:

**Request => Spring Security (Filter chain) => Controller => Service...**

Do chúng ta “**không sửa Spring Security**” => sẽ can thiệp vào controller

=> can thiệp vào request **sau khi đã qua Spring Security và trước khi gọi tới Controller**

=> Interceptor

Mô hình:

Request => Spring Security => **Interceptor** => Controller => Service...

### Ý tưởng:

- Mỗi lời gọi request đều kèm theo JWT (access token) => chúng ta biết được ai đang đăng nhập (email) => biết được user đấy có quyền hạn (permissions) gì
- Check target controller (url) và permission user có. Nếu tồn tại, cho request đi tiếp, còn ngược lại, ném ra exceptions

## **Bước 1: Khai báo interceptor**

//**PermissionInterceptor.java**

```
public class PermissionInterceptor implements HandlerInterceptor {
    @Override
    public boolean preHandle(
        HttpServletRequest request,
        HttpServletResponse response, Object handler)
        throws Exception {

        String path = (String) request.getAttribute(HandlerMapping.BEST_MATCHING_PATTERN_ATTRIBUTE);
        String requestURI = request.getRequestURI();
        String httpMethod = request.getMethod();
        System.out.println(">>> RUN preHandle");
        System.out.println(">>> path= " + path);
        System.out.println(">>> httpMethod= " + httpMethod);
        System.out.println(">>> requestURI= " + requestURI);

        return true;
    }
}
```

//Khai báo tại phần config **WebMvcConfigurer**

**PermissionInterceptorConfiguration.java**

```
@Configuration
public class PermissionInterceptorConfiguration implements WebMvcConfigurer {
    @Bean
    PermissionInterceptor getPermissionInterceptor() {
        return new PermissionInterceptor();
    }

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        String[] whiteList = {
            "/", "/api/v1/auth/**", "/storage/**",
            "/api/v1/companies", "/api/v1/jobs", "/api/v1/skills", "/api/v1/files"
        };
        registry.addInterceptor(getPermissionInterceptor())
            .excludePathPatterns(whiteList);
    }
}
```

## **Bước 2: Check Permissions**

Mô hình:

Request => Spring Security => **Interceptor** => Controller => Service...

Request gửi kèm JWT (access\_token) => Spring giải mã token, và lưu thông tin vào Security Context

=> trước khi tới Interceptor, chúng ta đã biết được email của user

=> query user theo email => lấy role => lấy permission của user

## **#119. Test Giao Diện Frontend (Part 3)**

Source code video này (#119):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### **1. Giải thích về Transactional**

@Transactional

### **2. Test giao diện frontend**

git checkout test-3-final

Với file .env.production và .env.development, các bạn sửa tham số:  
VITE\_ACL\_ENABLE=true

Ví dụ: ở frontend chưa chặn phân quyền:

<http://localhost:4173/admin/job/upsert?id=1>

=> xử lý tại backend (trả về phản hồi với mã lỗi)



## Chapter 13: Modules Subscribers

*Bài tập 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.*

### #120. Model Subscribers

Source code video này (#120):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

//Subscriber.java

id: long

name: String

email: String

@ManyToMany(fetch = FetchType.LAZY)

@JsonIgnoreProperties(value = { "subscribers" })

@JoinTable(name = "subscriber\_skill", joinColumns = @JoinColumn(name = "subscriber\_id"), inverseJoinColumns = @JoinColumn(name = "skill\_id"))

**private List<Skill> skills;**

//Skill.java

@ManyToMany(fetch = FetchType.LAZY, mappedBy = "skills")

@JsonIgnore

private List<Subscriber> subscribers;

## #121. Bài tập CRUD Subscribers

Lưu ý: truyền token ứng với tài khoản [admin@gmail.com](mailto:admin@gmail.com) (full quyền, và đã gán quyền crud với subscribers)

Viết 3 API (in đậm bên dưới):

CRUD Subscribers (làm tương tự như CRUD jobs, cần check các điều kiện cần thiết)

- **Create a subscriber**

POST <http://localhost:8080/api/v1/subscribers>

Check trùng email, check skills truyền lên (tương tự như job)

Data mẫu:

```
{
  "email": "admin@gmail.com",
  "name": "Eric",
  "skills": [
    {"id": 1 }, {"id": 2 }
  ]
}
```

- **Update a subscriber** (chỉ update skills)

PUT <http://localhost:8080/api/v1/subscribers>

check skills truyền lên (tương tự như job)

```
{
  "id": 1,
  "skills": [
    {"id": 1 }, {"id": 3 }
  ]
}
```

- Không cần làm API fetch data và xóa, vì không làm tại admin

**Update API Delete a Skills** => cần xóa subscriber\_skill

## #122. Chữa bài tập CRUD Subscribers

Source code video này (#122):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

## #123. Cấu hình Send Email với Spring

Source code video này (#123):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tham khảo:

<https://docs.spring.io/spring-framework/reference/integration/email.html>

<https://www.baeldung.com/spring-email>

### 1. Cài đặt thư viện

// <https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-mail>  
**implementation("org.springframework.boot:spring-boot-starter-mail")**

### 2. Tạo app password với Gmail

- Ngôn ngữ sử dụng là tiếng anh
- Tài khoản Gmail cần bật xác thực 2 lớp

<https://myaccount.google.com/apppasswords>

### 3. Cấu hình tham số môi trường

<https://docs.spring.io/spring-boot/appendix/application-properties/index.html#appendix.application-properties.mail>

<https://stackoverflow.com/questions/16594855/where-to-find-all-available-java-mail-properties>

spring.mail.host=smtp.gmail.com

spring.mail.port=587

**spring.mail.username**=your-email

**spring.mail.password**=your-gmail-app-password

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true

## **#124. Hello Word với Spring Email**

Source code video này (#124):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Mô hình gửi email:

Client gửi request lên server Java Spring -> gọi server Gmail -> gửi email tới client

### **1.Yêu cầu của video này**

Bạn cần hoàn thiện video #123, đã cấu hình email/password gửi email

### **2. Quá trình thực hiện**

//tạo service để reuse

<https://mailtrap.io/blog/spring-send-email/>

<https://stackoverflow.com/a/74750259>

//Interface JavaMailSender

<https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/mail/javamail/JavaMailSender.html>



## #125. Send Email với Template (Part 1)

Source code video này (#125):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Tham khảo:

[https://gitlab.com/public-starter-projects/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/service/MailService.java?ref\\_type=heads](https://gitlab.com/public-starter-projects/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/service/MailService.java?ref_type=heads)

### Lưu ý với hàm gửi email. Viết try/catch để bắt exception

Mục tiêu:

- Cấu hình template engine
- Gửi email với template ở trên

#### 1. Vấn đề đặt ra

- Gửi email với text đơn thuần thường có giao diện “khá xấu”. Chúng ta muốn có màu sắc, và với giao diện web (giao diện đọc email), chúng ta sẽ cần **CSS**
- Chúng ta muốn **tái sử dụng “format gửi email”** để gửi cùng lúc cho nhiều khách hàng khác nhau.

=> template engine ra đời

#### 2. Sử dụng Thymeleaf template

##### Bước 1: Cài đặt thư viện

Khi generate project, mình đã tích hợp sẵn thymeleaf cho spring

```
implementation("org.springframework.boot:spring-boot-starter-thymeleaf")  
implementation("org.thymeleaf.extras:thymeleaf-extras-springsecurity6")
```

//todo: minh họa cách làm với <https://start.spring.io/>

## **Bước 2: Cấu hình template**

//Gửi email với JavaMailSender

//gửi email với html

[https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/service/MailService.java?ref\\_type=heads#L59](https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/service/MailService.java?ref_type=heads#L59)

```
public void sendEmailSync(String to, String subject, String content, boolean isMultipart,
boolean isHtml) {
    // Prepare message using a Spring helper
    MimeMessage mimeTypeMessage = this.javaMailSender.createMimeMessage();
    try {
        MimeMessageHelper message = new MimeMessageHelper(mimeTypeMessage,
isMultipart, StandardCharsets.UTF_8.name());
        message.setTo(to);
        message.setSubject(subject);
        message.setText(content, isHtml);
        this.javaMailSender.send(mimeTypeMessage);
    } catch (MailException | MessagingException e) {
        System.out.println("ERROR SEND EMAIL: " + e);
    }
}
```

**//Gửi email với template**

[https://gitlab.com/public-starter-projects/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/service/MailService.java?ref\\_type=heads#L89](https://gitlab.com/public-starter-projects/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/service/MailService.java?ref_type=heads#L89)

```
public void sendEmailFromTemplateSync(String to, String subject, String
templateName) {
    Context context = new Context();
    String content = this.templateEngine.process(templateName, context);
    this.sendEmailSync(to, subject, content, false, true);
}
```

**//download file Template sử dụng trong video:**

<https://drive.google.com/drive/folders/1FRyhSzW5CEvWH5D30Dr2V0MFO7-JdpCV?usp=sharing>

**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)

## #126. Send Email với Template (Part 2)

Source code video này (#126):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njRLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njRLlu?usp=sharing)

### 1. Truyền data động cho template

<https://www.thymeleaf.org/doc/articles/springmail.html>

[https://gitlab.com/public-starter-projects/1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/service/MailService.java?ref\\_type=heads#L89](https://gitlab.com/public-starter-projects/1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/service/MailService.java?ref_type=heads#L89)

Truyền data từ controller qua template

//todo: hardcode data, sử dụng vòng lặp để render data

Tạo full template (đang hardcode data)

Sử dụng vòng lặp với thymeleaf: <https://stackoverflow.com/a/36745375>

Format currency: <https://stackoverflow.com/a/14165869>

```
<span th:text="${#numbers.formatDecimal(job.salary, 0, 'COMMA', 0, 'POINT')}">
</span>
```

## #127. Send Email với Template (Part 3).

Source code video này (#127):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### 1.Lấy động data để send email

```
public void sendSubscribersEmailJobs() {
    List<Subscriber> listSubs = this.subscriberRepository.findAll();
    if (listSubs != null && listSubs.size() > 0) {
        for (Subscriber sub : listSubs) {
            List<Skill> listSkills = sub.getSkills();
            if (listSkills != null && listSkills.size() > 0) {
                List<Job> listJobs = this.jobRepository.findBySkillsIn(listSkills);
                if (listJobs != null && listJobs.size() > 0) {

                    // List<ResEmailJob> arr = listJobs.stream().map(
                    // job -> this.convertJobToSendEmail(job)).collect(Collectors.toList());

                    this.emailService.sendEmailFromTemplateSync(
                        sub.getEmail(),
                        "Cơ hội việc làm hot đang chờ đón bạn, khám phá ngay",
                        "job",
                        sub.getName(),
                        listJobs);
                }
            }
        }
    }
}
```

Lưu ý: cần update thêm cho constructors như bên dưới ae nhé

```
private final SubscriberRepository subscriberRepository;  
private final SkillRepository skillRepository;  
private final JobRepository jobRepository;  
private final EmailService emailService;
```

```
public SubscriberService(  
    SubscriberRepository subscriberRepository,  
    SkillRepository skillRepository,  
    JobRepository jobRepository,  
    EmailService emailService) {  
    this.subscriberRepository = subscriberRepository;  
    this.skillRepository = skillRepository;  
    this.jobRepository = jobRepository;  
    this.emailService = emailService;  
}
```

## 2. Sử dụng async để gửi email

<https://www.danvega.dev/blog/sending-async-emails-in-spring>

```
public ResEmailJob convertJobToSendEmail(Job job) {  
    ResEmailJob res = new ResEmailJob();  
    res.setName(job.getName());  
    res.setSalary(job.getSalary());  
    res.setCompany(new ResEmailJob.CompanyEmail(job.getCompany().getName()));  
    List<Skill> skills = job.getSkills();  
    List<ResEmailJob.SkillEmail> s = skills.stream().map(skill -> new  
ResEmailJob.SkillEmail(skill.getName()))  
        .collect(Collectors.toList());  
    res.setSkills(s);  
    return res;  
}
```

## #128. Cron Job (Part 1)

Tham khảo:

<https://spring.io/blog/2020/11/10/new-in-spring-5-3-improved-cron-expressions>

<https://docs.spring.io/spring-framework/reference/integration/scheduling.html>

### 1. Khái niệm cron job

<https://en.wikipedia.org/wiki/Cron>

Cron là cách chúng ta đặt lịch (schedule) để tự động làm một công việc gì đấy (job)

### 2. Cách viết cron job

<https://docs.spring.io/spring-framework/reference/integration/scheduling.html#scheduling-cron-expression>

<https://spring.io/guides/gs/scheduling-tasks>

#### Bước 1: enabled

@EnableScheduling

#### Bước 2: sử dụng schedule với cron

[https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/scheduling/annotation/Scheduled.html#cron\(\)](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/scheduling/annotation/Scheduled.html#cron())

@Scheduled(fixedRate = 5000)

## **#129. Cron Job (Part 2)**

Source code video này (#129):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Cách viết cron job:

[https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/scheduling/annotation/Scheduled.html#cron\(\)](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/scheduling/annotation/Scheduled.html#cron())

<https://docs.spring.io/spring-framework/reference/integration/scheduling.html#scheduling-cron-expression>

Tham khảo: <https://stackoverflow.com/questions/45124756/spring-scheduling-cron-expression-for-everyday-at-midnight-not-working>



### #130. Nhận xét về dự án Frontend

Source code video này (#130):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

#### 1. Fix bug

Nên tách riêng API dành cho Client và API dành cho Admin => dễ phân quyền khi public API

//Code thêm tính năng download và giao diện subscribers

#### **Bổ sung api:**

POST /api/v1/subscribers/skills

=> lấy skill của user đã đăng ký

#### Test dự án frontend:

Version node.js mình sử dụng là **16.20.0**:

<https://nodejs.org/download/release/v16.20.0/>

Cài chính xác version để hạn chế tối đa lỗi có thể xảy ra. (npm)

Kiểm tra đã cài đặt thành công với câu lệnh: **node -v**

Link gitlab dự án frontend: <https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/03-react-vite-jobhunter>

git checkout master

## 2. Tổng kết ứng dụng:

### Người dùng chưa đăng nhập:

- Chức năng đăng ký/đăng nhập
- Xem được danh sách công ty (company)
- Xem được danh sách công việc (job)
- Tìm kiếm job theo các tiêu chí (skill/location)

### Người dùng đã đăng nhập:

- **Người dùng được tạo = /register** (normal user, không được phân role):
    - + Chức năng rải cv
    - + Chức năng xem lịch sử rải cv
    - + Chức năng đăng ký nhận email theo skill (subscribers)
  - **Người dùng được phân quyền** (theo permission và role):
    - + Bản chất của 1 permission (quyền hạn), là 1 API khai báo tại backend
    - + Người dùng admin (full quyền) được tạo = code
    - + Cần kết hợp giữa frontend/backend để thực hiện việc phân quyền (mặc định chặn phân quyền tại backend theo Interceptor)
- #117. Cách xử lý phân quyền tại Frontend (Bonus)  
và #118. Interceptor

## **Chapter 14: Tổng Kết**

*Tổng kết các kiến thức đã học và hướng dẫn tối ưu hóa dự án backend Java Spring*

### **#131. Xử lý Global Exception**

[https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/web/rest/errors/ExceptionHandler.java?ref\\_type=heads](https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/web/rest/errors/ExceptionHandler.java?ref_type=heads)

Lỗi đang default nhảy vào Custom Auth

```
@ExceptionHandler(Exception.class)
public ResponseEntity<RestResponse<Object>> handleAllException(Exception ex) {
    RestResponse<Object> res = new RestResponse<Object>();
    res.setStatusCode(HttpStatus.INTERNAL_SERVER_ERROR.value());
    res.setMessage(ex.getMessage());
    res.setError("Internal Server Error");
    return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(res);
}
```

## #132. Swagger

Source code video này (#132):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

### 1.Swagger là gì ?

<https://swagger.io/docs/specification/about/>

**Mục đích chúng ta sử dụng Swagger, là tìm cách viết document (tài liệu) cho REST APIs.**

Đi làm, có rất nhiều cách để viết “document”, và swagger là 1 trong số đó (và nên biết khi sử dụng với Java)

### 2.Cài đặt

<https://springdoc.org/#getting-started>

`implementation("org.springdoc:springdoc-openapi-starter-webmvc-ui:2.5.0")`

**Cấu hình Security: (allow access)**

```
"/v3/api-docs/**",  
"/swagger-ui/**",  
"/swagger-ui.html"
```

**//FormatRestResponse.java**

```
String path = request.getURI().getPath();  
if (path.startsWith("/v3/api-docs") || path.startsWith("/swagger-ui")) {  
    return body;  
}
```

**Truy cập swagger-ui tại:** <http://localhost:8080/swagger-ui/index.html>

(với service = /v3/api-docs)

### 3.Cấu hình Swagger

- Thêm Bearer Token
- Hướng dẫn test api

**Download file cấu hình:**

[https://drive.google.com/file/d/1OQ0\\_\\_SI9wWr-S0\\_cvOWDT8WKhQx6XFXw/view?usp=drive\\_link](https://drive.google.com/file/d/1OQ0__SI9wWr-S0_cvOWDT8WKhQx6XFXw/view?usp=drive_link)

Tham khảo:

<https://springdoc.org/#springdoc-openapi-core-properties>

<https://www.bezkoder.com/spring-boot-swagger-3/>

<https://www.baeldung.com/spring-boot-swagger-jwt>

### **#133. Logging**

Tài liệu: <https://docs.spring.io/spring-boot/how-to/logging.html>

<https://www.baeldung.com/spring-boot-logging>

Ví dụ:

[https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/service/impl/CountryServiceImpl.java?ref\\_type=heads#L21](https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/service/impl/CountryServiceImpl.java?ref_type=heads#L21)

### #134. Build Dự Án với Docker

Source code video này (#134):

[https://drive.google.com/drive/folders/1Zh-F\\_MDCUCnuC4OhLsB3TlawZ\\_njrLlu?usp=sharing](https://drive.google.com/drive/folders/1Zh-F_MDCUCnuC4OhLsB3TlawZ_njrLlu?usp=sharing)

Yêu cầu để thực hiện video này: bạn cần có kiến thức cơ bản về docker

Nếu chưa biết gì, tham khảo:

[https://www.youtube.com/playlist?list=PLncHg6Kn2JT4kLKJ\\_7uy0x4AdNrCHbe0n](https://www.youtube.com/playlist?list=PLncHg6Kn2JT4kLKJ_7uy0x4AdNrCHbe0n)

Tại sao không deploy local ?

Tư duy khi viết dockerfile cho dự án:

**Bước 1:** sử dụng gradle để build dự án (mặc định thành **file .jar**)

Do chúng ta không sử dụng view (như khóa mvc), nên không cần build file **.war**

**Bước 2:** định nghĩa docker compose file

Ghi đè các tham số môi trường, như:

- Thông tin kết nối tới database
- Thông tin nơi lưu file upload
- Tạo volume để backup database, và file upload

**Bước 3:** chạy dự án với docker compose

Do đã có file .jar (bước 1) => để chạy dự án, chỉ cần chuẩn bị môi trường java là chạy được

### #135. Nhận xét về cách code dự án Spring

Không có khái niệm, đâu là cách tốt ưu nhất để code dự án với Spring.

**Điều quan trọng, là cách bạn code, cách bạn tổ chức code, phục vụ và giải quyết tốt cho bài toán bạn đang gặp phải.**

Tuy nhiên, khi code, cần chú ý tới 3 yếu tố chính:

- **Code phải có khả năng mở rộng** (phát triển cho tương lai, ví dụ thêm feature mới)
- **Code phải bảo trì được** (maintain)
- **Code phải có khả năng test** (viết testcase)

//Giải thích cách code với implement:

Cách code spring với implement

Ví dụ: [https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/service/impl/CountryServiceImpl.java?ref\\_type=heads#L19](https://gitlab.com/public-starter-projects1/000-java/02-java-spring-restful/02-java-jhipster-with-filter/-/blob/master/src/main/java/com/mycompany/myapp/service/impl/CountryServiceImpl.java?ref_type=heads#L19)



## #136. Cách tự tạo dự án Spring Restful của bạn

### 1. Các lưu ý

**Không nên chọn version Java mới nhất**, vì có thể, thư viện (community) chưa kịp thời hỗ trợ => nên chọn 1 version “gần mới nhất” để đảm bảo tính ổn định.

Hãy chọn version java có từ LST (long term support)

<https://www.oracle.com/java/technologies/java-se-support-roadmap.html>

**Nên chọn các version mới của Spring để nhận những update mới nhất** (xịn nhất)

<https://github.com/spring-projects/spring-framework/releases>

<https://github.com/spring-projects/spring-framework/wiki/Spring-Framework-Versions>

Nếu bắt đầu dự án mới, bạn **ên sử dụng Spring Framework từ version 6.x**  
(ứng với Spring Security 6.x và Spring Boot 3.x)

### 2. Cách tự tạo dự án của riêng bạn

**Bạn có thể lấy lại dự án của mình (sử dụng trong khóa học) để code mở rộng thêm**

Ưu điểm: bạn không cần phải “cấu hình”, ví dụ như Spring Security

Nhược điểm: bạn cần xóa code đi, để phục vụ dự án của bạn

Hoặc, tạo dự án từ đầu:

<https://start.spring.io/>

Lưu ý, cấu hình của trang : <https://start.spring.io/> sẽ thay đổi theo thời gian (đây là lý do tại sao trong khóa học, mình yêu cầu kéo dự án từ github)

<https://gitlab.com/public-starter-projects/1/000-java/02-java-spring-restful/01-java-spring-jobhunter-starter>

### 3.Cách bạn tự code dự án Spring từ đầu

Truy cập: <https://start.spring.io/>

**Bước 1:** Bạn chọn version Spring boot, chọn build tool, language và dependency

Dependency hay dùng như: jpa, chọn loại database, security

**Bước 2:** Nếu bạn chọn database, cần cấu hình thông tin đăng nhập vào database  
Mình đã làm bước này trong khóa học, tại video #16

**Bước 3:** Tạo model cho dự án

Tạo base các model cho dự án và ràng buộc mối quan hệ (nếu có)

Đây chính là bước bạn cần phân tích database cho dự án của bạn, có thể chưa hoàn hảo, tuy nhiên cần code base các Model chính

**Bước 4:** Cấu hình Request/Response (nếu có)

**Tại bước này, bạn nên off Security** (disabled Spring Security)

//disable security

```
@SpringBootApplication(exclude = {  
    org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration.class,  
    org.springframework.boot.actuate.autoconfigure.security.servlet.ManagementWebSecurityAutoConfiguration.class  
})
```

**Cấu hình Security với Password Encoder (video #48)**

Sau đấy, tạo base CRUD User

**Bước 5:** Cấu hình Spring Security

Bao gồm viết file config cho Security và logic đăng nhập, cùng với JWT

**Bạn không cần nhớ code**, có thể tham khảo cách mình làm trong khóa học, **điều quan trọng, là bạn hiểu bạn đang làm gì**. Có thể debug để xem luồng code chạy

**Bước 6:** CRUD các tính năng cho dự án của bạn

## **#137. Nhận xét về dự án thực hành**

### **1. Nhận xét về dự án thực hành**

#### **Ưu điểm:**

- Thực hành sử dụng Spring như là backend Restful API với JWT (json web token)
- Phân quyền dự án ở mức độ basic (với role và permissions)

#### **Nhược điểm:**

- Nên tách riêng api dành cho client, dành cho admin
- Không nên format response (để cho frontend tự format)

### **2. Các kiến thức chưa đề cập**

- rate limit (throttle)
- actuator (logging, tracing...)
- Soft-delete: không nên xóa data

## #138. What's next ? Học gì tiếp theo

### 1. My Goal (Jhipster)

<https://www.jhipster.tech/>

Từ video #19, mình đã nói về mục tiêu của khóa học.

Như vậy, với 2 khóa học Spring MVC và Spring Restful, mình tin rằng, bây giờ, nếu đưa source code 1 dự án thực tế (như là Jhipster), **bạn có thể tự tin đọc & hiểu source code ở một mức độ nhất định (không phải là siêu nhân, nhưng ít nhất, chúng ta có thể hiểu các kiến thức trọng tâm)**

### 2. What's next ?

**Tự thực hành project :** nếu không có bước này, kiến thức không phải là của bạn. Nên tự thực hành để chuyển hóa kiến thức. Chỗ nào quên, thì xem lại source code và kết hợp google (đọc stackoverflow)

Video #136, mình đã đề cập về cách tự code dự án Spring của bạn.

**Học kiến thức frontend:** Bạn nên học thêm kiến thức frontend, vì có mỗi backend, không thể tạo nên 1 sản phẩm hoàn thiện.

Đồng thời, bạn sẽ thấy nó nhàm chán (boring) - do không nhìn thấy được tính “áp dụng” của sản phẩm bạn tạo ra

### Các kiến thức mới để khám phá:

- Spring cloud (microservice)
- Đa luồng (chạy parallel/ multi thread)
- Xử lý với transactions (rollback)
- Xử lý với nhiều data (batch)

### #139. Suy Nghĩ Về Chuyện Thực Tập & Làm Fresher (Bonus)

**Question:** Anh ơi, học xong lộ trình Spring của anh (hoặc khóa Spring Restful), em có thể đi thực tập/đi làm fresher không ạ ?

**Answer:**

Mình không muốn trả lời là Có hay Không ? Vì câu hỏi, nó đã sai ngay từ đầu.

<https://hoidanit.vn/tu-van#hoc-xong-khoa-hoc-cua-hoi-dan-it-co-di-lam-fresher-duoc-khong>

**Phân tích:**

- Để làm level fresher (người ít kinh nghiệm), bạn cần thực tập trước. Chưa biết gì, hoặc **chưa từng đi thực tập, mà làm fresher, ca này khó** (nếu không muốn nói là khó xảy ra)

Để làm fresher, ngoài kiến thức chuyên môn (kiến thức về java trong trường hợp này), bạn cần có "tác phong của người đi làm". Ví dụ như kỹ năng làm việc nhóm, kỹ năng giao tiếp, khả năng chịu áp lực, có tinh thần trách nhiệm trong công việc.

Nên nhớ, đi làm fresher, bạn làm công ăn lương (theo hợp đồng), không phải đi chơi. **Bạn được trả tiền công, dĩ nhiên, bạn cần làm được việc**

- **Để làm thực tập, điều cần có chính là kiến thức chuyên môn.** Tuy nhiên, rất ít công ty (may mắn thì gặp) là chấp nhận training cho thực tập sinh

Đa phần, là sẽ để cho tự bơi và tự học. Cuối kỳ thực tập, cho làm bài test để chọn ra những bạn giỏi nhất.

**Việc bạn code & học theo khóa học của mình, nó không có nghĩa là, bạn có thể làm được như khóa học của mình.**

Vì vậy, trước khi đi thực tập, **bạn hãy "tự làm dự án của bạn"**, có như vậy, kiến thức mới là của bạn, và giúp bạn "trụ lại" sau kì thực tập

Mình có video phân tích về kỹ năng ghi CV, và góc nhìn của cty khi đọc CV của bạn: <https://youtube.com/live/xy67XFdAR8Q>

Nên nhớ, kiến thức cần thời gian tiêu hóa và hấp thụ. Học được 1 tới 2 tháng, bạn có thể "được nhận thực tập", cơ mà, liệu bạn có thể trụ lại ?

Hãy dành từ 6 tháng tới 1 năm, học hành 1 cách nghiêm túc & dục tốt bất đạt. Sự thật đấy

## #140. Giải Thích Về Lỗi RestResponse cannot be cast to class String

Tài liệu:

<https://docs.spring.io/spring-framework/reference/6.2-SNAPSHOT/web/webmvc/message-converters.html>

<https://stackoverflow.com/questions/44121648/controlleradvice-responsebodyadvice-failed-to-enclose-a-string-response>

### 1. Giải thích về cơ chế Message Converter của Spring

//debug

```
if (!MediaType.APPLICATION_JSON.equals(selectedContentType)) {  
    return body;  
}  
  
String path = request.getURI().getPath();  
if (path.startsWith("/v3/api-docs") || path.startsWith("/swagger-ui")) {  
    return body;  
}
```

### 2. Cách debug để xem lỗi

**AbstractHttpMessageConverter**

Write -> **addDefaultHeaders**

**StringHttpMessageConverter**

protected void **addDefaultHeaders**(HttpHeaders headers, **String s**, @Nullable  
MediaType type)

### 3. Cách khắc phục

Không nên can thiệp vào quá trình “mặc định” sử dụng MessageConverter của Spring, vì không rõ yêu cầu sẽ thay đổi như thế nào trong tương lai, và có thể ảnh hưởng tới các thư viện (third party), ví dụ trong khóa học là swagger.

## **#141. Cấu Hình Lombok cho STS (Nếu Gặp Lỗi)**

**Mục tiêu của video:** hướng dẫn cách fix dự án Spring sử dụng thư viện lombok, tuy nhiên không chạy được với Spring Tool Suite

Ví dụ:

[https://www.reddit.com/r/SpringBoot/comments/12im5y6/lombok\\_not\\_working\\_with\\_sts/](https://www.reddit.com/r/SpringBoot/comments/12im5y6/lombok_not_working_with_sts/)

**Bước 1:** download lombok

<https://projectlombok.org/download>

Cần có file **.jar** của thư viện để thực thi bước 2

**Bước 2:** thực thi lombok

Sử dụng câu lệnh: **java -jar lombok.jar**

Chọn instance của Spring Tool Suite

**Bước 3:** Restart Spring Tool Suite

- Clear cache
- Build lại dự án

# Lời Kết

Như vậy là chúng ta đã cùng nhau trải qua hơn 140+ video về sử dụng framework Spring với Restful API dành cho backend Java

Tất cả các kiến thức mình chia sẻ, đều được lấy từ kinh nghiệm đi làm của mình và... các trang tài liệu về Spring.

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 Spring 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)