

## Section 1: Getting Started

### 1. What is Security & Why it is important

**INTRODUCTION TO SECURITY**  
WHAT & WHY

**eazy bytes**

- WHAT IS SECURITY?**  
Security is for protecting your data and business logic inside your web applications.
- DIFFERENT TYPES OF SECURITY**  
Security for a web application will be implemented in different way like using firewalls, HTTPS, SSL, Authentication, Authorization etc.
- SECURITY IS AN NON FUN REQ.**  
Security is very important similar to scalability, performance and availability. No client will specifically asks that I need security.
- WHY SECURITY IMPORTANT?**  
Security doesn't mean only loosing data or money but also the brand and trust from your users which you have built over years.
- SECURITY FROM DEV PHASE**  
Security should be considered right from development phase itself along with business logic
- AVOIDING MOST COMMON ATTACKS**  
Using Security we should also avoid most common security attacks like CSRF, Broken Authentication inside our application.

#### Bảo mật là gì ?

**Bảo mật là để bảo vệ dữ liệu và Business Logic của bạn trong các ứng dụng web của bạn.** Không phải mỗi khi bạn chỉ muốn bảo vệ dữ liệu của mình, đôi khi bạn cũng có thể muốn bảo vệ Business Logic của mình, như nếu chúng ta có thể nhìn vào công cụ tìm kiếm Google, họ có rất nhiều thuật toán, họ không muốn chia sẻ các thuật toán đó với Mọi người, vì vậy đó là Business Logic của họ, đang điều hành doanh nghiệp của họ hàng ngày. Đó là lý do tại sao Google không bao giờ muốn chia sẻ Business Logic của họ với bất kỳ ai và họ sẽ bảo vệ các thuật toán đó bằng các biện pháp bảo mật tuyệt vời. Vì vậy, đó là lý do tại sao xin lưu ý rằng việc đảm bảo ứng dụng web không chỉ dành cho dữ liệu của chúng tôi, mà còn là Business Logic của chúng tôi.

#### Bảo mật là một yêu cầu không chức năng

Và điểm tiếp theo mà tôi muốn làm nổi bật ở đây là bảo mật là một yêu cầu không chức năng. Không có khách hàng nào sẽ yêu cầu bạn xem xét bảo mật trong quá trình phát triển ứng dụng web. Theo mặc định và giả định rằng khách hàng của bạn sẽ thực hiện như một phần của các yêu cầu phi chức năng. Vì vậy, các yêu cầu phi chức năng là các yêu cầu không liên quan đến Business Logic khách hàng của bạn, như nếu chúng tôi thực hiện bảo mật, hiệu suất, tiêu chuẩn mã hóa, vì vậy đây là tất cả các yêu cầu không có chức năng mà mọi khách hàng mong đợi từ bạn 'xác định rõ ràng trong hợp đồng hoặc yêu cầu của họ.

#### Bảo mật ngay từ giai đoạn phát triển.

Chúng ta không bao giờ nên theo giả định rằng bảo mật đến sau khi thử nghiệm UAT hoặc sau khi triển khai sản xuất. Chúng ta nên luôn luôn xem xét bảo mật ngay từ giai đoạn phát triển để công việc

của bạn như một nhà phát triển sẽ rất dễ dàng. Vì vậy, hãy nghĩ về một kịch bản mà bạn không xem xét bảo mật từ giai đoạn phát triển và có thể sau khi hoàn thành việc kiểm tra UAT và hiệu suất, sau đó bạn bắt đầu tập trung vào bảo mật. Vì vậy, những loại kịch bản này sẽ dẫn đến một số vấn đề hồi quy. Đó là lý do tại sao luôn xem xét bảo mật từ giai đoạn phát triển.

### Có nhiều loại bảo mật khác nhau.

Vì vậy, có một số trách nhiệm bảo mật nhất định mà nhóm DevOps của chúng tôi hoặc Quản trị viên máy chủ của chúng tôi sẽ chú ý, như nếu chúng tôi thực hiện HTTPS, Tường lửa, Chứng chỉ SSL. Vì vậy, tất cả các loại bảo mật này sẽ được các quản trị viên máy chủ và DevOps chăm sóc, nhưng nếu bạn thấy, với tất cả các HTTPS, chứng chỉ SSL, tường lửa, họ đang bảo vệ máy chủ của chúng tôi. Họ sẽ không bảo vệ ứng dụng web hoặc Business Logic hoặc dữ liệu của chúng tôi. Vì vậy, trách nhiệm của chúng tôi là nhà phát triển để thực hiện authentication và authorization trong ứng dụng web của chúng tôi.

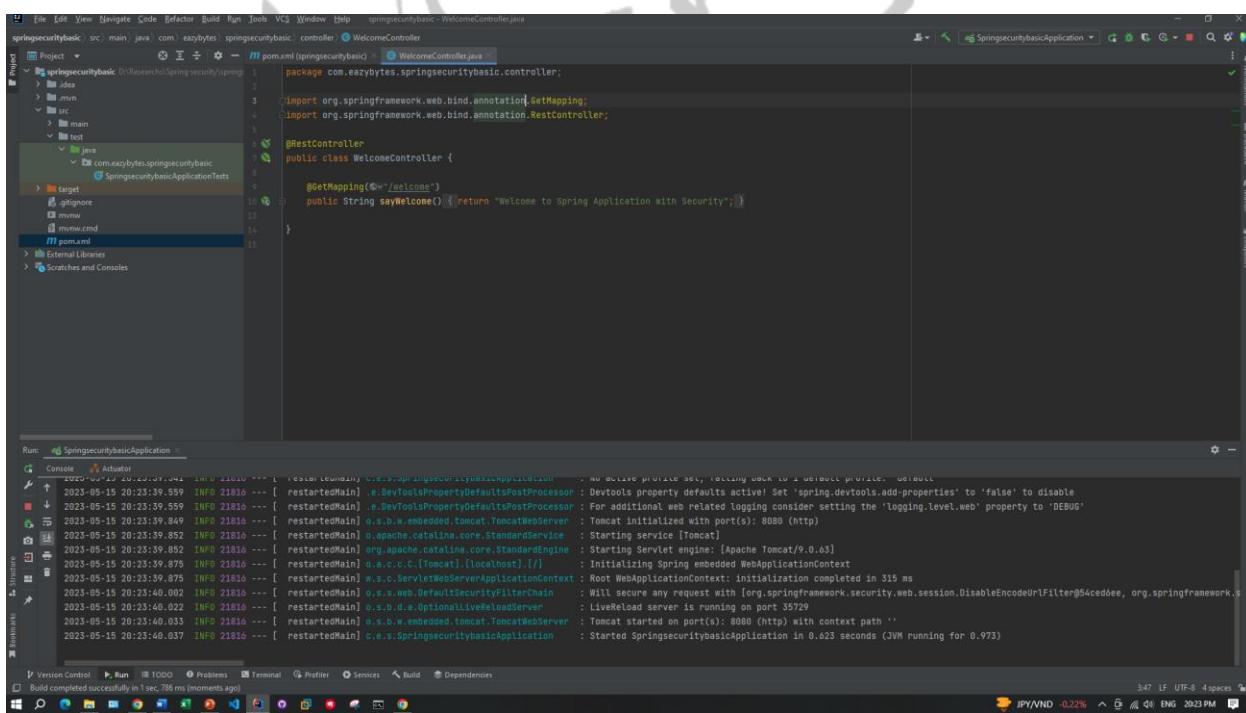
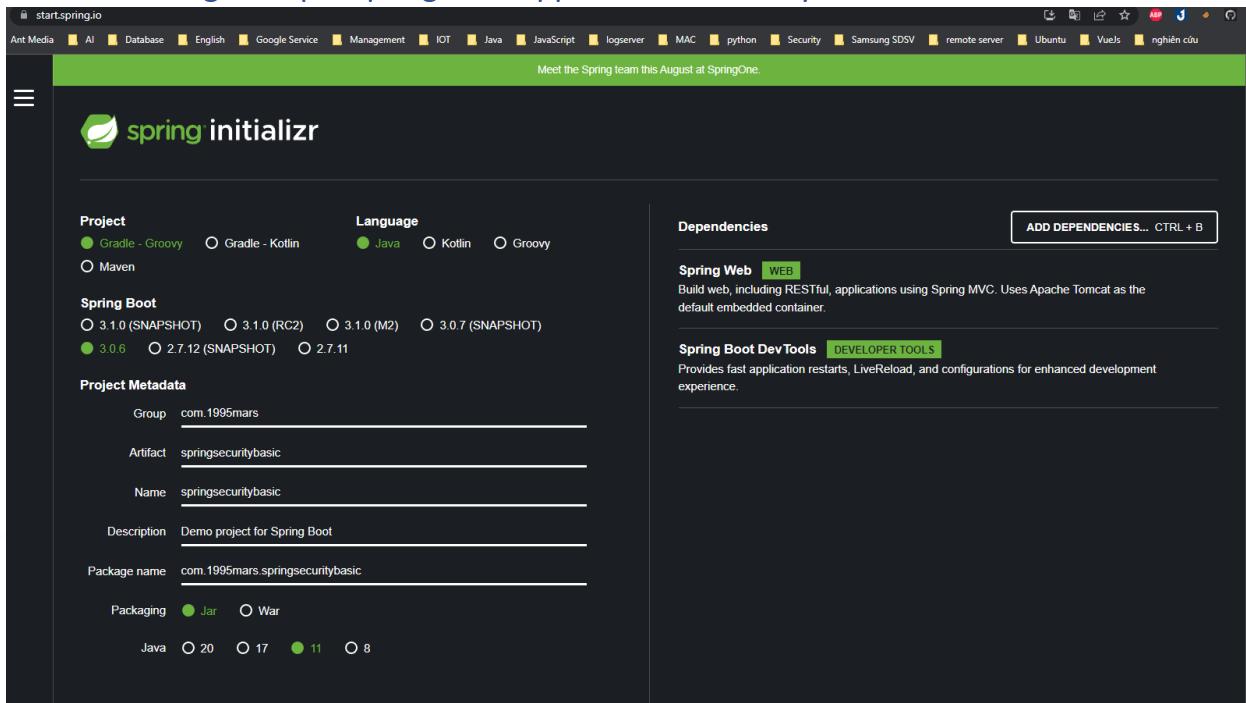
### Tại sao bảo mật lại quan trọng?

Bởi vì bất cứ khi nào bạn không có bảo mật, chắc chắn, sẽ có một số vi phạm dữ liệu, lỗ hổng bảo mật, các cuộc tấn công bảo mật sẽ xảy ra trong ứng dụng web của bạn, điều này có thể dẫn đến mất dữ liệu hoặc mất Business Logic và với loại kịch bản này, chắc chắn, công ty mà bạn đang làm việc hoặc tổ chức mà bạn đang làm việc, họ sẽ phải đổi mặt với các vấn đề thương hiệu và tin cậy từ user hoặc từ khách hàng và trong một số tình huống, các tổ chức của chúng tôi cũng sẽ phải đổi mặt với các vấn đề pháp lý từ khách hàng. Vì vậy, đó là lý do tại sao luôn xem xét, bảo mật là rất quan trọng.

### Tránh các cuộc tấn công phổ biến nhất

Bất cứ khi nào chúng tôi bảo mật ứng dụng web của mình, chúng tôi cũng đã thêm lợi thế, giống như có một số lượng lớn các cuộc tấn công bảo mật phổ biến nhất xảy ra trong ngành hàng ngày, như Tấn công CSRF, Cross-Site Scripting (XSS), các cuộc tấn công Broken Authentication, vì vậy đây là tất cả những gì chúng ta nên xem xét trong khi chúng ta đang bảo vệ các ứng dụng web của mình. Vì vậy, ở cấp độ cao, tôi chỉ muốn thuyết phục bạn rằng việc đảm bảo các ứng dụng web là rất quan trọng.

## 2. Creating a simple Spring Boot app with out security



### 3. Securing Spring Boot basic app using Spring Security

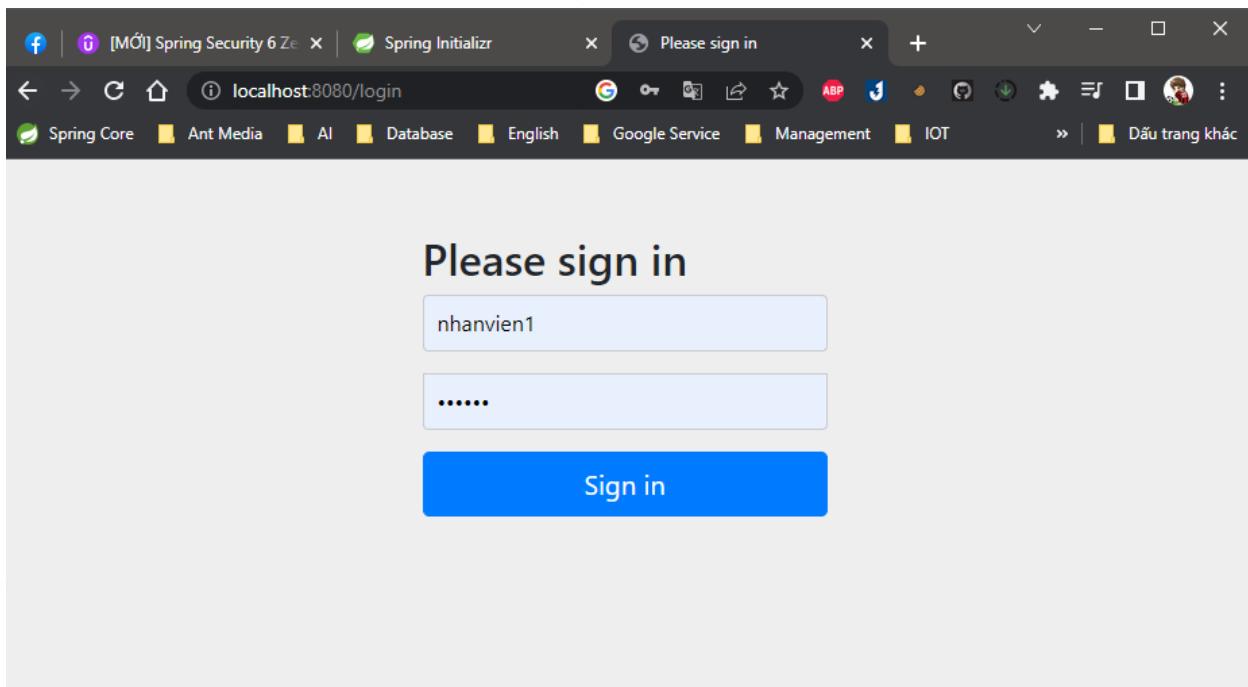
## Thêm các dependence

```
17      <java.version>17</java.version>
18  </properties>
19  <dependencies>
20      <dependency>
21          <groupId>org.springframework.boot</groupId>
22          <artifactId>spring-boot-starter-security</artifactId>
23      </dependency>
24      <dependency>
25          <groupId>org.springframework.boot</groupId>
26          <artifactId>spring-boot-starter-web</artifactId>
27      </dependency>
28
29      <dependency>
30          <groupId>org.springframework.boot</groupId>
31          <artifactId>spring-boot-devtools</artifactId>
32          <scope>runtime</scope>
33          <optional>true</optional>
34      </dependency>
35      <dependency>
36          <groupId>org.springframework.boot</groupId>
37          <artifactId>spring-boot-starter-test</artifactId>
38          <scope>test</scope>
39      </dependency>
40      <dependency>
41          <groupId>org.springframework.security</groupId>
42          <artifactId>spring-security-test</artifactId>
43          <scope>test</scope>
44      </dependency>
45  </dependencies>
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>
```

Bây giờ, nếu bạn cố truy cập localhost:8080/welcome, lần này bạn sẽ không nhận được bất kỳ phản hồi. Thay vào đó, Spring security của tôi đang yêu cầu tôi nhập thông tin đăng nhập của mình.



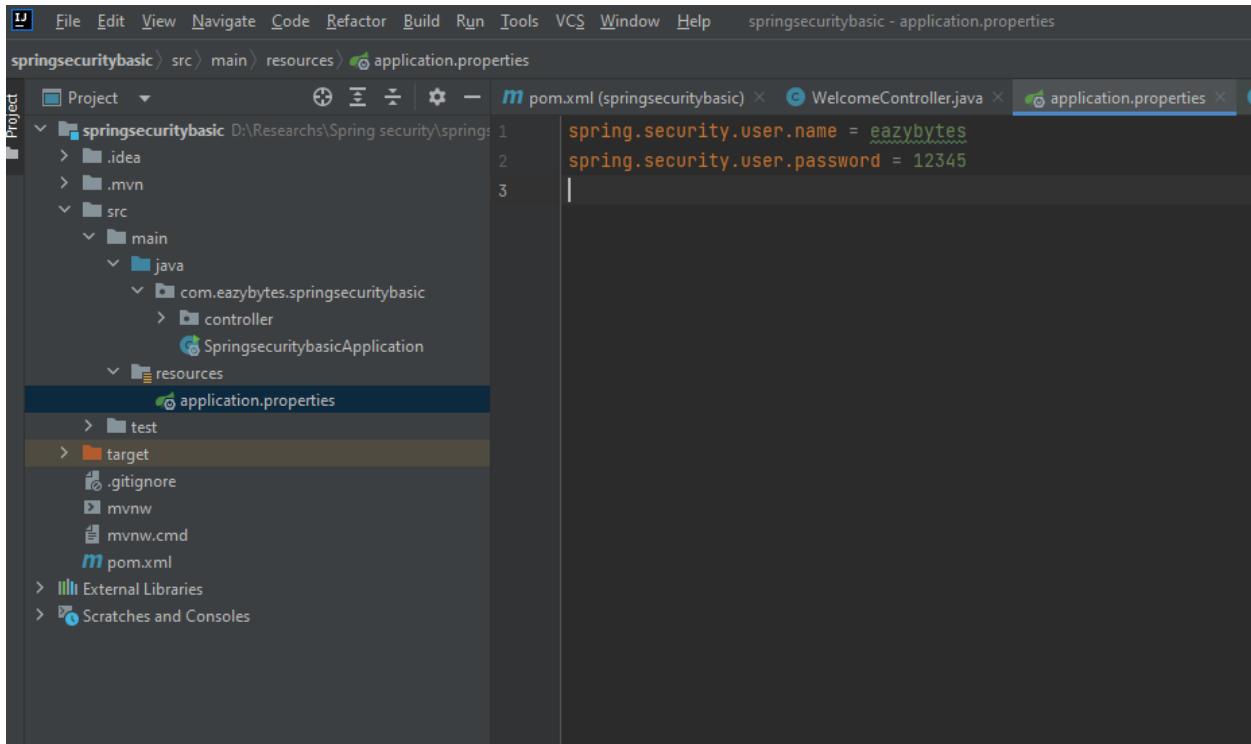
Tên người dùng mặc định bên trong Spring Security framework là admin. Vì vậy, mỗi khi bạn nhập người dùng, bạn cũng cần phải nhập mật khẩu. Không có giá trị mật khẩu mặc định. Mật khẩu sẽ được tạo và được in trên bảng điều khiển của bạn trong quá trình khởi động.

```
SpringsecuritybasicApplication x
Console Actuator
2023-05-15 20:27:16.555  INFO 22752 --- [ restartedMain] w.s.c.ServletWebServerApplication:2023-05-15 20:27:16.555  INFO 22752 --- [ restartedMain] w.s.c.ServletWebServerApplication:2023-05-15 20:27:16.664  WARN 22752 --- [ restartedMain] .s.s.UserDetailsServiceAutoConfig:
Using generated security password: 68e1fdc8-75a9-43b3-9216-fc089e93356f
This generated password is for development use only. Your security configuration must be
2023-05-15 20:27:16.696  INFO 22752 --- [ restartedMain] o.s.s.web.DefaultSecurityFilter:2023-05-15 20:27:16.707  INFO 22752 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer:2023-05-15 20:27:16.717  INFO 22752 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer:2023-05-15 20:27:16.722  INFO 22752 --- [ restartedMain] c.e.s.SpringsecuritybasicApplication:
```

Và thêm một điều kỳ diệu hay ho của Spring Security framework là nếu bạn làm mới bao nhiêu lần hoặc nếu bạn gửi yêu cầu nhiều lần framework không hỏi bạn để nhập thông tin đăng nhập của bạn mỗi lần. Nó sẽ chỉ hiển thị cho bạn trang đăng nhập và yêu cầu thông tin đăng nhập trong lần yêu cầu đầu tiên. Và lần sau, Spring Security framework sẽ đảm nhiệm việc lưu trữ Session ID hoặc chi tiết token, miễn là đang sử dụng cùng một trình duyệt nó sẽ không hỏi thông tin đăng nhập nhiều lần.

#### 4. Configure static credentials inside application properties file

Chúng ta có thể định cấu hình tên người dùng và mật khẩu của riêng mình bên trong ứng dụng web của chúng tôi và tương tự chúng ta có thể sử dụng để truy cập dịch vụ REST mà chúng tôi đã xây dựng cho đến nay. Vì vậy, để thay đổi hành vi mặc định này của Spring Security, những gì chúng ta phải làm là chúng ta có thể vào application.properties. Vì vậy, mọi ứng dụng web Spring Boot sẽ có một application.properties bên trong thư mục resource.



The screenshot shows the IntelliJ IDEA interface with a Spring Boot project named 'springsecuritybasic'. The project structure on the left includes .idea, .mvn, src (with main and resources), test, target, and various configuration files like pom.xml and mvnw. The application.properties file is selected in the resources directory. The code editor on the right contains the following content:

```
spring.security.user.name = eazybytes
spring.security.user.password = 12345
```

## 5. Tại sao nên sử dụng Spring Security framework

**WHY SPRING SECURITY ?**

 Application security is not fun and challenging to implement with our custom code/framework.

 Spring Security built by a team at Spring who are good at security by considering all the security scenarios. Using Spring Security, we can secure web apps with minimum configurations. So there is no need to re-invent the wheel here.

 Spring Security handles the common security vulnerabilities like CSRF, CORs etc. For any security vulnerabilities identified, the framework will be patched immediately as it is being used by many organizations.

 Using Spring Security we can secure our pages/API paths, enforce roles, method level security etc. with minimum configurations easily.

 Spring Security supports various standards of security to implement authentication, like using username/password authentication, JWT tokens, OAuth2, OpenID etc.

- Lý do đầu tiên tại sao chúng ta nên sử dụng Spring Security bảo mật ứng dụng không phải lúc nào cũng thú vị. Rất khó thực hiện hoặc bảo mật ứng dụng web của bạn với mã tùy chỉnh của riêng bạn hoặc với khuôn khổ tổ chức của riêng bạn. Lý do là vì mỗi ngày có hàng trăm vi phạm an ninh, lỗ hổng bảo mật đang được xác định. Càng nhiều lỗ hổng đang được các tin tức khám phá. Rất khó để chúng tôi ở chế độ chờ 24/7 và cập nhật mã tùy chỉnh của chúng tôi hàng ngày. Thay vào đó, chúng ta có thể dựa vào các khuôn khổ như Spring Security bởi vì Spring Security này, họ sẽ có một nhà phát triển chuyên dụng nhóm chỉ tập trung vào bảo mật. Vì vậy, với tư cách là nhà phát triển nếu chúng ta có thể xem xét Spring Security cho ứng dụng web của chúng tôi chúng ta có thể chỉ cần sử dụng Spring Security đó bên trong ứng dụng web của chúng tôi và chúng ta có thể tập trung, chúng ta có thể đặt phần lớn nỗ lực của mình bằng văn bản. Business Logic đang nâng cao ứng dụng web của chúng tôi với những yêu cầu mới mà khách hàng của chúng tôi đang đề xuất. Vì vậy, đó là điểm đầu tiên mà tôi muốn nhấn mạnh.
- Và điểm tiếp theo mà tôi muốn nhấn mạnh ở đây là Spring Security được xây dựng bởi nhóm có kinh nghiệm của các nhà phát triển bằng cách xem xét tất cả các tình huống bảo mật. Vì vậy, bất cứ khi nào bạn sử dụng Spring Security này, được xây dựng bởi các chuyên gia này, bạn có thể dễ dàng bảo mật ứng dụng web của mình với các cấu hình rất tối thiểu để bạn không phải phát minh lại khuôn khổ bảo mật lặp đi lặp lại, bạn chỉ có thể tận dụng điều này. Và vì nó là một framework mã nguồn mở không có tiền cũng tham gia.
- Và ngoài việc bảo mật ứng dụng web của bạn các Spring Security này cũng xử lý các lỗ hổng bảo mật phổ biến nhất như CSRF, CORS. Vì vậy, đối với bất kỳ lỗ hổng bảo mật nào được xác định trên thị trường trên cơ sở hàng ngày, đội an ninh Spring họ sẽ vá hoặc cập nhật framework ngay lập tức. Và nhiều tổ chức, đang sử dụng khuôn khổ này, họ có thể tận

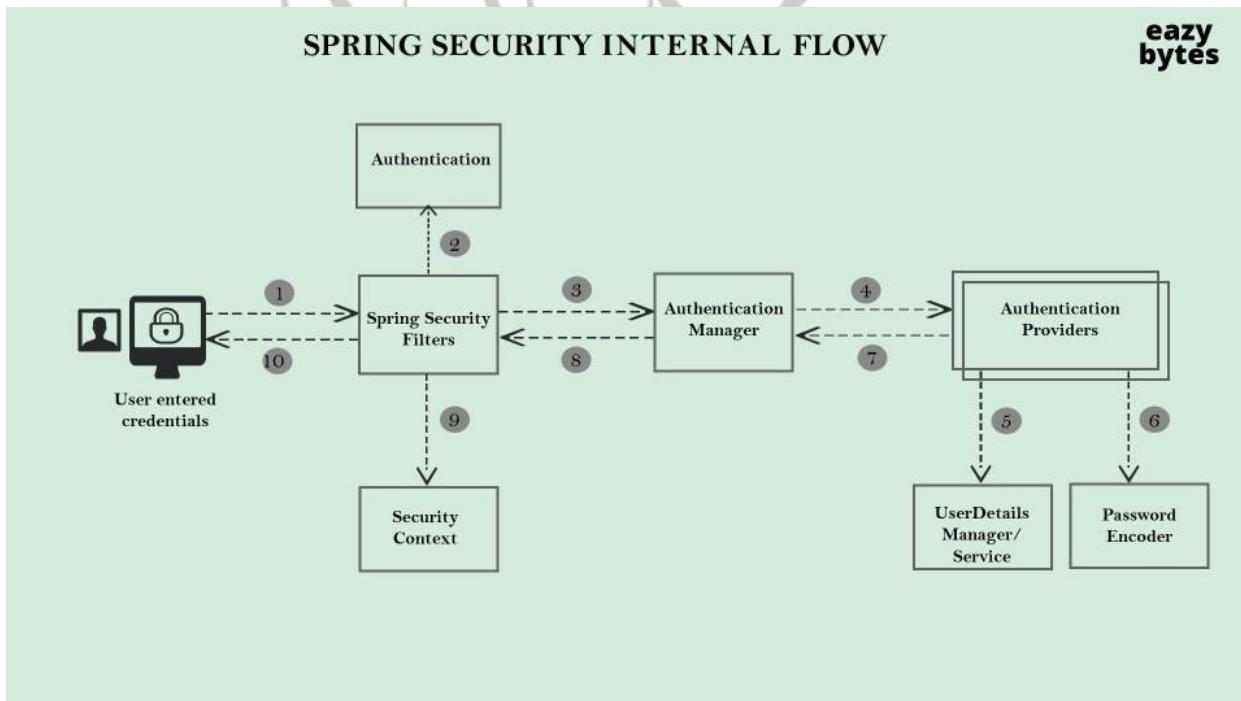
dụng những bản vá mới nhất này mà không đặt bất kỳ nỗ lực trùng lặp nào trong việc khắc phục những lỗ hổng mới này đang được xác định.

- Và với sự trợ giúp của Spring Security chúng tôi có thể bảo mật các trang web của mình, chúng tôi có thể bảo mật các API REST của mình, vi dịch vụ, và chúng tôi có thể thực thi các quy tắc authorization giống như các cơ quan quản lý cơ chế truy cập dựa trên vai trò. Chúng tôi cũng có thể có bảo mật cấp phương pháp như mức độ bảo mật thứ hai bên trong ứng dụng web của chúng tôi. Vì vậy, tất cả điều này có thể được đảm bảo với các cấu hình tối thiểu một cách dễ dàng. Vì vậy, bạn là một nhà phát triển, bạn có thể ngồi và thư giãn về mặt bảo mật khi bây giờ bạn đang sử dụng Spring Security framework.
- Và cuối cùng Spring Security cũng hỗ trợ các tiêu chuẩn bảo mật khác nhau trong việc thực hiện xác thực và authorization. Giống như bạn có thể thực hiện xác thực và authorization với sự giúp đỡ của HTTP đơn giản cơ bản sử dụng tên người dùng và mật khẩu, hoặc bạn có thể sử dụng các khái niệm nâng cao như JWT token, hoặc tới OpenID để bảo mật ứng dụng web của bạn.

## 6. Giới thiệu nhanh về Servlet & Filters

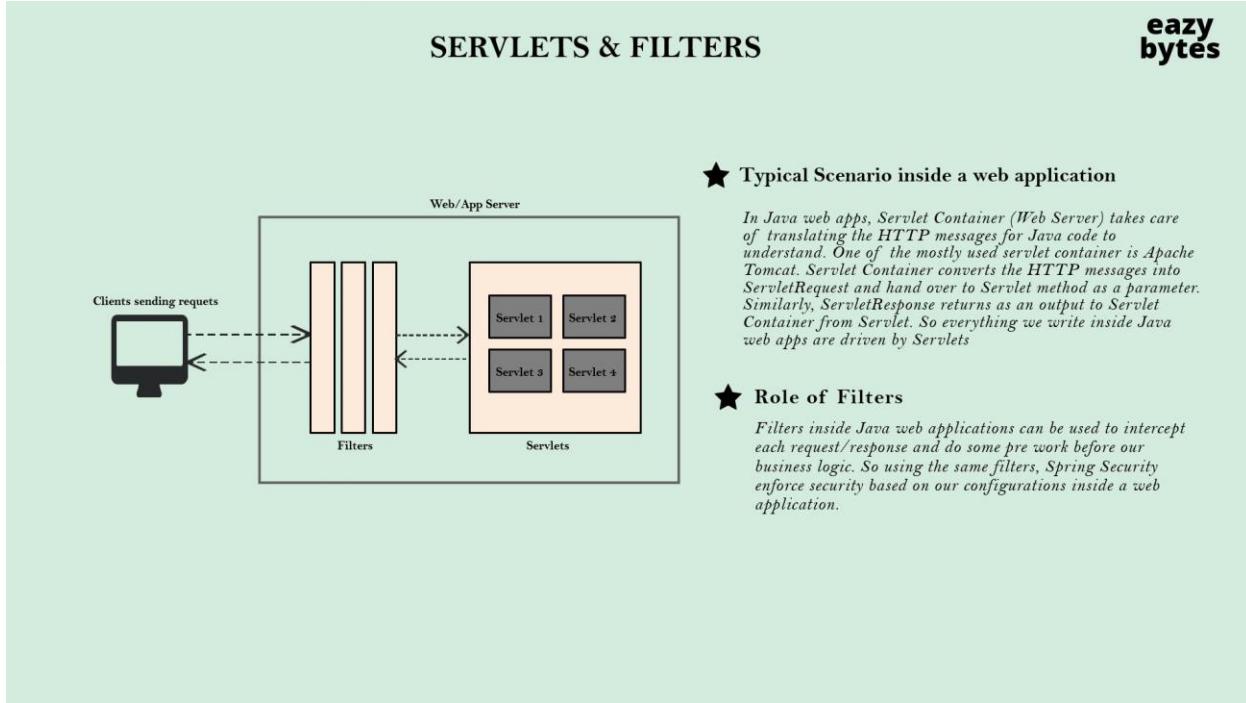
Theo mặc định, nó đang bảo vệ tất cả các API và nó cũng đang hiển thị một trang đăng nhập để user nhập thông tin đăng nhập của mình. Và một khi người dùng nhập thông tin đăng nhập của riêng mình từ yêu cầu thứ hai trở đi, Spring Security không hỏi thông tin đăng nhập lặp đi lặp lại nếu bạn cố gắng truy cập vào ứng dụng web nhiều lần sau lần đăng nhập đầu tiên. Vì vậy, đây là tất cả điều kỳ diệu mà Spring Security đang làm với các cấu hình mặc định.

Đây là luồng nội bộ Spring Security.



Các **Servlet & Filters** bên trong các ứng dụng web Java. Vì vậy, bất kỳ ứng dụng web Java nào nó chấp nhận yêu cầu và gửi phản hồi thông qua một giao thức HTTP, vì trình duyệt của chúng tôi chỉ có thể hiểu giao thức HTTP. Vì vậy, sử dụng giao thức HTTP này trình duyệt của tôi có thể gửi yêu cầu vào ứng

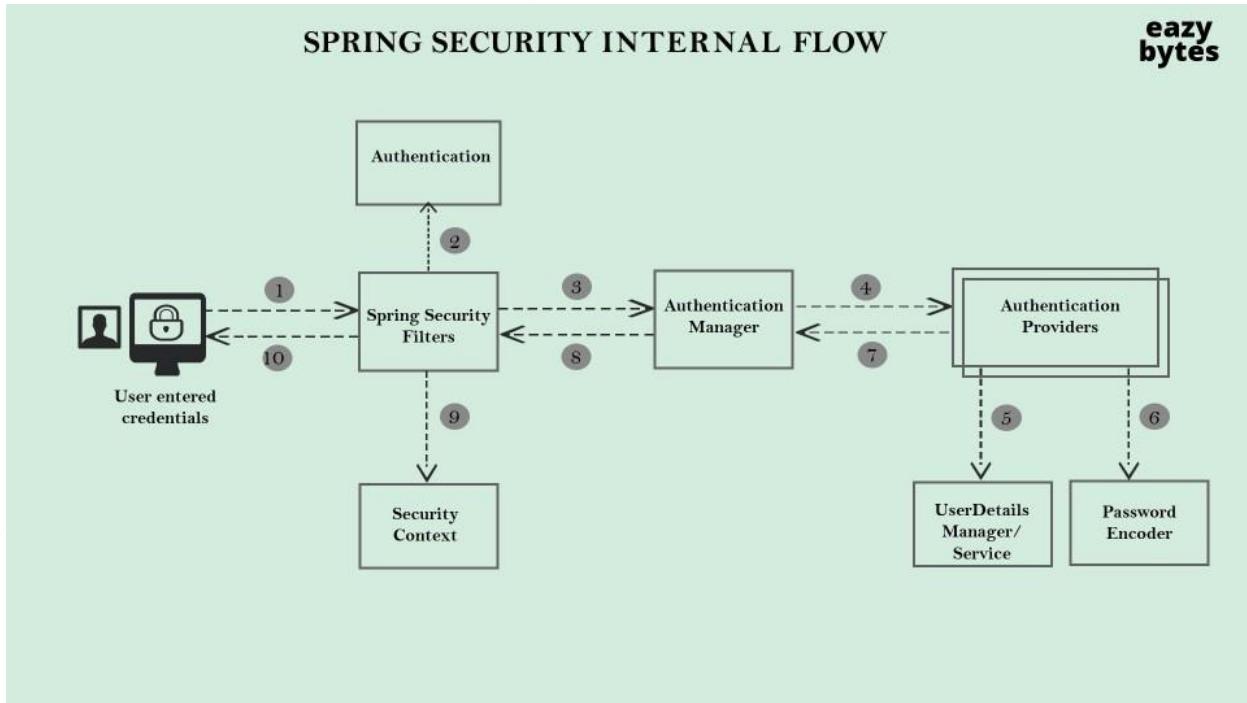
dụng web backend của tôi. Vì vì mã Java của tôi không thể hiểu được những Yêu cầu giao thức HTTP, có một người đàn ông trung niên đang ngồi giữa mã Java của tôi và trình duyệt của tôi. Vì vậy, người đàn ông trung gian này, chúng tôi gọi anh ta là solid container hoặc server web. Chúng tôi sử dụng các máy chủ web như Apache Tomcat, đó là solid container. Vì vậy, những gì solid container này sẽ làm là họ sẽ chuyển đổi HTTP token mà họ nhận được từ trình duyệt thành một HTTP solid request object.



Và cùng một đối tượng sẽ được trao cho lõi Java là khuôn khổ mà chúng tôi đang sử dụng bên trong ứng dụng web của chúng tôi. Và rất giống như vậy, khi chúng tôi đang cố gắng để gửi phản hồi trả lại trình duyệt, servlet container của tôi sẽ lấy đối tượng của phản hồi chắc chắn HTTP và nó sẽ lại chuyển đổi điều đó thành một HTTP token để trình duyệt của tôi có thể hiểu được. Vì vậy, tất cả điều này đang xảy ra bởi vì của khái niệm được gọi là servlet. Vì vậy, servlet đóng một vai trò rất quan trọng bên trong các ứng dụng web của chúng tôi, nhưng thật may nhung ngày nay không ai sử dụng chúng trực tiếp bởi vì chúng rất phức tạp. Vì vậy, để làm cho công việc của chúng tôi trở nên dễ dàng, Spring framework đã ra đời.. Và bên trong, spring boot và **spring framework** họ tạo ra các servlet. Và họ có trách nhiệm trong biện pháp xử lý logic phức tạp liên quan đến các servlet.

**Filters** là loại servlet đặc biệt mà chúng ta có thể sử dụng để đánh chặn mỗi và mọi yêu cầu đang đến đối với ứng dụng web của chúng tôi. Vì vậy, bên trong các bộ lọc này, chúng ta có thể định nghĩa bất kỳ logic trước hoặc công việc trước nào mà chúng tôi muốn làm trước khi actual business logic của chúng tôi thực thi. Như bạn có thể thấy ở đây client application của tôi đang cố gửi yêu cầu vào backend web application, mà tôi đã triển khai bên trong một solid container như Tomcat. Vì vậy, ở giữa máy client của tôi và các servlet thực tế, chúng ta có thể sử dụng các bộ lọc. Và những bộ lọc này chịu trách nhiệm để chặn từng và mọi yêu cầu của ứng dụng web. Và chúng thực thi logic mà chúng ta đã xác định bên trong các bộ lọc này.

## 7. Introduction to Spring Security Internal flow



Bước đầu tiên, người dùng sẽ nhập thông tin đăng nhập của riêng mình trên trang đăng nhập. Người dùng có thể thực hiện yêu cầu đến back-end server với sự trợ giúp của trình duyệt, với sự trợ giúp của ứng dụng di động. Người dùng sẽ gửi yêu cầu cùng với thông tin đăng nhập của mình đến ứng dụng web backend, nơi chúng tôi đang bảo mật nó với sự trợ giúp của Spring Security. Vì vậy, ngay khi solid container hoặc máy chủ Tomcat của tôi hoặc ứng dụng web của tôi nhận được yêu cầu đó.

Vì chúng tôi đã thêm Spring Security vào đường dẫn lớp của mình nên có một số bộ lọc Spring Security do chính nhóm Spring Security phát triển. Mục đích của các bộ lọc này là chúng sẽ giám sát từng và mọi yêu cầu đến với back-end server. Họ sẽ kiểm tra đường dẫn mà user của tôi đang cố truy cập. Vì vậy, dựa trên đường dẫn và cấu hình của tôi mà tôi đã thực hiện bên trong ứng dụng web, họ sẽ xác định xem đây là tài nguyên được bảo vệ hay đây là tài nguyên có thể truy cập công khai. Theo đó, các bộ lọc này sẽ quyết định xem có cần thực thi xác thực đối với user hay không. Họ sẽ hiển thị trang đăng nhập và điều này sẽ xảy ra lần đầu tiên khi user của tôi đang cố truy cập trang web được bảo vệ của tôi. Từ lần thứ hai trở đi, sau khi xác thực thành công, bộ lọc Spring Security của tôi cũng sẽ có logic để kiểm tra xem người dùng đã đăng nhập hay chưa. Theo đó, các bộ lọc này sẽ không thực thi bất kỳ trang đăng nhập nào. Họ sẽ chỉ tận dụng Session ID hiện có hoặc token hiện có đã được tạo trong giai đoạn đăng nhập ban đầu.

Trong Spring Security, có rất nhiều bộ lọc, chẳng hạn như hơn 20 bộ lọc có trong Spring Security. Mỗi bộ lọc sẽ có vai trò và trách nhiệm riêng. Và trong quá trình này, các bộ lọc Spring Security của tôi cũng sẽ trích xuất username và password mà user của tôi đang gửi và họ sẽ chuyển đổi tên người dùng và mật khẩu đó thành một đối tượng **Authentication** trong bước hai.

Bởi vì đối tượng **Authentication** là tiêu chuẩn cốt lõi trong Spring Security, để lưu trữ thông tin chi tiết về user của tôi. Vì vậy, trong bước hai, đối tượng **Authentication** của tôi sẽ chỉ có tên người dùng và thông tin xác thực. Và khi đối tượng **Authentication** này được hình thành, bộ lọc Spring Security

của tôi sẽ chuyển yêu cầu tới **Authentication Manager** (trình quản lý xác thực). Vì vậy, **Authentication Manager** là một interface hoặc class sẽ quản lý logic xác thực thực tế.

Vì vậy, những gì **Authentication Manager** này sẽ làm là, nó sẽ kiểm tra các nhà **Authentication providers** có sẵn bên trong ứng dụng web của tôi là gì. Và bên trong các authentication provider này, chúng ta có thể xác định logic xác thực thực tế. Cho dù tôi muốn xác thực thông tin đăng nhập của người dùng từ cơ sở dữ liệu hay từ máy chủ LDAP hay từ máy chủ authorization hay từ bộ đệm của tôi. Vì vậy, tất cả loại **business logic** đó chúng ta **có thể viết** bên trong **Authentication Providers**. Và nó cũng có thể là nhiều authentication provider, bạn có thể viết bên trong ứng dụng web của mình. Dựa trên yêu cầu của riêng bạn, bạn có thể xác định bất kỳ số lượng authentication provider nào. Vì vậy, trách nhiệm của **Authentication Manager** là xác định tất cả các authentication provider có sẵn cho yêu cầu cụ thể này là gì. Nó sẽ gửi yêu cầu đến các authentication provider này, cho đến khi đạt đến điểm xác thực thành công hoặc xác thực thất bại. Vì vậy, trong trường hợp này, trình quản lý xác thực của tôi sẽ không chỉ trả lại phản hồi cho user nói rằng đăng nhập của bạn không thành công. Thay vào đó, nó sẽ thử với tất cả các authentication provider có sẵn. Vì vậy, một khi nó thử tất cả các authentication provider và không có nhà cung cấp nào không được xác thực thành công thì chỉ có trình quản lý xác thực của tôi sẽ trả lời user nói rằng xác thực của bạn không thành công.

Và bên trong **Authentication Provider** này, tôi có thể viết logic xác thực thực tế của mình ngay lập tức hoặc tôi cũng có thể tận dụng mang lại các interface và class bảo mật được cung cấp, đó là trình User Detail và User Detail Service. Bởi vì bất cứ khi nào tôi muốn thực hiện một số xác thực, yêu cầu của tôi rõ ràng là tải thông tin chi tiết về người dùng từ một số hệ thống lưu trữ như cơ sở dữ liệu hoặc từ LDAP. Và khi tôi tải những UserDetails đó, tôi có thể so sánh những gì user được cung cấp và những gì tôi đã lưu trữ bên trong hệ thống lưu trữ của mình. Vì vậy, tất cả logic được xác định trước đó là logic chung được yêu cầu bởi phần lớn dự án, đã được cung cấp bên trong trình User Detail và interface User Detail Service này.

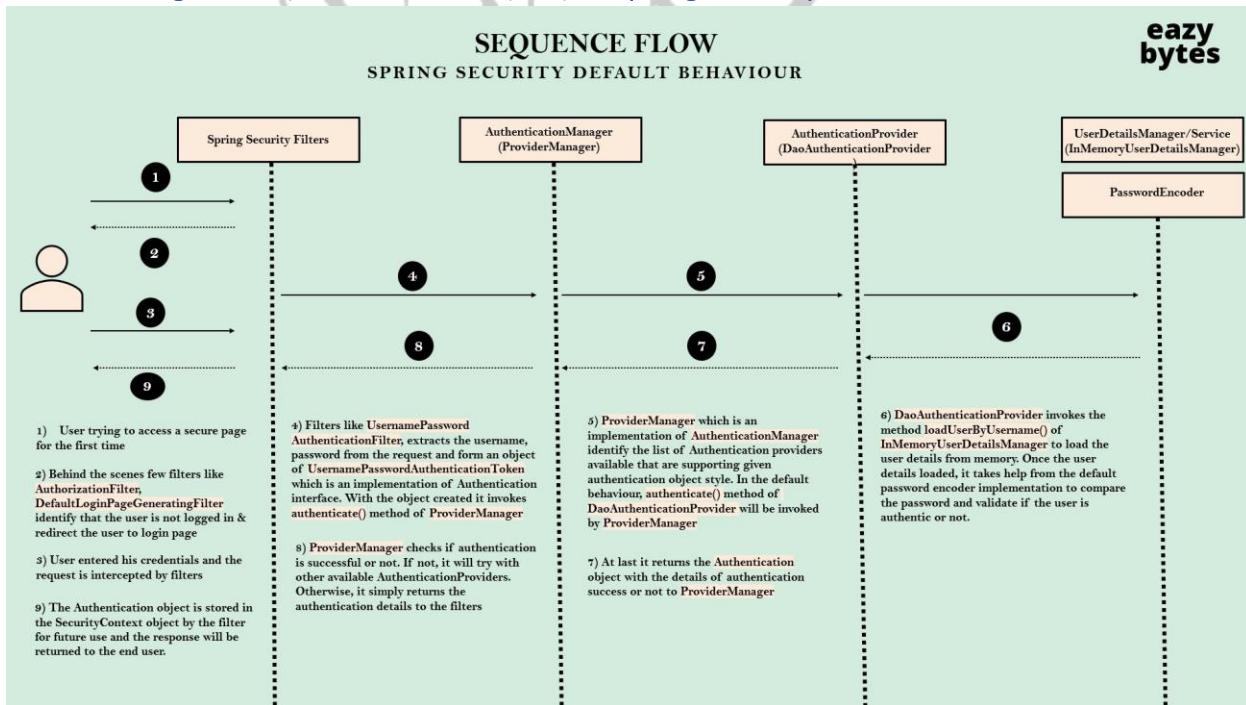
Nếu bạn muốn tận dụng các authentication provider được cung cấp mặc định bên trong Spring Security, thì đến lượt họ sẽ nhận trợ giúp từ User Detail Service và trình User Detail. Và vì chúng tôi có mật khẩu bên trong xác thực nên chúng tôi không muốn lưu trữ chúng hoặc thực hiện xác thực bằng văn bản thuần túy đối với mật khẩu vì điều đó rất dễ bị tổn thương về mặt bảo mật. Đó là lý do tại sao chúng ta phải luôn mã hóa hoặc băm mật khẩu của mình khi lưu trữ vào hệ thống lưu trữ hoặc khi chúng ta đang cố tìm nạp và xác thực bằng mật khẩu do user cung cấp. Vì vậy, User Detail Service của tôi hoặc trình **User Detail và Password Encoder**, chúng sẽ làm việc cùng nhau để xác định xem thông tin đăng nhập đã cho của user có hợp lệ hay không, sẽ cho authentication provider biết liệu nó có hợp lệ, thành công hay không. Và sau khi authentication provider của tôi xử lý xong, họ sẽ bàn giao lại cho Authentication Manager. Từ Authentication Manager, nó sẽ quay lại Spring Security filters. Và trước khi chúng tôi cố gắng xử lý phản hồi cho user, Spring Security filters của tôi, họ sẽ cố lưu trữ Authentication object mà họ đã tạo ở bước hai, bên trong Security Context.

Bây giờ ở bước chín, Authentication object mà chúng ta sẽ lưu trữ bên trong Security Context sẽ có các chi tiết như xác thực của tôi có thành công hay không, Session ID là gì? Vì vậy, vì Spring Security của tôi đang lưu trữ các chi tiết xác thực đó bên trong Security Context của bước chín từ yêu cầu thứ hai trở đi, nên nó sẽ không yêu cầu user nhập thông tin đăng nhập của anh ấy nếu xác thực của anh ấy thành công trong giai đoạn đăng nhập ban đầu. Vì vậy, đó là mục đích của bước chín. Vì vậy, khi việc lưu trữ đối tượng Authentication cùng với các chi tiết xác thực thành công được lưu trữ bên trong Security

Context, phản hồi từ tài nguyên được bảo vệ giống như có thể anh ta đang cố truy cập trang web hoặc dịch vụ. Vì vậy, phản hồi đó sẽ được gửi lại cho user ở bước 10.



## 8. Luồng trình tự của hành vi mặc định Spring Security



Đây là tất cả các thành phần quan trọng mà chúng ta có bên trong Spring Security framework. Và phía bên tay trái, chúng tôi có một user, theo sau là các Spring Security filters, tiếp theo là **Provider Manager**, đây là một triển khai của **Authentication Manager**.

Provider manager, chúng tôi có DAO authentication provider, đây là một triển khai của authentication provider. Và authentication provider này trong bộ sẽ nhận trợ giúp từ việc triển khai user details manager và password encoder.

Ở bước đầu tiên, user của tôi đang cố truy cập vào một đường dẫn API. Anh ta không biết liệu đường dẫn API được bảo mật hay công khai. Khi yêu cầu được bắt đầu bởi user trong bước thứ hai, các Spring Security filters của tôi như authorization filter, chúng sẽ chặn yêu cầu và nếu đường dẫn API là đường dẫn API được bảo mật, chúng sẽ hiển thị một trang đăng nhập để user nhập thông tin đăng nhập của mình. Sau khi user nhập thông tin đăng nhập của mình, yêu cầu sẽ bị chặn lại bởi các bộ lọc.

Vì vậy, lần này bộ lọc mà nó sẽ chặn là username-password authentication filter. Vì vậy, bộ lọc này trích xuất tên người dùng và mật khẩu từ yêu cầu và tạo username-password authentication token object, là một triển khai của interface **Authentication**. Bộ lọc này sẽ gọi phương thức **authenticate** bên trong Provider Manager. Provider Manager của tôi sẽ cố gắng xác định tất cả danh sách các authentication provider có thể áp dụng cho luồng của bạn và sẽ thử tất cả các authentication provider có sẵn bằng cách gọi phương thức xác thực có trong chúng. Và trong luồng mặc định, nó sẽ gọi DaoAuthenticationProvider và DaoAuthenticationProvider của tôi bên trong, nó sẽ gọi người dùng tải theo phương thức tên người dùng có sẵn bên trong lớp user details manager implementation trong bộ nhớ, đây là một triển khai của user details manager. Sau khi thông tin chi tiết về người dùng được tải từ bộ nhớ ứng dụng của tôi, mật khẩu sẽ được so sánh với sự trợ giúp của password encoder. Và nếu mật khẩu khớp, DAO authentication provider của tôi sẽ gửi phản hồi tới **Provider Manager** nói rằng xác thực thành công bằng cách điền các chi tiết cần thiết bên trong đối tượng Authentication. Và **Provider Manager** của tôi sẽ xác thực xem xác thực có thành công hay không. Nếu nó thành công, nó sẽ truyền tải điều tương tự đến các bộ lọc Spring Security. Nếu không, nó sẽ tiếp tục thử các authentication provider khác có sẵn.

Sau khi **Provider Manager** của tôi gửi phản hồi trở lại Spring Security filters, trong bước thứ chín, framework của tôi sẽ lưu trữ các chi tiết xác thực đó bên trong đối tượng Authentication để nếu user của tôi đang cố gọi một đường dẫn an toàn hơn, trong những trường hợp như vậy, framework của tôi không nên hỏi đi hỏi lại thông tin đăng nhập. Thay vào đó, nó sẽ tận dụng các chi tiết mà nó sẽ lưu trữ bên trong Security Context. Vì vậy, đây là luồng trình tự mà chúng ta đã thảo luận ở cấp độ cao.

## Section 2: Changing the default security configurations

### 1. Các dịch vụ REST backend cần thiết cho ứng dụng EazyBank

**BACKEND REST SERVICES  
FOR EAZYBANK APPLICATION**

**eazy  
bytes**

**Services with out any security**

*/contact – This service should accept the details from the Contact Us page in the UI and save to the DB.*

*/notices – This service should send the notice details from the DB to the 'NOTICES' page in the UI*

**Services with security**

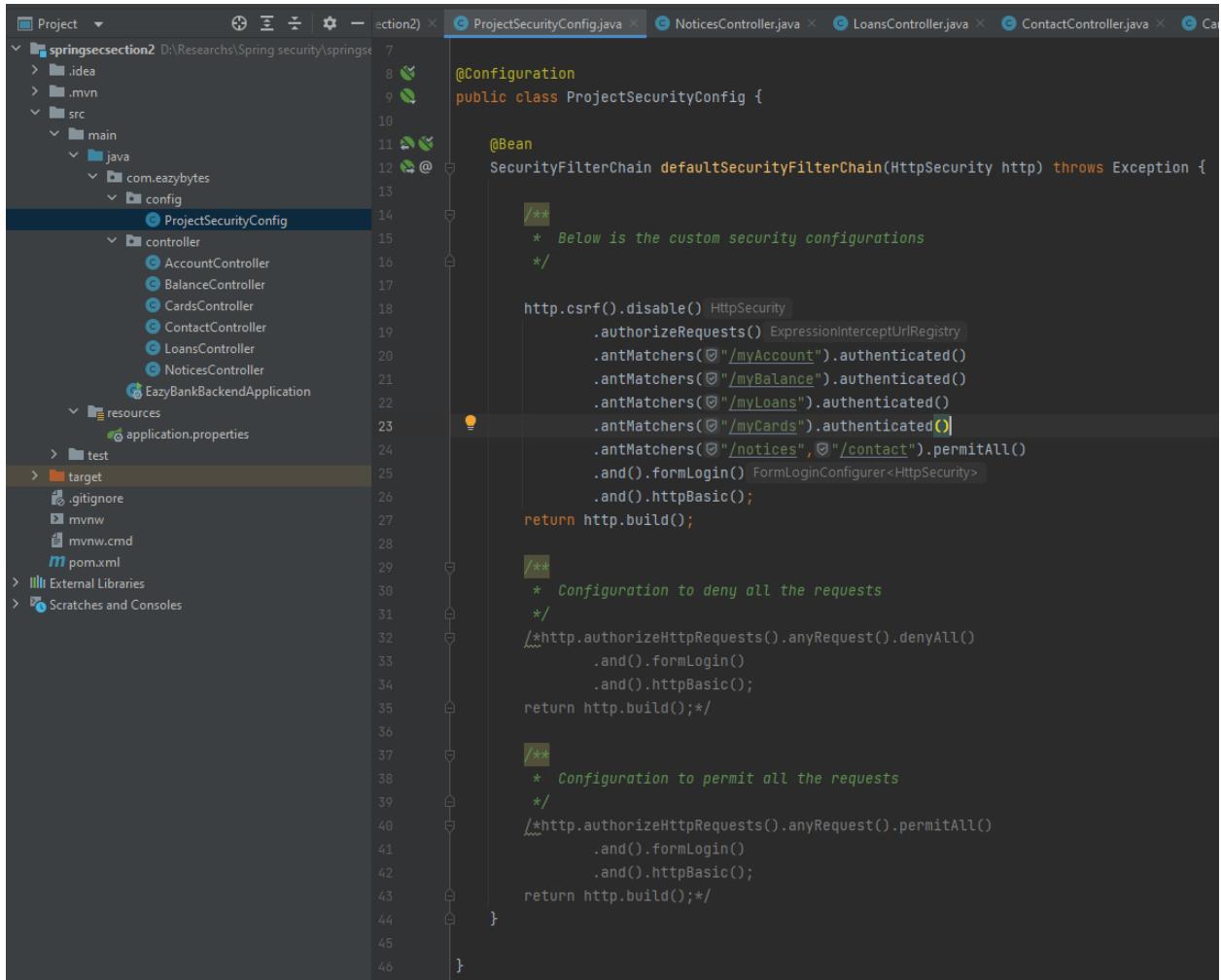
*/myAccount – This service should send the account details of the logged in user from the DB to the UI*

*/myBalance – This service should send the balance and transaction details of the logged in user from the DB to the UI*

*/myLoans – This service should send the loan details of the logged in user from the DB to the UI*

*/myCards – This service should send the card details of the logged in user from the DB to the UI*

- Viết API và config các Spring Security cơ bản



```
Project SecurityConfig.java NoticesController.java LoansController.java ContactController.java Ca

springsecsection2 D:\Researchs\Spring security\springsecsection2
src
  main
    java
      com.eazybytes
        config
          ProjectSecurityConfig
        controller
          AccountController
          BalanceController
          CardsController
          ContactController
          LoansController
          NoticesController
        EazyBankBackendApplication
      resources
        application.properties
  test
  target
  .gitignore
  mvnw
  mvnw.cmd
  pom.xml

ProjectSecurityConfig.java

@Configuration
public class ProjectSecurityConfig {

    @Bean
    SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {

        /**
         * Below is the custom security configurations
         */

        http.csrf().disable()
            .authorizeRequests()
                .antMatchers("/myAccount").authenticated()
                .antMatchers("/myBalance").authenticated()
                .antMatchers("/myLoans").authenticated()
                .antMatchers("/myCards").authenticated()
                .antMatchers("/notices", "/contact").permitAll()
                .and().formLogin()
                    .and().httpBasic();
        return http.build();

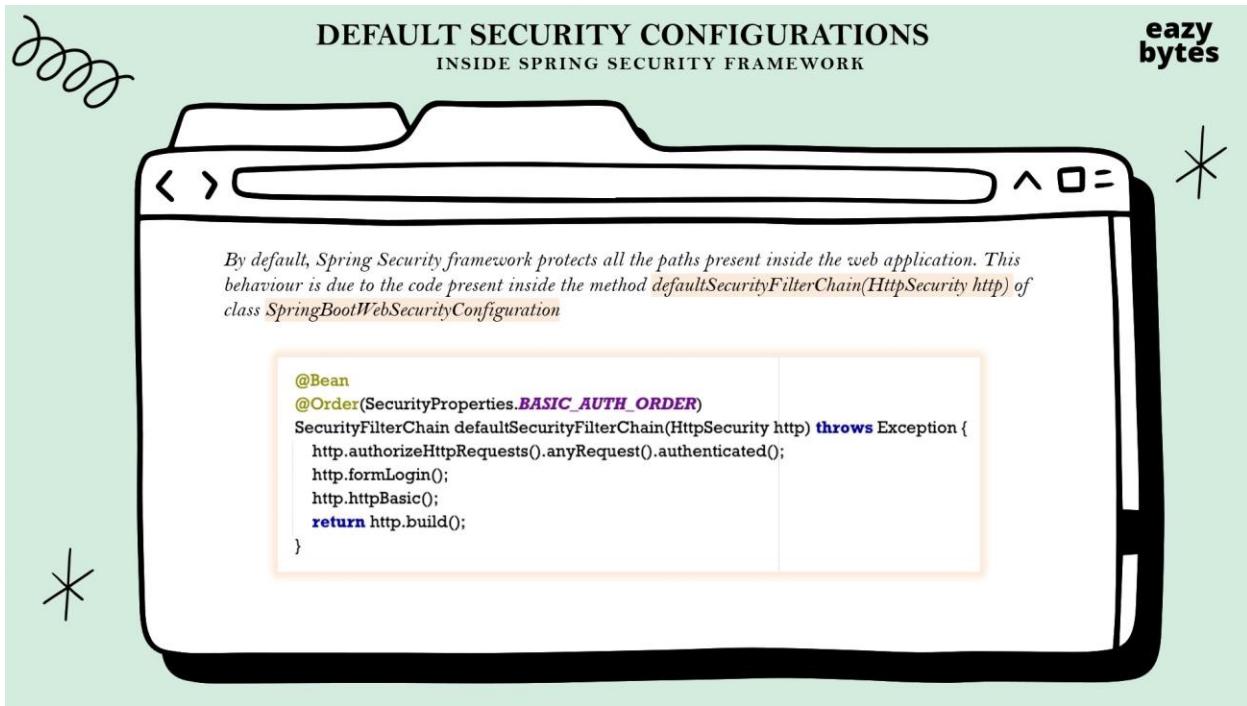
    }

    /**
     * Configuration to deny all the requests
     */
    /*http.authorizeHttpRequests().anyRequest().denyAll()
        .and().formLogin()
        .and().httpBasic();
    return http.build();*/

    /**
     * Configuration to permit all the requests
     */
    /*http.authorizeHttpRequests().anyRequest().permitAll()
        .and().formLogin()
        .and().httpBasic();
    return http.build();*/
    }
}
```

## 2. Kiểm tra cấu hình mặc định bên trong Spring Security framework

Theo mặc định, tất cả API REST của chúng tôi được bảo mật bởi Spring Security framework. Tuy nhiên, chúng tôi có các yêu cầu bảo mật tùy chỉnh của riêng mình. Giống như, chúng tôi muốn một số URL được bảo mật và các URL còn lại không được bảo mật. Vì vậy, để xác định tất cả các yêu cầu bảo mật tùy chỉnh này, trước tiên chúng ta cần hiểu bên trong Spring Security framework, logic nào chịu trách nhiệm bảo vệ tất cả các URL theo mặc định



Giống như bạn có thể thấy ở đây, bên trong framework, chúng ta có một lớp có tên là **Spring Boot Web Security Configuration**. Bên trong lớp này, chúng tôi có **Security Filter Chain** mặc định của phương thức chấp nhận HttpSecurity làm tham số đầu vào.

Bây giờ, đến với phương thức là **defaultSecurity FilterChain**. Bên trong phương thức này, như bạn có thể thấy, nó đang chấp nhận một tham số đầu vào có tên HttpSecurity và sử dụng HTTP Security, này, chúng tôi đang gọi một phương thức có tên là **authorizeHttpRequest**. Sau đó, chúng tôi yêu cầu bất kỳ yêu cầu nào đến với ứng dụng web của tôi phải được xác thực. Bằng mã này, chúng tôi muốn bất kỳ yêu cầu nào đến **Spring Boot Web Application** của chúng tôi phải được xác thực. Và, yêu cầu có thể đến thông qua ứng dụng UI hoặc thông qua một số lệnh gọi **REST API**. Đó là lý do tại sao chúng ta cũng cần gọi **formLogin** này và các phương thức HttpBasic với sự trợ giúp của các **HTTP security** này.

Với hai dòng này, điều mà **framework** đang cố gắng truyền tải ở đây là yêu cầu có thể đến thông qua **HTML form** hoặc thông qua **UI application**. Những yêu cầu như vậy tôi muốn được xác thực. Trong khi đó, nếu yêu cầu đến dưới dạng lệnh gọi **REST API** hoặc thông qua ứng dụng Postman, trong các tình huống như vậy, chúng tôi sẽ gửi yêu cầu cùng với thông tin đăng nhập của người dùng bằng cách tuân theo các tiêu chuẩn cơ bản của HTTP.

Sau ba dòng quan trọng này, bạn có thể thấy bằng cách sử dụng cùng một **HTTP variable**, ý tôi là làm việc với một phương thức có tên là **build** và phương thức **build** này sẽ trả về một đối tượng thuộc loại **Security Filter Chain**. Và bạn có thể thấy có một annotation **Bean** trên phương thức này, có nghĩa là, bất kỳ đối tượng nào mà phương thức này sẽ trả về, **Spring framework** sẽ duy trì nó dưới dạng Bean bên trong ngữ cảnh Spring. Nó dựa trên chiến lược đàm phán nội dung của **Spring Security's** để xác định loại xác thực nào sẽ sử dụng. Điều đó có nghĩa là, yêu cầu có thể đến thông qua **form login** hoặc thông qua **HTTP cơ bản**. Theo đó, **framework** sẽ cố gắng xác thực yêu cầu.

Bất cứ khi nào chúng tôi muốn xác định các yêu cầu bảo mật của riêng mình với sự trợ giúp của **Spring Security's**, chúng tôi cần tạo **Security Filter Chain** loại Bean như bạn có thể thấy bên trong phương thức này. Vì vậy, bước tiếp theo, những gì chúng ta có thể làm ở đây là cố gắng xác định bean của **Security Filter Chain**. Tương tự, tôi sẽ tạo new package. Tên package sẽ là config. Tôi sẽ tạo một lớp mới với tên **Project Security Config**. Vì vậy, khi tôi nhập tên này, một lớp mới sẽ được tạo. Trên đầu lớp này, tôi muốn xác định một annotation **Configuration**. Khi chúng tôi xác định annotation này, đó là dấu hiệu cho **Spring Security framework** rằng chúng tôi có một số cấu hình nhất định được xác định bên trong lớp này. Vì vậy, trong quá trình khởi động, nó sẽ quét tất cả các bean mà chúng ta đã khai báo bên trong lớp này.

```
@Configuration
public class ProjectSecurityConfig {

    @Bean
    SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {

        http.authorizeHttpRequests().anyRequest().permitAll() AuthorizationManagerRequestMat...
            .and().formLogin() FormLoginConfigurer<HttpSecurity>
            .and().httpBasic();
        return http.build();

    }
}
```

Với điều này, chúng tôi đã tạo Bean of **Security Filter Chain** của riêng mình.

### 3. Modifying the code as per my custom requirements – Tùy chỉnh code theo yêu cầu

```
4. @Configuration
public class ProjectSecurityConfig {

    @Bean
    SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http)
throws Exception {

    /**
     * Below is the custom security configurations
     */

    http.csrf().disable()
        .authorizeRequests()
    .antMatchers("/myAccount", "/myBalance", "/myBalance", "/myLoans", "/myCard
s").authenticated()
        .antMatchers("/notices", "/contact").permitAll()
        .and().formLogin()
        .and().httpBasic();
    return http.build();

    /**

```

```
    * Configuration to deny all the requests
    */
    /*http.authorizeHttpRequests().anyRequest().denyAll()
     .and().formLogin()
     .and().httpBasic();
    return http.build();*/

    /**
     * Configuration to permit all the requests
     */
    /*http.authorizeHttpRequests().anyRequest().permitAll()
     .and().formLogin()
     .and().httpBasic();
    return http.build();*/
}

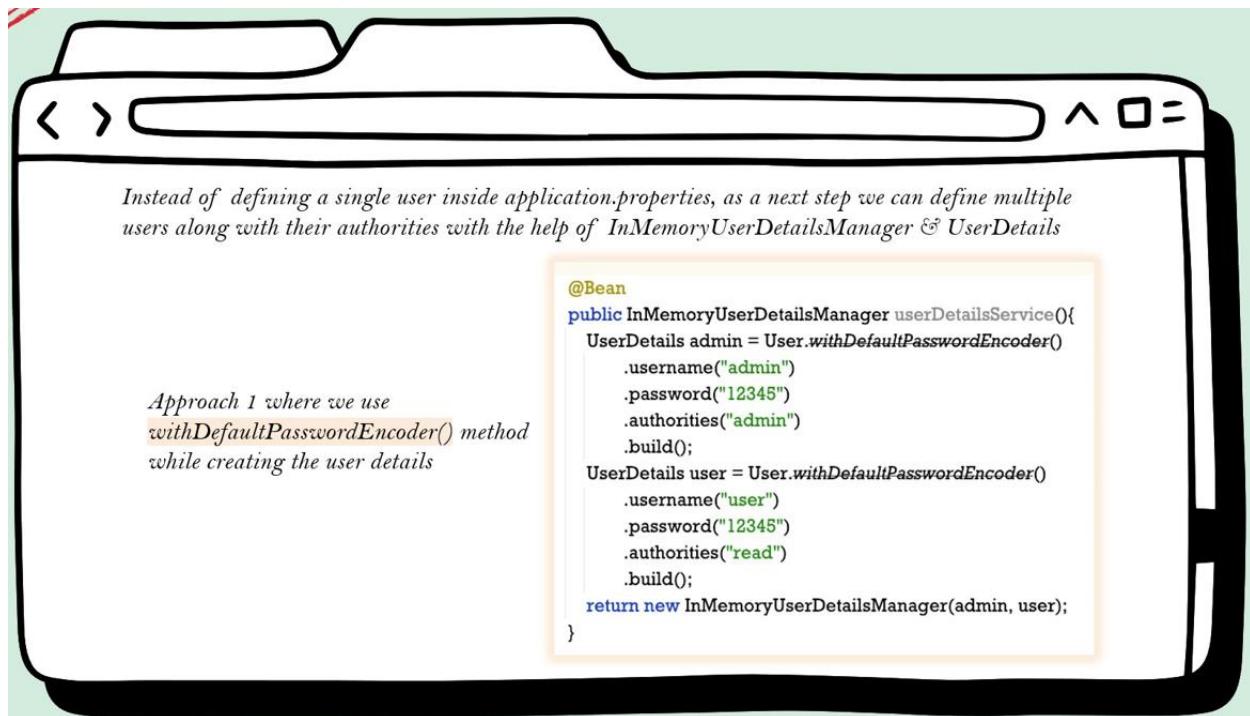
}
```



## Section 3: Defining & Managing Users

### 1. Configuring users using InMemoryUserDetailsManager – Approach

Phương pháp này vui lòng chỉ sử dụng cho các ứng dụng non-product hoặc cho các ứng dụng có mức độ nghiêm trọng thấp hoặc cho một số ứng dụng demo mà bạn đang thực hiện cho khách hàng của mình.



Vì vậy, giống như bạn có thể thấy ở đây, để tạo người dùng, nhiều người dùng bên trong bộ nhớ của ứng dụng web Spring Boot của bạn, bạn cần tạo Bean kiểu InMemoryUserDetailsManager, đây một trong những lớp mở rộng interface userDetailsManager. Vì vậy, khi chúng tôi cố gắng tạo một đối tượng của InMemoryUserDetailsManager này, chúng tôi có thể tạo bao nhiêu người dùng tùy thích cho hàm tạo của lớp này.

Ngay trước khi trả về một đối tượng mới của lớp này, tôi chỉ chuyển thông tin chi tiết về admin và user đã tạo ở đầu phương thức. Vì vậy, câu hỏi tiếp theo mà bạn có thể có ở đây là "làm cách nào tôi có thể tạo cho người dùng tên người dùng, mật khẩu và quyền hạn của họ?" Vì vậy, tương tự, chúng tôi có sẵn một lớp User bên trong Spring Security Framework. Trước tiên, sử dụng cùng một lớp User, chúng ta có thể đặt username và password mà chúng ta phải sử dụng là gì. Chúng ta có thể gọi phương thức Build để xây dựng UserDetails và cùng UserDetails mà chúng ta có thể chuyển đến hàm tạo InMemoryUserDetailsManager.

Sử dụng InMemoryUserDetailsManager để tạo dữ liệu:

```
//ProjectSecurityConfig.java
/** *Approach 1 where we use withDefaultPasswordEncoder() method
while creating the user details*/
@Bean
public InMemoryUserDetailsManager userDetailsService() {
```

```

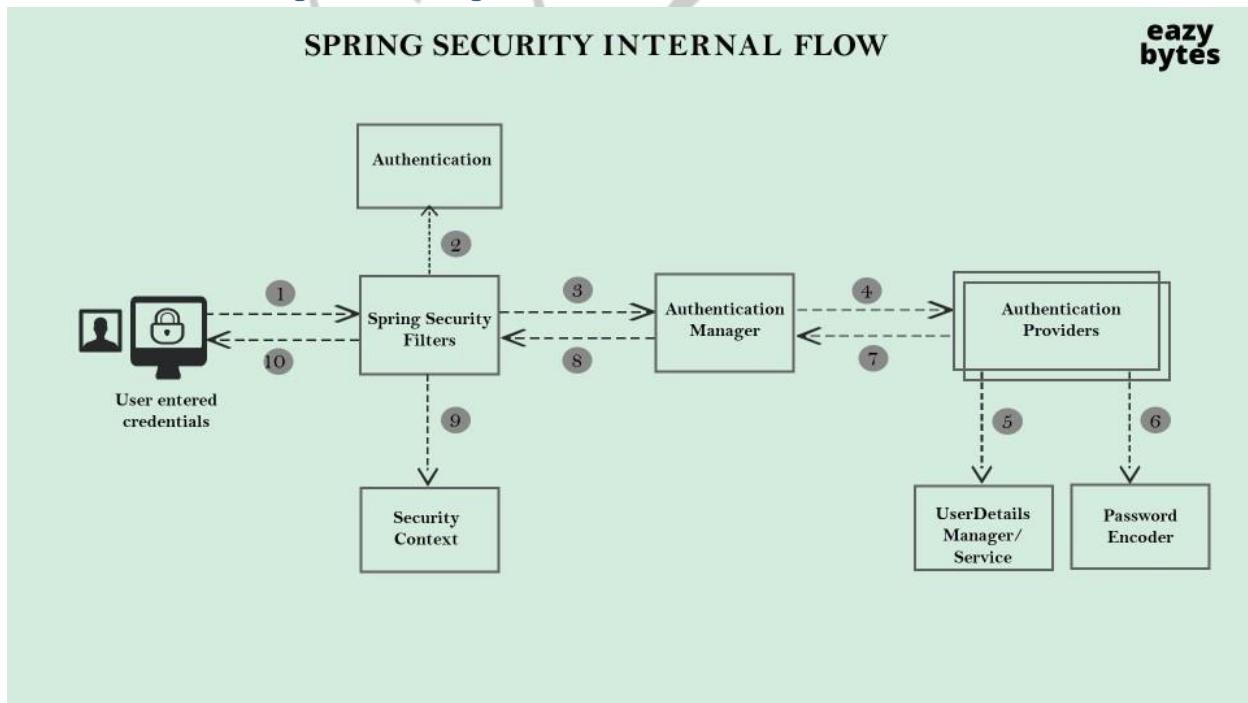
UserDetails admin = User.withDefaultPasswordEncoder()
    .username("admin")
    .password("12345")
    .authorities("admin")
    .build();
UserDetails user = User.withDefaultPasswordEncoder()
    .username("user")
    .password("12345")
    .authorities("read")
    .build();
return new InMemoryUserDetailsManager(admin, user);

}

/*Approach 2 where we use NoOpPasswordEncoder Bean
while creating the user details*/
@Bean
public InMemoryUserDetailsManager userDetailsService() {
    UserDetails admin = User.withUsername("admin")
        .password("12345")
        .authorities("admin")
        .build();
    UserDetails user = User.withUsername("user")
        .password("12345")
        .authorities("read")
        .build();
    return new InMemoryUserDetailsManager(admin, user);
}

```

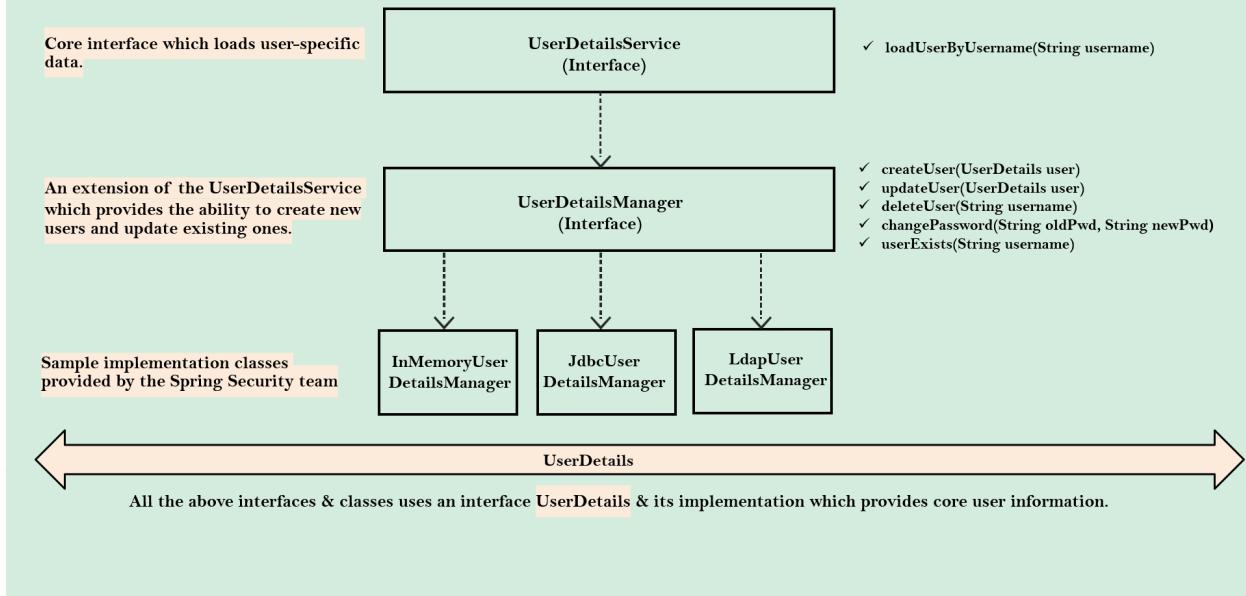
## 2. Understanding User Management interfaces and Classes



## USER MANAGEMENT

### IMPORTANT CLASSES & INTERFACES

eazy  
bytes



Như bạn có thể thấy ở đây, interface đầu tiên mà chúng tôi có là **UserDetailsService**. Service này sẽ có một phương thức trừu tượng với tên **loadUserByUsername**. Hoạt động đầu tiên mà bạn muốn làm là, bạn muốn tải UserDetails để bạn có thể so sánh với các UserDetails mà bạn có bên trong hệ thống lưu trữ và những gì user đã nhập bên trong trình duyệt. Vì đây là một kịch bản rất phổ biến. Spring Security đã tạo ra một interface riêng có một phương pháp trừu tượng duy nhất với tên **loadUserByUsername**.

Tại sao chúng tôi đang tải UserDetails từ hệ thống lưu trữ chỉ với tên người dùng? Tại sao không có cả tên người dùng và mật khẩu? Có hai lý do cho nó. Lý do đầu tiên là, chúng ta không bao giờ nên gửi mật khẩu một cách không cần thiết. Nếu bạn bắt đầu sử dụng mật khẩu bên trong điều kiện web của bạn và các câu lệnh SQL của bạn, bạn đang gửi mật khẩu thực tế của người dùng vào **database server** và **locks of the database**, đó không phải là cách tiếp cận được khuyến nghị.

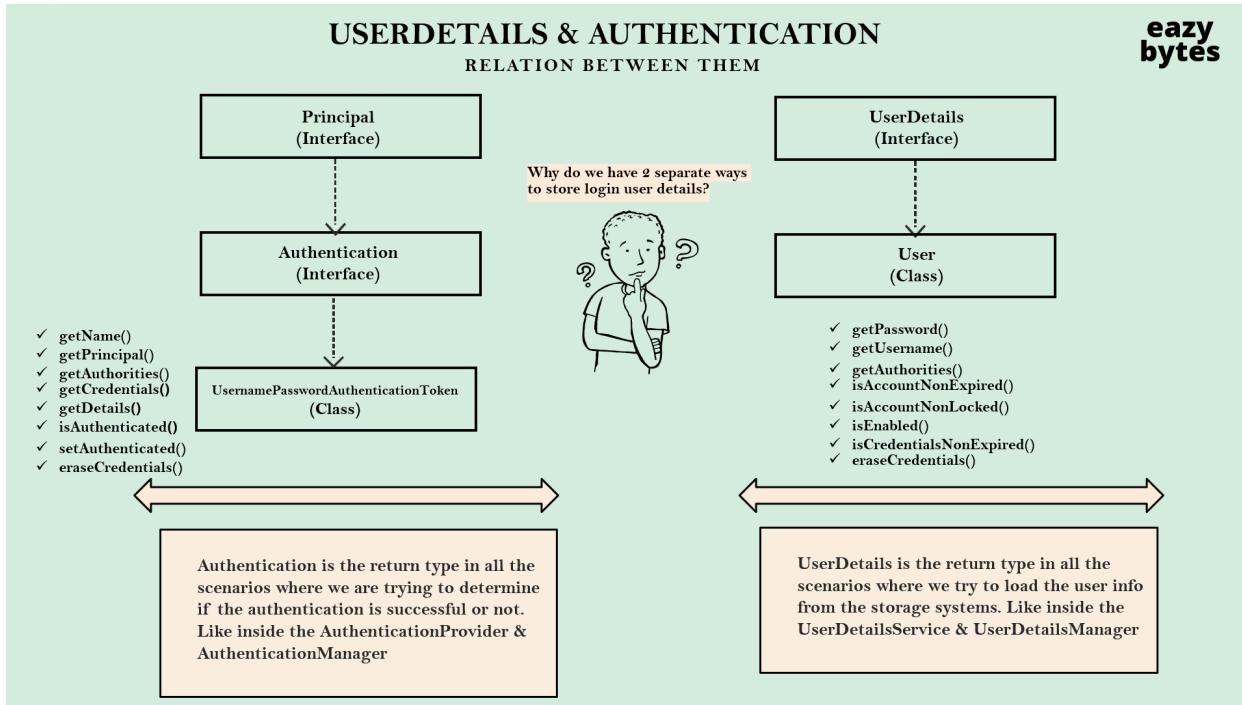
Vì vậy, đó là lý do tại sao những gì Spring Security đề xuất là, trước tiên, bạn chỉ cần tải UserDetails với sự giúp đỡ của tên người dùng. Sau này, bạn có thể có một logic của việc so sánh các mật khẩu. Và theo đó, chúng ta có thể thực hiện một hành động xác thực có thành công hay không.

Chúng ta có thêm một interface, đang mở rộng UserDetailsService. Vì vậy, tên interface này là **UserDetailsManager**, interface này sẽ giúp chúng ta để quản lý UserDetails. Giống như ngoài việc tải các chi tiết từ cơ sở dữ liệu hoặc từ hệ thống lưu trữ, đôi khi chúng tôi có thể muốn tạo một người dùng hoàn toàn mới. Chúng tôi muốn cập nhật các UserDetails. Chúng tôi muốn xóa UserDetails, đổi mật khẩu. Vì vậy, để phù hợp với tất cả các tình huống đó, interface UserDetailsManager này, nó có tất cả các phương thức như createUser, updateUser, deleteUser.v.v. Và dĩ nhiên, nó cũng kế thừa phương thức loadUserByUsername từ interface UserDetails.

Và Spring Security **cho các nhà phát triển để viết logic của riêng họ**. Thay vào đó, những gì họ đã làm là, họ đã tạo ra một số triển khai mẫu của UserDetailsManager. Vì vậy, những triển khai mẫu này

giống như **InMemoryUserDetailsManager**, mà chúng ta đã sử dụng trong các bài giảng trước. Và cái thứ hai là **JdbcUserDetailsManager**, mà chúng ta có thể sử dụng bất cứ khi nào chúng ta đang cố gắng để lấy các UserDetails từ cơ sở dữ liệu. Vì vậy, khi chúng ta đang giao dịch với cơ sở dữ liệu trong quá trình xác thực, chúng ta chỉ có thể sử dụng **JdbcUserDetailsManager**. Và rất tương tự, nếu bạn đang sử dụng máy chủ Ldap để lưu trữ các UserDetails, thì bạn có thể sử dụng LdapUserDetailsManager. Vì vậy, đây là ba lớp triển khai quan trọng được cung cấp bởi Spring Security framework. Và ở một mức độ lớn, chúng ta có thể tận dụng chúng. Nhưng nếu bạn có logic xác thực của riêng mình, mọi thứ bạn muốn tự viết, thì chắc chắn điều bạn có thể làm là, bạn có thể xác định **Authentication Provider** của riêng mình. Bên trong Authentication Provider của bạn, bạn có thể viết logic của riêng bạn. Nhưng kể từ bây giờ, chúng tôi sẽ với Authentication Provider mặc định đó là **DaoAuthenticationProvider**, nó đang nhận sự giúp đỡ từ việc triển khai của UserDetailsManager và UserDetailsService đó là InMemoryUserDetailsManager, JdbcUserDetailsManager và LdapUserDetailsManager.

### 3. Deep Dive of UserDetails Interface & User class



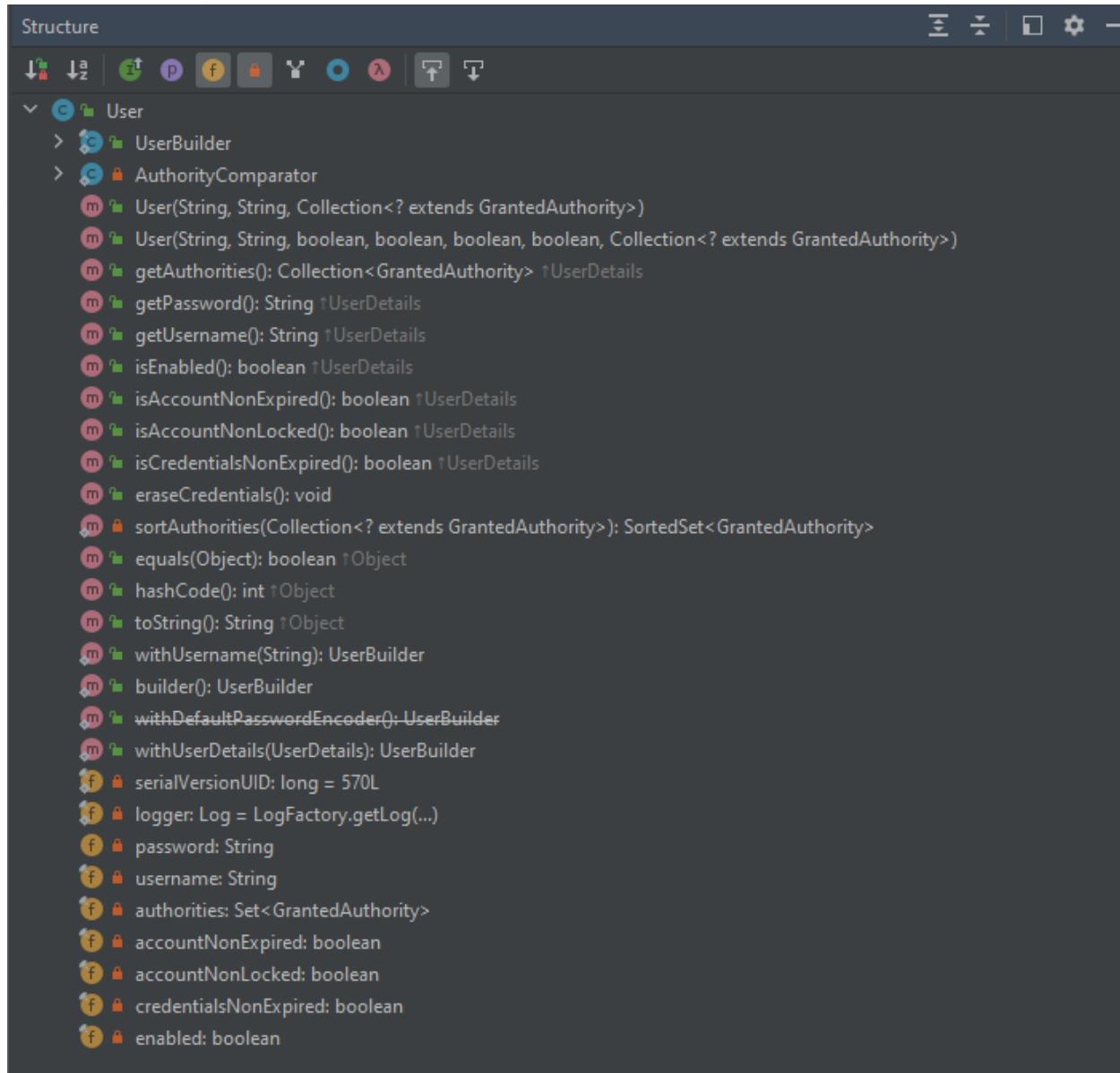
Như bạn có thể thấy ở đây, đây là interface 'UserDetails'. Vì nó là một interface nên nó sẽ có tất cả các phương thức trừu tượng.

Các phương thức có trong interface:

- Phương thức đầu tiên là 'getAuthorities', chứa danh sách các quyền hoặc vai trò của user, để chúng ta có thể sử dụng chúng để triển khai cơ chế truy cập dựa trên vai trò hoặc authorization mà chúng ta sẽ thảo luận trong các bài giảng sắp tới.
- Còn 'getPassword', 'getUsername' thì chúng sẽ trả về mật khẩu và tên người dùng của user.
- Nó cũng hỗ trợ một số phương pháp khác giúp chúng tôi xác định xem tài khoản người dùng đã hết hạn chưa, tài khoản người dùng có bị khóa hay không hoặc liệu thông tin đăng nhập của người dùng đã hết hạn hay tài khoản người dùng đã được bật chưa hoặc bị vô hiệu hóa. Bây giờ đến với các

phương thức còn lại mà chúng ta có như 'isAccountNonExpired'. Vì vậy, bất cứ khi nào giá trị này là đúng, điều đó có nghĩa là tài khoản người dùng hợp lệ, nó chưa hết hạn, trong khi sai cho biết rằng tài khoản người dùng không còn hiệu lực vì tài khoản đã hết hạn. Vì vậy, điều tương tự cũng áp dụng cho 'isAccountNonLocked', 'isCredentialsNonExpired' và 'isEnabled'.

Spring Security, không chỉ dừng lại ở việc xác định một interface. Họ cũng đưa ra một số triển khai mẫu của interface này mà chúng ta có thể sử dụng trong các dự án của mình và trong quá trình phát triển hàng ngày của chúng ta. Vì vậy, giống như bạn có thể thấy ở đây, họ đang nói hãy đi và giới thiệu lớp **User** để triển khai tham khảo. Nếu bạn muốn, chúng tôi có thể viết triển khai của riêng mình.



Tùy thuộc vào việc bạn muốn sử dụng **User** hay bạn muốn viết lớp triển khai 'UserDetails' của riêng mình. Bên trong lớp **User** này, tất cả các phương thức mà chúng ta đã thấy bên trong interface 'UserDetails', cùng một phương thức có thể đã được ghi đè bên trong lớp này. Có một phương thức

'getPassword' chỉ trả về giá trị mật khẩu nhưng ở đây bạn có thể có một câu hỏi như giá trị mật khẩu này sẽ điền vào nó như thế nào.

```
public User(String username, String password, Collection<? extends GrantedAuthority> authorities) {  
    this(username, password, enabled: true, accountNonExpired: true, credentialsNonExpired: true, accountNonLocked: true, authorities);  
}
```

Vì vậy, nếu bạn kiểm tra các hàm tạo của lớp **User** này, bất kỳ ai cũng có thể tạo một đối tượng của lớp này bằng cách sử dụng hai hàm tạo này, một là không chuyển bất kỳ giá trị Boolean nào, bạn chỉ cần chuyển tên người dùng, mật khẩu và quyền hạn và còn lại tất cả các giá trị Boolean, chúng sẽ chỉ điền true, true, true cho biết rằng tài khoản chưa hết hạn, chưa bị khóa và chưa bị vô hiệu hóa.

Nếu bạn kiểm tra phương thức mà chúng ta có bên trong 'InMemoryUserDetailsManager', thì có một phương thức gọi là 'LoadUserByUsername' sẽ được gọi bởi 'AuthenticationProvider'.

```
public UserDetails loadUserByUsername(String username) throws  
UsernameNotFoundException {  
    UserDetails user = (UserDetails)this.users.get(username.toLowerCase());  
    if (user == null) {  
        throw new UsernameNotFoundException(username);  
    } else {  
        return new User(user.getUsername(), user.getPassword(),  
user.isEnabled(), user.isAccountNonExpired(), user.isCredentialsNonExpired(),  
user.isAccountNonLocked(), user.getAuthorities());  
    }  
}
```

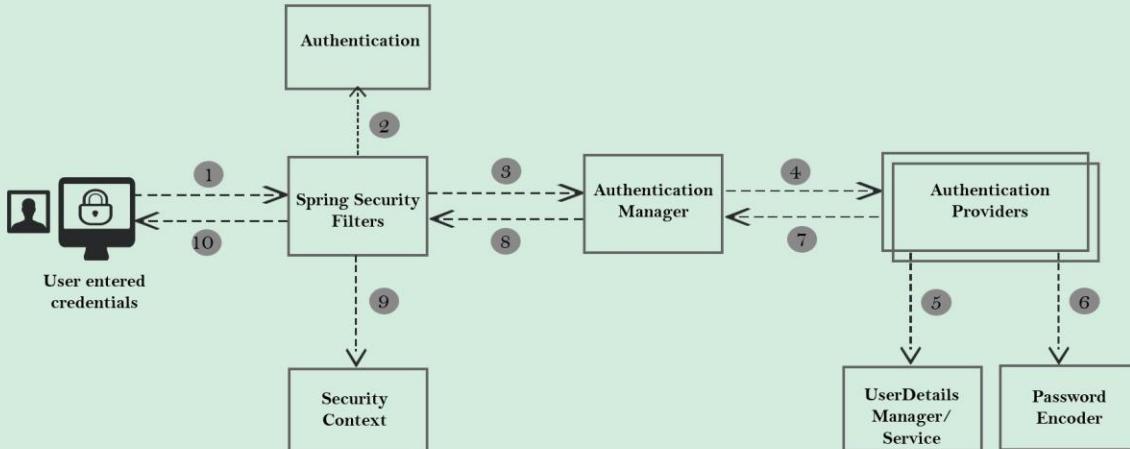
Vì vậy, ở đây bạn có thể thấy dựa trên tên người dùng, chúng tôi đang cố tải các chi tiết từ map. Map này có thể đã được tạo trong quá trình khởi động ứng dụng của tôi, dựa trên 'UserDetails' mà tôi đã định cấu hình và lưu trữ bên trong bộ nhớ của ứng dụng web.

Trong luồng nội bộ, tôi đã nói tất cả các chi tiết xác thực như liệu xác thực có thành công hay không và tên người dùng như 'UserDetails', tất cả các chi tiết đó sẽ được lưu trữ bên trong Authentication object. Vì vậy, đó là những gì tôi đã nói. Và bạn cũng có thể thấy ở bước hai, mọi thứ được điền vào đối tượng **Authentication** và đối tượng tương tự sẽ được lưu trữ bên trong Security Context sau khi xác thực thành công.

Vì vậy, bây giờ ở đây bạn có thể có một câu hỏi như tại sao lại có đối tượng **Authentication**, tại sao lại có interface **UserDetails** và lớp **User**, mối quan hệ giữa chúng là gì?

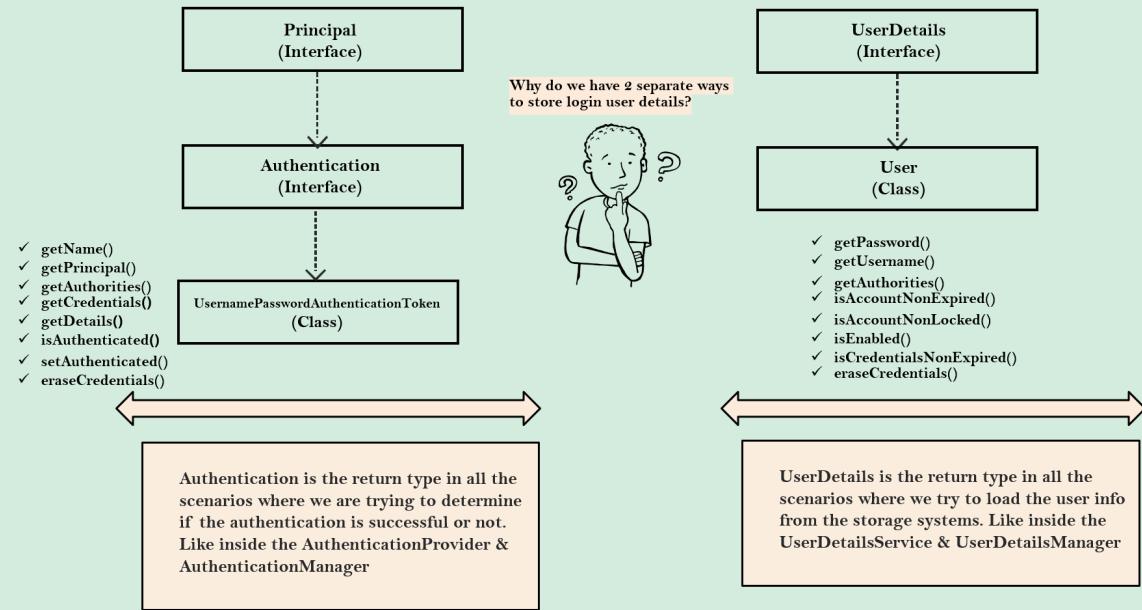
## SPRING SECURITY INTERNAL FLOW

eazy  
bytes



## USERDETAILS & AUTHENTICATION RELATION BETWEEN THEM

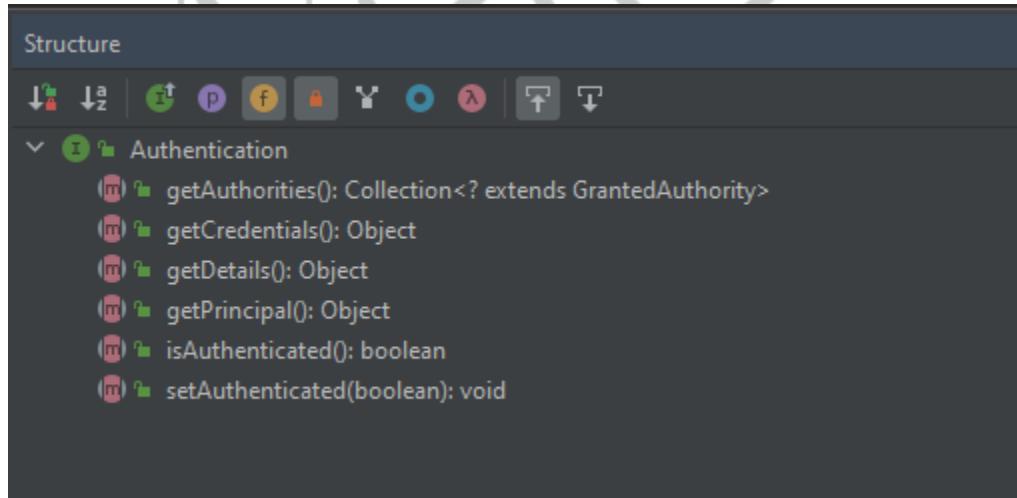
eazy  
bytes



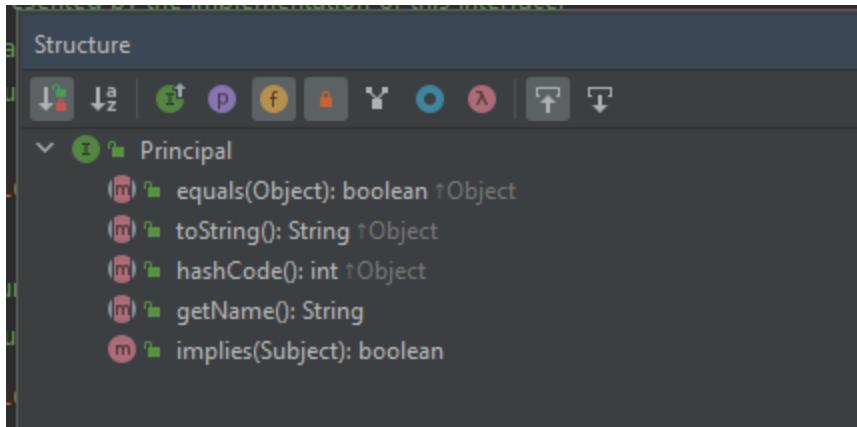
Vì vậy, tại sao chúng ta có hai cách riêng biệt để lưu trữ user detail đăng nhập? Cái đầu tiên là lớp 'UserDetails' và 'User'. Và bên trong lớp và interface này, chúng ta có các phương thức 'getPassword', 'getUsername', 'getAuthorities', v.v. Vì vậy, 'UserDetails' và 'User' này, chúng tôi sẽ sử dụng bất cứ khi nào chúng tôi đang cố tải thông tin chi tiết về người dùng từ hệ thống lưu trữ như bộ nhớ cơ sở dữ liệu của ứng dụng, đặc biệt là bên trong các interface và lớp triển khai của 'UserDetailsService' và 'UserDetailsManager' .

Khi user detail được tải từ cơ sở dữ liệu, thì chắc chắn những chi tiết này sẽ được chia sẻ trở lại 'AuthenticationProviders'. Vì vậy, bên trong 'AuthenticationProviders', sau khi xác thực thành công, những 'AuthenticationProviders' này chịu trách nhiệm chuyển đổi tất cả thông tin đó cùng với các chi tiết xác thực thành công thành loại dữ liệu đối tượng Authentication. Vì vậy, Authentication là kiểu trả về trong tất cả các tình huống mà chúng tôi đang cố gắng xác định xem xác thực có thành công hay không. Ngoài 'AuthenticationProviders' như bên trong Provider Manager và bên trong các bộ lọc, chúng tôi luôn xử lý đối tượng Authentication vì sử dụng Authentication object, chúng tôi chỉ có thể lưu trữ các chi tiết liên quan đến xác thực như tên người dùng, mật khẩu, quyền hạn cho dù xác thực có thành công hay không. Nhưng chúng tôi không phải lưu trữ các chi tiết như tài khoản đã hết hạn hay chưa, tài khoản có bị vô hiệu hóa hay được kích hoạt hay không vì chỉ khi các điều kiện này được thỏa mãn thì chỉ việc xác thực mới thành công. Vì vậy, đó là lý do tại sao tôi không có lý do gì để mang lại những user detail đó cho các thành phần bên trong của 'Spring Security'. Vì vậy, đó là nơi 'AuthenticationProvider' của tôi sẽ thực hiện phép thuật chuyển đổi các user detail này mà chúng tôi đã nhận được từ 'UserDetailsService' và 'UserDetailsManager' thành đối tượng Authentication.

Và tất nhiên, một trong những triển khai xác thực mà chúng tôi có là 'UsernamePasswordAuthenticationToken'. Vì vậy, interface xác thực này lần lượt mở rộng interface chính từ thư viện Java. Interface 'Principal' chỉ đơn giản đại diện cho tên của một thực thể, tên của một cá nhân hoặc id đăng nhập. Từ 'Principal', interface 'Authentication' này kế thừa một phương thức có tên 'getPrincipal' và bên trong 'Authentication', chúng ta có các phương thức khác như 'getName', 'getAuthorities'.



Bây giờ chúng ta hãy thử khám phá interface 'Authentication'.



Vì vậy, phương pháp duy nhất mà chúng tôi có bên trong principal là 'getName'. Bây giờ nếu bạn đi và kiểm tra '**Authentication**', bên trong này, chúng tôi có các phương thức như 'getPrincipal', 'getDetails', 'getAuthorities', 'getCredentials', 'isAuthenticated', 'setAuthenticated'. Vì vậy, 'isAuthenticated' được framework sử dụng để hiểu liệu người dùng đã cho có được xác thực thành công hay không. Và 'setAuthenticated' được framework sử dụng để đặt giá trị của 'isAuthenticated'. Bất cứ khi nào quá trình xác thực thành công hoàn tất thì phương thức này sẽ được sử dụng để đặt giá trị thực, nếu không, giá trị sai sẽ được đặt. Nếu bạn đi và kiểm tra 'InMemoryUserDetailsManager', ở đây chúng tôi đang tải thông tin chi tiết về người dùng từ hệ thống lưu trữ. Đó là lý do tại sao kiểu trả về là 'UserDetails'.

Trong khi nếu bạn đi và kiểm tra 'DaoAuthenticationProvider' nơi chúng tôi đang cố xác thực người dùng, nếu bạn cuộn xuống, sau khi user detail được điền bên trong phương thức authenticate này, bạn có thể thấy ở cuối khi xác thực thành công, chúng tôi chỉ gọi một phương thức gọi là 'createSuccessAuthentication'.

```
public Authentication authenticate(Authentication authentication) throws
AuthenticationException {
    Assert.isInstanceOfUsernamePasswordAuthenticationToken.class,
    authentication, () -> {
        return
    this.messages.getMessage("AbstractUserDetailsAuthenticationProvider.onlySuppo
rts", "Only UsernamePasswordAuthenticationToken is supported");
    });
    String username = this.determineUsername(authentication);
    boolean cacheWasUsed = true;
    UserDetails user = this.userCache.getUserFromCache(username);
    if (user == null) {
        cacheWasUsed = false;
        try {
            user = this.retrieveUser(username,
            (UsernamePasswordAuthenticationToken)authentication);
        } catch (UsernameNotFoundException var6) {
            this.logger.debug("Failed to find user '" + username + "'");
            if (!this.hideUserNotFoundExceptions) {
                throw var6;
            }
            throw new
        BadCredentialsException(this.messages.getMessage("AbstractUserDetailsAuthenti
cationProvider.badCredentials", "Bad credentials"));
    }
}
```

```

        }
        Assert.notNull(user, "retrieveUser returned null - a violation of the
interface contract");
    }

    try {
        this.preAuthenticationChecks.check(user);
        this.additionalAuthenticationChecks(user,
(UsernamePasswordAuthenticationToken)authentication);
    } catch (AuthenticationException var7) {
        if (!cacheWasUsed) {
            throw var7;
        }
        cacheWasUsed = false;
        user = this.retrieveUser(username,
(UsernamePasswordAuthenticationToken)authentication);
        this.preAuthenticationChecks.check(user);
        this.additionalAuthenticationChecks(user,
(UsernamePasswordAuthenticationToken)authentication);
    }

    this.postAuthenticationChecks.check(user);
    if (!cacheWasUsed) {
        this.userCache.putUserInCache(user);
    }

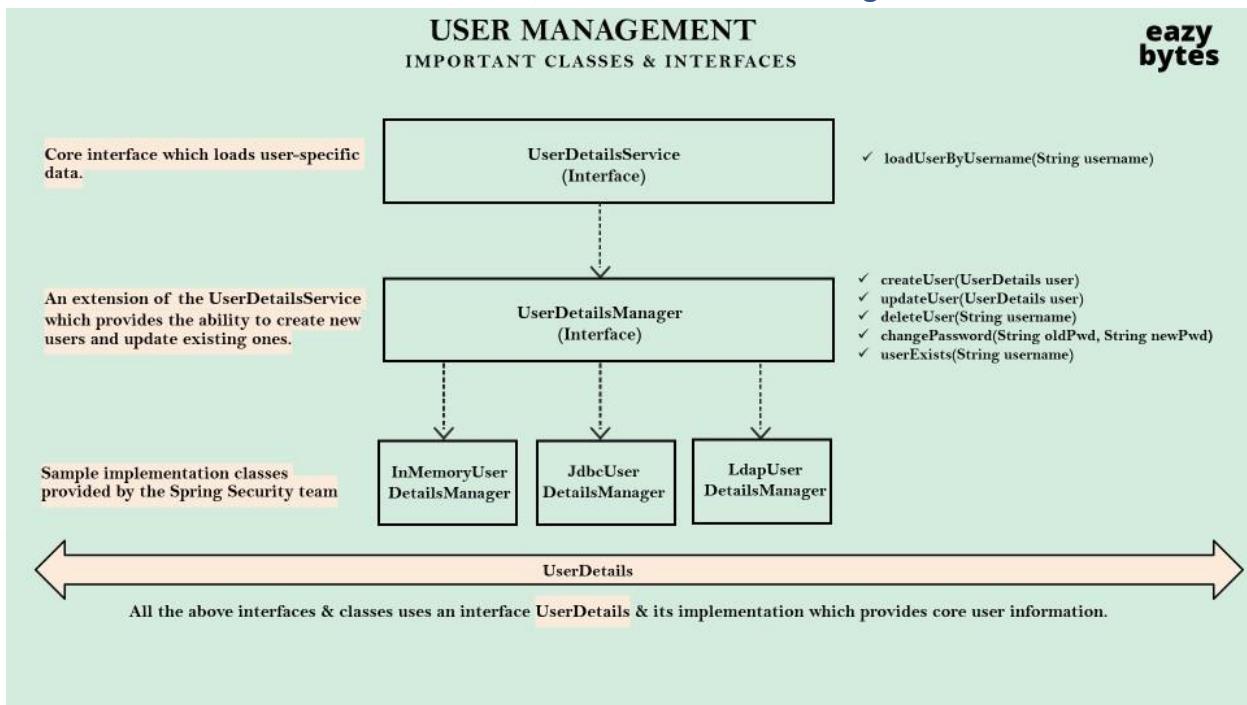
    Object principalToReturn = user;
    if (this.forcePrincipalAsString) {
        principalToReturn = user.getUsername();
    }

    return this.createSuccessAuthentication(principalToReturn,
authentication, user);
}

```

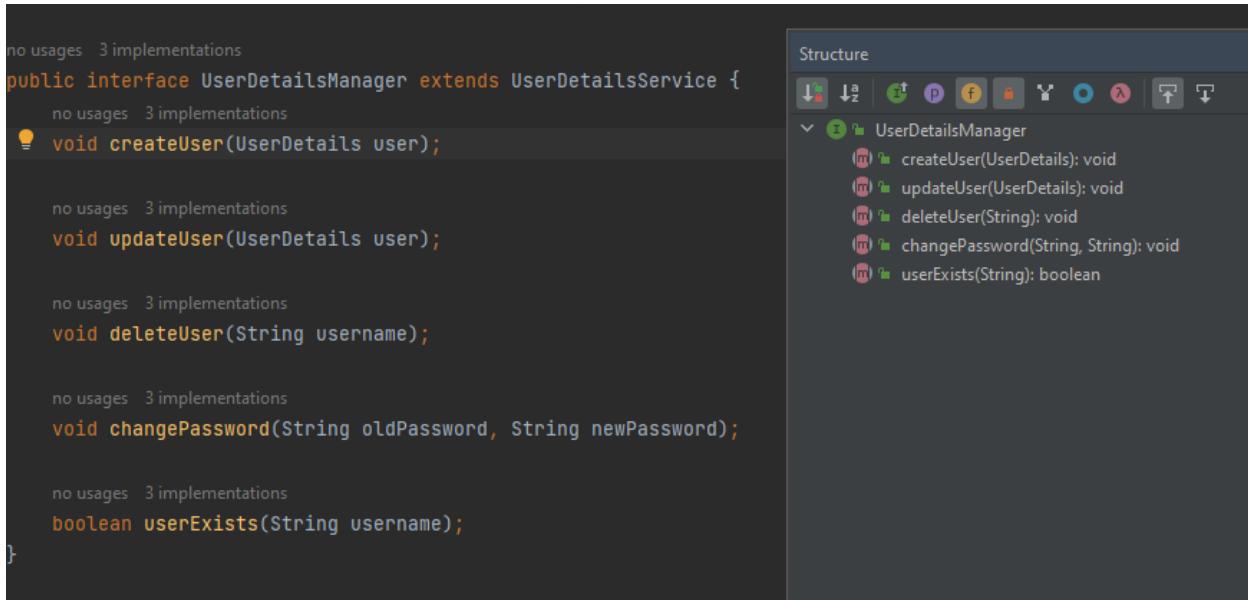
Vì vậy, phương pháp này, những gì nó sẽ làm là, nó sẽ điền vào tất cả các chi tiết cần thiết từ **UserDetails** đến '**Authentication**' và thông tin tương tự sẽ được trả lại. Khi phương thức này được gọi, bạn có thể thấy có một câu lệnh trả về từ đây và token của phương thức cũng như cách xác thực dưới dạng kiểu trả về của phương thức.

#### 4. Tìm hiểu sâu về UserDetailsService & UserDetailsManager Interfaces



Interface này, là **User Detail Service**, được xây dựng để phù hợp với kịch bản phổ biến nhất, đó là tải dữ liệu cụ thể của người dùng. Nếu không tải dữ liệu cụ thể của người dùng từ hệ thống lưu trữ, chúng tôi thực sự không thể thực hiện xác thực. Vì vậy, đây là trường hợp phổ biến nhất, đối với trường hợp sử dụng này, chúng có một interface riêng giống như **User Detail Service**. Và interface này có một phương thức với tên **loadUserByUsername**. Vì vậy, đối với phương thức này, chúng ta chỉ cần chuyển tên người dùng được nhập bởi user và dựa trên tên người dùng, user detail từ hệ thống lưu trữ sẽ được lớp triển khai phương thức này tải và trả về.

Interface **User Detail Service** này được mở rộng bởi một interface khác, đó là **UserDetailsManager**. Bên trong **User Detail**, chúng tôi chỉ có một dịch vụ muốn tải thông tin chi tiết của người dùng. Trong khi đến với **UserDetailsManager**, nó sẽ giúp chúng tôi hoặc nó sẽ cung cấp khả năng tạo người dùng mới, cập nhật những người dùng hiện có.



```

no usages 3 implementations
public interface UserDetailsManager extends UserDetailsService {
    no usages 3 implementations
    void createUser(UserDetails user);

    no usages 3 implementations
    void updateUser(UserDetails user);

    no usages 3 implementations
    void deleteUser(String username);

    no usages 3 implementations
    void changePassword(String oldPassword, String newPassword);

    no usages 3 implementations
    boolean userExists(String username);
}

```

Structure view:

- UserDetailsManager
  - createUser(UserDetails): void
  - updateUser(UserDetails): void
  - deleteUser(String): void
  - changePassword(String, String): void
  - userExists(String): boolean

Vì vậy, ở đây bạn có thể thấy nó có nhiều phương pháp khác nhau. Chúng ta có thể ghi đè phương thức này và triển khai logic bên trong nó.

Hai interface này rất quan trọng trong Spring Security và có rất ít lớp triển khai cho các interface này. Vì một số lý do, nếu bạn không muốn tuân theo tất cả các trình quản lý user detail, **User Detail Service** này, bạn chỉ muốn viết logic của riêng mình, thì chắc chắn bạn có thể xác định **Authentication Provider** của riêng mình và bạn có thể viết tất cả logic mà bạn muốn bên trong **Authentication Provider** tùy chỉnh đó.

## 5. UserDetailsManager Implementation classes

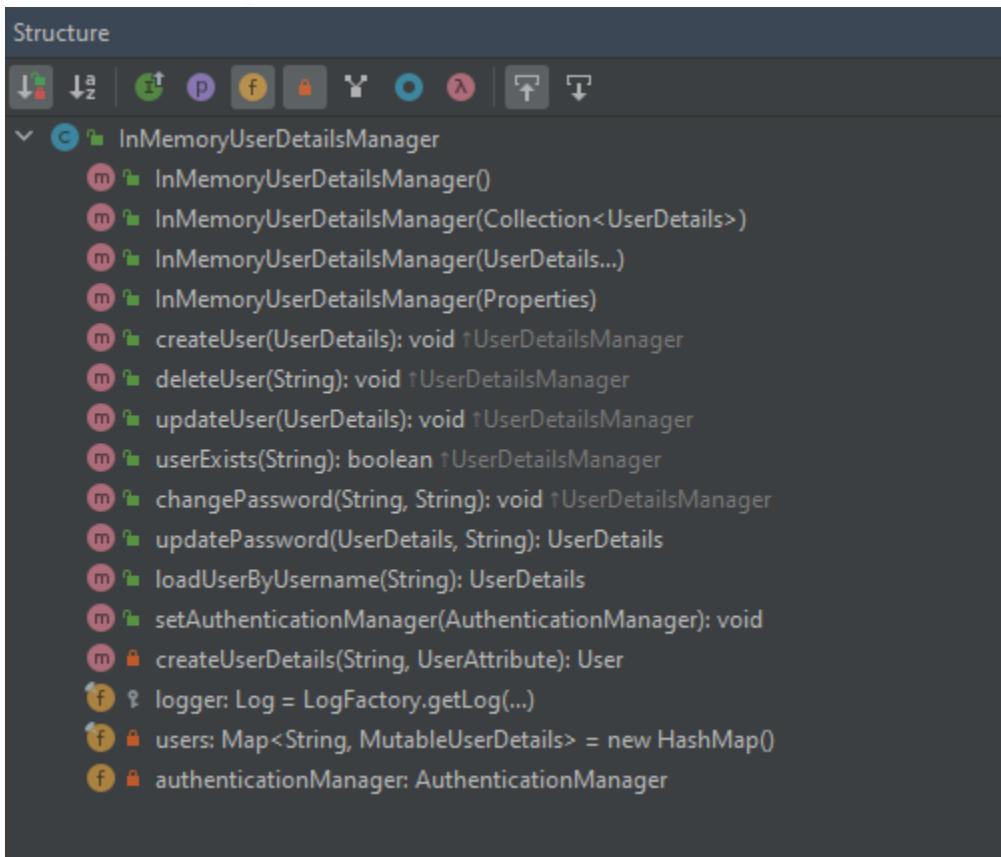
Implementation **UserDetailsManager** trong Spring Security chịu trách nhiệm quản lý user detail, chẳng hạn như thông tin người dùng và xác thực. Nó cung cấp các hoạt động để tạo, cập nhật và xóa tài khoản người dùng, cũng như truy xuất thông tin chi tiết của người dùng.

Bản thân interface **UserDetailsManager** là high-level abstraction và việc triển khai thực tế của nó được cung cấp bởi các lớp khác nhau trong Spring Security. Chúng ta hãy đi sâu vào một số lớp triển khai **UserDetailsManager** thường được sử dụng:

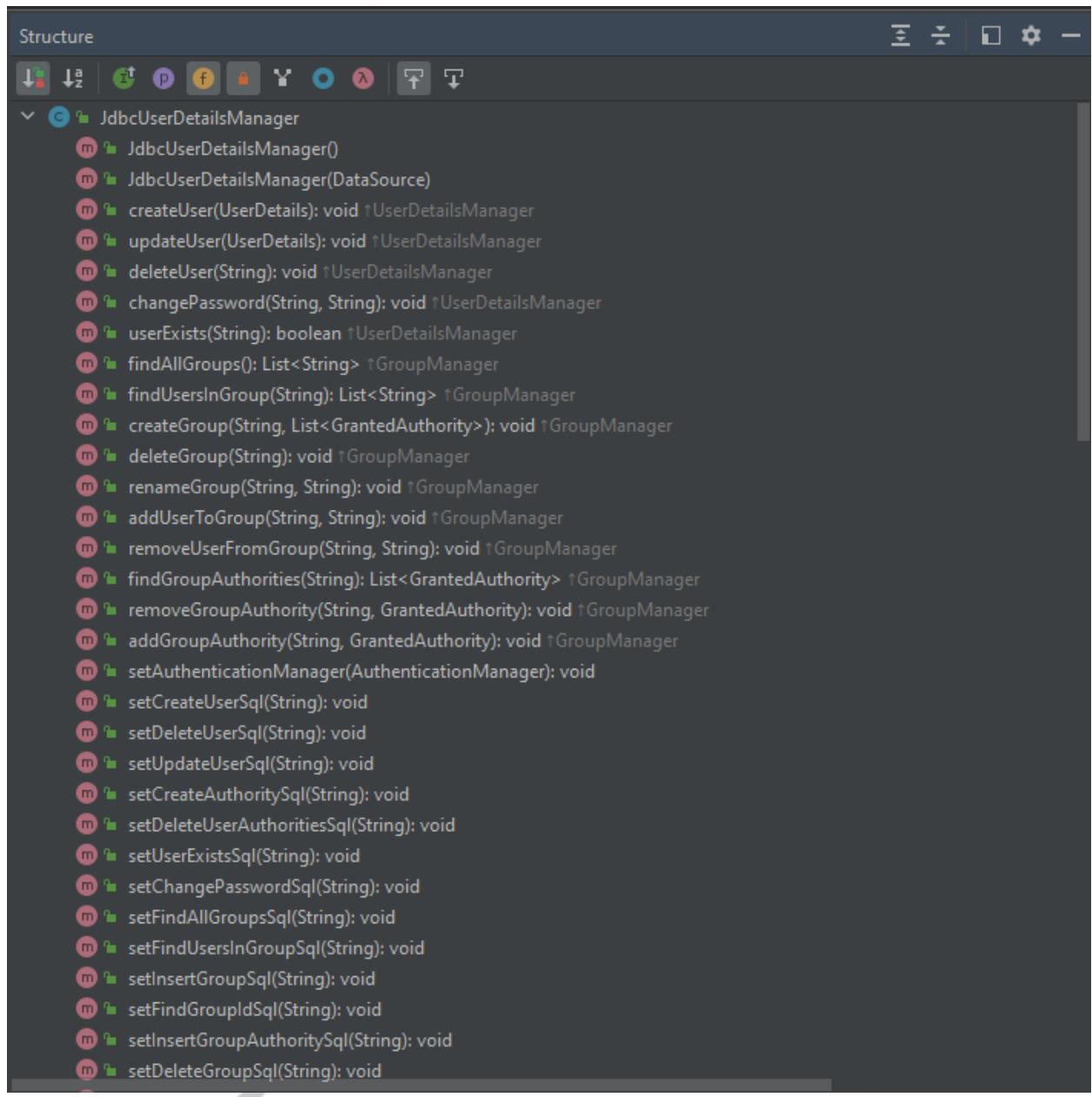
- **JdbcUserDetailsManager**: Việc triển khai này được thiết kế để hoạt động với cơ sở dữ liệu dựa trên JDBC để lưu trữ người dùng. Nó tận dụng **JdbcTemplate** của Spring để tương tác với cơ sở dữ liệu và thực hiện các thao tác quản lý người dùng. Nó hỗ trợ các nhà cung cấp cơ sở dữ liệu khác nhau và cho phép bạn định cấu hình các truy vấn SQL được sử dụng để quản lý người dùng.
- **InMemoryUserDetailsManager**: Việc triển khai này lưu trữ user detail trong bộ nhớ, làm cho nó phù hợp với các ứng dụng quy mô nhỏ hoặc mục đích thử nghiệm. Thông tin người dùng thường được cung cấp thông qua cấu hình và thông tin này không lưu giữ dữ liệu giữa các lần khởi động lại ứng dụng.
- **UserDetailsServiceWrapper**: Lớp này là một trình bao bọc xung quanh triển khai **UserDetailsService**, được sử dụng để tải thông tin chi tiết về người dùng từ một nguồn bên ngoài, chẳng hạn như cơ sở dữ liệu hoặc LDAP. Lớp **UserDetailsServiceWrapper** thêm chức năng quản lý người dùng bổ sung vào **UserDetailsService** được bao bọc, biến nó thành một **UserDetailsManager** một cách hiệu quả.

- **LdapUserDetailsManager**: Việc triển khai này được thiết kế để quản lý user detail được lưu trữ trong thư mục LDAP. Nó tận dụng các hoạt động LDAP do Spring LDAP cung cấp để thực hiện các hoạt động quản lý người dùng, chẳng hạn như tạo, cập nhật và xóa tài khoản người dùng trong thư mục LDAP.

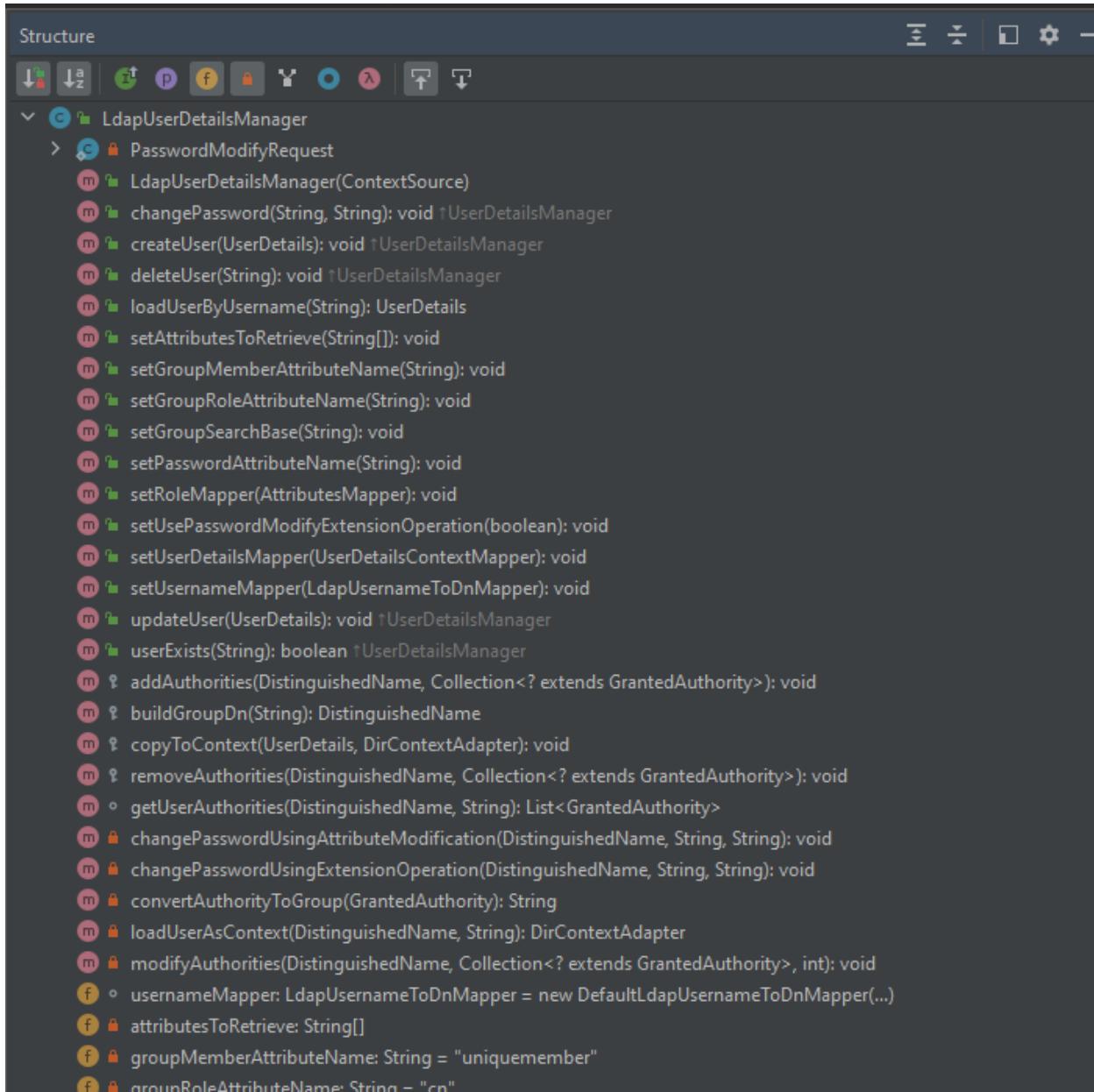
Chúng ta khám phá những gì hiện diện bên trong InMemoryUserDetailsManager này.



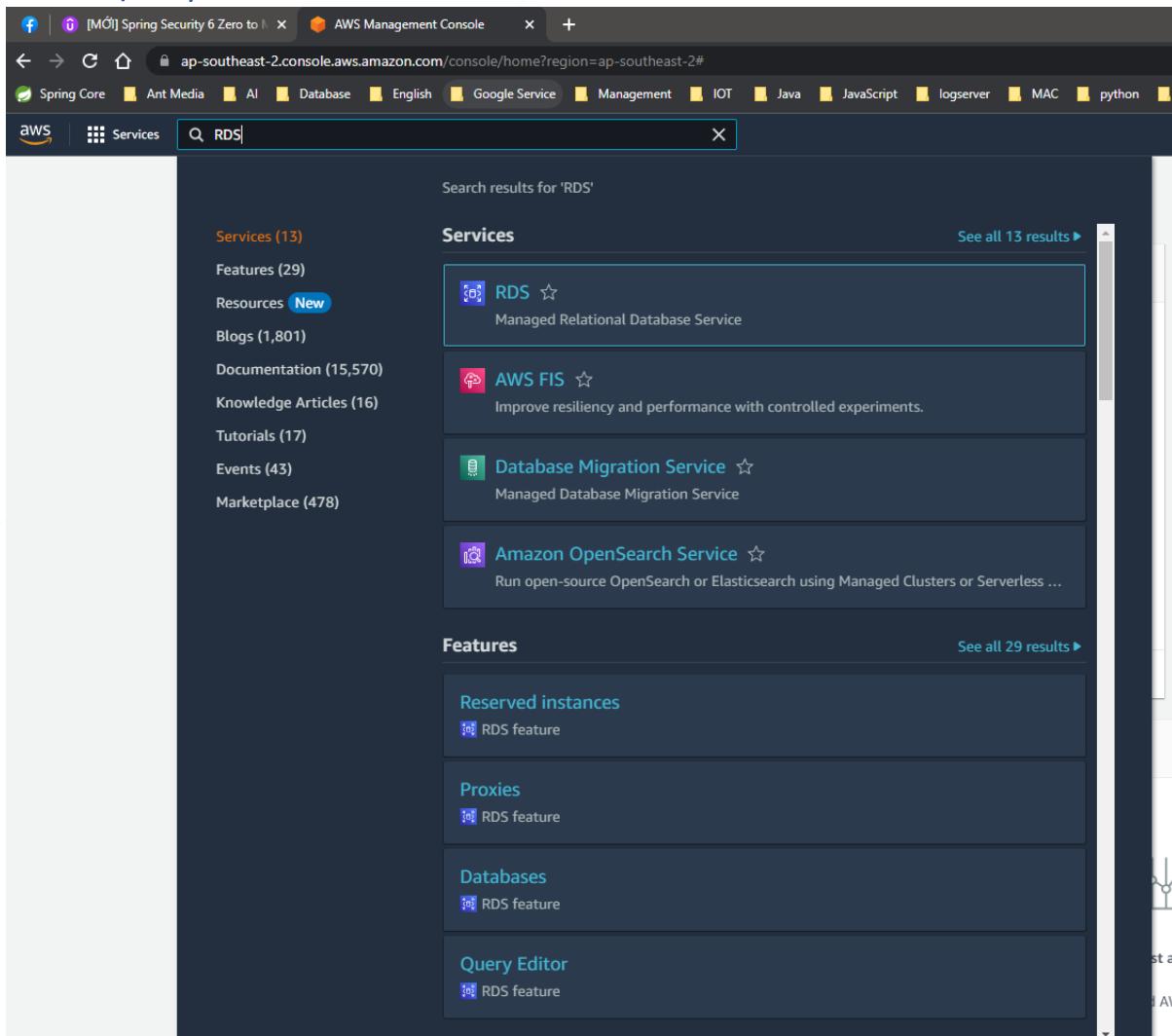
## JdbcUserDetailsManager



Và một lần nữa LdapUserDetailsManager, nó không phải là lớp triển khai được sử dụng phổ biến vì trừ khi bạn có máy chủ LDAP làm hệ thống lưu trữ của mình, bạn có thể không sử dụng được UserDetailsManager này.



## 6. Tạo MySQL với AWS RDS



The screenshot shows the AWS Management Console search results for 'RDS'. The search bar at the top contains 'RDS'. The results are categorized into 'Services' and 'Features'.

**Services (13)**

- Features (29)
- Resources **New**
- Blogs (1,801)
- Documentation (15,570)
- Knowledge Articles (16)
- Tutorials (17)
- Events (43)
- Marketplace (478)

**Services**

- RDS** ☆  
Managed Relational Database Service
- AWS FIS** ☆  
Improve resiliency and performance with controlled experiments.
- Database Migration Service** ☆  
Managed Database Migration Service
- Amazon OpenSearch Service** ☆  
Run open-source OpenSearch or Elasticsearch using Managed Clusters or Serverless ...

**See all 13 results ▶**

**Features**

- Reserved instances  
RDS feature
- Proxies  
RDS feature
- Databases  
RDS feature
- Query Editor  
RDS feature

**See all 29 results ▶**

AWS Services Search [Alt+S]

## Amazon RDS

**Dashboard**

- Databases
- Query Editor
- Performance insights
- Snapshots
- Exports in Amazon S3
- Automated backups
- Reserved instances
- Proxies

Subnet groups

Parameter groups

Option groups

Custom engine versions

Events

**Introducing Aurora I/O-Optimized**  
Aurora's I/O-Optimized  is a new cluster storage configuration that

**Try the new Amazon RDS Multi-AZ deployment option**  
For your Amazon RDS for MySQL and PostgreSQL workloads, you can now run multiple instances by deploying the Multi-AZ DB cluster [Learn more](#)

**Create database**

Or, [Restore Multi-AZ DB Cluster from Snapshot](#)

## Resources

You are using the following Amazon RDS resources in the Asia Pacific (Singapore) Region

**DB Instances (0/40)**  
Allocated storage (0 TB/100 TB)  
[Increase DB instances limit !\[\]\(aa7cbacc0677753def4ac2d02cf166b5\_img.jpg\)](#)

**DB Clusters (0/40)**

**Reserved instances (0/40)**

**Snapshots (0)**



Standard create

You set all of the configuration options, including ones for availability, security, backups, and maintenance.

Easy create

Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

## Configuration

Engine type [Info](#)

Aurora (MySQL Compatible)



Aurora (PostgreSQL Compatible)



MySQL



MariaDB



PostgreSQL



Oracle

**ORACLE**

Microsoft SQL Server



Edition

MySQL Community

DB instance size

Production

db.r6g.xlarge  
4 vCPUs  
32 GiB RAM  
500 GiB  
1.209 USD/hour

Dev/Test

db.r6g.large  
2 vCPUs  
16 GiB RAM  
100 GiB  
0.274 USD/hour

Free tier

db.t3.micro  
2 vCPUs  
1 GiB RAM  
20 GiB  
0.030 USD/hour

**DB instance identifier**

Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.

The DB instance identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

**Master username [Info](#)**

Type a login ID for the master user of your DB instance.

1 to 16 alphanumeric characters. First character must be a letter.

 **Auto generate a password**

Amazon RDS can generate a password for you, or you can specify your own password.

**Master password [Info](#)**

Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), '(single quote), "(double quote) and @ (at sign).

**Confirm master password [Info](#)**

 **Invalid password**

**► Set up EC2 connection - *optional***

You can also set up a connection to an EC2 instance after creating the database. Go to the database list page or the database details page, choose **Actions**, and then choose **Set up to EC2 connection**.

**► View default settings for Easy create**

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

 You are responsible for ensuring that you have all of the necessary rights for any third-party products or services that you use with AWS services.

**Cancel****Create database**

⇒ Đợi 1 lúc để tạo cơ sở dữ liệu mẫu

⇒ Sau đó Modify DB instance: springsecurity => Connectivity => Additional configuration

## Connectivity

Network type [Info](#)  
To use dual-stack mode, make sure that you associate an IPv6 CIDR block with a subnet in the VPC you specify.

**IPv4**  
Your resources can communicate only over the IPv4 addressing protocol.

**Dual-stack mode**  
Your resources can communicate over IPv4, IPv6, or both.

DB subnet group

Security group  
List of DB security groups to associate with this DB instance.  
  
**default** 

Certificate authority [Info](#)  
Using a server certificate provides an extra layer of security by validating that the connection is being made to an Amazon database. It does so by checking the server certificate that is automatically installed on all databases that you provision.

▼ Additional configuration

Public access

**Publicly accessible**  
RDS assigns a public IP address to the database. Amazon EC2 instances and other resources outside of the VPC can connect to your database. Resources inside the VPC can also connect to the database. Choose one or more VPC security groups that specify which resources can connect to the database.

**Not publicly accessible**  
No IP address is assigned to the DB instance. EC2 instances and devices outside the VPC can't connect.

Database port  
Specify the TCP/IP port that the DB instance will use for application connections. The application connection string must specify the port number. The DB security group and your firewall must allow connections to the port. [Learn more](#) 

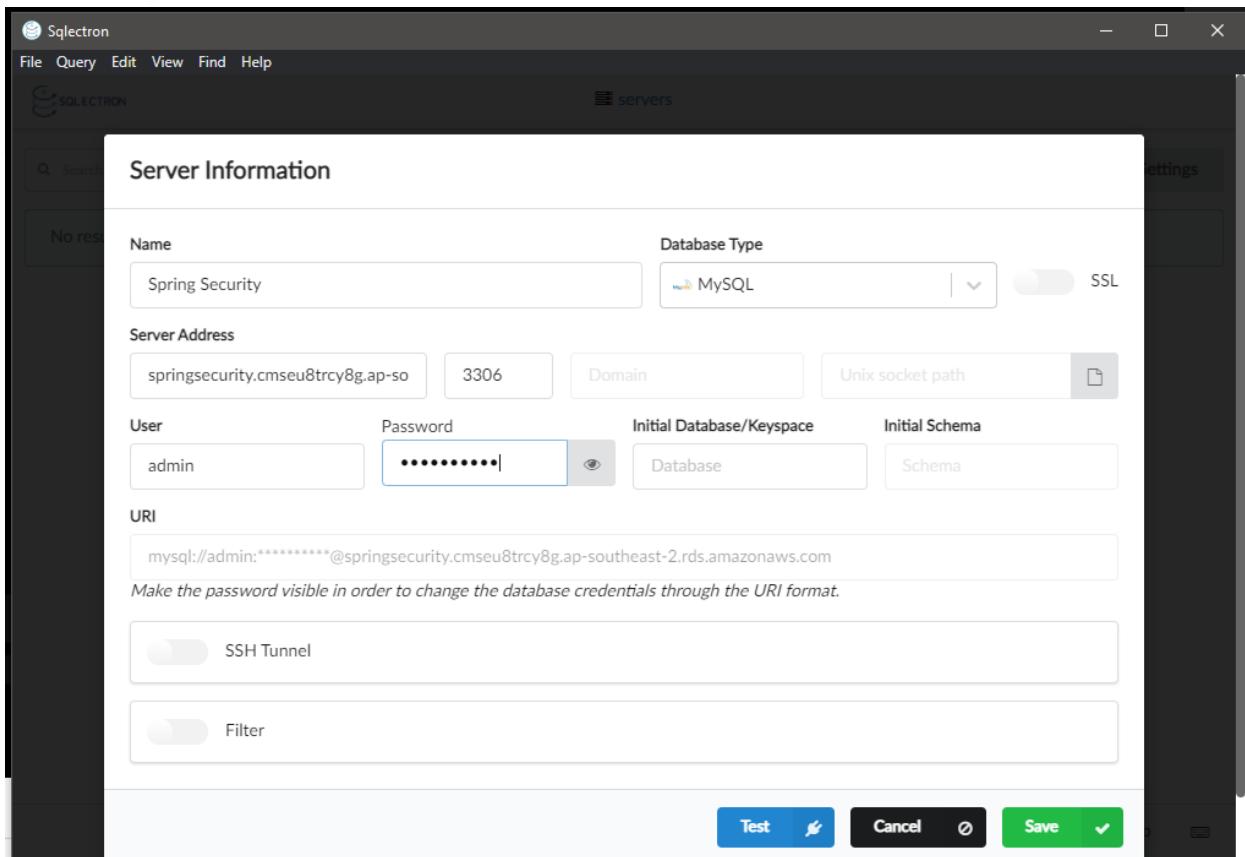
## 7. Kết nối với DB & Tạo người dùng bên trong DB theo JdbcUserDetailsManager

### Cài đặt Sqlectron

Sqlectron là một ứng dụng đồ họa UI application (GUI) được sử dụng để quản lý cơ sở dữ liệu. Nó cung cấp một giao diện đồ họa dễ sử dụng cho việc truy vấn, chỉnh sửa và quản lý cơ sở dữ liệu.

Sqlectron hỗ trợ nhiều loại cơ sở dữ liệu phổ biến như MySQL, PostgreSQL, SQLite, SQL Server, Oracle và nhiều hệ thống quản lý cơ sở dữ liệu khác. Với Sqlectron, người dùng có thể thực hiện các tác vụ như thực thi truy vấn SQL, duyệt và chỉnh sửa bảng dữ liệu, xem thông tin cơ sở dữ liệu và quản lý người dùng.

<https://sqlectron.github.io/>



springsecurity

**Summary**

DB identifier springsecurity	CPU <div style="width: 3.02%;">3.02%</div>	Status <span style="color: green;">Available</span>	Class db.t3.micro
Role Instance	Current activity <div style="width: 0%;">0 Connections</div>	Engine MySQL Community	Region & AZ ap-southeast-2a

**Connectivity & security**

Endpoint & port	Networking	Security
Endpoint <a href="#">springsecurity.cmseu8trcy8g.ap-southeast-2.rds.amazonaws.com</a>	Availability Zone ap-southeast-2a	VPC security groups <a href="#">default (sg-057d945defdfa30b8)</a> <span style="color: green;">Active</span>
Port 3306	VPC <a href="#">vpc-0625c58925365661d</a>	Publicly accessible No
	Subnet group <a href="#">default-vpc-0625c58925365661d</a>	Certificate authority <a href="#">Info</a> <a href="#">rds-ca-2019</a>
	Subnets <a href="#">subnet-09199ba970a4f122f</a> <a href="#">subnet-0fe08055fd322951e</a> <a href="#">subnet-0c4dba6e317d23404</a>	Certificate authority date August 23, 2024, 00:08 (UTC+07:00)
	Network type IPv4	DB instance certificate expiration date August 23, 2024, 00:08 (UTC+07:00)

## 8. Sử dụng JdbcUserDetailsManager để thực hiện xác thực

Thêm các **dependency** cần thiết:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Cấu hình kết nối đến cơ sở dữ liệu:

```
#application.properties

spring.datasource.url=jdbc:mysql://springsecurity.cjdg8jrihf3.us-east-2.rds.amazonaws.com/eazybank
spring.datasource.username=admin
spring.datasource.password=Tung090895
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

Phương thức **userDetailsService** được định nghĩa để cung cấp một **UserDetailsService** dựa trên một nguồn dữ liệu ( **dataSource**).

**UserDetailsService** là một interface trong Spring Security được sử dụng để tải thông tin người dùng từ nguồn dữ liệu (ví dụ: cơ sở dữ liệu) và cung cấp thông tin về người dùng cho việc xác thực và phân quyền trong ứng dụng.

Trong phương thức `userDetailsService`, bạn cần cung cấp một đối tượng `DataSource` để xác định nguồn dữ liệu từ đó `UserDetailsService` sẽ truy xuất thông tin người dùng.

```
--ProjectSecurityConfig.java

@Bean
public PasswordEncoder passwordEncoder() {
    return NoOpPasswordEncoder.getInstance();
}

@Bean
public UserDetailsService userDetailsService(DataSource dataSource) {
    return new JdbcUserDetailsManager(dataSource);
}
```

## 9. Tạo các table cho Authentication

```
CREATE TABLE `users` (
`id` INT NOT NULL AUTO_INCREMENT,
`username` VARCHAR(45) NOT NULL,
`password` VARCHAR(45) NOT NULL,
`enabled` INT NOT NULL,
PRIMARY KEY (`id`));

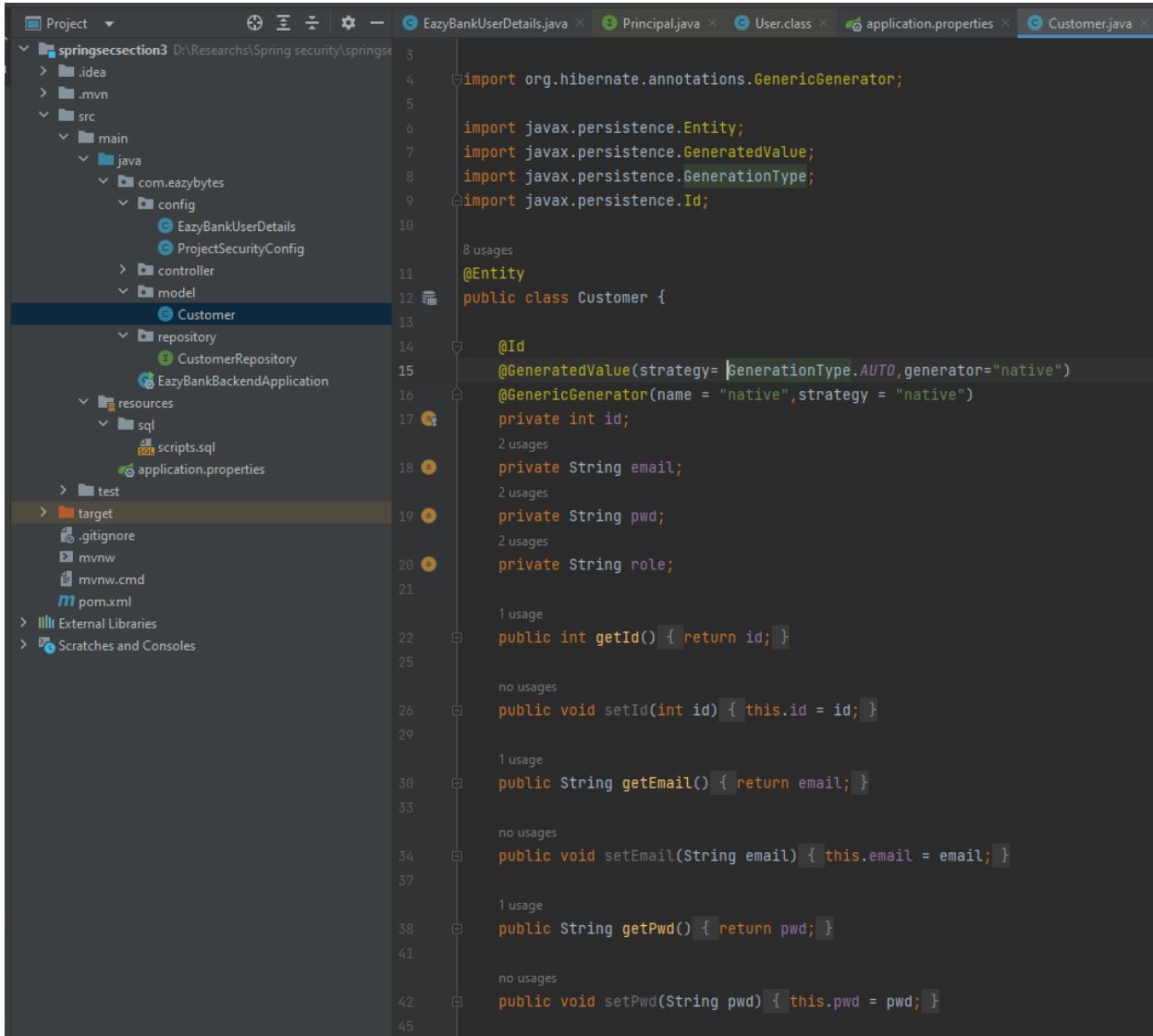
CREATE TABLE `authorities` (
`id` int NOT NULL AUTO_INCREMENT,
`username` varchar(45) NOT NULL,
`authority` varchar(45) NOT NULL,
PRIMARY KEY (`id`));

INSERT IGNORE INTO `users` VALUES (NULL, 'happy', '12345', '1');
INSERT IGNORE INTO `authorities` VALUES (NULL, 'happy', 'write');

CREATE TABLE `customer` (
`id` int NOT NULL AUTO_INCREMENT,
`email` varchar(45) NOT NULL,
`pwd` varchar(200) NOT NULL,
`role` varchar(45) NOT NULL,
PRIMARY KEY (`id`)
);

INSERT INTO `customer` (`email`, `pwd`, `role`)
VALUES ('johndoe@example.com', '54321', 'admin');
```

## 10. Creating JPA Entity and repository classes for new table



```
3 import org.hibernate.annotations.GenericGenerator;
4
5 import javax.persistence.Entity;
6 import javax.persistence.GeneratedValue;
7 import javax.persistence.GenerationType;
8 import javax.persistence.Id;
9
10 @Entity
11 public class Customer {
12
13     @Id
14     @GeneratedValue(strategy = GenerationType.AUTO, generator = "native")
15     @GenericGenerator(name = "native", strategy = "native")
16     private int id;
17
18     private String email;
19
20     private String pwd;
21
22     private String role;
23
24     public int getId() { return id; }
25
26     public void setId(int id) { this.id = id; }
27
28     public String getEmail() { return email; }
29
30     public void setEmail(String email) { this.email = email; }
31
32     public String getPwd() { return pwd; }
33
34     public void setPwd(String pwd) { this.pwd = pwd; }
35
36
37
38
39
40
41
42
43
44
45
```

Có thể sử dụng Lombok để rút gọn code trong Java.

### Tạo CustomerRepository

Interface **CustomerRepository** mở rộng từ **CrudRepository<Customer, Long>**. Điều này cho biết **CustomerRepository** là một interface đại diện cho việc quản lý đối tượng **Customer** trong cơ sở dữ liệu.

**CrudRepository** là một interface được cung cấp bởi Spring Data JPA, cung cấp các phương thức cơ bản để thực hiện các hoạt động CRUD (Create, Read, Update, Delete) trên các đối tượng trong cơ sở dữ liệu. Bằng cách mở rộng **CrudRepository<Customer, Long>**, **CustomerRepository** sẽ thừa kế các phương thức cơ bản này và có thể sử dụng chúng để thao tác với đối tượng **Customer** trong cơ sở dữ liệu. Ví dụ, sau khi đăng ký **CustomerRepository** trong ứng dụng Spring, bạn có thể sử dụng các phương thức như **save**, **findById**, **findAll**, **delete**,... để thực hiện các hoạt động như thêm mới khách hàng, tìm

kiểm khách hàng theo ID, lấy danh sách tất cả khách hàng, xóa khách hàng, và nhiều hoạt động khác liên quan đến quản lý dữ liệu khách hàng trong cơ sở dữ liệu.

```
@Repository
public interface CustomerRepository extends CrudRepository<Customer, Long> {

    List<Customer> findByEmail(String email);
}
```

## 11. Creating our own custom implementation of UserDetailsService

Lớp **EazyBankUserDetails** implements **UserDetailsService** để cung cấp thông tin người dùng (user details) cho quá trình xác thực và phân quyền trong ứng dụng EazyBank.

**UserDetailsService** là một interface trong Spring Security, được sử dụng để tìm kiếm thông tin người dùng dựa trên tên người dùng (username) và trả về một đối tượng **UserDetails**. Đối tượng **UserDetails** chứa các thông tin cần thiết để xác thực và phân quyền, chẳng hạn như tên người dùng, mật khẩu đã mã hóa, vai trò (roles), quyền (authorities), và các thông tin khác liên quan.

Trong trường hợp của lớp **EazyBankUserDetails**, nó implement **UserDetailsService** để cung cấp thông tin người dùng cho quá trình xác thực và phân quyền trong ứng dụng EazyBank. Các phương thức trong **EazyBankUserDetails** sẽ được triển khai để tìm kiếm thông tin người dùng trong cơ sở dữ liệu hoặc hệ thống lưu trữ khác, và trả về một đối tượng **UserDetails** chứa thông tin người dùng tương ứng.

Sau khi triển khai **EazyBankUserDetails** và đăng ký nó trong ứng dụng Spring, nó có thể được sử dụng bởi Spring Security để xác thực và phân quyền người dùng khi họ truy cập vào các tính năng và tài nguyên của EazyBank.

```
@Service
public class EazyBankUserDetails implements UserDetailsService {

    @Autowired
    private CustomerRepository customerRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        String userName, password = null;
        List<GrantedAuthority> authorities = null;
        List<Customer> customer = customerRepository.findByEmail(username);
        if (customer.size() == 0) {
            throw new UsernameNotFoundException("User details not found for
the user : " + username);
        } else{
            userName = customer.get(0).getEmail();
            password = customer.get(0).getPwd();
            authorities = new ArrayList<>();
            authorities.add(new
SimpleGrantedAuthority(customer.get(0).getRole()));
        }
    }
}
```

```
        return new User(username, password, authorities);
    }
}
```

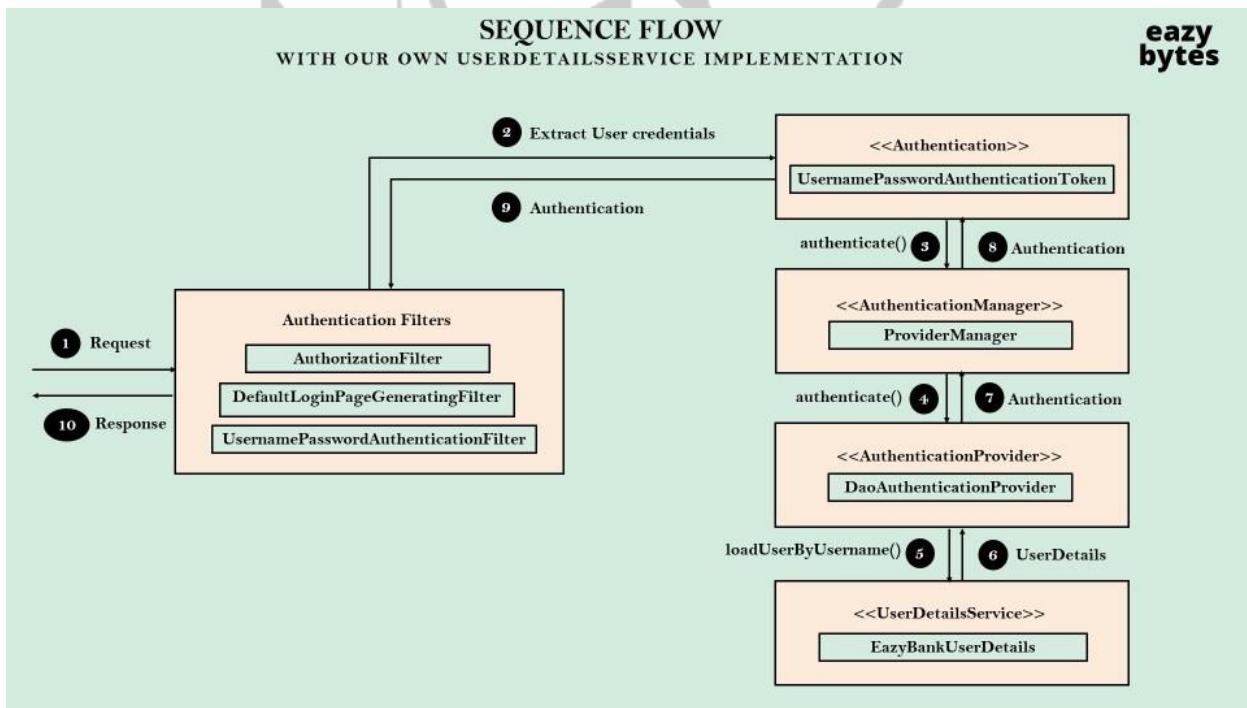
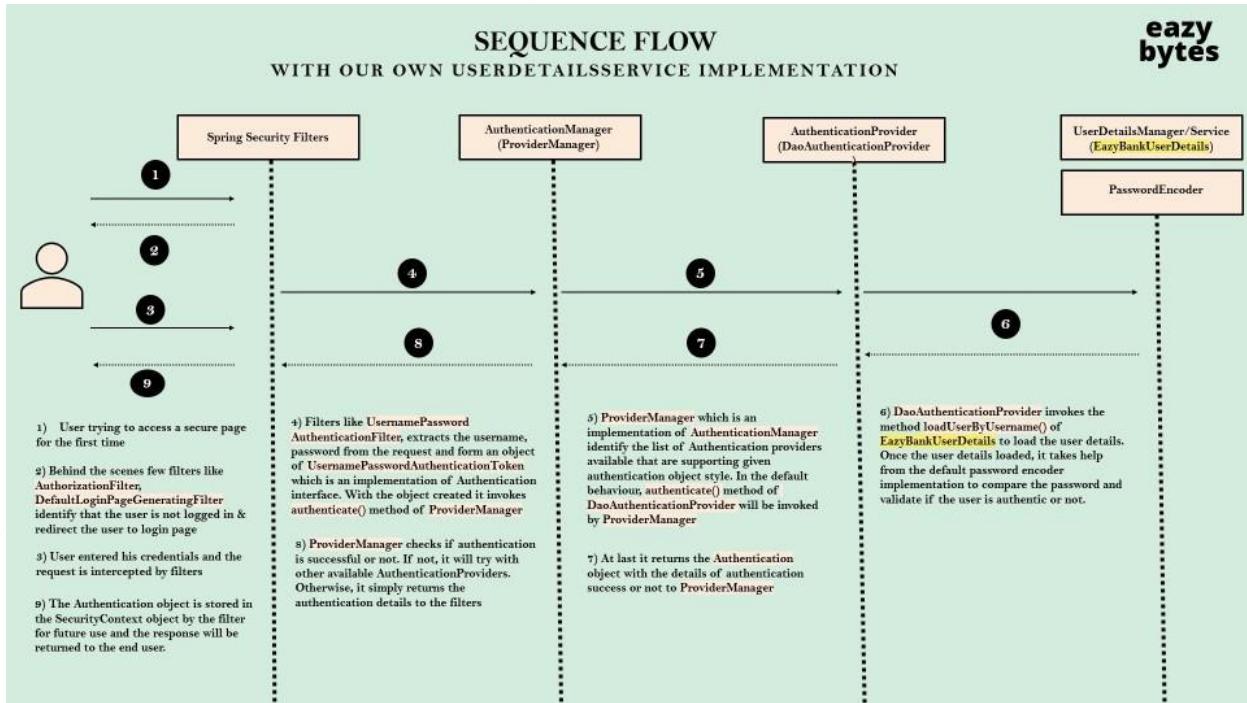
## 12. Building a new REST API to allow the registration of new User

```
@RestController
public class LoginController {

    @Autowired
    private CustomerRepository customerRepository;

    @PostMapping("/register")
    public ResponseEntity<String> registerUser(@RequestBody Customer
customer) {
        Customer savedCustomer = null;
        ResponseEntity response = null;
        try {
            savedCustomer = customerRepository.save(customer);
            if (savedCustomer.getId() > 0) {
                response = ResponseEntity
                    .status(HttpStatus.CREATED)
                    .body("Given user details are successfully
registered");
            }
        } catch (Exception ex) {
            response = ResponseEntity
                .status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("An exception occurred due to " + ex.getMessage());
        }
        return response;
    }
}
```

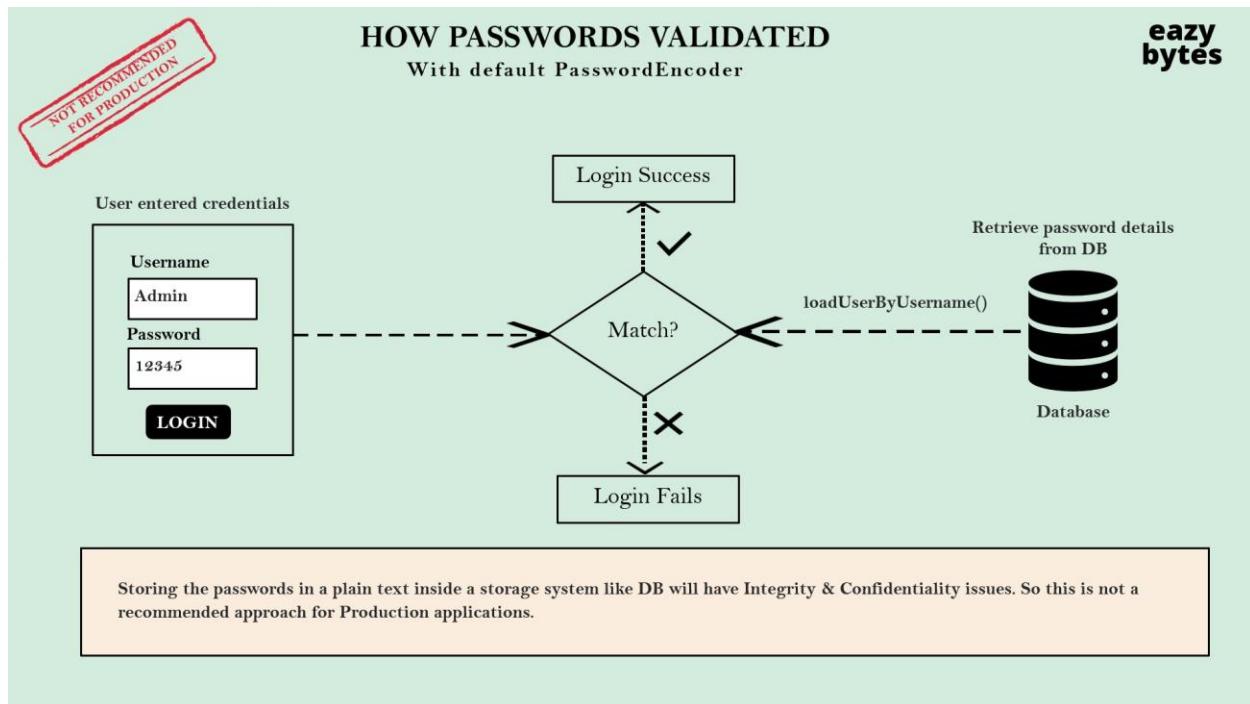
## 13. UserDetailsService Implementation



## Section 4: Password Management with PasswordEncoders

### 1. Cách xác thực mật khẩu trong Spring Security theo mặc định

User của tôi sẽ nhập thông tin đăng nhập của riêng anh ấy, tên người dùng và mật khẩu của anh ấy là gì. Và ngay sau khi anh ấy nhấp vào nút đăng nhập hoặc nút đăng nhập, đăng sau hậu trường, Spring Security của tôi sẽ thực thi tất cả logic có bên trong **Authentication Provider** và nó sẽ cố tải các chi tiết từ hệ thống lưu trữ với sự trợ giúp của phương thức `loadUserByUsername` có sẵn bên trong các lớp triển khai của **User Details Manager**.



Khi **user details** được tải, Spring Security của tôi sẽ cố gắng so sánh mật khẩu được cung cấp bởi user của tôi và mật khẩu được tải từ cơ sở dữ liệu. Vì vậy, phép so sánh này sẽ diễn ra dưới dạng so sánh văn bản gốc giống như cách chúng ta so sánh hai chuỗi bên trong Java với sự trợ giúp của phương thức `equals`. Tương tự như vậy, nó sẽ so sánh và nếu mật khẩu trùng khớp thì thông tin đăng nhập của tôi sẽ thành công, nếu không, thông tin đăng nhập của tôi sẽ không thành công. Tất cả mật khẩu của chúng tôi, chúng tôi đang lưu trữ ở định dạng văn bản gốc. Đó là lý do tại sao phương pháp này hoàn toàn không được khuyến khích.

Ở đây, bên trong `DaoAuthenticationProvider` của tôi, nơi xác thực thực tế sẽ diễn ra. Tôi có thể tìm phương pháp xác thực mật khẩu đang diễn ra. Bên trong phương thức `authenticate` này, chúng tôi sẽ thử tải chi tiết tên người dùng từ hệ thống lưu trữ hoặc từ cơ sở dữ liệu. Khi thông tin chi tiết về người dùng được tải, chúng tôi sẽ thực thi một số phương thức như `PreAuthenticationChecks` mà chúng tôi sẽ thực hiện. Vì vậy, bên trong `PreAuthenticationChecks` này, Spring Security sẽ kiểm tra xem tài khoản của tôi đã hết hạn chưa, tài khoản của tôi có bị khóa hay tài khoản của tôi bị vô hiệu hóa hay không. Vì vậy, tất cả những kịch bản tiêu cực nó sẽ kiểm tra. Vì vậy, sau khi các kiểm tra đó hoàn tất thành công, nó sẽ gọi một phương thức có tên là **AuthenticationChecks**.

```

62     cacheWasUsed = false;
63
64     try {
65         user = this.retrieveUser(username, (UsernamePasswordAuthenticationToken)authentication);
66     } catch (UsernameNotFoundException var6) {
67         this.logger.debug("Failed to find user '" + username + "'");
68         if (!this.hideUserNotFoundExceptions) {
69             throw var6;
70         }
71
72         throw new BadCredentialsException(this.messages.getMessage( code: "AbstractUserDetailsAuthenticationP
73     }
74
75     Assert.notNull(user, message: "retrieveUser returned null - a violation of the interface contract");
76 }
77
78     try {
79         this.preAuthenticationChecks.check(user);
80         this.additionalAuthenticationChecks(user, (UsernamePasswordAuthenticationToken)authentication);
81     } catch (AuthenticationException var7) {
82         if (!cacheWasUsed) {
83             throw var7;
84         }
85
86         cacheWasUsed = false;
87         user = this.retrieveUser(username, (UsernamePasswordAuthenticationToken)authentication);
88         this.preAuthenticationChecks.check(user);
89         this.additionalAuthenticationChecks(user, (UsernamePasswordAuthenticationToken)authentication);
90     }
91
92     this.postAuthenticationChecks.check(user);
93     if (!cacheWasUsed) {
94         this.userCache.putUserInCache(user);
95     }
96
97     Object principalToReturn = user;
98     if (this.forcePrincipalAsString) {
99         principalToReturn = user.getUsername();
100    }
101
102     return this.createSuccessAuthentication(principalToReturn, authentication, user);

```

Nếu tôi bấm vào phương thức này, nó là một phương thức trừu tượng có sẵn bên trong lớp này. Vì vậy, phương pháp này được triển khai bên trong DaoAuthenticationProvider của tôi.

```
--AbstractUserDetailsAuthenticationProvider

protected abstract void additionalAuthenticationChecks(UserDetails
userDetails, UsernamePasswordAuthenticationToken authentication) throws
AuthenticationException;
```

Vì vậy, đây là phương pháp mà tôi đang nói. Vì vậy, tôi sẽ đặt một điểm dừng bên trong phương pháp này và hãy xem mật khẩu của tôi được xác thực như thế nào với PasswordEncoder mặc định mà chúng tôi đã sử dụng. Tương tự, hãy để tôi vào trình duyệt và thử đăng nhập. Vì vậy, ở đây, tôi đang cố truy cập một API là tài khoản của tôi và nó sẽ hỏi tôi thông tin đăng nhập. Tên người dùng mà tôi có thể nhập là john Doe@example.com và mật khẩu là 54321. Vì vậy, ngay sau khi tôi nhập vào đăng nhập, bạn

có thể thấy quá trình thực thi của tôi bị dừng bên trong phương thức addAuthenticationChecks của DaoAuthenticationProvider.

```
--DaoAuthenticationProvider
```

```
protected void additionalAuthenticationChecks(UserDetails userDetails,
UsernamePasswordAuthenticationToken authentication) throws
AuthenticationException {
    if (authentication.getCredentials() == null) {
        this.logger.debug("Failed to authenticate since no credentials
provided");
        throw new
BadCredentialsException(this.messages.getMessage("AbstractUserDetailsAuthenti
cationProvider.badCredentials", "Bad credentials"));
    } else {
        String presentedPassword =
authentication.getCredentials().toString();
        if (!this.passwordEncoder.matches(presentedPassword,
userDetails.getPassword())) {
            this.logger.debug("Failed to authenticate since password does not
match stored value");
            throw new
BadCredentialsException(this.messages.getMessage("AbstractUserDetailsAuthenti
cationProvider.badCredentials", "Bad credentials"));
        }
    }
}
```

Đầu tiên, nó đang kiểm tra xem thông tin đăng nhập có sẵn bên trong đối tượng **Authentication** của tôi hay không. Vì vậy, các thông tin đăng nhập có sẵn. Nếu thông tin đăng nhập có sẵn, nó sẽ được gán cho chuỗi của bạn, đó là mật khẩu được cung cấp. Vì vậy, bất kỳ mật khẩu nào bạn nhận được từ đối tượng **Authentication** đều là mật khẩu mà user của tôi đã nhập. Ở đây, chúng tôi đang gọi một phương thức có tên là các trận đấu có sẵn bên trong PasswordEncoder.

Nếu bạn có thể đánh dấu trên PasswordEncoder này, thì PasswordEncoder hiện tôi đang sử dụng là NoOpPasswordEncoder. PasswordEncoder này không được bảo mật, thay vào đó hãy sử dụng một bộ mã hóa mật khẩu khác có sẵn. Và họ cũng nhấn mạnh rằng điều này rất hữu ích để kiểm tra nơi có thể ưu tiên làm việc với mật khẩu văn bản gốc. Và bên trong phương thức **matches** này, chúng tôi chỉ đơn giản là kiểm tra sự bằng nhau của hai chuỗi là mật khẩu mà tôi nhận được từ user và mật khẩu khác mà tôi đã tải từ cơ sở dữ liệu. Vì vậy, tham số thứ hai khác mà chúng tôi nhận được từ đối tượng user detail được tải từ cơ sở dữ liệu. Vì vậy, nếu hai chuỗi này bằng nhau thì nó sẽ trả về giá trị Boolean true. Vì một số lý do, nếu thông tin đăng nhập không khớp thì chúng tôi sẽ nhận được ngoại lệ này, đây là thông tin đăng nhập không hợp lệ. Vì mật khẩu khớp với nhau nên nó sẽ không nhập vào bên trong (không nghe được). Vì vậy, nhận được phản hồi thành công.

## 2. Encoding Vs Encryption Vs Hashing

### Encoding Vs Encryption Vs Hashing

eazy  
bytes

## Different ways of Pwd management

Encoding	Encryption	Hashing
<ul style="list-style-type: none"><li>✓ Encoding is defined as the process of converting data from one form to another and has nothing to do with cryptography.</li><li>✓ It involves no secret and completely reversible.</li><li>✓ Encoding can't be used for securing data. Below are the various publicly available algorithms used for encoding.</li></ul> <p>Ex: ASCII, BASE64, UNICODE</p>	<ul style="list-style-type: none"><li>✓ Encryption is defined as the process of transforming data in such a way that guarantees confidentiality.</li><li>✓ To achieve confidentiality, encryption requires the use of a secret which, in cryptographic terms, we call a "key".</li><li>✓ Encryption can be reversible by using decryption with the help of the "key". As long as the "key" is confidential, encryption can be considered as secured.</li></ul>	<ul style="list-style-type: none"><li>✓ In hashing, data is converted to the hash value using some hashing function.</li><li>✓ Data once hashed is non-reversible. One cannot determine the original data from a hash value generated.</li><li>✓ Given some arbitrary data along with the output of a hashing algorithm, one can verify whether this data matches the original input data without needing to see the original data</li></ul>

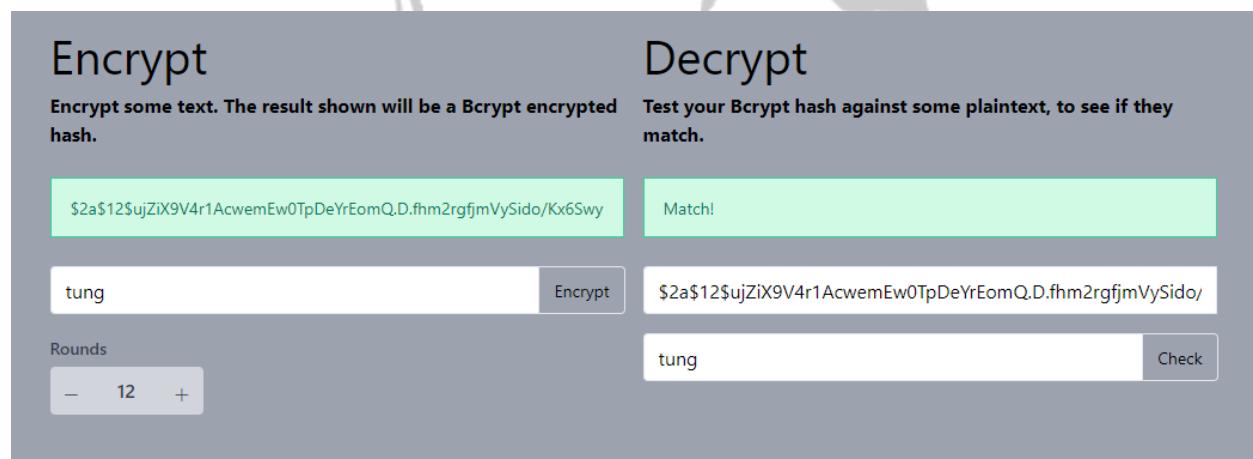
**Encoding - Mã hóa** là một quá trình sẽ chuyển đổi dữ liệu của bạn từ dạng này sang dạng khác và nó không liên quan đến bất kỳ tính bảo mật nào, điều đó có nghĩa là nếu tôi tuân theo mã hóa và cố gắng chuyển đổi văn bản đơn giản của mình thành mật khẩu thành giá trị được mã hóa thì bất kỳ ai cũng có thể lấy giá trị được mã hóa đó và họ có thể theo dõi quá trình giải mã. Và khi họ thực hiện việc giải mã này, họ có thể dễ dàng nhìn thấy văn bản ban đầu thành mật khẩu mà họ đã sử dụng trong quá trình mã hóa là gì.

Cùng với đó, chúng ta có thể chuyển sang tùy chọn thứ hai mà chúng ta có, đó là **encryption - mã hóa dữ liệu**. Vì vậy, **encryption** cũng đang trong quá trình chuyển đổi dữ liệu đơn giản của bạn theo cách đảm bảo tính bảo mật. Vì vậy, cách chúng tôi sẽ đạt được tính bảo mật này là bất cứ khi nào chúng tôi cố gắng **encryption** một số dữ liệu, chúng tôi tuân theo một số thuật toán và với thuật toán mã hóa của tôi có thể mã hóa văn bản thuần túy của tôi thành mật khẩu thành định dạng mà không ai có thể hiểu được. Và một khi chúng tôi mã hóa văn bản thuần túy thành mật khẩu nếu bạn muốn hiểu văn bản thuần túy thành mật khẩu cho một giá trị được mã hóa nhất định là gì thì chúng tôi phải thực hiện việc giải mã đó. Vì vậy, **decryption - giải mã** là một quá trình mã hóa ngược lại. Vì vậy, bất cứ khi nào ai đó muốn giải mã văn bản thuần túy thành mật khẩu với một giá trị được mã hóa nhất định, thì chắc chắn họ phải sử dụng cùng một thuật toán mà tôi đã sử dụng và cùng một bí mật là khóa mà tôi đã sử dụng trong quá trình mã hóa. Vì vậy, thuật toán mã hóa này và các khóa bí mật thường sẽ được duy trì dưới dạng dữ liệu bí mật bên trong back-end application.

Vì vậy, đó là lý do tại sao điều này tốt hơn rất nhiều so với encoding miễn là bạn có thể bảo vệ chi tiết thuật toán của mình và chi tiết khóa bí mật mà bạn có thể yên tâm ngủ bằng cách coi như không ai có thể hack mật khẩu của bạn, nhưng điều này cũng có một vấn đề. Vấn đề là quản trị viên máy chủ của bạn là nhà phát triển của bạn hoặc một số người thử nghiệm có quyền truy cập vào các biến này

như bí mật và thuật toán. Họ có thể dễ dàng tận dụng những thứ này và họ có thể giải mã các giá trị được mã hóa và họ có thể sử dụng sai mật khẩu, điều đó có nghĩa là có khả năng đảo ngược các giá trị được mã hóa thành mật khẩu văn bản thuần túy của bạn. Vì vậy, đó là lý do tại sao điều này cũng không được khuyến nghị xem xét để quản lý mật khẩu. Tất nhiên, bạn có thể đã thấy mã hóa và giải mã này được sử dụng nhiều trong các ấn phẩm web để lưu trữ mật khẩu của cơ sở dữ liệu hoặc một số máy chủ thay vì đề cập trực tiếp đến dữ liệu nhạy cảm bên trong thuộc tính của bạn, các nhà phát triển tệp luôn tuân theo kiểu **encryption** này để lưu trữ các thuộc tính bên trong tệp thuộc tính của bạn.

**Hashing**, dữ liệu của bạn sẽ được chuyển đổi thành giá trị băm bằng một số hàm băm toán học. Vì vậy, một khi bạn áp dụng hàm băm trên mật khẩu văn bản thuần túy hoặc trên dữ liệu của mình, nó sẽ không thể đảo ngược được. Vì nó không thể đảo ngược và không ai có thể nhìn thấy mật khẩu văn bản thuần túy bằng cách biết giá trị băm, điều này đã trở thành tiêu chuẩn công nghiệp để lưu trữ mật khẩu bên trong các hệ thống lưu trữ như cơ sở dữ liệu. Cùng với đó, chẳng hạn như nếu bạn không thể đảo ngược mật khẩu văn bản thuần túy thì chúng tôi sẽ xác thực bằng mật khẩu do user nhập trong quá trình đăng nhập như thế nào. Tương tự, chúng tôi có một tùy chọn bên trong hàm băm. Vì vậy, bất cứ khi nào bạn đang cố gắng so sánh mật khẩu văn bản thuần túy do người dùng nhập với giá trị đã được băm sẵn được lưu trữ bên trong cơ sở dữ liệu. Vì vậy, trong các loại tình huống này, chúng tôi phải áp dụng hàm băm trên mật khẩu do user nhập. Vì vậy, bạn sẽ có hai giá trị băm.



The screenshot shows a web-based Bcrypt hasher tool with two main sections: 'Encrypt' on the left and 'Decrypt' on the right.

**Encrypt Section:**  
Text input: \$2a\$12\$ujZiX9V4r1AcwemEw0TpDeYrEomQ.D.fhm2rgfjmVySido/Kx6Swy  
Text input: tung  
Rounds: 12

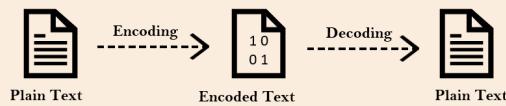
**Decrypt Section:**  
Text input: Match!  
Text input: \$2a\$12\$ujZiX9V4r1AcwemEw0TpDeYrEomQ.D.fhm2rgfjmVySido/  
Text input: tung  
Check button

Bạn có thể coi mã hóa này là một hàm băm ở đây, bạn cũng có thể thấy họ đã đề cập đến hàm băm được mã hóa này. Vì vậy, **encrypt** là một thuật ngữ chung mà chúng tôi có thể sử dụng bất cứ khi nào chúng tôi cố gắng chuyển đổi chuỗi của mình thành dữ liệu bí mật. Và mặc định số vòng là 12 như các bạn xem tại đây. Vì vậy, thuật toán Bcrypt này, nó cũng sẽ chấp nhận bao nhiêu vòng bạn muốn xem xét bằng thuật toán này để tạo ra giá trị băm.

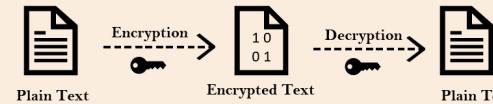
## Encoding Vs Encryption Vs Hashing

eazy  
bytes

### Encoding & Decoding



### Encryption & Decryption



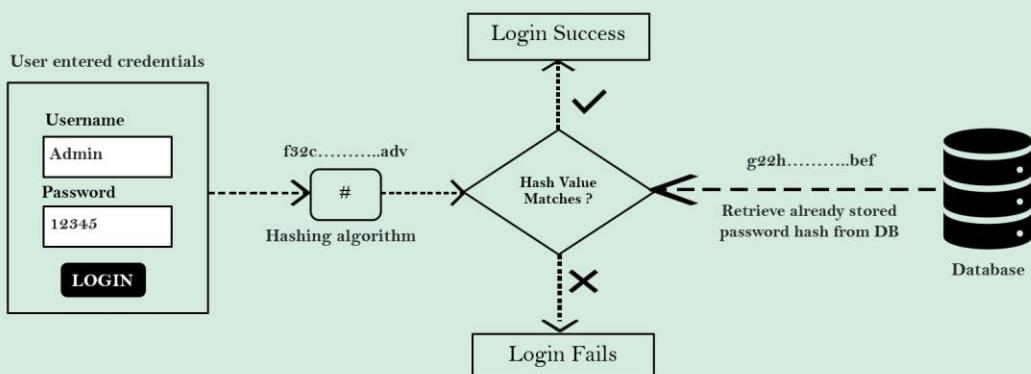
### Hashing (Only 1-way)



### 3. Cách mật khẩu sẽ được xác thực bằng Hashing & PasswordEncoders

#### HOW PASSWORDS VALIDATED With Hashing & PasswordEncoders

eazy  
bytes



Storing & managing the passwords with hashing is the recommended approach for Production applications. With various PasswordEncoders available inside Spring Security, it makes our life easy.

Thao tác đầu tiên mà người dùng của tôi đã nhập thông tin đăng nhập của anh ấy, giống như quản trị viên và 1, 2, 3, 4, 5. **Spring Security** sẽ tận dụng một trong những bộ mã hóa mật khẩu mà chúng ta sẽ thảo luận trong các bài giảng sắp tới và với sự trợ giúp của bộ mã hóa mật khẩu này, nó sẽ cố gắng băm mật khẩu được nhập bởi user của tôi. Trước tiên, hãy nghĩ rằng mật khẩu băm này bắt đầu bằng F32C, sau đó là ADV. Vì vậy, đây là cách chuỗi băm của mật khẩu này sẽ trông như thế nào. Vì vậy,

sau khi quá trình băm này hoàn tất, Spring Security của tôi cũng sẽ tải giá trị băm hiện có được tạo trong quá trình đăng ký của user của tôi từ cơ sở dữ liệu. Và nó sẽ giữ giá trị băm này bên trong đối tượng User Detail. Và chuỗi băm này mà chúng tôi đã tìm nạp từ cơ sở dữ liệu trông giống như hàm băm G22, theo sau là BEF. Vì vậy, hãy nghĩ rằng đây là cách chuỗi băm sẽ trông giống như chuỗi mà chúng ta tìm nạp từ cơ sở dữ liệu. Nếu bạn nhìn vào cả hai chuỗi băm, chúng thực sự khác nhau, nhưng giá trị băm sẽ giống nhau.

Và nếu giá trị băm này giống nhau thì thao tác đăng nhập của tôi sẽ thành công, nếu không thì thao tác đăng nhập của tôi sẽ thất bại. Vì vậy, bộ mã hóa mật khẩu của tôi sẽ xử lý tất cả các kết hợp chuỗi băm này và tìm giá trị băm của chúng và đảm bảo liệu giá trị băm có khớp hay không. Vì vậy, trong toàn bộ hoạt động này, bạn có thể thấy chúng tôi không bao giờ lưu trữ văn bản thuần túy cho mật khẩu bên trong cơ sở dữ liệu. Đồng thời, trong khi so sánh các mật khẩu, chúng tôi cũng không so sánh chúng ở dạng văn bản thuần túy với mật khẩu. Mọi thứ chúng ta đang xử lý theo kiểu chuỗi băm. Vì vậy, tiêu chuẩn băm này mang lại rất nhiều lợi thế khi chúng ta xử lý việc quản lý mật khẩu vì nó siêu, siêu an toàn và không thể đảo ngược. Cùng với đó, tôi có thể đảm bảo tính bảo mật và toàn vẹn cho user của mình rằng mật khẩu của bạn an toàn với chúng tôi và chúng tôi có thể yên tâm ngủ mà không phải lo lắng về mật khẩu.

#### 4. Deep dive of PasswordEncoder interface

**DETAILS OF PASSWORDENCODER**

**Methods inside PasswordEncoder Interface**

```
public interface PasswordEncoder {  
    String encode(CharSequence rawPassword);  
    boolean matches(CharSequence rawPassword, String encodedPassword);  
    default boolean upgradeEncoding(String encodedPassword) {  
        return false;  
    }  
}
```

**Different implementations of PasswordEncoder inside Spring Security**

★ NoOpPasswordEncoder <small>(Not recommended for Prod apps)</small>	★ BCryptPasswordEncoder
★ StandardPasswordEncoder <small>(Not recommended for Prod apps)</small>	★ SCryptPasswordEncoder
★ Pbkdf2PasswordEncoder	★ Argon2PasswordEncoder

PasswordEncoder là một interface và nó có hai phương thức trừu tượng và một phương thức mặc định. Vì vậy, phương thức trừu tượng đầu tiên mà chúng ta có là **encode**. Vì vậy, phương pháp mã hóa này, chúng tôi có thể sử dụng trong quá trình đăng ký của user. Vì vậy, đối với phương thức mã hóa này, nếu bạn xem mật khẩu văn bản thuần túy mà user của tôi đã nhập trong quá trình đăng ký, nó sẽ chuyển đổi mật khẩu thô hoặc mật khẩu văn bản thuần túy đó thành chuỗi băm hoặc giá trị được mã hóa dựa trên PasswordEncoder mà bạn đang sử dụng. Mặc dù tên phương thức đang nói mã hóa,

nhưng không có quá trình xử lý mã hóa nào đang diễn ra. Vì vậy, dựa trên PasswordEncoder mà bạn sử dụng, thuật toán băm tương ứng và quá trình băm sẽ diễn ra ở hậu trường.

match là một phương thức mà chúng ta cần sử dụng trong quá trình đăng nhập để so sánh mật khẩu do người dùng nhập với mật khẩu đã được lưu trữ bên trong cơ sở dữ liệu. Vì vậy, phương thức khớp này sẽ chấp nhận rawPassword do user của tôi nhập trong quá trình đăng nhập. Và tham số thứ hai là mật khẩu được mã hóa hoặc mật khẩu băm mà chúng tôi lấy từ cơ sở dữ liệu với sự trợ giúp của loadUserByUsername. Vì vậy, hai tham số này, nếu bạn vượt qua, đăng sau hậu trường, phương thức so khớp này sẽ có logic băm mật khẩu thô của tôi trước tiên bằng cách sử dụng cùng một thuật toán băm. Nó sẽ cố gắng so sánh hai chuỗi băm và lấy giá trị băm cho cả hai chuỗi. Nếu các giá trị băm giống nhau, thì phương thức so khớp này sẽ trả về true và trong trường hợp đó, thao tác đăng nhập sẽ thành công. Mặt khác, nếu các giá trị băm không khớp, phương thức khớp này sẽ trả về false và cùng với đó, thao tác đăng nhập của tôi sẽ không thành công.

Và phương thức cuối cùng bên trong interface này mà chúng ta có là upgradeEncoding. Vì vậy, đây là một phương thức mặc định và nó có một số logic mặc định trả về giá trị sai mọi lúc. Vì vậy, mục đích của phương pháp này là đôi khi bất cứ khi nào bạn muốn làm cho cuộc sống của một hacker trở nên rất, rất khó khăn để hack và giải mã mật khẩu của bạn và cố gắng hiểu mật khẩu văn bản thuần túy của các giá trị băm của bạn là gì, nếu bạn muốn làm cho cuộc sống của hacker trở nên khó khăn, sau đó bạn có thể mã hóa mật khẩu của mình hai lần. Vì vậy, thay vì chỉ thực hiện băm một lần, bạn có thể thực hiện băm hai lần. Vì vậy, về cơ bản, upgradeEncoding này với giá trị trả về là true sẽ tăng cường bảo mật cho mật khẩu của bạn. Theo mặc định, điều này là sai vì chúng ta nên sử dụng mật khẩu văn bản thuần túy mặc định, băm một lần.

## 5. Tìm hiểu sâu về PasswordEncoder implementation

Có nhiều lớp implementation khác nhau của PasswordEncoder trong Java Spring Security Framework. Một số implementation phổ biến bao gồm:

1. BCryptPasswordEncoder: Sử dụng thuật toán BCrypt để mã hóa mật khẩu.  
BCryptPasswordEncoder sử dụng salt để ngăn chặn các cuộc tấn công bằng cách thêm dữ liệu ngẫu nhiên vào mật khẩu trước khi mã hóa. (khuyên dùng)
2. PbKdf2PasswordEncoder: Sử dụng thuật toán PBKDF2 để mã hóa mật khẩu.  
PBKDF2PasswordEncoder cũng sử dụng salt như BCryptPasswordEncoder, nhưng nó sử dụng một số lần lặp để làm chậm quá trình mã hóa, từ đó tăng tính bảo mật. Vì vậy, bộ mã hóa mật khẩu này tôi cũng không khuyên bạn nên sử dụng bên trong các ứng dụng sản xuất nhưng bạn vẫn có thể thấy một số ứng dụng được phát triển có thể năm năm trở lại hoặc sáu năm trở lại. Họ có thể đang sử dụng bộ mã hóa mật khẩu này bởi vì trong thời gian đó, nó được coi là an toàn. Nhưng ngày nay, với những tiến bộ mới nhất xảy ra với CPU và GPU, điều này không còn an toàn nữa. Vì vậy, nếu hacker nào đó có máy GPU cao cấp, giống như máy đơn vị xử lý đồ họa, có khả năng xử lý nhiều dữ liệu và hướng dẫn, họ có thể dễ dàng áp dụng các cuộc tấn công vũ phu vào giá trị băm của bạn và họ có thể đoán mật khẩu văn bản đơn giản.
3. SCryptPasswordEncoder: Sử dụng thuật toán SCrypt để mã hóa mật khẩu.  
SCryptPasswordEncoder cũng sử dụng salt và có thể được cấu hình với các tham số khác nhau để tăng tính bảo mật.
4. Argon2PasswordEncoder: đây thậm chí còn là thuật toán băm mới nhất, trong đó có ba chiều. Vì vậy, bất cứ khi nào ai đó đang cố gắng tận dụng điều này Thuật toán băm

Argon2PasswordEncoder hoặc Argon2, hacker của tôi hoặc bất kỳ mã nào, họ phải chỉ định ba loại tài nguyên. Một là tính toán, thứ hai là bộ nhớ, và cái thứ ba là số lượng chủ đề hoặc nhiều lõi của CPU. Vì vậy, với tất cả điều này, thực sự là không thể thực hiện được tấn công bất cứ khi nào chúng tôi sử dụng thuật toán băm Argon2. Vì vậy, đó là lý do tại sao ngay cả Argon2PasswordEncoder cũng vậy, là một sự cân nhắc tốt cho các ứng dụng web sản xuất của bạn. Nhưng ở đây, bạn có thể rất bị cám dỗ để sử dụng Argon2PasswordEncoder bởi vì nó là mạnh nhất.

Để sử dụng PasswordEncoder trong ứng dụng Spring Security, bạn cần cấu hình một bean của một implementation cụ thể và sử dụng nó để mã hóa và so sánh mật khẩu. Ví dụ:

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    // ...
}
```

Trên đây là một giới thiệu tổng quan về PasswordEncoder và các implementation phổ biến. Điều quan trọng là chọn một implementation phù hợp với yêu cầu bảo mật của ứng dụng của bạn.

## 6. Demo đăng ký người dùng mới với bộ mã hóa mật khẩu Bcrypt

Cấu hình BCryptPasswordEncoder trong lớp cấu hình Spring Security của bạn:

```
@Configuration
public class ProjectSecurityConfig {
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

- Tạo RestController để thực hiện đăng ký.

```
@RestController
public class LoginController {

    @Autowired
    private CustomerRepository customerRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @PostMapping("/register")
    public ResponseEntity<String> registerUser(@RequestBody Customer
customer) {
```

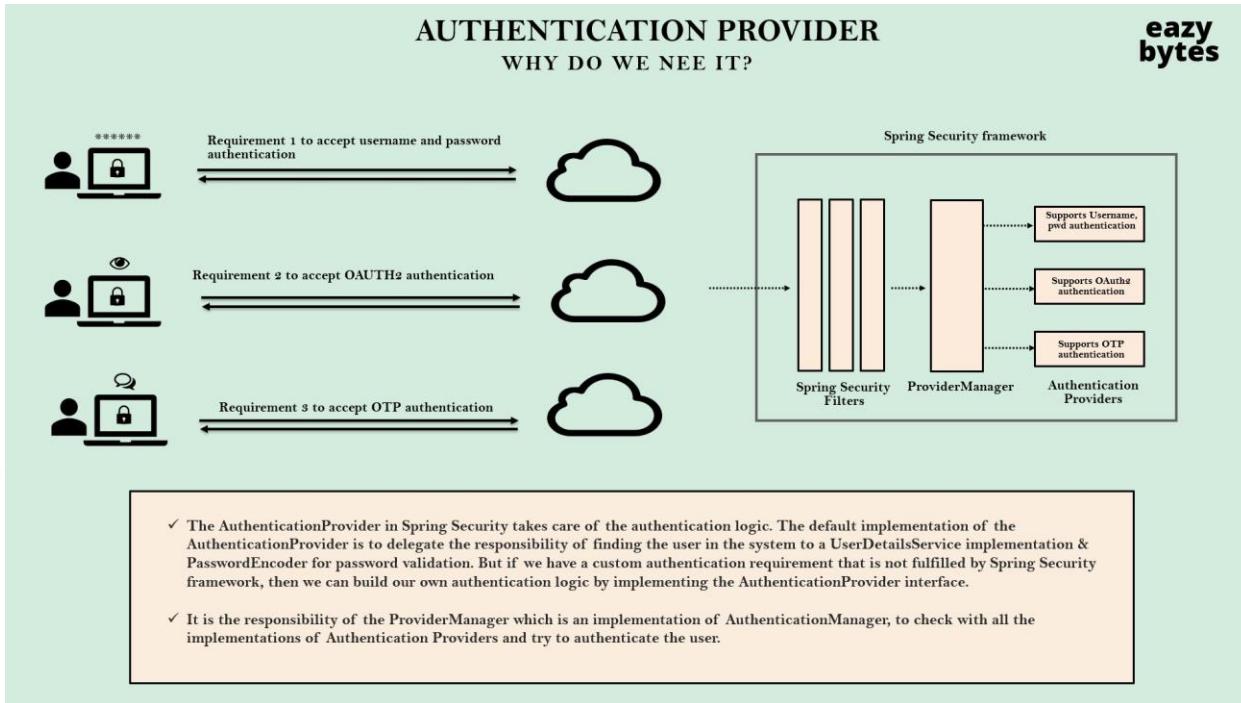
```
Customer savedCustomer = null;
ResponseEntity response = null;
try {
    String hashPwd = passwordEncoder.encode(customer.getPwd());
    customer.setPwd(hashPwd);
    savedCustomer = customerRepository.save(customer);
    if (savedCustomer.getId() > 0) {
        response = ResponseEntity
            .status(HttpStatus.CREATED)
            .body("Given user details are successfully
registered");
    }
} catch (Exception ex) {
    response = ResponseEntity
        .status(HttpStatus.INTERNAL_SERVER_ERROR)
        .body("An exception occurred due to " + ex.getMessage());
}
return response;
}

}
```



## Section 5: Understanding Authentication Provider and Implementing it

### 1. Tại sao chúng ta nên xem xét việc tạo AuthenticationProvider của riêng mình



Bất cứ khi nào chúng tôi sử dụng default authentication provider với tư cách là nhà phát triển, trách nhiệm của chúng tôi chỉ là tải thông tin user detail từ hệ thống lưu trữ như cơ sở dữ liệu và định cấu hình mật khẩu và code mà chúng tôi muốn sử dụng. Vì vậy, với hai thay đổi này và nếu chúng tôi chuyển tiếp các user detail đó tới DaoAuthenticationProvider, thì DaoAuthenticationProvider của tôi sẽ đảm nhận việc xác thực user bằng cách so sánh mật khẩu và cùng với so sánh mật khẩu, nó cũng sẽ kiểm tra các tham số khác như tài khoản được đăng nhập hay không, tài khoản đã hết hạn hay chưa, thông tin đăng nhập đã hết hạn chưa.

Vì vậy, DaoAuthentication này cung cấp hầu hết các kịch bản mà chúng tôi muốn xem xét trong quá trình xác thực user và logic triển khai mặc định này mà chúng tôi có bên trong DaoAuthenticationProvider là quá đủ cho nhiều dự án, nhưng trong thế giới thực, chúng tôi sẽ có một số yêu cầu phức tạp bất cứ khi nào chúng tôi muốn xác thực user. Trong những tình huống như vậy, ngoài việc tải thông tin chi tiết về người dùng từ hệ thống lưu trữ, chúng tôi cũng muốn thực thi logic xác thực của riêng mình. Có thể khách hàng của tôi có một số yêu cầu riêng như chỉ cho phép user trên 18 tuổi vào hệ thống hoặc họ cũng có thể có yêu cầu chỉ cho phép user đăng nhập từ danh sách các quốc gia được phép. Vì vậy, yêu cầu tùy chỉnh có thể là bất cứ điều gì. Có thể có nhiều tình huống mà bạn có thể nghĩ đến nhưng điểm mà tôi muốn nhấn mạnh ở đây là bất cứ khi nào bạn muốn có logic xác thực tùy chỉnh của riêng mình, thì chắc chắn bạn phải viết **Authentication Provider** của riêng mình. Và bên trong đó, bạn cần viết tất cả logic xác thực của mình.

Cách tiếp cận đầu tiên muốn xem xét là sử dụng tên người dùng và mật khẩu. Và yêu cầu thứ hai là có thể đối với môi trường sản xuất là một số ứng dụng có độ nhạy cao là chấp nhận xác thực OAuth 2.0, đây là authentication và authorization framework nâng cao. Và yêu cầu xác thực thứ ba chấp nhận OTP như một phần của xác thực. Vì vậy, ở đây nếu bạn thấy, có ba yêu cầu khác nhau cho cùng một ứng

dụng dựa trên các tình huống khác nhau. Trong loại tình huống này, chắc chắn chúng ta có thể viết nhiều **Authentication Provider**.

Đầu tiên các bộ lọc Spring Security của tôi sẽ thực hiện công việc của chúng như chuyển đổi thông tin đăng nhập của người dùng thành một Authentication object, **Provider Manager** sẽ xuất hiện và **Provider Manager** này chịu trách nhiệm gọi tất cả **Authentication Providers** có bên trong ứng dụng của mình để xác thực thành công hoặc thất bại, vào thời điểm **Provider Manager** gọi tất cả các **Authentication Provider**. Vì vậy, vì bên trong thế giới thực hoặc bên trong các dự án thực, có thể có nhiều tình huống trong đó bạn phải viết logic xác thực tùy chỉnh của riêng mình, bạn nên hiểu cách xác định **Authentication Providers** của riêng mình và cách đảm bảo thực hiện xác thực với sự trợ giúp của **Authentication Providers** của riêng bạn.

## 2. Tìm hiểu về AuthenticationProvider methods

Bên trong interface **Authentication Provider**, chúng tôi có hai phương thức trừu tượng.

```
public interface AuthenticationProvider {  
    Authentication authenticate(Authentication authentication)  
        throws AuthenticationException;  
    boolean supports(Class<?> authentication);  
}
```

★ The `authenticate()` method receives and returns authentication object. We can implement all our custom authentication logic inside `authenticate()` method.

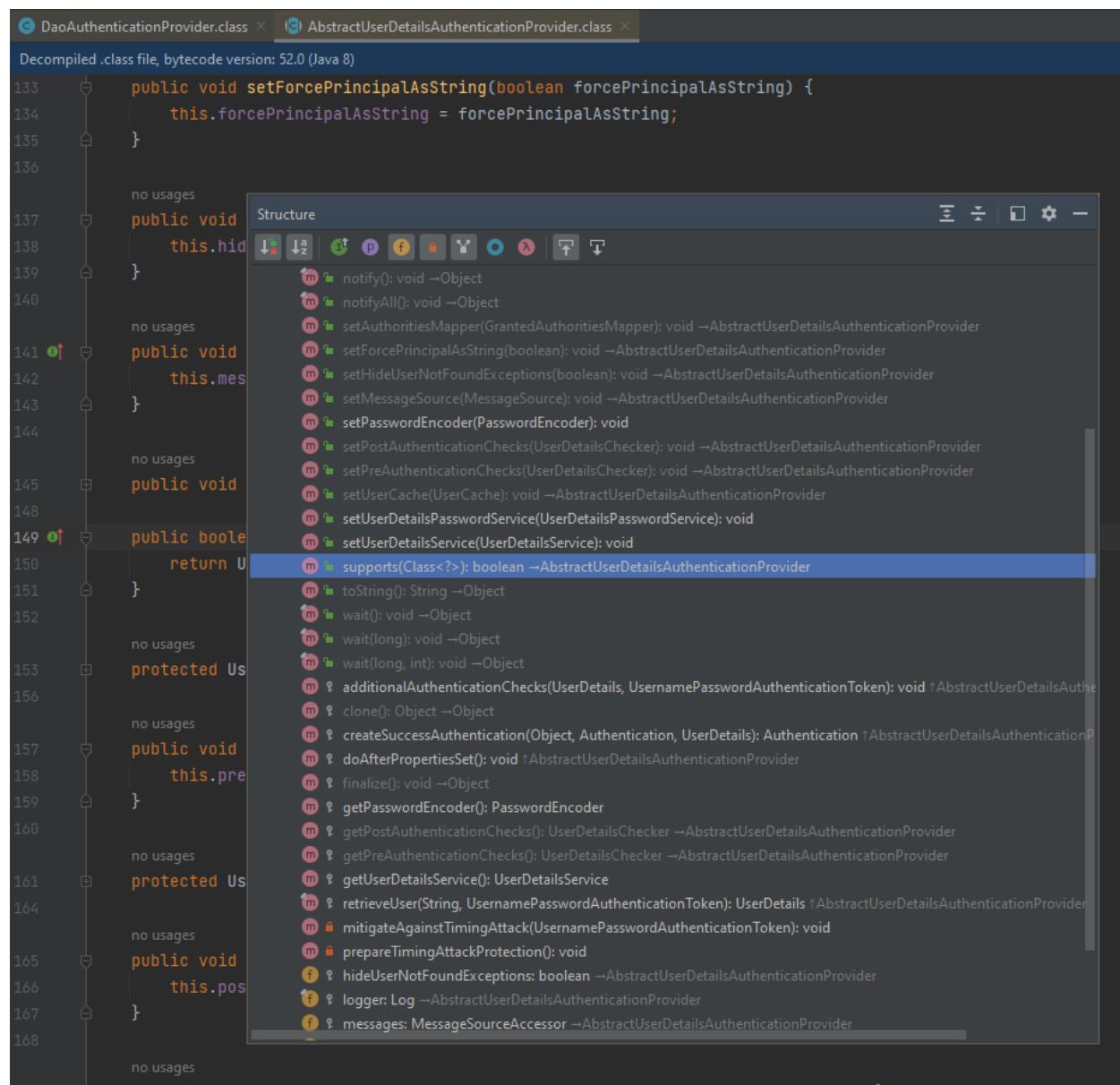
★ The second method in the `AuthenticationProvider` interface is `supports(Class<?> authentication)`. You'll implement this method to return true if the current `AuthenticationProvider` supports the type of the `Authentication` object provided.

Cái đầu tiên là phương thức **authenticate**. Vì vậy, đây là phương pháp quan trọng mà bạn cần viết tất cả logic xác thực của mình. Vì vậy, bạn có thể thấy đối với phương thức này, tham số đầu vào là đối tượng **Authentication** sẽ có tên người dùng và thông tin đăng nhập của user được điền bởi các **Spring Security filters**. Và khi bạn đã thực thi tất cả business logic có trong phương thức `authenticate` này, bạn phải đảm bảo rằng bạn cũng đang nhập thông tin xác thực thành công và viết đối tượng **Authentication** làm đầu ra từ phương thức này, nhưng lần này bên trong **Authentication object** mà chúng ta đang trả về từ **Authentication Provider**. Nó phải có thông tin xác thực có thành công hay không, để **Provider Manager** của tôi có thể quyết định có gọi các **Authentication Provider** khác có sẵn hay không.

Và phương pháp trừu tượng thứ hai mà chúng tôi có là **supports**. Vì vậy, bằng cách sử dụng các phương thức hỗ trợ này, chúng ta cần thông báo cho Spring Security framework loại xác thực nào mà tôi

muốn hỗ trợ với sự trợ giúp của **Authentication Provider**. Loại xác thực được sử dụng phổ biến nhất là với sự trợ giúp của **username và password**. Bằng cách sử dụng phương pháp hỗ trợ này, chúng tôi đang thông báo cho **Spring Security framework** về loại xác thực mà **Authentication Provider** của tôi phải được gọi.

Khi **Authentication Provider** của tôi không thể thực hiện xác thực thành công, thì rõ ràng nó sẽ thử **Authentication Provider** tiếp theo có sẵn trong dự án của chúng tôi. Phương pháp thứ hai mà chúng tôi có là **supports**. Vì vậy, hãy cố gắng hiểu cách viết mã bên trong phương thức **supports**. Trước tiên, tôi cần hiểu những gì có bên trong **DaoAuthenticationProvider**. Vì vậy, nếu bạn đi tìm **DaoAuthenticationProvider** và kiểm tra phương thức **supports**(extends **AbstractUserDetailsAuthenticationProvider**), bên trong phương thức **supports** này, bạn có thể thấy nó có một mã đơn giản nói rằng, "Tôi sẽ xử lý tất cả các xác thực thuộc loại tên người dùng backend application xác thực mật khẩu."



```

133     public void setForcePrincipalAsString(boolean forcePrincipalAsString) {
134         this.forcePrincipalAsString = forcePrincipalAsString;
135     }
136
137     no usages
138     public void hid() {
139     }
140
141     no usages
142     public void mes() {
143     }
144
145     no usages
146     public void
147     public boolean supports(Class<?> clazz) {
148         return U
149     }
150
151     no usages
152     protected Us
153
154     no usages
155     public void pre() {
156     }
157
158     no usages
159     protected Us
160
161     no usages
162     public void pos() {
163     }
164
165     no usages
166     public void
167     no usages
168
169     no usages

```

```
--> AbstractUserDetailsAuthenticationProvider
public boolean supports(Class<?> authentication) {
    return
UsernamePasswordAuthenticationToken.class.isAssignableFrom(authentication);
}
```

Vì hầu hết chúng ta muốn xác thực user với sự trợ giúp của tên người dùng và mật khẩu, chúng ta nên luôn xem xét backend application xác thực này bên trong lớp hỗ trợ của mình vì ngay cả khi bạn muốn triển khai OTP, đây là tham số thứ ba mà bạn muốn xây dựng trong quá trình xác thực, thì trước tiên bạn cần xác thực đúng người dùng bằng tên người dùng và mật khẩu. Nếu thành công, thì bạn cần đưa người dùng đến trang OTP. Và OTP là một business logic riêng biệt mà bạn có thể xác định dựa trên yêu cầu của riêng mình

### 3. Implementing and Customising the AuthenticationProvider inside our application

Và tên lớp mà tôi muốn cung cấp ở đây là **EasyBankUsernamePasswordAuthenticationProvider**. Khi chúng tôi xác định lớp này, chúng tôi cần đảm bảo rằng chúng tôi đang implementing một interface. Vậy tên interface bạn đã biết đó là **AuthenticationProvider**. Một khi tôi triển khai những điều này, chắc chắn tôi cần ghi đè hai phương thức. Vì vậy, ở đây tôi sẽ cố gắng thực hiện cả hai phương pháp này. Hãy để tôi nhấp vào phương thức triển khai này và tôi không muốn sao chép tài liệu Java, tôi chỉ muốn ghi đè lên chúng.

Thay đổi đầu tiên mà tôi muốn thực hiện là, bên trong phương thức **supports**, tôi chỉ muốn chuyển đến **Spring Security framework** của mình rằng tôi muốn hỗ trợ kiểu xác thực tên người dùng, mật khẩu. Vì vậy, tương tự, điều tôi có thể làm là Dao Authentication Provider và tìm phương thức **supports** để tôi có thể sao chép toàn bộ mã này vào **Authentication Provider** mới của mình

Bây giờ, bên trong phương thức **authenticate** này, chúng tôi có trách nhiệm xác định tất cả **authenticate method** ngay từ khi tải thông tin chi tiết người dùng từ hệ thống lưu trữ đó, so sánh mật khẩu. Và sau đó, chúng ta nên tạo một Authentication object thành công với thông tin xác thực thành công hay không. Vì vậy, để làm điều tương tự, bước đầu tiên mà tôi muốn thực hiện bên trong quá trình xác thực là, trước tiên tôi muốn tìm nạp thông tin chi tiết về người dùng của mình từ cơ sở dữ liệu. Và bước thứ hai mà tôi muốn thực hiện sau khi tìm nạp thông tin chi tiết về người dùng từ cơ sở dữ liệu là so sánh các mật khẩu. Vì vậy, đối với hai bước này, tôi cần một **Bean** để được **Autowired** với lớp này. Những beans này giống như **customer repository** mà chúng tôi có thể tìm nạp thông tin chi tiết về người dùng từ cơ sở dữ liệu. Và bean thứ hai mà tôi muốn đưa vào là password encoder. Vì vậy, hiện tại, password encoder mà chúng tôi đang sử dụng trong dự án của mình là **BCryptPasswordEncoder**. Vì vậy, với hai hạt này được đưa vào và Autowired, chúng ta nên bắt đầu logic xác thực của mình.

```
@Component
public class EazyBankUsernamePwdAuthenticationProvider implements
AuthenticationProvider {

    @Autowired
    private CustomerRepository customerRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Override
```

```

public Authentication authenticate(Authentication authentication) throws
AuthenticationException {
    String username = authentication.getName();
    String pwd = authentication.getCredentials().toString();
    List<Customer> customer = customerRepository.findByEmail(username);
    if (customer.size() > 0) {
        if (passwordEncoder.matches(pwd, customer.get(0).getPwd())) {
            List<GrantedAuthority> authorities = new ArrayList<>();
            authorities.add(new
SimpleGrantedAuthority(customer.get(0).getRole()));
            return new UsernamePasswordAuthenticationToken(username, pwd,
authorities);
        } else {
            throw new BadCredentialsException("Invalid password!");
        }
    } else {
        throw new BadCredentialsException("No user registered with this
details!");
    }
}

@Override
public boolean supports(Class<?> authentication) {
    return
(UsernamePasswordAuthenticationToken.class.isAssignableFrom(authentication));
}

}

```

Hai dòng đầu tiên rất đơn giản để hiểu. Chúng tôi chỉ đang cố tải tên người dùng và mật khẩu mà user của tôi đã cung cấp vào tên người dùng này và các biến mật khẩu và sử dụng tên người dùng này, trong trường hợp này, email, tôi đang cố tải thông tin chi tiết về người dùng với sự trợ giúp của phương pháp này, đó là findByEmail. Vì vậy, khi chúng tôi có đối tượng khách hàng này, chúng tôi sẽ kiểm tra xem có bất kỳ bản ghi nào được tìm nạp thành công từ cơ sở dữ liệu hay không. Nếu không, chúng ta chỉ đơn giản ném BadCredentialsException.

Mặt khác, nếu có một bản ghi được tìm từ cơ sở dữ liệu cho một email nhất định, nếu bạn thấy ở đây, thì chúng tôi chỉ đang xác thực mật khẩu do user của tôi nhập khớp với mật khẩu mà chúng tôi đã lưu trữ bên trong cơ sở dữ liệu. Đó là lý do tại sao chúng tôi đang sử dụng phương thức passwordEncoder.matches này. Đối với phương pháp đối sánh này, chúng tôi đang cung cấp mật khẩu thô do user của tôi nhập và mật khẩu băm, được lưu trữ bên trong cơ sở dữ liệu. Nếu mật khẩu phù hợp, chúng tôi sẽ tham gia vào khối if này. Nếu không, chúng ta sẽ ném BadCredentialsException. Bên trong if này, bạn có thể thấy tôi đang cố điền thông tin chi tiết về quyền hạn của user của mình. Vì vậy, các chi tiết về quyền hạn này hiện diện bên trong cột vai trò này trong bảng **customer** của tôi. Tôi chỉ đang cố gắng chuyển đổi giá trị vai trò chuỗi đó thành một lớp GrantedAuthority được cấp đơn giản. Và tương tự, tôi sẽ thêm vào danh sách authorities.

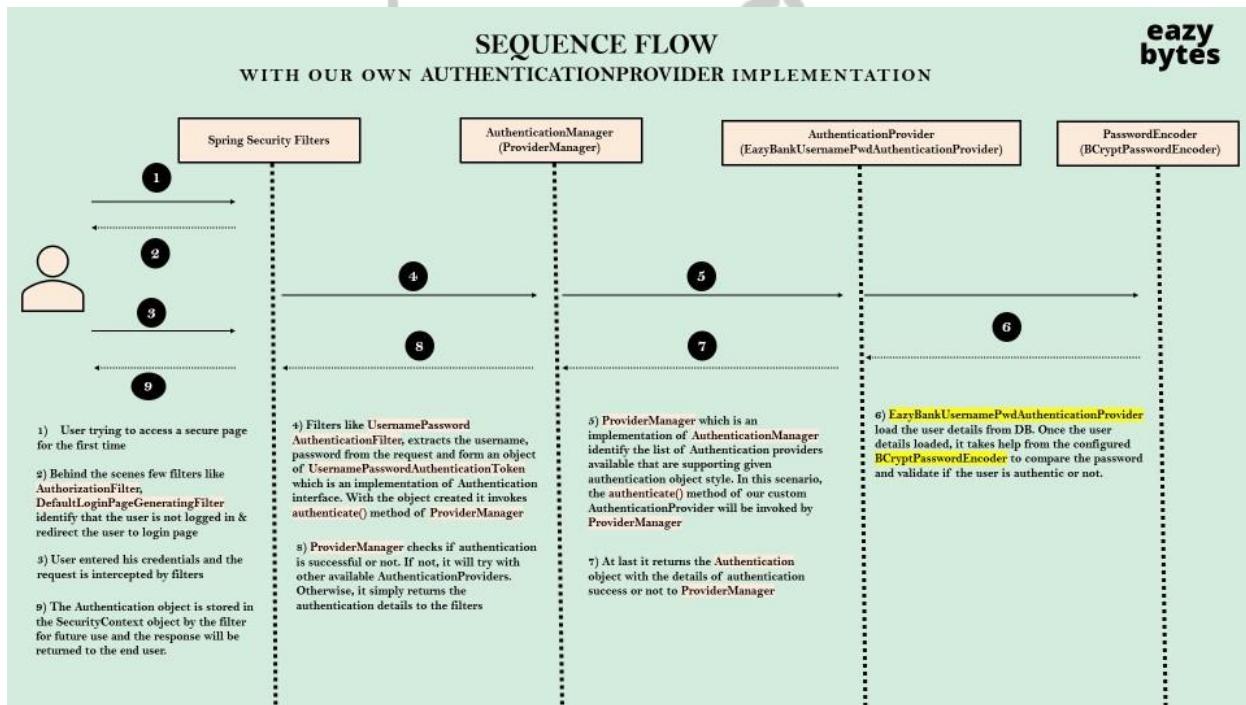
```
return new UsernamePasswordAuthenticationToken(username, pwd, authorities);
```

Và bây giờ bạn có thể thấy tôi vừa tạo một đối tượng UsernamePasswordAuthenticationToken.

Nếu bạn vào bên trong hàm tạo này, bạn có thể thấy chúng tôi chỉ đang đặt tên người dùng, thông tin đăng nhập và quyền hạn này vào Authentication object. Nhưng đồng thời, chúng tôi cũng thông báo rằng xác thực của tôi đã thành công. Bên trong hàm tạo này với dòng mã này, chúng tôi đang truyền đạt tới **ProviderManager** rằng xác thực của tôi thành công.

#### 4. Spring Security Sequence flow with custom AuthenticationProvider

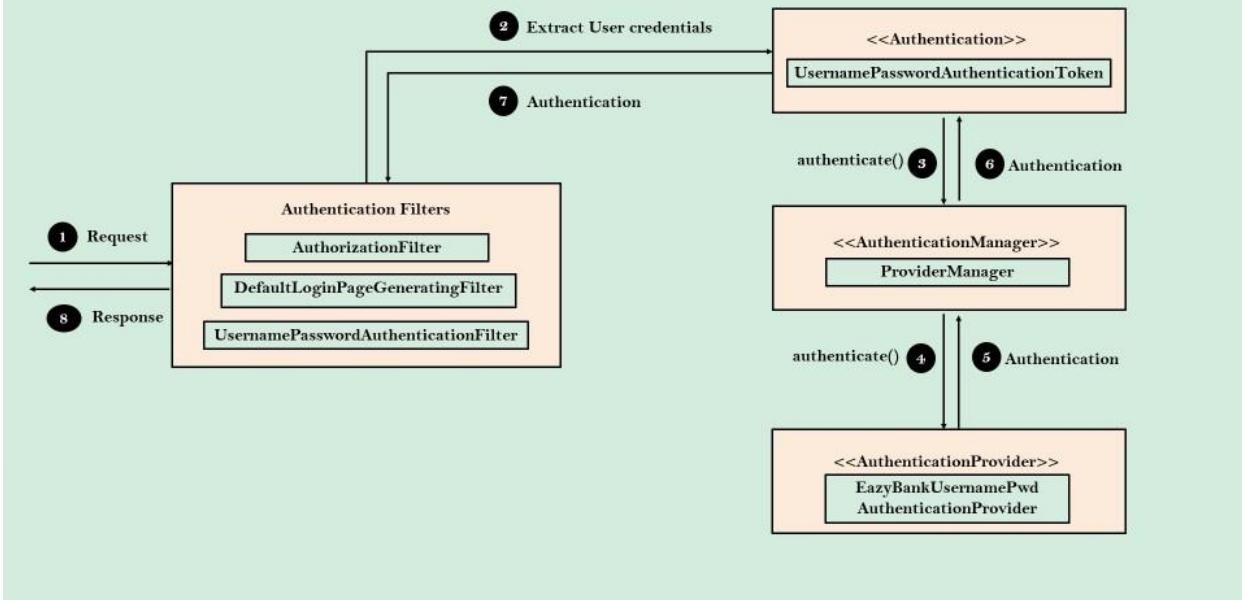
Chúng tôi đã triển khai **Authentication Provider** của riêng mình, đó là **username, password authentication provider** Eazybank. Vì vậy, **Provider Manager** của tôi sẽ gọi phương thức `authenticate` có trong **Authentication Provider** của tôi. Và bên trong **Authentication Provider** của tôi, tôi sẽ tìm nạp tất cả thông tin chi tiết về người dùng và tôi sẽ làm việc với bộ mã hóa mật khẩu để kiểm tra xem các mật khẩu có khớp hay không. Vì vậy, khi tôi hài lòng với tất cả Business Logic của mình, tôi sẽ trả lại Authentication object cho **Provider Manager**. Vì vậy, sự khác biệt quan trọng với các luồng trình tự mà chúng ta đã thảo luận trước đây là ở đây, chúng ta không còn tận dụng **UserDetailsService** hoặc các implement **UserDetailsService** như JWC user details manager, hay custom implementation của user detail service, hoặc user details manager, bởi vì chúng tôi đã viết chi tiết người dùng tìm nạp logic cũng bên trong chính phương thức `authenticate` của chúng tôi.



Nó hiện trông đơn giản, bởi vì chúng tôi không còn tận dụng **các user detail service và user details manager implementation classes**. Vì vậy, với điều đó, tôi cho rằng bạn cực kỳ hiểu rõ về các **Authentication Provider** cũng như cách ghi đè và xác định **Authentication Provider** của riêng bạn.

**SEQUENCE FLOW**  
WITH OUR OWN AUTHENTICATIONPROVIDER IMPLEMENTATION

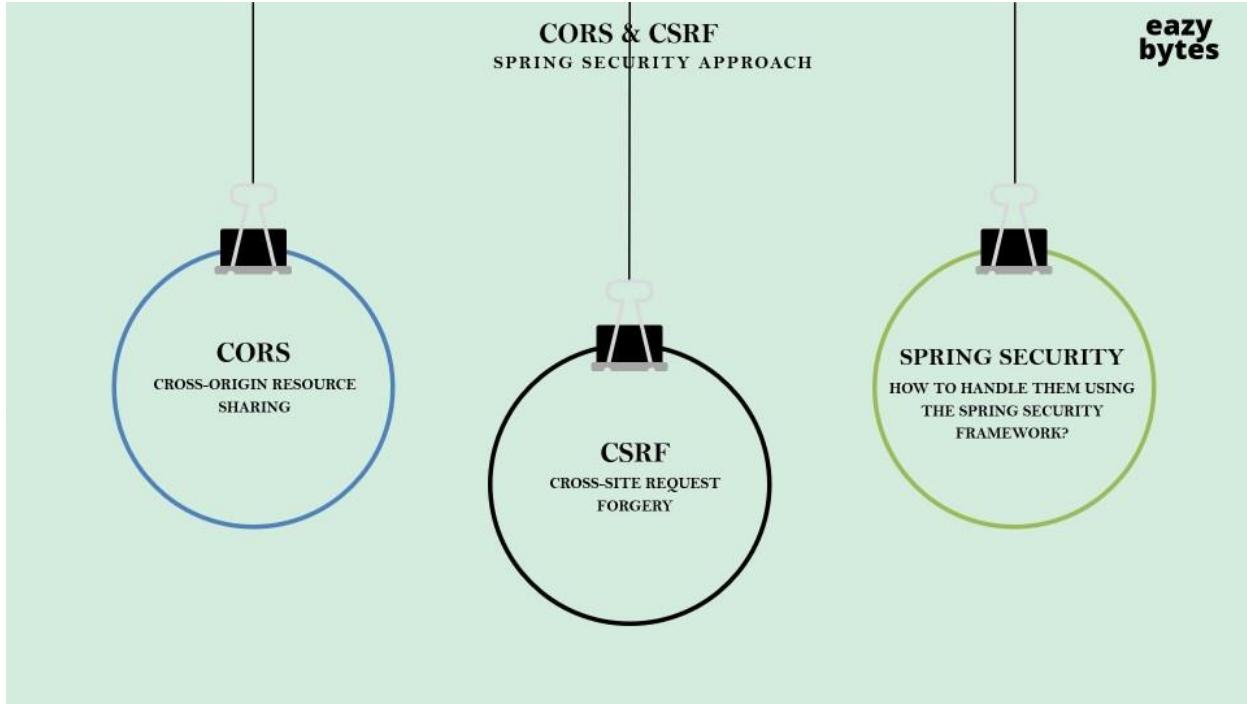
**eazy  
bytes**



Mar

## Section 6: Understanding CORs & CSRF

### 1. Setting up the EazyBank UI project



#### Angular Project Setup

- 1) Download and Install the node.js from <https://nodejs.org/en/download/> as per your operating system and check if the installation is successful using the command 'node -v' which will print the current installed version.
- 2) Run the command 'npm install -g @angular/cli' to install Angular CLI.
- 3) Download the visual code using the link <https://code.visualstudio.com/> and import the angular project provided.
- 4) Navigate to the folder inside the project where package.json is present and run the command 'npm install' by opening a terminal. This will install all the dependencies inside the new folder created with the name 'node\_modules'
- 5) Run the command 'ng serve' and it will start the application at 'http://localhost:4200/' by default.

#### Run project BankAppUi

This project was generated with Angular CLI version 14.1.2.

#### Development server

Run ng serve for a dev server. Navigate to <http://localhost:4200/>. The application will automatically reload if you change any of the source files.

#### Code scaffolding

Run `ng generate component component-name` to generate a new component. You can also use `ng generate directive|pipe|service|class|guard|interface|enum|module`.

### Build

Run `ng build` to build the project. The build artifacts will be stored in the `dist/` directory.

### Running unit tests

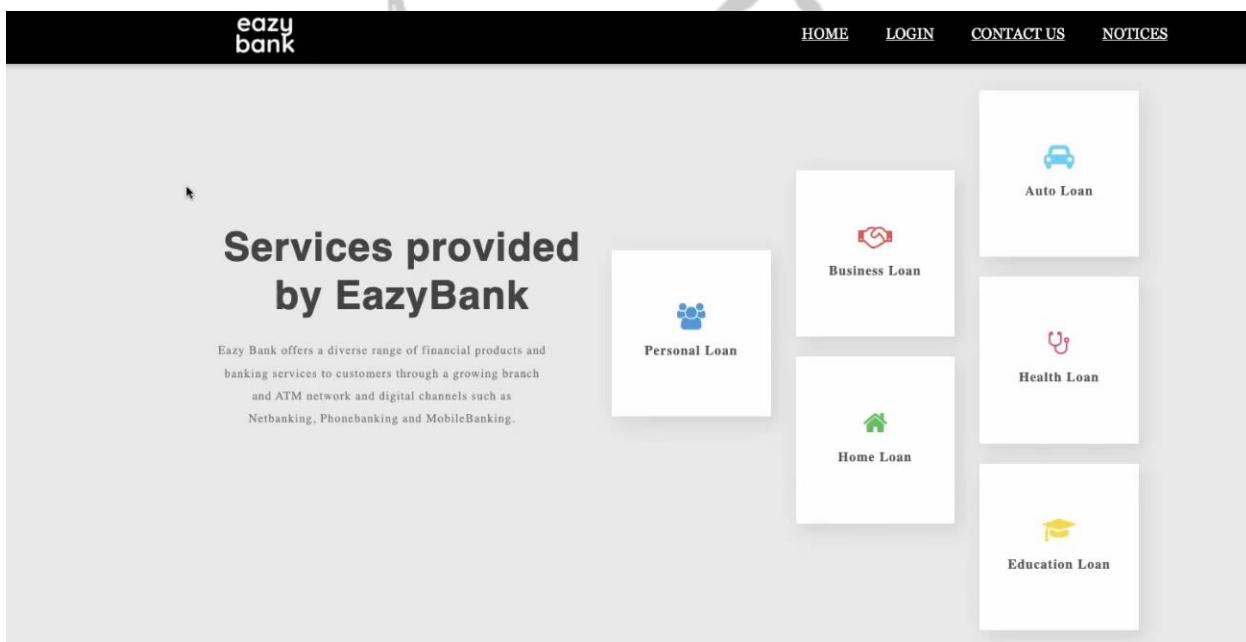
Run `ng test` to execute the unit tests via Karma.

### Running end-to-end tests

Run `ng e2e` to execute the end-to-end tests via a platform of your choice. To use this command, you need to first add a package that implements end-to-end testing capabilities.

### Further help

To get more help on the Angular CLI use `ng help` or go check out the [Angular CLI Overview and Command Reference page](#).



## 2. Understanding the UI project and walkthrough of the Angular code

Dưới đây là các bước cơ bản để tạo một dự án Angular và call API đến back end:

1. Cài đặt Angular CLI: Angular CLI là một công cụ dòng lệnh giúp bạn tạo và quản lý các dự án Angular. Để cài đặt Angular CLI, bạn cần mở cửa sổ dòng lệnh và chạy lệnh sau: `npm install -g @angular/cli`
2. Tạo một dự án Angular mới: Sau khi cài đặt Angular CLI, bạn có thể tạo một dự án Angular mới bằng cách chạy lệnh sau trong cửa sổ dòng lệnh: `ng new ten-du-an`. Thay `ten-du-an` bằng tên của dự án của bạn.

3. Chạy dự án Angular: Để chạy dự án Angular, bạn cần di chuyển đến thư mục chứa dự án và chạy lệnh sau: ng serve. Sau đó, bạn có thể truy cập vào địa chỉ http://localhost:4200 trong trình duyệt để xem ứng dụng của mình.
4. Tạo một service để call API: Trong Angular, bạn có thể sử dụng service để tương tác với back end và lấy dữ liệu từ API. Để tạo một service mới, bạn có thể chạy lệnh sau trong cửa sổ dòng lệnh: ng generate service ten-service. Thay ten-service bằng tên của service của bạn.
5. Sử dụng HttpClientModule để call API: Để call API từ service của bạn, bạn cần import HttpClientModule vào module chính của ứng dụng và thêm nó vào mảng imports. Sau đó, bạn có thể inject HttpClient vào service của mình và sử dụng nó để call API.

Ví dụ:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
```

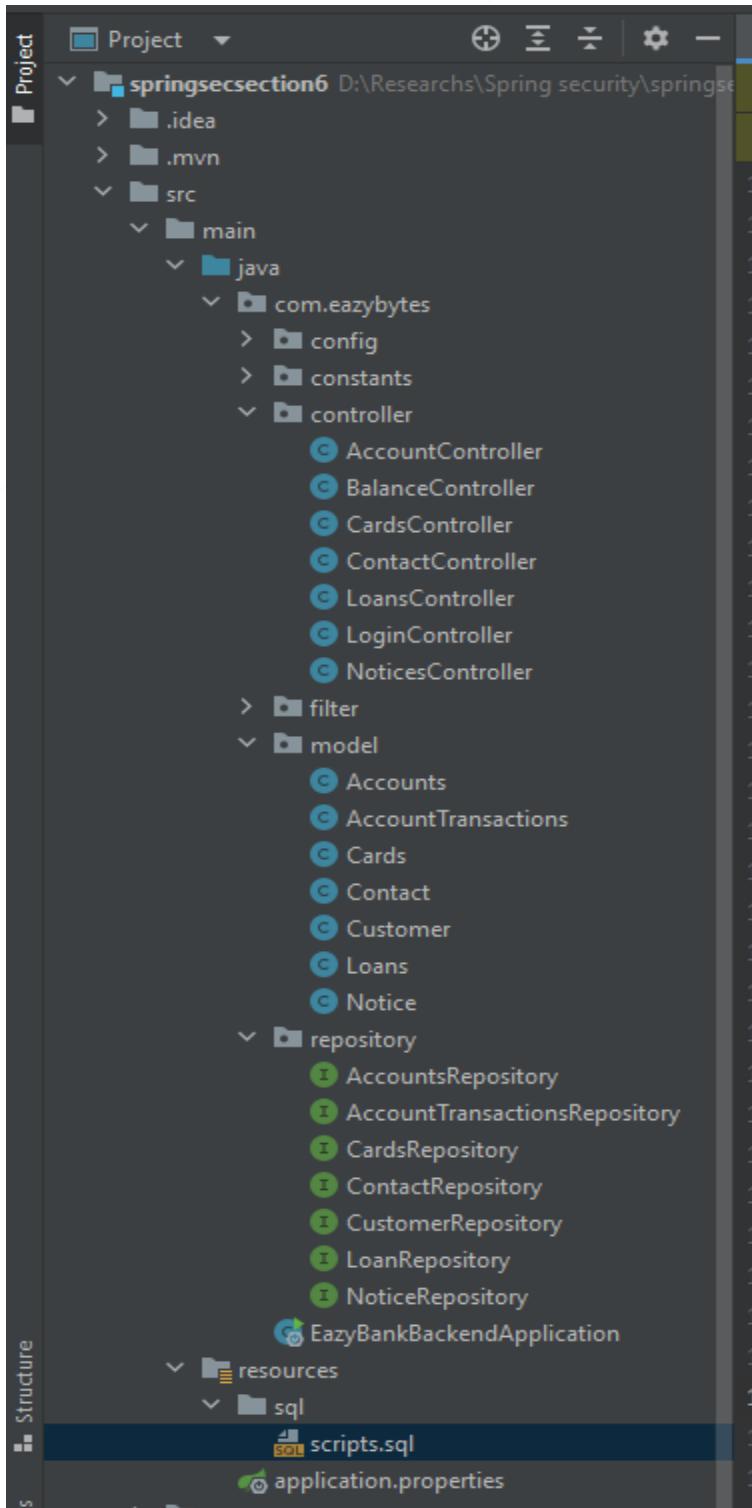
```
@Injectable({
  providedIn: 'root'
})
export class DataService {
  constructor(private http: HttpClient) {}

  getData() {
    return this.http.get('<URL>');
  }
}
```

Trong ví dụ trên, chúng ta đã tạo một service có tên là DataService và inject HttpClient vào service. Sau đó, chúng ta đã sử dụng phương thức get() của HttpClient để call API và lấy dữ liệu.

Đây chỉ là các bước cơ bản để tạo một dự án Angular và call API đến back end. Bạn có thể tìm hiểu thêm chi tiết về từng bước trong các hướng dẫn trực tuyến hoặc tài liệu chính thức của Angular.

3. Creating new DB schema for EazyBank and Updating Backend project based on the latest DB schema



#### 4. CORS Error

Lỗi CORS trong frontend xảy ra khi trình duyệt chặn các yêu cầu từ ứng dụng frontend tới một nguồn tài nguyên (resource) trên một miền (domain) khác. Đây là một biện pháp bảo mật của trình duyệt để ngăn chặn các yêu cầu xâm nhập từ các nguồn không đáng tin cậy.

Một số thông báo lỗi CORS phổ biến mà bạn có thể gặp trong frontend bao gồm:

1. "Access to XMLHttpRequest at 'URL' from origin 'Origin' has been blocked by CORS policy": Đây là thông báo lỗi cho biết yêu cầu đã bị chặn bởi chính sách CORS. Trình duyệt không cho phép gửi yêu cầu từ nguồn (origin) hiện tại tới nguồn yêu cầu (URL).
2. "No 'Access-Control-Allow-Origin' header is present on the requested resource": Thông báo này chỉ ra rằng server không trả về header "Access-Control-Allow-Origin" trong phản hồi. Trình duyệt yêu cầu server cung cấp header này để xác định xem yêu cầu từ nguồn nào được chấp nhận.
3. "Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource": Thông báo này xuất hiện khi trình duyệt thực hiện preflight request, một yêu cầu kiểm tra trước khi gửi yêu cầu thực sự. Server phải trả về header "Access-Control-Allow-Origin" để cho phép yêu cầu tiếp theo.
4. "Credentials mode 'include' is not allowed in CORS preflight channel": Thông báo này xảy ra khi bạn gửi yêu cầu CORS với thông tin định danh (credentials), nhưng server không được cấu hình cho phép chia sẻ thông tin định danh trong kênh preflight.

Để khắc phục lỗi CORS trong frontend, bạn có thể thực hiện các biện pháp sau:

- ⇒ Cấu hình server: Đảm bảo server được cấu hình đúng chính sách CORS và trả về các header CORS phù hợp như "Access-Control-Allow-Origin" và "Access-Control-Allow-Headers" để cho phép các yêu cầu từ các miền khác.

## 5. Giới thiệu về CORS

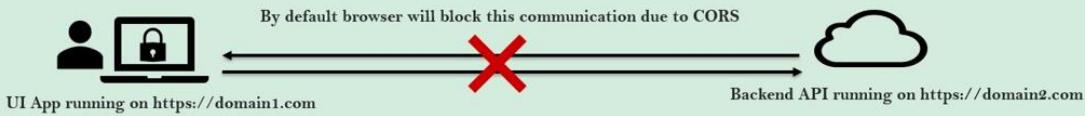
### CROSS-ORIGIN RESOURCE SHARING (CORS)

CORS is a protocol that enables scripts running on a browser client to interact with resources from a different origin. For example, if a UI app wishes to make an API call running on a different domain, it would be blocked from doing so by default due to CORS. It is a specification from W3C implemented by most browsers.

So CORS is not a security issue/attack but the default protection provided by browsers to stop sharing the data/communication between different origins.

"other origins" means the URL being accessed differs from the location that the JavaScript is running from, by having:

- a different scheme (HTTP or HTTPS)
- a different domain
- a different port



Hình thức đầy đủ của CORS là chia sẻ tài nguyên gốc chéo (cross origin resource sharing), có nghĩa là hai **Origin - nguồn gốc** khác nhau. Họ đang cố gắng chia sẻ tài nguyên của họ. Vì vậy, ở đây có hai origin khác nhau giống như client application và back-end application. Tại sao chúng được gọi là origin khác nhau. Vì vậy, trước hết, origin là gì? Origin là một URL, là sự kết hợp của giao thức HTTP URL và tên miền hoặc tên máy chủ lưu trữ và số cổng nơi ứng dụng web được triển khai là gì. Sự kết hợp của ba tham số này như giao thức HTTP, tên miền và cổng sẽ kết hợp với nhau, chúng tôi gọi nó là **Origin**.

Giả sử nếu hai ứng dụng khác nhau đang cố gắng giao tiếp với nhau thì chắc chắn **Origin** của chúng sẽ khác nhau vì số cổng sẽ khác hoặc tên máy chủ sẽ khác hoặc giao thức HTTP có thể khác hoặc tất cả chúng có thể khác nhau vì chúng là những ứng dụng rất khác nhau được triển khai trên các máy chủ khác nhau. Chúng tôi không thể mong đợi số cổng, tên máy chủ lưu trữ và giao thức HTTP giống nhau. Vì vậy, những loại kịch bản này sẽ dẫn đến **cross origin - nguồn gốc chéo**, nghĩa là hai **origin** khác nhau. Họ đang cố gắng giao tiếp với nhau.

Bất cứ khi nào loại giao tiếp này diễn ra bên trong các trình duyệt hiện đại như Chrome, Firefox và Safari, giao tiếp mặc định sẽ bị trình duyệt dừng lại. Lý do là tính năng bảo mật được cung cấp bởi quy trình hiện đại. Trong thế giới web, bạn biết đấy có rất nhiều tin tức và có rất nhiều lỗ hổng bảo mật đã xảy ra hàng ngày. Vì vậy, để tránh hầu hết các lỗ hổng bảo mật đó, quy trình hiện đại mà họ sẽ làm là nếu hai **origin** khác nhau đang cố liên lạc với nhau mà không có bất kỳ cấu hình thích hợp nào thì họ sẽ dừng liên lạc với lỗ hổng ta đã thấy trong bài giảng trước. Bạn có thể xem xét một tình huống trong đó tôi có một máy chủ web sản xuất đang chạy hợp lệ và một số tin tức đang cố liên lạc với máy chủ của tôi từ máy chủ của anh ta hoặc từ ứng dụng web của anh ta. Và nếu bạn không có CORS, thì nó sẽ tạo ra nhiều vấn đề bảo mật và tôi cần xử lý việc này theo cách thủ công bên trong ứng dụng web của mình. Để tránh khỏi tất cả các lỗ hổng bảo mật và tin tức hoặc web, tất cả các trình duyệt hiện đại,

chúng đã bật các CORS này theo mặc định. CORS không phải là một luồng bảo mật. Nó là một lớp bảo vệ khỏi các lỗ hổng bảo mật mà chúng tôi có bên trong web. Nếu ai đó nói CORS là tấn công bảo mật, vui lòng ngăn họ lại và giải thích cho họ. Nó không phải là một cuộc tấn công bảo mật, nó là một **class-layer** bảo vệ khỏi các cuộc tấn công bảo mật. Giống như cách chúng tôi có ứng dụng Angular đang cố gắng giao tiếp với back-end application rất giống nhau và triển khai ứng dụng web của họ, ứng dụng của chúng tôi ở các tên máy chủ khác nhau, số cổng và giao thức HTTP, nhưng họ vẫn có thể có lý do chính đáng hợp lệ để giao tiếp với nhau. Vì vậy, trong những tình huống như vậy, chúng tôi sẽ nói với trình duyệt như thế nào, vui lòng cho phép giao tiếp giữa hai này.

## 6. Possible options to fix the CORs issue

Hiện tại, giao tiếp giữa ứng dụng Angular của chúng tôi và ứng dụng Springboard backend đã bị chặn bởi sự cố CORS này, đây là một biện pháp bảo mật được kích hoạt bên trong tất cả các trình duyệt hiện đại. Vậy tại sao giao tiếp bắt đầu giữa hai ứng dụng khác nhau của tôi? Bởi vì mặc dù tên miền và giao thức HTTP của chúng giống nhau, nhưng các cổng lại khác nhau. Ứng dụng Angular được triển khai ở cổng 4200, trong khi ứng dụng web Springboard của tôi được triển khai ở cổng 8080. Vì vậy, vì chúng có nguồn gốc khác nhau, CORS của tôi bên trong trình duyệt đang ngừng liên lạc giữa chúng. Nếu bạn đang tìm kiếm các tùy chọn khả thi, có hai cách để chúng tôi có thể xử lý vấn đề này với sự trợ giúp của Spring Security framework.

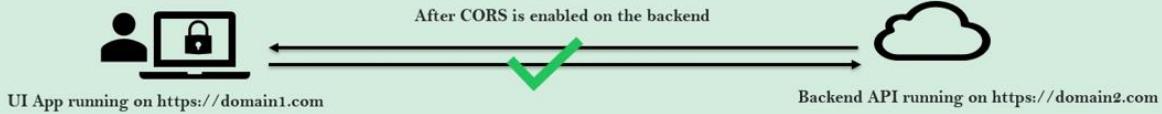
### SOLUTION TO HANDLE CORS

If we have a valid scenario, where a Web APP UI deployed on a server is trying to communicate with a REST service deployed on another server, then these kind of communications we can allow with the help of `@CrossOrigin` annotation. `@CrossOrigin` allows clients from any domain to consume the API.

`@CrossOrigin` annotation can be mentioned on top of a class or method like mentioned below,

```
@CrossOrigin(origins = "http://localhost:4200") // Will allow on specified domain
```

```
@CrossOrigin(origins = "*") // Will allow any domain
```



Tùy chọn đầu tiên mà chúng tôi có nằm ở đầu lớp trình điều khiển nơi bạn đã chú thích bằng bộ điều khiển REST hoặc nơi bạn đã xác định các API REST của mình, bạn có thể sử dụng annotation `@CrossOrigin`. Và bất cứ khi nào bạn đang sử dụng annotation `@CrossOrigin`, bạn có thể cho biết bạn muốn chấp nhận giao tiếp từ Origin nào. Vì vậy, bạn, với tư cách là một backend application, bạn phải định cấu hình Origin khác mà bạn muốn chấp nhận giao tiếp. Vì vậy, ở đây, nếu tôi vừa đề cập đến Origin bằng `http://localhost:4200`, thì tôi sẵn sàng chấp nhận giao tiếp chỉ từ UI application, được triển

khai tại máy chủ lưu trữ cục bộ 4200. Hoặc bạn cũng có thể định cấu hình Origin với giá trị dấu hoa thị, do đó, điều này cho trình duyệt biết rằng backend application của tôi có thể giao tiếp với bất kỳ loại miền nào bên trong internet. Vì vậy, với các cấu hình đó, nếu bạn có thể thấy, giao tiếp CrossOrigins của tôi sẽ được trình duyệt của tôi cho phép.

Ở đây, bạn có thể có một câu hỏi như làm thế nào trình duyệt của tôi hiểu được các cấu hình mà tôi đang thực hiện trên backend application. Thông thường các trình duyệt, họ sẽ đưa ra yêu cầu trước chuyến bay. Vậy ý nghĩa của yêu cầu trước chuyến bay là gì? Giả sử nếu bạn muốn gọi API thông báo từ ứng dụng Angular sang backend application. Vì vậy, trong trường hợp này, trình duyệt của tôi biết rõ đây là giao tiếp CrossOrigin và bất cứ khi nào giao tiếp CrossOrigin này đang cố diễn ra, trước tiên, trình duyệt của tôi sẽ không gửi yêu cầu API thông báo. Thay vào đó, nó sẽ đưa ra một yêu cầu trước chuyến bay tới back-end server nói rằng, "CrossOrigin đang cố liên lạc với bạn. Bạn có ổn với điều đó không?" Nếu backend application nói rằng, "Được rồi, đây là nguồn gốc hợp lệ và tôi tin rằng nguồn gốc đó là ứng dụng, vui lòng gửi yêu cầu thực tế cho tôi," thì trình duyệt sẽ gửi yêu cầu API thông báo thực tế đến back-end server.



## 7. Fixing CORs issue using Spring Security

Triển khai giải pháp liên quan đến CORS bên trong ứng dụng web Spring Boot với sự trợ giúp của Spring Security framework. Giống như chúng ta đã thảo luận trước đây, chúng ta có thể xác định cấu hình CORS ở cấp bộ controller hoặc trên global-toàn cầu bên trong ứng dụng web của bạn. Vì giải pháp global là lựa chọn tốt nhất.. Ở đây, bên trong lớp ProjectSecurityConfig, bên trong phương thức mà chúng ta đang cố gắng tạo một bean của SecurityFilterChain, ngay sau biến http này, chúng ta có thể gọi một phương thức mới gọi là cors. Khi chúng ta gọi phương thức cors này, chúng ta cần gọi một phương thức khác, đó là configureSource. Đối với phương thức configureSource này, chúng ta cần truyền một đối tượng của CorsConfigurationSource.

### SOLUTION TO HANDLE CORS

Instead of mentioning `@CrossOrigin` annotation on all the controllers inside our web app, we can define CORS related configurations globally using Spring Security like shown below,

```
@Bean
SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {
    http.cors().configurationSource(new CorsConfigurationSource() {
        @Override
        public CorsConfiguration getCorsConfiguration(HttpServletRequest request) {
            CorsConfiguration config = new CorsConfiguration();
            config.setAllowedOrigins(Collections.singletonList("http://localhost:4200"));
            config.setAllowedMethods(Collections.singletonList("*"));
            config.setAllowCredentials(true);
            config.setAllowedHeaders(Collections.singletonList("*"));
            config.setMaxAge(3600L);
            return config;
        }
    }).and().authorizeHttpRequests()
        .requestMatchers("/myAccount", "/myBalance", "/myLoans", "/myCards", "/user").authenticated()
        .requestMatchers("/notices", "/contact", "/register").permitAll()
        .and().formLogin()
        .and().httpBasic();
    return http.build();
}
```

Vì CorsConfigurationSource là một interface nên tôi đã viết một lớp bên trong ẩn anonymous. Bạn có thể thấy tất cả đây là một lớp bên trong anonymous trong đó bên trong lớp bên trong anonymous này, tôi đang cố gắng tạo một đối tượng của interface này bằng cách ghi đè phương thức trừu tượng có trong interface này và tên phương thức này là `getCorsConfiguration`. Bên trong phương thức này, như bạn có thể thấy, đầu tiên, chúng ta cần tạo một đối tượng CorsConfiguration. **Cấu hình rất quan trọng ở đây là `setAllowedOrigins`**. Chỉ sử dụng phương pháp này, chúng tôi đang cho biết tất cả các `origin` hoặc miền hoặc máy chủ được phép giao tiếp với back-end server này là gì.

Sau khi thực hiện những thay đổi này, chúng tôi gặp lỗi biên dịch vì CORS và CSRF này là hai cấu hình khác nhau. Khi quá trình xây dựng hoàn tất, chúng tôi có thể khởi động lại ứng dụng web của mình để đảm bảo rằng các cấu hình này đang phản ánh chính xác. Tương tự như vậy, trước tiên, hãy để tôi dừng phiên bản đang chạy hiện có. Và bây giờ tôi sẽ mở EasyBankBackendApplication. Và ở đây tôi sẽ khởi động ứng dụng ở chế độ gỡ lỗi. Thao tác này sẽ khởi động ứng dụng của tôi tại cổng 8080. Giờ đây,

Ứng dụng backend Spring Boot của tôi đã có thể giao tiếp với Angular UI application mà chúng tôi đã xây dựng

Dưới đây là một số cấu hình quan trọng trong CorsConfiguration:

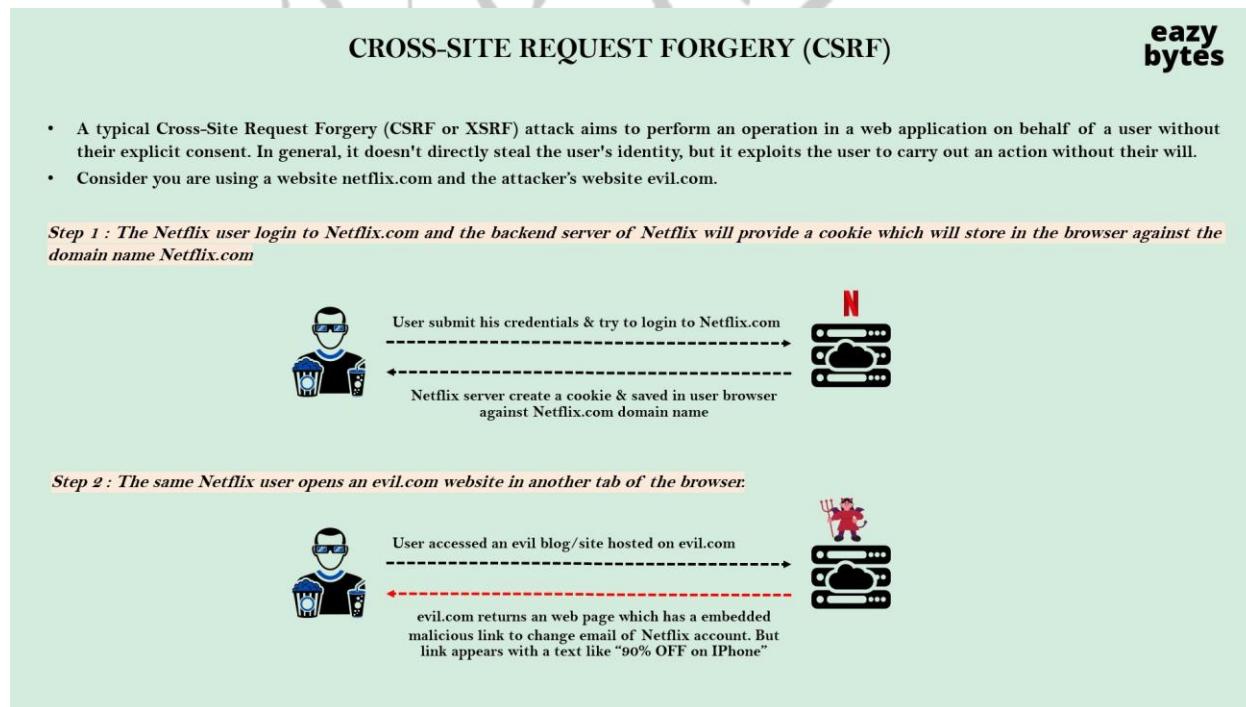
1. allowedOrigins: Xác định các nguồn gốc (origin) được phép truy cập vào tài nguyên từ máy chủ. Đây có thể là một danh sách các URL hoặc "\*", cho phép truy cập từ tất cả các nguồn gốc.
2. allowedMethods: Xác định các phương thức HTTP được phép trong yêu cầu CORS. Ví dụ: GET, POST, PUT, DELETE, v.v.
3. allowedHeaders: Xác định các header HTTP được phép trong yêu cầu CORS. Ví dụ: "Content-Type", "Authorization", v.v.
4. exposedHeaders: Xác định các header HTTP mà máy chủ cho phép truy cập từ phía client. Header này sẽ được gửi đến trình duyệt như một phần của phản hồi.
5. allowCredentials: Xác định liệu máy chủ cho phép gửi các thông tin xác thực (credentials) như cookies, phần tử localStorage, v.v. trong yêu cầu CORS. Giá trị mặc định là false.
6. maxAge: Xác định thời gian tối đa (tính bằng giây) mà client có thể lưu trữ phiên bản tạm thời của cấu hình CORS. Thời gian này chỉ định cấu hình trên trình duyệt của client.



## 8. Introduction to CSRF attack

CSRF là một cuộc tấn công bảo mật hoặc lỗ hổng bảo mật được sử dụng bởi tin tặc. Vì vậy, đây là một kỹ thuật. Bất cứ khi nào tin tặc thực hiện cuộc tấn công này, anh ta sẽ không đánh cắp thông tin đăng nhập, session ID cookie của bạn. Thay vào đó, anh ta sẽ lừa bạn thực hiện một số hành động mà bạn không muốn, điều đó có nghĩa là bạn không bao giờ biết rằng tin tặc đó đang lợi dụng bạn bằng cuộc tấn công CSRF và anh ta đang cố lấy một số thông tin backend application, từ tài khoản tương ứng của bạn. Bạn sẽ không bao giờ biết những chi tiết như vậy bất cứ khi nào tin tặc thực hiện hành vi giả mạo yêu cầu trên nhiều trang web này

Bạn với tư cách là user của Netflix. Vào tối thứ Sáu, bạn đã quyết định xem một số bộ phim Netflix hoặc sê-ri web Netflix. Vì vậy, bạn đã mang theo bóng ngô, Coke, mọi thứ bạn thiết lập và bạn đăng nhập vào tài khoản Netflix bên trong máy tính xách tay hoặc bên trong hệ thống của mình. Và sau khi bạn nhập thông tin đăng nhập của mình, máy chủ Netflix sẽ tạo một cookie và lưu cookie đó bên trong trình duyệt của bạn, đối với tên miền [netflix.com](http://netflix.com). Bạn với tư cách là người dùng hợp pháp, bạn đã đăng nhập vào ứng dụng web của Netflix và ở hậu trường, Netflix sẽ tạo một số cookie nhất định để không phải hỏi lại thông tin đăng nhập của bạn. Và những cookie này, nó sẽ lưu trữ bên trong trình duyệt của bạn dựa trên tên miền [netflix.com](http://netflix.com). Vì vậy, cookie là một khái niệm bên trong trình duyệt. Những cookie này có ưu điểm và nhược điểm riêng. Ưu điểm mà tôi muốn nhấn mạnh ở đây là, nếu một số tên miền như [netflix.com](http://netflix.com) tạo cookie và lưu trữ bên trong trình duyệt của bạn, không có tên miền nào khác như [facebook.com](http://facebook.com) hoặc [amazon.com](http://amazon.com), thì họ không thể đánh cắp cookie này từ trình duyệt. Trình duyệt đủ thông minh để chỉ chia sẻ cookie này với miền ban đầu đã tạo cookie này ban đầu. Vì vậy, trong trường hợp này, tên miền ban đầu là [netflix.com](http://netflix.com).



Bây giờ, trong bước hai, điều xảy ra là sau khi xem phim, có thể sau một giờ, user của tôi đã truy cập một trang web khác có tên [evil.com](http://evil.com). Bên trong [evil.com](http://evil.com) này, mà user của tôi đã mở trong cùng một trình duyệt, nhưng trong một tab khác. Vì vậy, bất cứ khi nào người dùng của tôi truy cập [evil.com](http://evil.com), đây

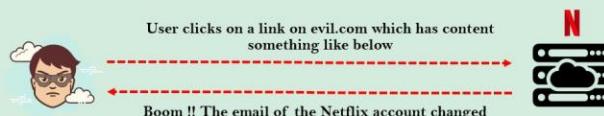
là trang web do hacker của tôi lưu trữ, bạn có thể thấy trên back-end server của evil.com, một hacker đang đứng ở đó để thực hiện cuộc tấn công CSRF. Vì vậy, ngay khi bạn cố gắng mở evil.com, anh ấy sẽ gửi một trang web tuyệt đẹp với tất cả thông tin mà bạn cần. Và cùng với thông tin mà bạn cần, anh ấy cũng sẽ có một số chi tiết về ưu đãi hấp dẫn, chẳng hạn như giảm giá 90% cho iPhone, nghĩa là anh ấy sẽ giảm giá 90% cho iPhone và anh ấy sẽ dụ bạn nhấp vào liên kết đó bởi vì nó trông rất hấp dẫn đối với bất cứ ai. User của tôi đã bị cám dỗ và anh ấy nhấp vào liên kết đó.

Trong bước ba, người dùng nhấp vào liên kết độc hại có trong evil.com, ngay sau liên kết đó, hacker của tôi sẽ có một mã nhúng. Nếu bạn thấy đây là một trong những mã nhúng mẫu mà bạn có thể hình dung. Có một biểu mẫu đang cố gửi yêu cầu đăng lên back-end server. Và nếu bạn thấy, hành động anh ấy đang thực hiện là truy cập netflix.com và /changeEmail. Vì vậy, với mã này, anh ta đang cố thay đổi email Netflix của bạn để anh ta có thể đăng nhập bằng email của chính mình và user sẽ không bao giờ biết vì mã này được nhúng sau liên kết đó. Ngay khi tôi nhấp vào liên kết đó, trình duyệt của tôi sẽ gửi yêu cầu đăng ký đường dẫn changeEmail của netflix.com. Và bất cứ khi nào nó gửi yêu cầu này tới phần cuối của netflix.com, trình duyệt của tôi cũng sẽ gửi chi tiết cookie cho miền netflix.com, vì yêu cầu nó đang gửi tới netflix.com. Và vì yêu cầu sẽ chuyển đến netflix.com, cùng với đó, nó sẽ tự động gửi tất cả các cookie liên quan đến netflix.com bên trong yêu cầu. Và netflix.com của tôi, nếu họ không xử lý cuộc tấn công CSRF đúng cách, thì họ thực sự không thể phân biệt liệu yêu cầu này đến từ trang web phù hợp của họ hay liệu nó đến từ trang web evil.com. Vì vậy, vì họ không thể phân biệt giữa một trang web xấu và trang web của chính họ, nên họ sẽ xử lý yêu cầu này và họ sẽ thay đổi email của user, và bùm, email của tài khoản Netflix sẽ bị thay đổi, và tin tức sẽ chiếm đoạt tài khoản Netflix của user của tôi. Ở đây, user của tôi không bao giờ biết rằng điều này sẽ xảy ra. Tin tức của tôi đã lừa user của tôi.

### CROSS-SITE REQUEST FORGERY (CSRF)

eazy  
bytes

*Step 3 : User tempted and clicked on the malicious link which makes a request to Netflix.com. And since the login cookie already present in the same browser and the request to change email is being made to the same domain Netflix.com, the backend server of Netflix.com can't differentiate from where the request came. So here the evil.com forged the request as if it is coming from a Netflix.com UI page.*



```
<form action="https://netflix.com/changeEmail"
      method="POST" id="form">
      <input type="hidden" name="email" value="user@evil.com">
<form>

<script>
  document.getElementById('form').submit()
</script>
```

## 9. Solution to handle CSRF attacks

Để đánh bại một cuộc tấn công CSRF, ứng dụng backend của tôi, họ cần một cách để xác định xem yêu cầu HTTP đến từ người dùng hợp pháp thông qua UI application ứng dụng của riêng họ hay nó đến từ trang web của tin tức? Vì vậy, ngay bây giờ, ứng dụng web backend của tôi không thể phân biệt được vì cả hai yêu cầu đều có cookie xác thực hợp lệ.

Để phân biệt giữa người dùng hợp pháp và tin tức, chúng tôi sẽ mang đến một token khác có tên CSRF. Vì vậy, CSRF token là token ngẫu nhiên an toàn được nhiều ứng dụng web trong ngành sử dụng để ngăn chặn cuộc tấn công CSRF bên trong ứng dụng web của họ. Token này phải là duy nhất cho mỗi session của user và nó phải là một giá trị ngẫu nhiên rất lớn để rất khó đoán. Vì vậy, bây giờ, chúng tôi có một tùy chọn như sử dụng CSRF token. Backend Server của tôi có thể phân biệt giữa người dùng hợp pháp của tôi và tin tức nhưng chúng tôi cần hiểu CSRF token sẽ giúp tránh cuộc tấn công CSRF như thế nào. Tương tự như vậy, lần này hãy diễn lại kịch bản tương tự, với ngoại lệ là backend được triển khai một giải pháp với sự trợ giúp của CSRF token.

### SOLUTION TO CSRF ATTACK

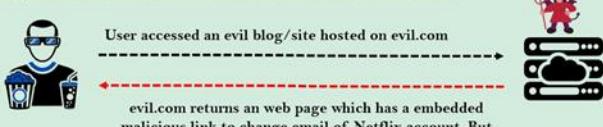
**eazy  
bytes**

- To defeat a CSRF attack, applications need a way to determine if the HTTP request is legitimately generated via the application's user interface. The best way to achieve this is through a CSRF token. A CSRF token is a secure random token that is used to prevent CSRF attacks. The token needs to be unique per user session and should be of large random value to make it difficult to guess.
- Let's see how this solve CSRF attack by taking the previous Netflix example again,

*Step 1 : The Netflix user login to Netflix.com and the backend server of Netflix will provide a cookie which will store in the browser against the domain name Netflix.com along with a randomly generated unique CSRF token for this particular user session. CSRF token is inserted within hidden parameters of HTML forms to avoid exposure to session cookies.*



*Step 2 : The same Netflix user opens an evil.com website in another tab of the browser.*



Câu chuyện tương tự, vào tối thứ Sáu, bạn ngồi trước netflix.com, mở nó và nhập thông tin đăng nhập của mình. Nhưng lần này, netflix.com, cùng với các cookie liên quan đến xác thực, nó cũng tạo ngẫu nhiên CSRF token trên backend server và token tương tự được gửi đến UI application và CSRF token cũng sẽ được gửi dưới dạng cookie. Nhưng có nhiều cách về cách chúng tôi muốn giao tiếp CSRF token đó từ phần backend đến UI application, cách tiếp cận được sử dụng phổ biến nhất là gửi CSRF token bên trong chính cookie. Vì vậy, bây giờ, trình duyệt của tôi nhận được hai cookie từ chương trình backend Netflix của tôi. Một là cookie liên quan đến xác thực và một là cookie liên quan đến CSRF.

Bên trong bước hai, user của tôi đã mở evil.com bên trong một tab khác của cùng một trình duyệt. Và lần này, evil.com của tôi cũng trả về cùng một trang web có liên kết độc hại được nhúng để thay đổi email của tài khoản Netflix. Nhưng liên kết nhúng này, nó trông rất hấp dẫn với dòng chữ nói rằng giảm giá 90% hoặc giảm giá 90% cho iPhone.

Bây giờ trong bước ba, như bạn có thể mong đợi, user của tôi sẽ nhấp vào liên kết độc hại đó.

**SOLUTION TO CSRF ATTACK**

**eazy bytes**

*Step 3 : User tempted and clicked on the malicious link which makes a request to Netflix.com. And since the login cookie already present in the same browser and the request to change email is being made to the same domain Netflix.com. This time the Netflix.com backend server expects CSRF token along with the cookie. The CSRF token must be same as initial value generated during login operation*

User clicks on a link on evil.com which has content something like below

Boom !! The Netflix threw an error 403

The CSRF token will be used by the application server to verify the legitimacy of the end-user request if it is coming from the same App UI or not. The application server rejects the request if the CSRF token fails to match the test.

Và dăng sau hậu trường, bất kỳ mã HTML và mã JavaScript nào mà tin tức của tôi gửi lại bên trong evil.com của hắn, nó sẽ yêu cầu netflix.com thay đổi email của user của tôi. Vì yêu cầu được gửi đến netflix.com, nên trình duyệt của tôi sẽ gửi hai cookie mà nó có, một cookie xác thực là cookie và cookie còn lại là cookie liên quan đến backend application CSRF. Nhưng lần này backend server netflix.com của tôi sẽ từ chối yêu cầu với lỗi 403. Vì vậy, ở đây, bạn có thể có một câu hỏi như, trình duyệt của tôi đang gửi cả hai cookie nhưng tại sao backend server netflix.com của tôi lại nhận ra liệu nó đến từ người dùng hợp pháp hợp lệ hay từ tin tức.

Để đạt được điều đó, có một mẹo chúng ta cần làm theo bên trong các ứng dụng web và UI application của mình. Bất cứ khi nào bạn nhận được backend application CSRF ban đầu từ thao tác đăng nhập, bạn sẽ nhận được backend application đó bên trong cookie. Nhưng thay vì tận dụng kiểu mặc định để gửi cookie đó đến backend với sự trợ giúp của trình duyệt, bạn cần viết một đoạn mã nhỏ bên trong UI application dùng để đọc cookie đó theo cách thủ công và gửi cookie đó vào trong header hoặc bên trong ứng dụng của bạn, ở bất cứ đâu bạn muốn hoặc bất cứ nơi nào bạn có thỏa thuận với backend. Vì vậy bạn đang đọc cookie này từ cùng một miền, đó là netflix.com, nên bạn sẽ có thể đọc cookie từ mã JavaScript và từ mã HTML của mình, nơi mà tin tức không thể làm điều tương tự từ evil.com của hắn. Bởi vì nếu bạn cố gắng loại trừ một số mã JavaScript bên trong evil.com, thì anh ta đang chạy mã JavaScript đó từ một trang web có tên evil.com và miền đó sẽ không có quyền truy cập vào cookie của backend application CSRF. Với hạn chế đó, tin tức của tôi không thể gửi backend application CSRF đó bên trong header hoặc bên trong payload. Anh ta sẽ luôn dựa vào kiểu tự động gửi cookie của trình duyệt đến backend server. Nhưng lần này backend server rất thông minh. Nó sẽ không xử lý yêu cầu miễn là CSRF token được đưa vào bên trong và có payload khác dựa trên thỏa thuận mà nó có với UI application.

Đây là một trong những phương pháp được sử dụng phổ biến nhất để xử lý các cuộc tấn công CSRF với sự trợ giúp của CSRF token. Và tất nhiên, có thể có nhiều cách khác nhưng khuyến nghị của tôi là, ít nhất hãy cố gắng hiểu kịch bản này, cách xử lý CSRF token và chúng tôi sẽ cố gắng triển khai giải pháp này bên trong ứng dụng web của mình với sự trợ giúp của Spring Security. Nhưng như tôi đã nói, chúng ta cũng có tùy chọn tắt hoàn toàn CSRF chỉ bằng cách gọi `csrf().disable()`. bên trong cấu hình Spring Security.



Nhưng chúng ta không bao giờ nên khám phá tùy chọn đó. Có thể điều đó tốt cho product application thấp hơn hoặc một số ứng dụng không nghiêm trọng. Nhưng nếu bạn đang xây dựng một số product application, đừng bao giờ làm điều này. Nó sẽ thu hút rất nhiều cuộc tấn công CSRF bên trong ứng dụng web của bạn và user của bạn sẽ phải chịu đựng rất nhiều do cuộc tấn công CSRF.

## 10. Ignoring CSRF protection for public APIs

Bỏ qua bảo vệ CSRF (Cross-Site Request Forgery) cho các API công khai trong Java Spring không được khuyến nghị, vì việc này có thể làm gia tăng nguy cơ bảo mật của ứng dụng. CSRF là một cuộc tấn công mà kẻ tấn công có thể lừa người dùng thực hiện các hành động không mong muốn trên ứng dụng của họ bằng cách sử dụng quyền truy cập đã được xác thực.

Java Spring cung cấp bảo vệ CSRF mặc định thông qua sử dụng token CSRF, nơi một token được tạo và gắn vào mọi yêu cầu có phương thức thay đổi trạng thái (như POST, PUT, DELETE). Khi yêu cầu gửi đi, token này phải được gửi kèm theo và được so khớp với token đã được tạo. Điều này đảm bảo rằng yêu cầu chỉ được thực hiện từ nguồn tin cậy.

Tuy nhiên, nếu bạn cho rằng các API công khai không chứa những yêu cầu có tác động thay đổi trạng thái, bạn có thể xem xét việc vô hiệu hóa bảo vệ CSRF cho các API đó. Để làm điều này, bạn có thể sử dụng cấu hình Spring Security để loại bỏ CSRF cho các API cụ thể.

Dưới đây là một ví dụ về cách vô hiệu hóa CSRF cho các API được đánh dấu là công khai trong Java Spring:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .csrf()
            .ignoringAntMatchers("/api/public/**"); // Thay đổi "/api/public/**" thành đường dẫn của API
    }
}
```

Trong ví dụ trên, chúng ta sử dụng phương thức `ignoringAntMatchers()` để chỉ định các đường dẫn mà CSRF sẽ không được áp dụng, và `/api/public/**` là đường dẫn của API công khai. Bạn có thể thay đổi đường dẫn này để phù hợp với cấu trúc đường dẫn của ứng dụng của bạn.

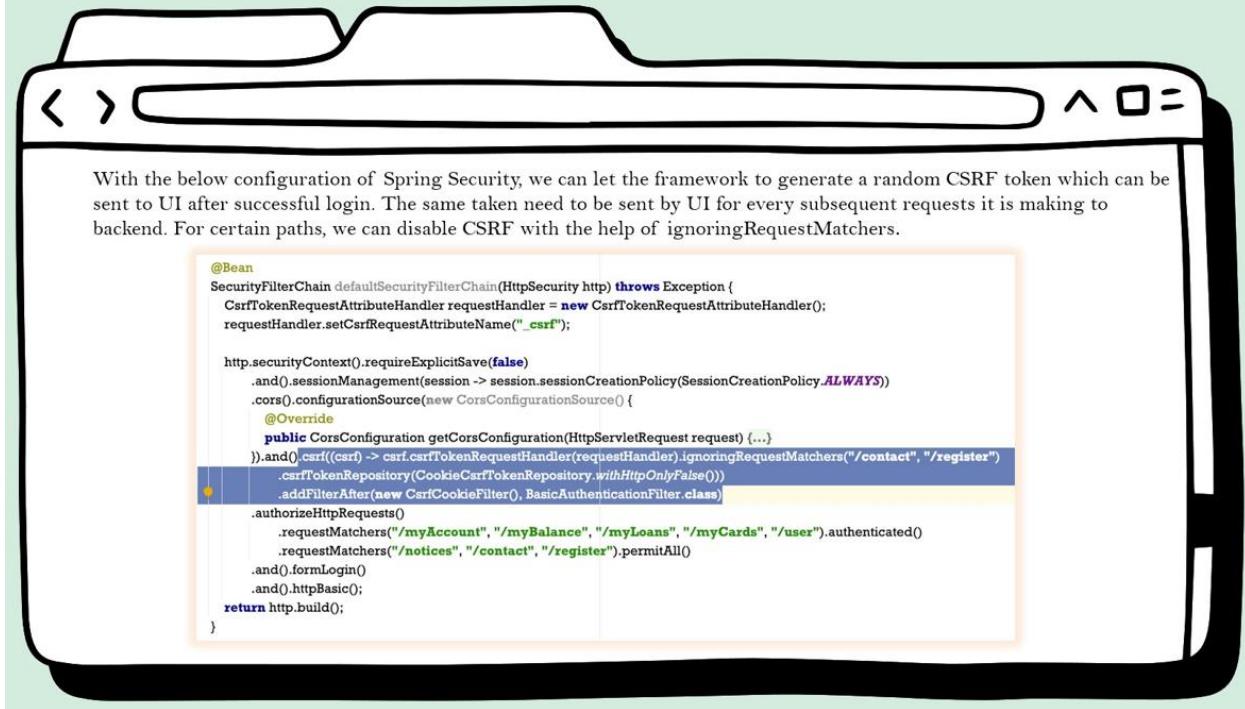
Tuy nhiên, hãy cẩn thận khi vô hiệu hóa CSRF cho các API công khai, hãy đảm bảo rằng không có yêu cầu nào có tác động thay đổi trạng thái được thực hiện thông qua các API này.

## 11. Implementing CSRF token solution inside our web application

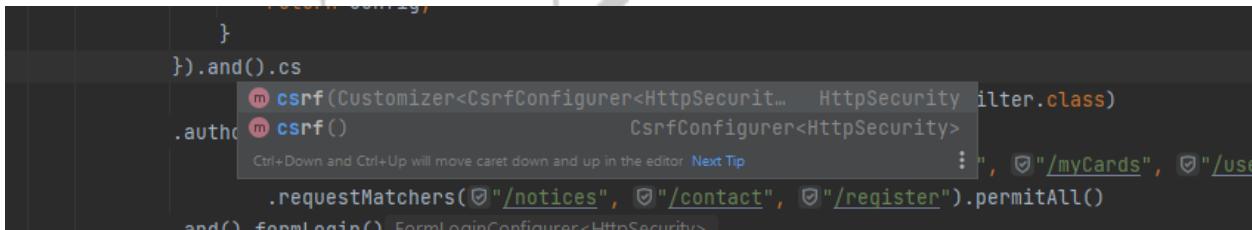
Trước tiên, sẽ tạo một đối tượng `CsrfTokenRequestAttributeHandler`. Nếu bạn có thể vào bên trong lớp này, thì mục đích của lớp này là, nó là một triển khai của `CsrfTokenRequestHandler`. Lớp này có khả năng cung cấp `CsrfToken` dưới dạng thuộc tính yêu cầu và giải quyết `value token`, dưới dạng `parameter value`. Ý tôi muốn nói với tuyên bố này là, Spring Security của bạn sẽ tạo giá trị `CsrfToken`, nhưng để xử lý giá trị đó và hiển thị giá trị đó dưới dạng `header` và `cookie` cho `UI application`, chúng tôi cần nhận trợ giúp từ thuộc tính này lớp xử lý. Vì vậy, đó là lý do tại sao chúng ta cần tạo một đối tượng của trình xử lý thuộc tính này bằng cách sử dụng cùng một tên đối tượng mà tôi đang cố gọi phương thức này: `setCsrfRequestAttributeName`. Và với phương thức này, tôi chuyển tên thuộc tính là `_csrf`.

## CSRF ATTACK SOLUTION INSIDE SPRING SECURITY

by

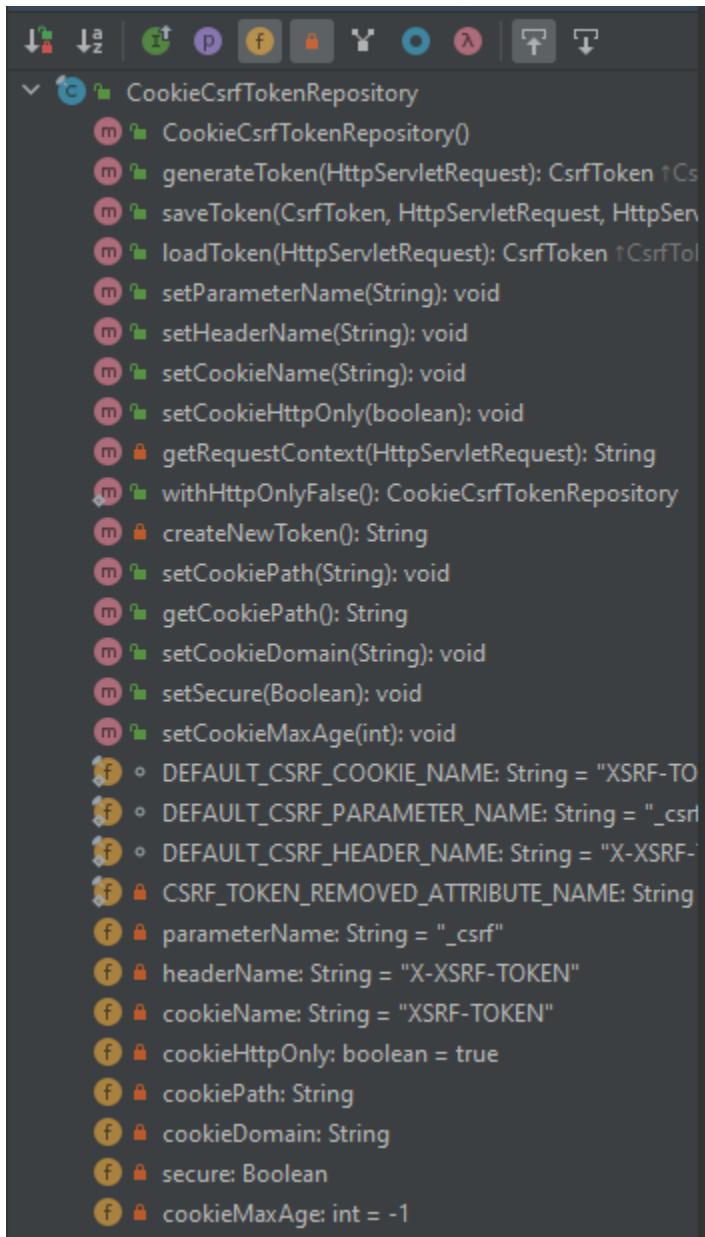


Sau khi xác định đối tượng này, tiếp theo chúng ta cần đi đến vị trí cấu hình CSRF. Vì vậy, như bây giờ, bạn có thể thấy, tôi đang gọi một phương thức gọi là csrf không chấp nhận bất kỳ đối số phương thức nào. Nhưng bây giờ tôi cần gọi thêm một phương thức CSRF có sẵn bên trong Spring Security cho cùng một phương thức. Đầu tiên, hãy để tôi xóa tất cả mã trước đó. Như bạn có thể thấy ở đây, tôi có hai loại phương thức csrf. Phương thức thứ hai sẽ chấp nhận CsrfConfigurer làm tham số đầu vào của phương thức.



Vì vậy, hãy để tôi chọn phương pháp thứ hai này. Đối với phương pháp này, chúng ta cần chuyển tất cả các cấu hình CSRF. Vì vậy, ở đây tôi sẽ dán hai dòng mã lambda. Bằng cách sử dụng đối tượng CsrfConfigurer này, trước tiên, tôi cần gọi CsrfTokenRequestHandler là gì. Chúng tôi đã tạo đối tượng requestHandler này ở trên cùng, cùng một đối tượng mà tôi đang cố chuyển. Khi chúng ta đề cập đến CsrfTokenRequestHandler, thì tôi sẽ gọi IgnoringRequestMatchers này, như chúng ta đã thảo luận trong các bài giảng trước. Tôi cần gọi phương thức này và tôi cần đề cập đến các URL API công khai để đối với các tập hợp URL này, tính năng bảo vệ CSRF sẽ không được áp dụng. Sau những chi tiết này, tôi cần gọi phương thức có tên là csrfTokenRepository. Đối với phương thức này, tôi cần chuyển mã này, đó là CookieCsrfTokenRepository.withHttpOnlyFalse.

Hãy thử xem có gì bên trong CookieCsrfTokenRepository này.



Giống như bạn có thể thấy ở đây, lớp này chịu trách nhiệm duy trì CSRF token trong cookie có tên này và đọc từ header có tên này. Vì vậy, nó tuân theo các quy ước tương tự của AngularJS. Không chỉ AngularJS. Hầu hết các UI application framework, cũng như ReactJS, chúng tuân theo cùng tên cookie và header name. Vì vậy, bạn có thể thấy, đây là tên mà nó sẽ xem xét cho cookie của bạn và đây là tên mà nó sẽ xem xét cho header mà nó sẽ gửi đến UI application. Chỉ sử dụng lớp này, Spring Security framework sẽ thực hiện tất cả những điều kỳ diệu của giải pháp CsrfToken.

Bây giờ, ở đây bạn có thể có một câu hỏi như, tại sao chúng tôi gọi điều này với iHttpOnlyFalse. Bất cứ khi nào cookie được tạo, thường là các backend, chúng có tùy chọn cho phép UI application dùng

đọc giá trị cookie hoặc chúng cũng có thể ngăn UI application hoặc mã JavaScript được triển khai bên trong UI application đọc giá trị cookie. Nhưng với sự trợ giúp của điều này với `HttpOnlyFalse`, chúng tôi đang nói với Spring Security, "Vui lòng tạo cookie CSRF với cấu hình là `HttpOnlyFalse` để mã JavaScript của tôi được triển khai bên trong angular application có thể đọc giá trị cookie đó. Bạn cũng có thể đi và kiểm tra bên trong lớp này. Bạn có thể thấy, khi sử dụng các UI application, như AngularJS, để giao tiếp với backend server, chúng ta cần đảm bảo rằng chúng ta đang sử dụng những ứng dụng này với `HttpOnlyFalse`.

Sau khi thực hiện những thay đổi này, bây giờ framework có khả năng tạo ra CSRF token có khả năng lưu trữ nó bên trong bộ nhớ của ứng dụng và nó cũng có khả năng chấp nhận các giá trị tên header và tên cookie từ UI application và nó sẽ xác thực giống nhau. Nhưng bước duy nhất còn lại ở đây là trước tiên, backend, nó phải gửi cookie và value header đến UI application sau lần đăng nhập ban đầu. Nếu không, UI application của tôi sẽ biết giá trị CSRF token là gì? Đó là lý do tại sao Spring Security của bạn cũng nên gửi token được tạo như một phần của phản hồi cho UI application.

Vì vậy, để gửi CSRF token cho mỗi và mọi phản hồi mà chúng tôi sẽ gửi tới UI application, chúng tôi cần tạo một lớp bộ lọc. Tương tự, hãy để tôi tạo một package tại đây. Package là `com.eazybytes.filter` và, tôi sẽ tạo một Filter mới và tên bộ lọc sẽ là `CsrfCookieFilter`. Vì vậy, bên trong lớp này, trước tiên, để biến lớp này thành bộ lọc, tôi cần extend một trong các filter do Spring Security cung cấp và tên bộ lọc là `OncePerRequestFilter`. Ngay sau khi chúng tôi mở rộng bộ lọc này, như bạn có thể thấy, chúng tôi nhận được thông báo lỗi rằng chúng tôi cần ghi đè một phương thức bên trong `CsrfCookieFilter` của mình. Vì vậy, hãy để tôi nhấp vào phương thức và chúng ta cần ghi đè phương thức này, đó là `doFilterInternal`. Đây là phương thức trống đã được tạo và bên trong phương thức này, chúng ta cần viết logic của riêng mình.

```
public class CsrfCookieFilter extends OncePerRequestFilter {  
  
    @Override  
    protected void doFilterInternal(HttpServletRequest request,  
HttpServletResponse response, FilterChain filterChain)  
        throws ServletException, IOException {  
        CsrfToken csrfToken = (CsrfToken)  
request.getAttribute(CsrfToken.class.getName());  
        if(null != csrfToken.getHeaderName()) {  
            response.setHeader(csrfToken.getHeaderName(),  
        csrfToken.getToken());  
        }  
        filterChain.doFilter(request, response);  
    }  
}
```

Đầu tiên, tôi đang cố đọc `CsrfToken` có sẵn bên trong `HttpServletRequest`. Vì vậy, bất cứ khi nào giá trị token được tạo ban đầu bởi backend, nó sẽ có sẵn dưới dạng thuộc tính yêu cầu và chúng tôi cũng đang cố gắng đọc nó và typecast nó cho đối tượng của `CsrfToken`. Vì vậy, bằng cách sử dụng đối tượng `csrfToken` này, trước tiên, chúng tôi sẽ kiểm tra xem có giá trị tên header bên trong đối tượng này không. Nếu nó không phải là `null`, điều đó có nghĩa là framework có thể đã tạo `csrfToken`. Bên trong khối `if` này, chúng tôi đang điền cùng một `HeaderName` và đó là một giá trị token bên trong `response` header và cùng một phản hồi sẽ được chuyển đến filter tiếp theo bên trong **Filter Chain**. Bằng cách này, cuối cùng, bất cứ khi nào chúng tôi gửi phản hồi UI application, giá trị `csrfToken` của tôi sẽ xuất hiện bên

trong header. Chúng tôi chỉ gửi header chứ không gửi cookie. Với tư cách là nhà phát triển, khi bạn điền giá trị CsrfToken như một phần của response header, Spring Security framework của tôi sẽ đảm nhiệm việc tạo cookie CSRF và gửi cookie tương tự tới trình duyệt hoặc ứng dụng UI application như một phần của phản hồi. Vì vậy, bây giờ chúng tôi đã tạo một bộ lọc nhưng chúng tôi cần giao tiếp về bộ lọc này với Spring Security framework.

Tương tự, chúng ta hãy chuyển đến lớp ProjectSecurityConfig này. Ở đây, ngay sau các cấu hình CSRF của chúng ta, chúng ta cần xác định một dòng mã ở đây, như bạn có thể thấy ở đây, tôi chỉ đang gọi một phương thức addFilter và với phương thức addFilter này, tôi đang chuyển đổi tương của bộ lọc tùy chỉnh của riêng chúng ta mà chúng tôi đã tạo và argument thứ hai tôi đang chuyển, BasicAuthenticationFilter. Vì vậy, BasicAuthenticationFilter là bộ lọc của Spring Security sẽ xuất hiện bất cứ khi nào chúng tôi đang sử dụng HTTP Basic Authentication.

```
.and() .csrf((csrf) ->
    csrf.csrfTokenRequestHandler(requestHandler).ignoringRequestMatchers("/contact", "/register")
        .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse()))
        .addFilterAfter(new CsrfCookieFilter(),
    BasicAuthenticationFilter.class)
```

Nhưng với dòng này, những gì chúng tôi đang nói với Spring Security là "Vui lòng thực thi CsrfCookieFilter này sau BasicAuthenticationFilter." Vì chỉ sau BasicAuthenticationFilter này, thao tác đăng nhập của tôi sẽ hoàn tất và ngay sau khi thao tác đăng nhập hoàn tất, CsrfToken của tôi sẽ được tạo. Cùng một giá trị CsrfToken, tôi muốn duy trì bên trong response với sự trợ giúp của CsrfCookieFilter này.

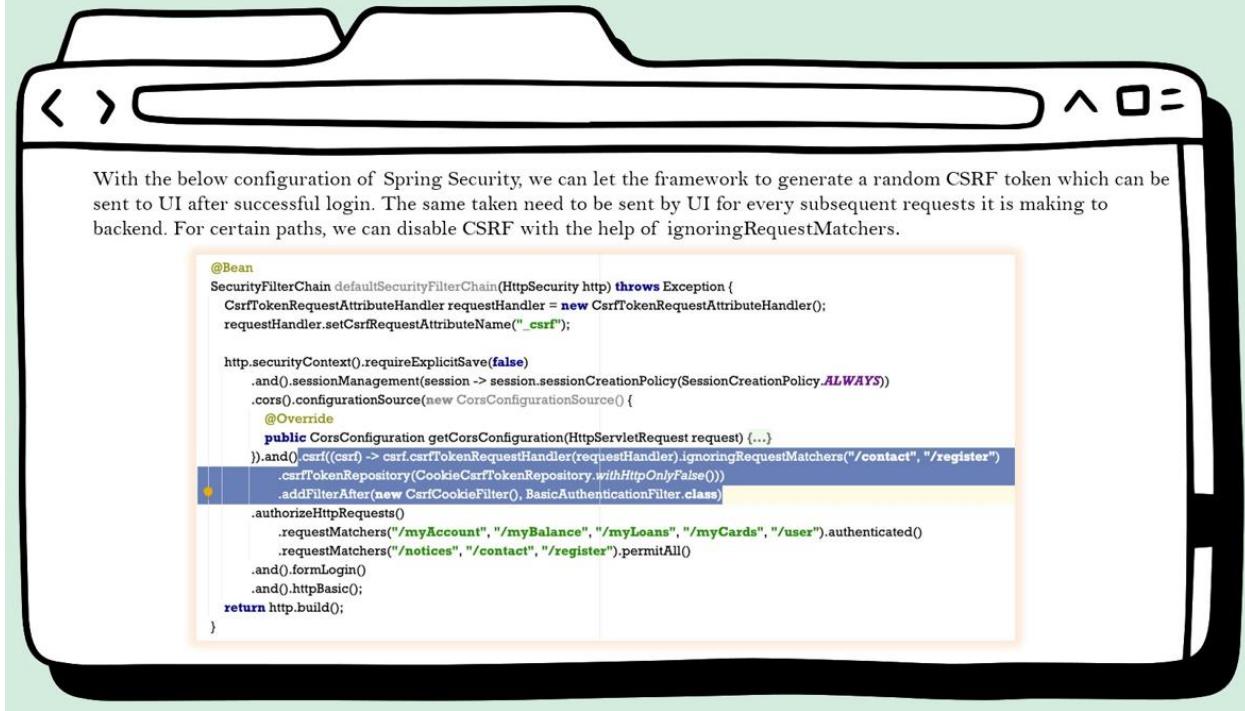
Bây giờ là bước cuối cùng, chúng ta cần tạo thêm một cấu hình ở đây, nhưng điều này không liên quan gì đến giải pháp CSRF.

```
http.securityContext().requireExplicitSave(false)
    .and().sessionManagement(session ->
    session.sessionCreationPolicy(SessionCreationPolicy.ALWAYS))
```

Giống như bạn có thể thấy ở đây, đây là cấu hình mới mà tôi đã dán ở đây. Mục đích của cấu hình này là: trước đây, bất cứ khi nào chúng tôi cố gắng truy cập các API thông qua trình duyệt, chúng tôi thường truy cập trực tiếp các backend API REST này thông qua trình duyệt và đã từng có một trang đăng nhập được xây dựng bởi Spring Security framework và chúng tôi nhập thông tin đăng nhập của mình và nó sẽ tạo một JSESSIONID và sử dụng cùng một JSESSIONID, chúng tôi cố gắng truy cập tất cả các yêu cầu tiếp theo mà không cần nhập thông tin đăng nhập. Nhưng bây giờ trở đi, vì chúng ta sẽ sử dụng một ứng dụng UI application riêng biệt để đăng nhập và truy cập tất cả các API REST, nên chúng ta cần cho phép Spring Security framework, "Vui lòng tạo JSESSIONID bằng cách theo dõi sessionManagement mà tôi đã tạo ở đây." Vì vậy, với cấu hình này, chúng tôi đang nói với Spring Security, "Vui lòng luôn tạo JSESSIONID sau khi hoàn tất đăng nhập ban đầu." Và cùng một JSESSIONID mà nó sẽ gửi đến UI application và UI application có thể tận dụng điều tương tự cho tất cả các yêu cầu tiếp theo mà nó sẽ thực hiện sau lần đăng nhập ban đầu.

## CSRF ATTACK SOLUTION INSIDE SPRING SECURITY

by



Nếu không có hai dòng này, bạn phải đăng nhập mỗi khi cố gắng truy cập API được bảo mật từ ứng dụng Angular của mình. Vì vậy, đó là lý do tại sao, vui lòng đảm bảo rằng bạn đang đề cập đến hai dòng này và sử dụng requireExplicitSave này, chúng tôi đang nói với Spring Security, "Tôi sẽ không chịu trách nhiệm lưu các chi tiết xác thực bên trong SecurityContextHolder". Theo mặc định, giá trị này là true. Bằng cách ghi đè các giá trị mặc định này, chúng tôi giao trách nhiệm tạo JSESSIONID và lưu trữ các chi tiết xác thực vào SecurityContextHolder cho framework. Tất cả điều này chúng tôi phải làm vì chúng tôi đang tận dụng một UI application riêng biệt để giao tiếp với backend của chúng tôi. Bởi vì với những thay đổi này, chỉ backend mới có khả năng gửi các giá trị CsrfToken đến UI application và UI application cũng phải chấp nhận giá trị CsrfToken và nó phải gửi cùng một giá trị CsrfToken cho mọi yêu cầu tiếp theo mà nó sẽ thực hiện.

Hãy thử thực hiện những thay đổi đó bên trong ứng dụng UI application. Tương tự, hãy để tôi chuyển đến Angular UI application. Vì vậy, lớp ở đây mà tôi muốn mở là login.component.ts. Bởi vì đây là nơi mà thao tác đăng nhập sẽ được gọi. Sau khi thao tác đăng nhập được gọi và hoàn thành, chúng tôi sẽ nhận được tất cả phản hồi, chẳng hạn như, bao gồm cookie và nội dung bên trong dữ liệu phản hồi này. Vì vậy, bên trong lớp này, tôi cần đọc cookie đó và lưu trữ nó bên trong sessionStorage hoặc dưới bất kỳ hình thức nào khác để tôi có thể gửi đi gửi lại cùng một backend application cho tất cả các yêu cầu trong tương lai user của tôi sẽ thực hiện cho backend. Vì vậy, để đọc cookie, chúng ta cần nhập một hàm từ bản thảo. Bạn có thể thấy, bên trong hai dòng này, tôi đang cố gắng đọc cookie có tên XSRF-TOKEN. Bởi vì Spring Security của tôi sẽ gửi cookie có tên XSRF-TOKEN. Đồng thời, nó sẽ mong đợi cookie này từ ứng dụng UI application bên trong header có tên X-XSRF-TOKEN. Vì vậy, trước tiên chúng ta cần đọc cookie có cùng tên. Giống như, đây là tên cookie. Vì vậy, đó là lý do tại sao tôi đã chuyển tên

cookie này cho hàm getCookie này. Và một khi chúng ta có nó bên trong một biến, tôi sẽ lưu trữ nó bên trong sessionStorage, với tên thuộc tính là XSRF-TOKEN.

```
validateUser(loginForm: NgForm) {
  this.loginService.validateLoginDetails(this.model).subscribe(
    responseData => {
      this.model = <any> responseData.body;

      this.model.authStatus = 'AUTH';
      window.sessionStorage.setItem("userdetails", JSON.stringify(this.model));
      let xsrf = getCookie('XSRF-TOKEN')!;
      window.sessionStorage.setItem("XSRF-TOKEN", xsrf);
      this.router.navigate(['dashboard']);
    });
}
```

Phần tiếp theo mà tôi cần xử lý là, tôi cần đảm bảo rằng tôi luôn gửi CsrfToken này tới backend của mình bất cứ khi nào tôi thực hiện một yêu cầu tiếp theo bên trong ứng dụng web của mình. Vì vậy, nơi thích hợp để thực hiện những thay đổi này là interceptor class. Vì vậy, bên trong interceptor class của chúng tôi, chúng tôi sẽ interceptor từng yêu cầu đến từ ứng dụng UI application của tôi. Bên trong interceptor class này, chúng ta có thể thêm một vài dòng mã.

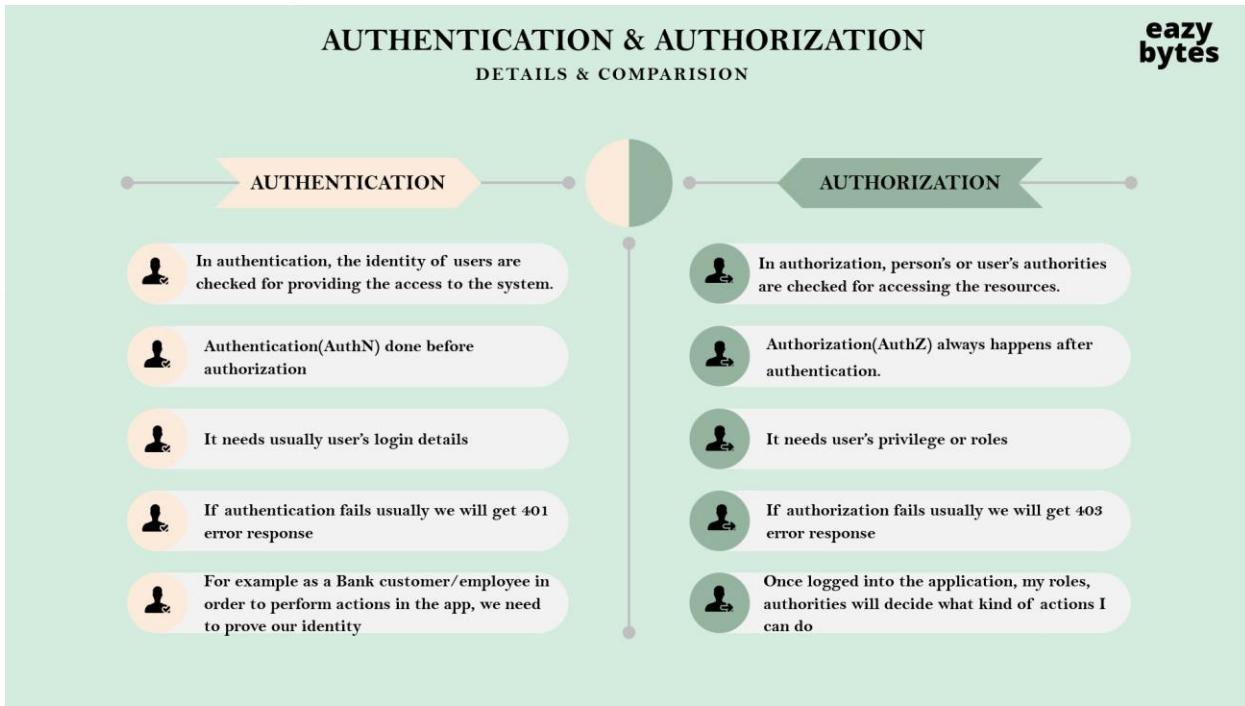
Trước tiên tôi đang cố đọc chi tiết biến XSRF-TOKEN từ sessionStorage. Và nếu xsrf này không phải là null, thì điều tôi đang làm là, tôi đang cố gắng gửi một httpHeader bằng cách tạo một header mới và nối thêm vào các httpHeader này có sẵn bên trong mã Angular. Tôi cần đảm bảo rằng tôi đang sử dụng cùng một tên header mà chúng ta đã thấy bên trong backend, chẳng hạn như X-XSRF-TOKEN. Và với header này, chúng ta cần chuyển giá trị CsrfToken. Cùng với đó, ứng dụng Angular của tôi sẽ gửi backend application XSRF, bất cứ khi nào nó nhìn thấy bên trong phiên của tôi. Mỗi public pages dành cho các tình huống mà bạn chưa đăng nhập. Sau đó, rõ ràng XSRF này sẽ không có giá trị và header này sẽ không chuyển đến backend và điều đó hoàn toàn ổn trong tình huống của chúng tôi vì các API public không yêu cầu bất kỳ CsrfToken nào trong tình huống của chúng tôi.

```
let xsrf = sessionStorage.getItem('XSRF-TOKEN');
if(xsrf){
  httpHeaders = httpHeaders.append('X-XSRF-TOKEN', xsrf);
}
```

## Section 7: Understanding & Implementing Authorization

### 1. Authentication Vs Authorization

Sự khác biệt giữa xác thực và authorization là gì? Tôi thấy nhiều lần bên trong các dự án, nhà phát triển và người thử nghiệm, sử dụng **Authentication Vs Authorization** này mà không biết sự khác biệt giữa chúng. Chúng ta tập trung vào việc hiểu sự khác biệt giữa **Authentication Vs Authorization** là gì.



Đến với **sự khác biệt đầu tiên**, Nếu bạn hỏi tôi định nghĩa authentication là gì, như một phần của authentication, chúng tôi sẽ cố gắng xác định ai là người dùng đang cố truy cập ứng dụng web của chúng tôi. Nếu không có authentication, chúng tôi không thể yêu cầu bất kỳ người nào hoặc bất kỳ user nào truy cập các API bảo mật của chúng tôi.

Trong khi đó, **Authorization** sẽ có hiệu lực sau khi hoàn tất authentication thành công. Là một phần của Authorization, chúng tôi sẽ kiểm tra các quyền hoặc đặc quyền hoặc vai trò (authorities or privileges or roles) mà một user cụ thể có. Theo đó, chúng tôi sẽ xây dựng business logic bên trong ứng dụng web của mình để anh ấy có thể truy cập các tình huống khác nhau hoặc các chức năng khác nhau bên trong ứng dụng web của chúng tôi dựa trên các đặc quyền(privileges) của anh ấy.

Nếu bạn nhận bất kỳ ứng dụng ngân hàng nào, sẽ có nhiều vai trò liên quan như thư ký, thủ quỹ, quản lý, giám sát viên, vì vậy có nhiều vai trò liên quan và mỗi vai trò sẽ có quyền hạn hoặc đặc quyền riêng. Vì vậy, kiểu bắt buộc user truy cập vào bên trong hệ thống của chúng tôi dựa trên các đặc quyền của họ, chúng tôi gọi đó là **Authorization**. Đó là định nghĩa đầu tiên và sự khác biệt cơ bản giữa authentication và Authorization.

Đến với **sự khác biệt thứ hai**, authentication, mà ở dạng ngắn gọn mà chúng tôi gọi là AuthN, sẽ luôn xuất hiện trước authorization. Trong khi đó, authorization, mà chúng ta có thể gọi ngắn gọn là

AuthZ, luôn xảy ra sau khi authentication . Sẽ không bao giờ có trường hợp chúng tôi thực hiện authorization mà không cần authentication . Trước tiên, chúng tôi luôn yêu cầu user đăng nhập vào hệ thống của chúng tôi. Chỉ sau điều đó, chúng tôi sẽ lo lắng về các đặc quyền, cấp độ truy cập và vai trò của anh ấy. Theo đó, chúng tôi có thể thực thi authorization.

Đến với **sự khác biệt thứ ba**, để thực hiện authentication , chúng tôi sẽ chỉ hỏi chi tiết đăng nhập của người dùng. Email của bạn là gì hoặc số điện thoại di động của bạn là gì và thông tin đăng nhập hoặc OTP của bạn là gì? Vì vậy, tất cả những chi tiết này chúng tôi thu thập từ user để đạt được authentication thành công user vào trang web của tôi. Trong khi bên trong authorization, chúng tôi sẽ không bao giờ lo lắng về thông tin đăng nhập của người dùng. Chúng tôi chỉ lo lắng về **privileges, access levels, and roles**. Chỉ dựa trên các **authorities, roles and privileges**, chúng tôi quyết định cấp độ truy cập của anh ấy trong ứng dụng web của chúng tôi.

Chuyển **sự khác biệt thứ tư**. Bất cứ khi nào authentication không thành công, chúng tôi sẽ nhận được mã lỗi 401, có nghĩa là authentication không thành công. Trong khi đó, trong trường hợp authorization, chúng tôi nhận được mã lỗi 403. Vì vậy, bất cứ khi nào chúng tôi nhận được 403 nghĩa là authorization thành công, nhưng người dùng này không được phép truy cập chức năng này. Đó là lý do tại sao chúng tôi nhận được 403, đây là lỗi bị cấm bất cứ khi nào authorization không thành công.

Và cuối cùng, nếu bạn hỏi tôi các ví dụ thời gian thực để nói về authentication và authorization, như tôi đã nói, bạn luôn có thể đăng ký ngân hàng. Vì vậy, bên trong một ứng dụng ngân hàng, chúng ta có thể có nhiều vai trò khác nhau như khách hàng, quản trị viên, thủ quỹ, quản lý ngân hàng, giám sát viên. Vì vậy, đây là tất cả các vai trò liên quan đến tất cả những người dùng này, trước tiên, trong ứng dụng ngân hàng của chúng tôi, chúng tôi sẽ luôn yêu cầu thông tin đăng nhập của họ để chứng minh danh tính của họ. Vì vậy, **đó là một phần của authentication** bên trong bất kỳ ứng dụng ngân hàng nào. Sau khi user được authentication và đăng nhập vào ứng dụng của tôi, thì dựa trên vai trò, đặc quyền và quyền hạn của anh ta, chức năng mà anh ta có thể truy cập là các trang mà anh ta có thể thấy trên ứng dụng web sẽ khác giữa người dùng này với người dùng khác. **Đó là authorization**.

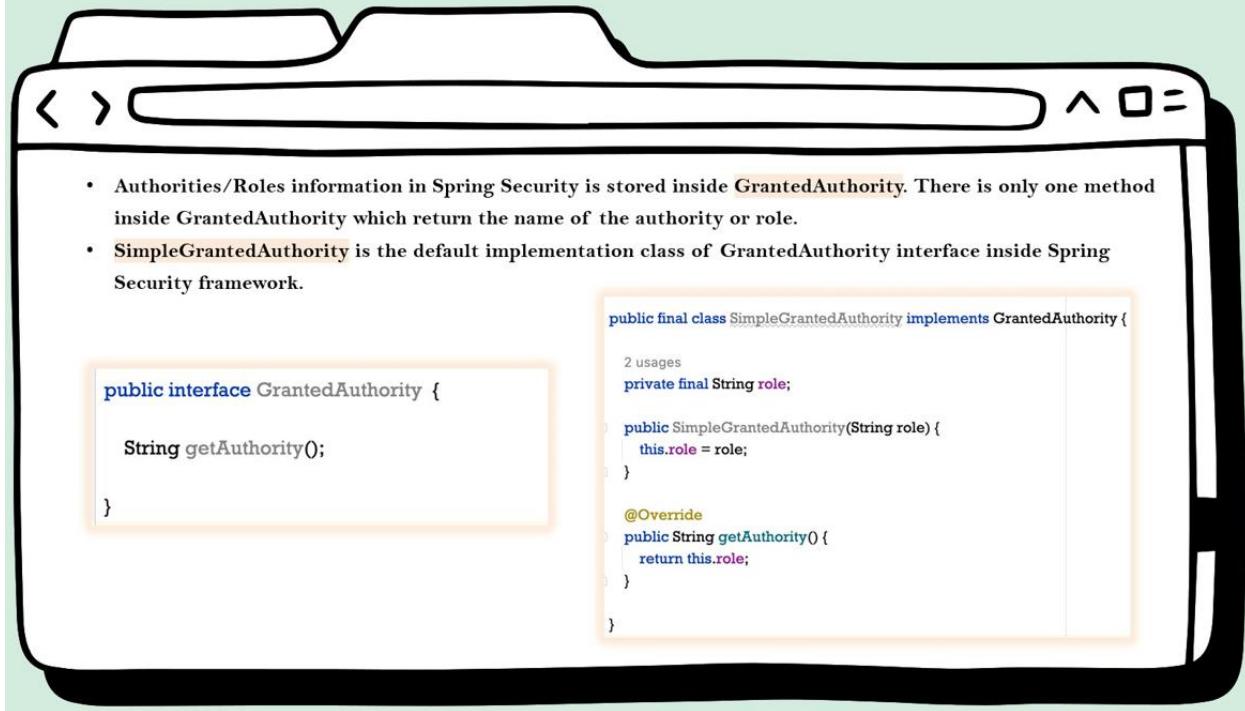
## 2. How Authorities stored inside Spring Security

Bởi vì để thực thi authorization bên trong trang web của tôi, trước tiên, tôi cần hiểu cách chúng tôi lưu trữ các quyền này hoặc các chi tiết đặc quyền bên trong Spring Security. Trong Spring Security, chúng tôi gọi các quyền hoặc đặc quyền này bằng hai thuật ngữ. Thuật ngữ đầu tiên là authorities và thuật ngữ thứ hai là roles. Có sự khác biệt giữa authorities và roles và chúng ta nên biết khi nào nên sử dụng authorities và khi nào nên sử dụng và bên trong Spring Security. Nhưng cách chúng được lưu trữ bên trong Spring Security là tương tự đối với cả hai authorities và roles này.

Trước tiên, chúng ta có thể bắt đầu thảo luận với authorities, sau đó khi chúng ta chuyển sang roles, tôi sẽ cho bạn biết sự khác biệt giữa authorities và roles cũng như sự khác biệt giữa chúng và khi nào nên sử dụng authorities hoặc roles. Các authorities và roles này được lưu trữ ở đâu trong Spring Security? Có một interface được gọi là called granted authority. Bên trong interface này, chúng ta có một phương thức getter và kiểu viết của phương thức getter này là chuỗi như bạn đã biết, Spring Security sẽ không bao giờ rời bỏ chúng ta chỉ bằng cách xác định các interface. Nó luôn cung cấp một số implementation mẫu của các interface này để chúng tôi có thể tận dụng chúng trong quá trình phát triển ứng dụng web của mình. Phải nói rằng, ngay cả đối với interface có thẩm quyền được cấp, cũng có một lớp được implement bên trong Spring Security và tên lớp này là Simple Granted Authority.

## HOW AUTHORITIES STORED ? INSIDE SPRING SECURITY

by



Vì vậy, bên trong lớp này, bạn có thể thấy có một constructor. Bất kỳ ai muốn gán authority or role cho một user cụ thể, họ cần gọi hàm tạo này của simple granted authority với tên vai trò ở định dạng chuỗi và cuối cùng tên đó sẽ được lưu trữ bên trong đối tượng của simple granted authority. Tại một số thời điểm, bất cứ khi nào spring framework của tôi muốn hiểu authority và role của user này là gì, thì nó sẽ chuyển sang phương thức `getAuthority` có sẵn bên trong Simple Granted Authority. Vì vậy, hãy thử nhanh chóng xác thực giao diện và lớp này bên trong Spring Security.

Giống như bạn có thể thấy ở đây, chúng ta có một interface **GrantedAuthority** bên trong Spring Security và bên trong đó chúng ta có một phương thức trừu tượng duy nhất có tên `getAuthority`.

```
package org.springframework.security.core;  
  
import java.io.Serializable;  
  
public interface GrantedAuthority extends Serializable {  
    String getAuthority();  
}
```

Đối với interface này, chúng tôi có một implementation với tên Simple Granted Authority, được cung cấp bởi chính Spring Security.

```

public final class SimpleGrantedAuthority implements GrantedAuthority {
    no usages
    private static final long serialVersionUID = 600L;
    no usages
    private final String role;

    1 usage
    public SimpleGrantedAuthority(String role) {
        Assert.hasText(role, message: "A granted authority textual representation is required");
        this.role = role;
    }

    public String getAuthority() { return this.role; }

    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        } else {
            return obj instanceof SimpleGrantedAuthority ? this.role.equals(((SimpleGrantedAuthority)obj).role) : false;
        }
    }

    public int hashCode() { return this.role.hashCode(); }

    public String toString() { return this.role; }
}

```

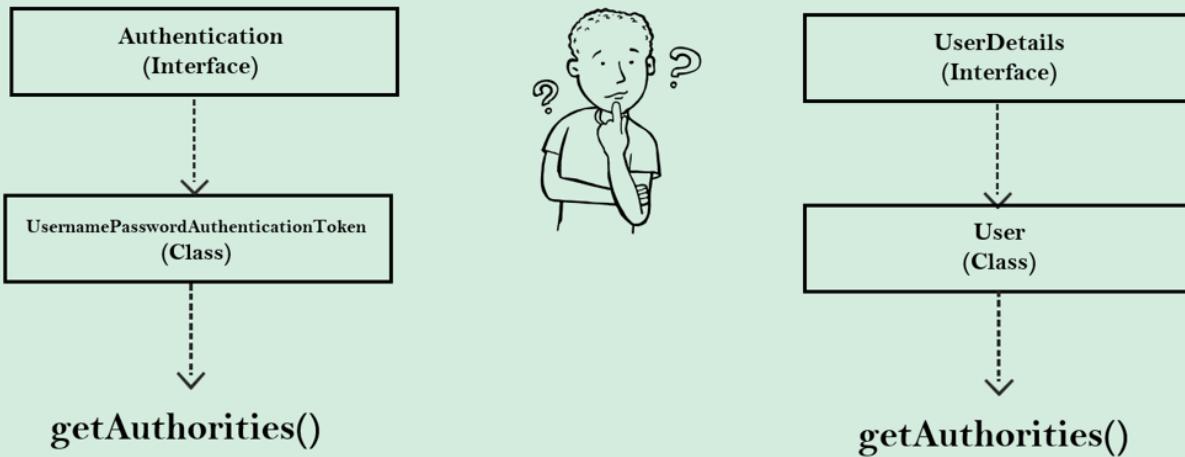
Nếu bạn có thể thấy ở đây, nó đang thực hiện quyền được cấp và bên trong đó, nó có một danh sách gọi biến thuộc loại final. Giống như bất cứ khi nào ai đó cố gắng tạo roles hoặc authorities, họ chỉ cần gọi constructor của lớp này. Điều đó có nghĩa là một roles và authorities được tạo và gán cho user, không ai có thể thay đổi các chi tiết này sau khi đối tượng được tạo. Đó là lý do tại sao chúng ta cũng không có bất kỳ phương thức setter nào bên trong lớp này, chúng ta chỉ có phương thức get. Sử dụng phương thức getAuthority, Spring Security, nó sẽ lấy thông tin chi tiết về roles hoặc authorities của user. Đây là cách Spring Security của tôi sẽ lưu trữ roles hoặc authorities của user.

Hiện tại chúng tôi đang thực hiện authentication bằng cách viết logic của riêng mình với sự trợ giúp của **user detail service and authentication provider**. Các authorities này được lưu trữ ở đâu bên trong các đối tượng của user details and authentication interfaces vì chúng tôi đang tận dụng chúng rất nhiều trong quá trình xác thực, giống như bạn có thể thấy bên dưới user, đó là một lớp triển khai **user detail**, có một phương thức gọi là **getAuthorities** sẽ được gọi bởi Spring Security của tôi để tìm nạp quyền của một người dùng cụ thể mà tôi đã tải từ cơ sở dữ liệu.

# HOW AUTHORITIES STORED ?

## INSIDE SPRING SECURITY

How does Authorities information stored inside the objects of UserDetails & Authentication interfaces which plays a vital role during authentication of the user ?



Nhưng khi bạn đến với kịch bản của **authentication provider**, nơi bạn sẽ tạo một đối tượng **Username Password Authentication Token**, ngay cả trong kịch bản này, framework của tôi sẽ gọi cùng một phương thức, lấy quyền có sẵn trong authentication interface. Vì vậy, đây là cách nó sẽ xử lý.

Chúng tôi cũng có thể xác thực điều tương tự trong các dự án của mình một lần.

```
-- EazyBankUsernamePwdAuthenticationProvider

@Override
public Authentication authenticate(Authentication authentication) throws AuthenticationException {
    String username = authentication.getName();
    String pwd = authentication.getCredentials().toString();
    List<Customer> customer = customerRepository.findByEmail(username);
    if (customer.size() > 0) {
        if (passwordEncoder.matches(pwd, customer.get(0).getPwd())) {
            List<GrantedAuthority> grantedAuthorities = new ArrayList<>();
            for (Authority authority : customer.get(0).getAuthorities()) {
                grantedAuthorities.add(new SimpleGrantedAuthority(authority.getName()));
            }
            return new UsernamePasswordAuthenticationToken(username, pwd, grantedAuthorities);
        } else {
            throw new BadCredentialsException("Invalid password!");
        }
    } else {
        throw new BadCredentialsException("No user registered with this details!");
    }
}
```

Trước tiên, nếu bạn có thể kiểm tra việc triển khai authentication provider như thế nào, nơi chúng tôi đã viết tất cả authentication của mình. Trong khi chúng tôi đang tạo đối tượng của authentication này dưới dạng **username password authentication token**, chúng tôi đang chuyển tên người dùng là gì, mật khẩu là gì và danh sách các authorities là gì. Vì vậy, trong khi chúng tôi đang cố gắng chuyển danh sách authorities, chúng tôi đang gửi một đối tượng **granted authority** dưới dạng lớp **Simple Granted Authority** bằng cách sử dụng mã này nơi chúng tôi đang cố gắng tạo một đối tượng có **Simple Granted Authority** và đối với hàm tạo này, chúng tôi là chuyển thông tin về role mà chúng tôi đã tìm nạp từ cơ sở dữ liệu ở định dạng chuỗi và cuối cùng bạn có thể thấy bên trong hàm tạo này, authorities sẽ được lưu trữ bằng cách thực thi dòng này. Vì vậy, ở đây chúng tôi đang lưu trữ tất cả các authorities dưới dạng một danh sách chưa sửa đổi, có nghĩa là không ai có thể sửa đổi danh sách này sau khi các quyền được gán cho người dùng và bất cứ khi nào framework của tôi cần các quyền này, nó chỉ cần gọi phương thức, đó là **getAuthorities** (extend từ các class cha).

```
public Collection<GrantedAuthority> getAuthorities() {  
    return this.authorities;  
}
```

Và bất cứ khi nào framework gọi phương thức **getAuthorities**, dòng mã này sẽ được thực thi và đơn giản là sẽ lấy danh sách các quyền mà chúng tôi đã tải từ hệ thống lưu trữ.

Ở đây, chúng tôi chỉ tải thông tin **user details** từ cơ sở dữ liệu và chúng tôi đang gửi một đối tượng chứa thông tin **user details** tới framework. Ngoài ra, ở đây, chúng tôi sẽ thực hiện theo chiến lược tương tự, bạn có thể thấy đối với **user constructor** này, chúng tôi đang chuyển tên người dùng, mật khẩu và danh sách các quyền mà chúng tôi đã tải từ cơ sở dữ liệu bằng cách tạo đối tượng **SimpleGrantedAuthority**. Ngay cả khi bạn đi và kiểm tra lớp user này, vẫn có một phương pháp gọi là **lấy quyền**. Sử dụng phương pháp này, framework của tôi sẽ hiểu thông tin về quyền hạn hoặc vai trò của user của tôi là gì. Vì vậy, bất kể bạn đang sử dụng quyền hạn hay vai trò, khuôn khổ của tôi sẽ luôn sử dụng cùng một cấu trúc. Mặc dù cái tên có nghĩa là có được chính quyền, nhưng nó cũng sẽ được sử dụng ngay cả cho các vai trò. Không có phương pháp nào khác. Một cái gì đó giống như nhận vai trò bên trong Spring Security. Với điều đó, tôi cho rằng bạn đã rất rõ ràng về nơi lưu trữ thông tin về quyền hạn hoặc vai trò trong Spring Security và bạn cũng hiểu rõ về cách các chi tiết quyền hạn này sẽ được tìm nạp từ các đối tượng chi tiết người dùng và interface xác thực bên trong framework bất cứ khi nào nó đang cố gắng thực hiện authorization. Bây giờ chúng tôi rất rõ ràng về các cơ quan chức năng, nơi chúng được cất giấu.

```
public final class SimpleGrantedAuthority implements GrantedAuthority {  
    private static final long serialVersionUID = 600L;  
    private final String role;  
  
    public String getAuthority() {  
        return this.role;  
    }  
}
```

### 3. Creating new table authorities to store multiple roles or authorities

```
CREATE TABLE `users` (
`id` INT NOT NULL AUTO_INCREMENT,
`username` VARCHAR(45) NOT NULL,
`password` VARCHAR(45) NOT NULL,
`enabled` INT NOT NULL,
PRIMARY KEY (`id`));

CREATE TABLE `authorities` (
`id` int NOT NULL AUTO_INCREMENT,
`username` varchar(45) NOT NULL,
`authority` varchar(45) NOT NULL,
PRIMARY KEY (`id`));

INSERT IGNORE INTO `users` VALUES (NULL, 'happy', '12345', '1');
INSERT IGNORE INTO `authorities` VALUES (NULL, 'happy', 'write');

CREATE TABLE `customer` (
`id` int NOT NULL AUTO_INCREMENT,
`email` varchar(45) NOT NULL,
`pwd` varchar(200) NOT NULL,
`role` varchar(45) NOT NULL,
PRIMARY KEY (`id`)

);

INSERT INTO `customer` (`email`, `pwd`, `role`)
VALUES ('johndoe@example.com', '54321', 'admin');
```

### 4. Making backend changes to load authorities from new DB table

Tạo Entity: Authority

```
@Entity
@Table(name = "authorities")
public class Authority {

    @Id
    @GeneratedValue(strategy= GenerationType.AUTO, generator="native")
    @GenericGenerator(name = "native", strategy = "native")
    private Long id;

    private String name;

    @ManyToOne
    @JoinColumn(name = "customer_id")
    private Customer customer;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
```

```

        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

}

```

### Thêm role và authority trong Customer

```

@Entity
public class Customer {

    @Id
    @GeneratedValue(strategy= GenerationType.AUTO, generator="native")
    @GenericGenerator(name = "native", strategy = "native")
    @Column(name = "customer_id")
    private int id;

    private String role;

    @JsonIgnore
    @OneToMany(mappedBy="customer", fetch=FetchType.EAGER)
    private Set<Authority> authorities;

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }

    public Set<Authority> getAuthorities() {
        return authorities;
    }

    public void setAuthorities(Set<Authority> authorities) {
        this.authorities = authorities;
    }

}

```

### Chỉnh sửa EazyBankUsernamePwdAuthenticationProvider

```
public class EazyBankUsernamePwdAuthenticationProvider implements AuthenticationProvider {

    @Autowired
    private CustomerRepository customerRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Override
    public Authentication authenticate(Authentication authentication) throws AuthenticationException {
        String username = authentication.getName();
        String pwd = authentication.getCredentials().toString();
        List<Customer> customer = customerRepository.findByEmail(username);
        if (customer.size() > 0) {
            if (passwordEncoder.matches(pwd, customer.get(0).getPwd())) {
                return new UsernamePasswordAuthenticationToken(username, pwd,
getGrantedAuthorities(customer.get(0).getAuthorities()));
            } else {
                throw new BadCredentialsException("Invalid password!");
            }
        } else {
            throw new BadCredentialsException("No user registered with this
details!");
        }
    }

    private List<GrantedAuthority> getGrantedAuthorities(Set<Authority>
authorities) {
        List<GrantedAuthority> grantedAuthorities = new ArrayList<>();
        for (Authority authority : authorities) {
            grantedAuthorities.add(new
SimpleGrantedAuthority(authority.getName())));
        }
        return grantedAuthorities;
    }
}
```

## 5. Configuring Authorities inside web application using Spring Security-Theory

Hiện tại bên trong ứng dụng web, chúng tôi đang lưu trữ quyền hạn - authorities của users. Sử dụng cùng một quyền hạn, chúng tôi cần thực thi authorization bên trong Spring boot web application.

Chúng tôi có ba phương pháp mà chúng tôi có thể sử dụng bên trong Spring Security.

# CONFIGURING AUTHORITIES

INSIDE SPRING SECURITY



In Spring Security the authorities requirements can be configured using the following ways,

**hasAuthority()** — Accepts a single authority for which the endpoint will be configured and user will be validated against the single authority mentioned. Only users having the same authority configured can invoke the endpoint.

**hasAnyAuthority()** — Accepts multiple authorities for which the endpoint will be configured and user will be validated against the authorities mentioned. Only users having any of the authority configured can invoke the endpoint.

**access()** — Using Spring Expression Language (SpEL) it provides you unlimited possibilities for configuring authorities which are not possible with the above methods. We can use operators like OR, AND inside access() method.

Phương thức đầu tiên là hasAuthority, chúng ta có thể định cấu hình một quyền duy nhất đối với dịch vụ mảng điểm cuối, để bất kỳ ai authority đó, họ chỉ có thể truy cập điểm cuối cụ thể đó.

[requestMatchers\("/myAccount"\).hasAuthority\("VIEWACCOUNT"\)](#)

Nhưng nếu bạn đang ở trong một tình huống mà bạn muốn định cấu hình nhiều quyền cho một dịch vụ API còn lại cụ thể hoặc một đường dẫn API, thì bạn có thể sử dụng hasAnyAuthority. Vì vậy, phương pháp thứ hai, đó là hasAnyAuthority, nó sẽ chấp nhận danh sách các authorities có thẩm quyền giống như bạn có thể vượt qua nhiều authorities.

[.requestMatchers\("/myBalance"\).hasAnyAuthority\("VIEWACCOUNT", "VIEWBALANCE"\)](#)

Và nếu user có bất kỳ authority nào được liệt kê thì anh ta sẽ có thể truy cập vào chức năng cụ thể đó hoặc API phần còn lại cụ thể. Nếu không, anh ta sẽ gặp lỗi 403. Và phương pháp cuối cùng mà chúng ta có là access. Sử dụng phương thức truy cập này, chúng tôi có thể định cấu hình các quy tắc authorization phức tạp của mình với sự trợ giúp của (ngôn ngữ biểu thức) spring expression language. Có thể bạn có một số yêu cầu phức tạp khi bạn muốn sử dụng các toán tử logic như điều kiện AND, OR và NOT bất cứ khi nào bạn muốn định cấu hình các quyền hạn này. Vì vậy, trong những tình huống đó với sự trợ giúp của phương thức truy cập bằng cách sử dụng spring expression language, bạn có thể thực thi authorization bằng cách xác định authority và điều kiện của chúng.

Bước tiếp theo, chúng ta hãy cố gắng hiểu chúng ta cần gọi các phương thức này từ đâu. Các phương thức này, như hasAuthority, hasAnyAuthority và các phương thức access, chúng ta có thể gọi request to matches phương thức.



Giống như bạn có thể thấy ở đây, ngay sau khi tôi thực hiện xong **requestMatchers** cùng với đường dẫn API, tôi có thể gọi các phương thức authorization này như hasAuthority, hasAnyAuthority và các phương thức access.

Đối với authority, chúng tôi có thể chuyển một chi tiết authority duy nhất trong khi hasAnyAuthority sẽ chấp nhận nhiều thông tin chi tiết authority. Và nếu bạn không sử dụng hasAuthority hoặc hasAnyAuthority, bạn có thể chỉ cần đề cập đến .authenticated, điều đó có nghĩa là bất kỳ người dùng nào cũng có thể truy cập ứng dụng web của chúng tôi dù đã đăng nhập và chúng tôi không muốn thực thi bất kỳ authorization nào đối với API cụ thể đó. Vì vậy, ở đây đối với người dùng dấu gạch chéo, chúng tôi không muốn có bất kỳ authorization nào, đó là lý do tại sao chúng tôi chỉ giữ nó dưới dạng các public API được xác thực và còn lại, chúng tôi không nên chạm vào chúng. Họ vẫn nên có giấy phép tất cả. Vì vậy, theo cách này, chúng tôi có thể định config authorities và thực hiện authorization bên trong Spring boot web application.

## 6. Authority Vs Role in Spring Security

**Authority** giống như một đặc quyền cá nhân mà người dùng có thể có hoặc nó cũng có thể đại diện cho một hành động cá nhân mà user của tôi có thể thực hiện bên trong ứng dụng web của tôi. Nếu bạn kiểm tra các authority như VIEWACCOUNT, VIEWCARDS, VIEWLOANS, thì khi sử dụng các authority này, anh ấy chỉ có thể thực hiện một trong các hành động, chẳng hạn như xem chi tiết tài khoản, chi tiết thẻ hoặc chi tiết khoản vay của mình. Vì các authority này đại diện cho một đặc quyền cá nhân hoặc một hành động cá nhân, nên chúng tôi có thể sử dụng các authority này để hạn chế quyền truy cập một cách chi tiết.

Trong khi role đại diện cho một nhóm các authority. Đôi khi, nhiều ứng dụng web dành cho doanh nghiệp, chúng sẽ có hàng nghìn hàng nghìn hành động mà ứng dụng doanh nghiệp của tôi sẽ hỗ trợ. Ở đây, chúng tôi chỉ có bốn hoặc năm API REST, nhưng trong thế giới thực, bạn có thể tưởng tượng

sẽ dễ dàng có hàng trăm dịch vụ và mỗi dịch vụ sẽ có một cơ quan tương ứng. Vì vậy, trong những trường hợp như vậy, việc đối phó và làm việc với các authority sẽ trở nên vô cùng phức tạp. Đó là lý do tại sao chúng ta có thể nhóm các authority này theo role.

Vì vậy, thay vì tạo rất nhiều authority bên trong ứng dụng web của mình, tôi có thể cho rằng người dùng của mình có thể thực hiện các hoạt động như vậy, admin của tôi có thể thực hiện các hoạt động như vậy và theo đó, tôi có thể tạo các role nhất định bên trong ứng dụng web của mình. Vì vậy, các role thường đại diện cho một nhóm các đặc quyền hoặc hành động. Bất cứ khi nào bạn thực thi authorization với sự trợ giúp của các role, thì bạn đang hạn chế quyền truy cập theo cách thức chi tiết. Quản lý chi tiết có nghĩa là bạn sẽ đi vào chi tiết rất nhỏ đối với các cấp hành động của API REST như VIEWACCOUNT, VIEWCARDS, trong khi authorization chi tiết có nghĩa là bạn sẽ không đi vào chi tiết lắm. Ở cấp độ cao, bạn sẽ tạo một số role và bạn sẽ chỉ thực thi authorization với sự trợ giúp của các role. Và các cơ quan này là các role, những tên này dành riêng cho Spring Security. Nhiều ứng dụng, họ sẽ gọi chúng bằng các quy ước đặt tên khác nhau. Một số dự án, họ gọi nó là privileges. Một số dự án, họ gọi nó là actions. Bất kể tên mà chúng ta theo dõi bên trong các dự án là gì, nhưng khi đến với Spring Security, chúng ta nên chọn các authority hoặc role để thực thi authorization.

Tiêu chuẩn đó là, bạn phải luôn có tiền tố ROLE\_ bên trong hệ thống lưu trữ của mình để phân biệt xem chuỗi cụ thể đang đại diện cho authority hay role bên trong ứng dụng web của bạn. Bước tiếp theo, bên trong cơ sở dữ liệu của chúng tôi, chúng tôi có thể tạo một số role như ROLE\_ADMIN và ROLE\_USER bên trong cơ sở dữ liệu. Tôi đã tạo hai role mới cho các user.

## 7. Configuring Roles Authorization inside web app using Spring Security

Chúng ta có thể sử dụng bất kỳ phương pháp nào trong ba phương pháp này.

**CONFIGURING AUTHORITIES  
INSIDE SPRING SECURITY**

**eazy  
bytes**

**🔒** In Spring Security the ROLES requirements can be configured using the following ways,

**hasRole()** — Accepts a single role name for which the endpoint will be configured and user will be validated against the single role mentioned. Only users having the same role configured can invoke the endpoint.

**hasAnyRole()** — Accepts multiple roles for which the endpoint will be configured and user will be validated against the roles mentioned. Only users having any of the role configured can call the endpoint.

**access()** — Using Spring Expression Language (SpEL) it provides you unlimited possibilities for configuring roles which are not possible with the above methods. We can use operators like OR, AND inside access() method.

**Note :**

- ROLE\_ prefix only to be used while configuring the role in DB. But when we configure the roles, we do it only by its name.
- access() method can be used not only for configuring authorization based on authority or role but also with any special requirements that we have. For example we can configure access based on the country of the user or current time/date.

Phương thức đầu tiên là hasRole sẽ chấp nhận một role duy nhất làm đầu vào. Vì user có role cụ thể đó, họ chỉ có thể truy cập API cụ thể đó khi bạn đang sử dụng phương thức **hasRole** này, nhưng nếu bạn có yêu cầu trong đó một đường dẫn API cụ thể có thể được truy cập bởi bất kỳ danh sách role nào, thì bạn có thể sử dụng **hasAnyRole**. Họ rất giống với hasAuthority hoặc anyAuthority.

Và phương pháp cuối cùng mà chúng ta có thể sử dụng là phương pháp tiếp cận mà chúng ta đã thảo luận khi nói về authority. Vì vậy, đây là phương pháp tương tự mà chúng ta có thể thực thi một số quy tắc **Authorization** với sự trợ giúp của role và spring expression language. Và chúng ta có thể sử dụng bất kỳ toán tử logic nào như điều kiện OR và NOT, điều này có thể xảy ra khi bạn sử dụng spring expression language. Nhưng một thông tin quan trọng mà tôi muốn nhấn mạnh ở đây là bên trong cơ sở dữ liệu của chúng tôi, chúng tôi đang duy trì lệnh gọi tiền tố, ROLE\_ cho tất cả các vai trò của mình. Nhưng khi bạn đang cố gắng sử dụng hasRole hoặc hasAnyRole hoặc phương thức truy cập này, bạn không nên đề cập đến tiền tố đó. Nếu không có giá trị tiền tố đó, bạn nên đề cập đến lý do là nội bộ, bất cứ khi nào bạn gọi phương thức hasRole hoặc hasAnyRole này, Spring Security sẽ thêm giá trị tiền tố đó. Đó là lý do tại sao bạn không cần phải đề cập đến giá trị tiền tố đó theo cách thủ công bất cứ khi nào bạn đang cố gắng sử dụng các phương thức này bên trong cấu hình dự án của mình.

Like shown below, we can configure ROLES requirements for the APIs/Paths.

```
@Bean
SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {
    CsrfTokenRequestAttributeHandler requestHandler = new CsrfTokenRequestAttributeHandler();
    requestHandler.setCsrfRequestAttributeName("_csrf");

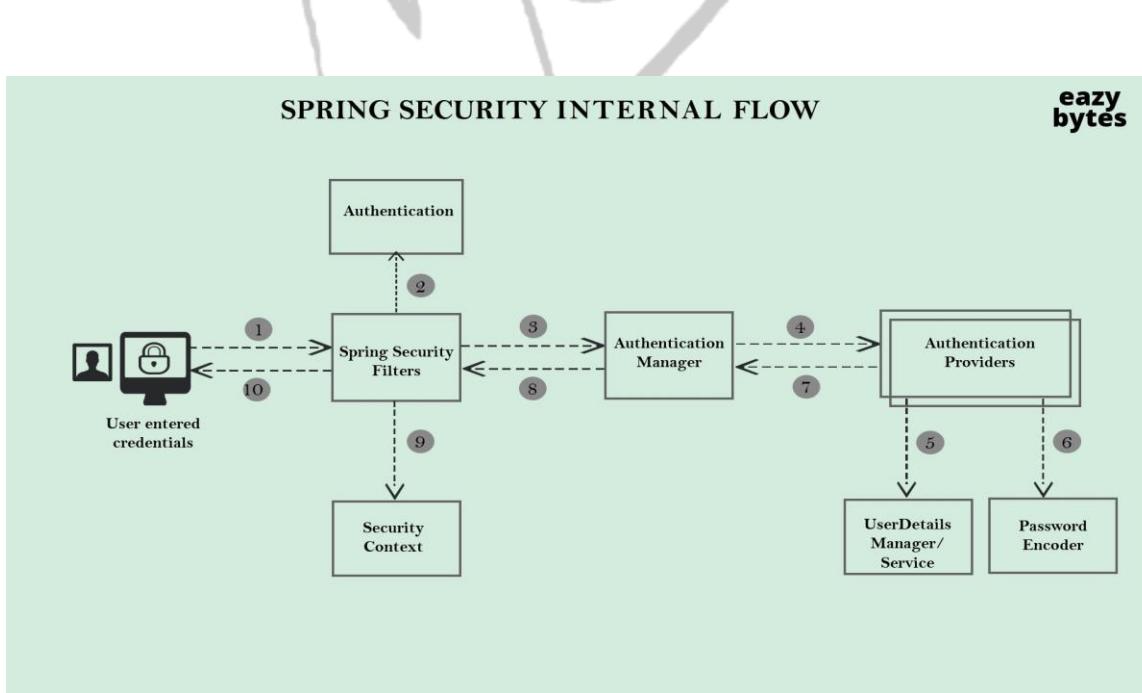
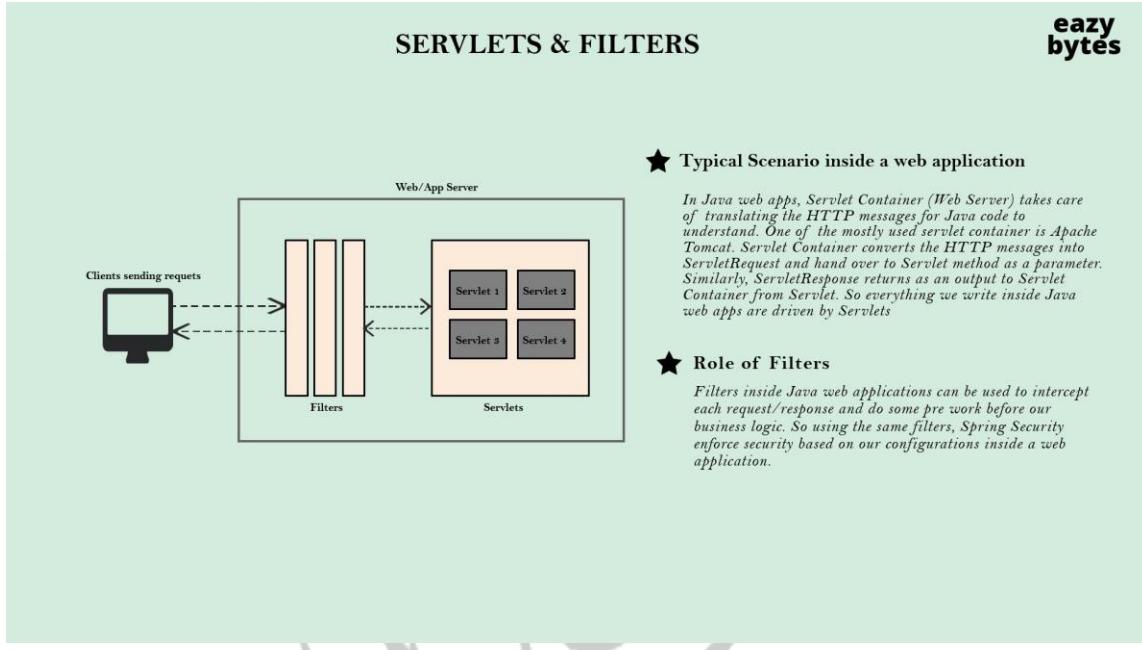
    http.securityContext().requireExplicitSave(false)
        .and().sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.ALWAYS))
        .cors().configurationSource(new CorsConfigurationSource() {...}).and().csrf((csrf) -> csrf.csrfTokenRequestHandler(requestHandler)
            .ignoringRequestMatchers("/contact", "/register")
            .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
            .addFilterAfter(new CsrfCookieFilter(), BasicAuthenticationFilter.class)
            .authorizeHttpRequests()
                .requestMatchers("/myAccount").hasRole("USER")
                .requestMatchers("/myBalance").hasAnyRole("USER", "ADMIN")
                .requestMatchers("/myLoans").hasRole("USER")
                .requestMatchers("/myCards").hasRole("USER")
                .requestMatchers("/user").authenticated()
                .requestMatchers("/notices", "/contact", "/register").permitAll()
        .and().formLogin()
        .and().httpBasic();
    return http.build();
}
```

May

## Section 8: Writing our own Custom Filters in Spring Security

### 1. Introduction to Filters in Spring Security and the sample use cases

Các **Filters** là một thành phần bên trong ứng dụng web Java, có thể được sử dụng để chặn từng yêu cầu hoặc phản hồi bên trong ứng dụng web của chúng ta. Và trong quá trình chặn này, chúng tôi có thể thực thi một số logic trước khi làm việc mà chúng tôi muốn được thực thi trước business logic thực tế của mình. Đó là lý do tại sao sử dụng khái niệm Filters, Spring Security của tôi đang thực thi tất cả bảo mật dựa trên các cấu hình mà chúng tôi thực hiện bên trong bất kỳ ứng dụng web.



Lý do tại sao lại thảo luận về các bộ lọc là trong phần này, chúng ta sẽ xác định các bộ lọc tùy chỉnh của riêng mình có thể được thực thi trong quá trình authentication và authorization. Tại đây, bạn có thể đặt lại câu hỏi tại sao tôi muốn sử dụng filter tùy chỉnh bên phải bên trong Spring Security?

Nguyên nhân có thể do nhiều nguyên nhân. Rất nhiều lần bên trong các ứng dụng doanh nghiệp phức tạp, chúng tôi sẽ gặp một số tình huống nhất định hoặc yêu cầu của khách hàng để thực hiện một số hoạt động trong quy trình authentication và authorization. Bạn có thể muốn thực hiện một số vi phạm đầu vào trước khi authentication. Bạn có thể muốn thêm một số chi tiết theo dõi hoặc kiểm tra và báo cáo vào yêu cầu của mình. Bạn có thể muốn ghi lại một số chi tiết nhất định của user đang cố gắng truy cập vào hệ thống của bạn trong quá trình authentication. Có thể bạn muốn ghi địa chỉ IP, chi tiết quốc gia của anh ấy, bất kỳ chi tiết nào bạn muốn ghi nếu bạn muốn thực hiện một số mã hóa và giải mã dữ liệu đầu vào trước khi authentication hoặc nếu bạn muốn thực thi một số authentication đa yếu tố bằng OTP. Vì vậy, các kịch bản có thể là bất cứ điều gì.

## FILTERS IN SPRING SECURITY

**eazy  
bytes**

- ✓ Lot of times we will have situations where we need to perform some house keeping activities during the authentication and authorization flow. Few such examples are,

- Input validation
- Tracing, Auditing and reporting
- Logging of input like IP Address etc.
- Encryption and Decryption
- Multi factor authentication using OTP

- ✓ All such requirements can be handled using HTTP Filters inside Spring Security. Filters are servlet concepts which are leveraged in Spring Security as well.

- ✓ We already saw some built-in filters of Spring security framework like UsernamePasswordAuthenticationFilter, BasicAuthenticationFilter, DefaultLoginPageGeneratingFilter etc. in the previous sections.

- ✓ A filter is a component which receives requests, processes its logic and handover to the next filter in the chain.

- ✓ Spring Security is based on a chain of servlet filters. Each filter has a specific responsibility and depending on the configuration, filters are added or removed. We can add our custom filters as well based on the need.

Nếu bạn tham gia vào bất kỳ dự án thời gian thực nào, sẽ có rất nhiều tình huống trong đó bạn phải viết logic tùy chỉnh của riêng mình trước khi authentication thực tế diễn ra sau khi authentication hoặc authorization. Vì vậy, đối với tất cả các loại tình huống này, chúng tôi cần viết các custom filter của riêng mình và chúng tôi cần đưa chúng vào luồng của Spring Security filters. Vì chúng tôi muốn chặn từng request hoặc responses, nên tùy chọn rõ ràng mà chúng tôi có là HTTP Filter. Giống như chúng ta biết, các filter là loại servlet đặc biệt cũng được sử dụng bên trong Spring Security. Chúng tôi đã thấy một số filter sẵn có được sử dụng bởi Spring Security trong quá trình authentication người dùng. filter này như UsernamePasswordAuthenticationFilter, BasicAuthenticationFilter, DefaultLoginPageGeneratingFilter. Vì vậy, tất cả các bộ lọc sẵn có của Spring Security này mà chúng tôi đã thấy rất nhiều lần trong khóa học này. Nếu bạn nhìn vào chuỗi bộ lọc Spring Security, sẽ có rất nhiều bộ lọc sẽ được thực thi bất cứ khi nào chúng tôi cố gắng authentication. Số có hơn 10 bộ lọc Spring Security sẽ có vai trò và trách nhiệm riêng. Cách các filter này thực thi theo kiểu dây chuyền. Vì vậy, bên trong các filter chain này, tất cả các bộ lọc sẽ được xử lý từng cái một, giống như filter một sẽ được thực

thi sau đó từ filter một sẽ hoạt động, filter hai tiếp theo là filter ba. Vì vậy, chuỗi này sẽ tiếp tục hoạt động với các filter tiếp theo cho đến khi chúng đạt đến filter cuối cùng bên trong chuỗi-chain.

## 2. How to create our own custom filter

Bất cứ khi nào bạn muốn viết filter tùy chỉnh của riêng mình, bạn chỉ cần extend filter interface, có sẵn trong các thư viện Java của chúng tôi. Khi bạn triển khai filter interface này, bên trong lớp nơi bạn đang triển khai filter, bạn cần viết tất cả business logic của mình hoặc bạn cần viết tất cả logic mà bạn muốn được thực thi như một phần của filter đó bên trong một phương thức có tên là doFilter. Vì vậy, phương thức doFilter này sẽ cung cấp cho bạn hoặc sẽ chấp nhận ba tham số đầu vào. Tham số đầu tiên là HttpServletRequest, đại diện cho yêu cầu đầu vào HTTP đến từ user của bạn. Và tham số thứ hai là HttpServletResponse và điều này thể hiện phản hồi HTTP mà chúng tôi sẽ gửi lại cho user hoặc client. Và tham số thứ ba là FilterChain. Giống như chúng ta đã thảo luận trước đây, FilterChain đại diện cho một tập hợp các bộ lọc với thứ tự được xác định mà chúng sẽ thực thi. Những gì chúng ta sẽ làm bằng cách lấy tham số FilterChain này là khi chúng ta hoàn thành việc thực thi của mình, chúng ta cần gọi bộ lọc tiếp theo có sẵn bên trong FilterChain. Đó là lý do tại sao chúng ta có tham số thứ ba này cho phương thức doFilter.

### IMPLEMENTING CUSTOM FILTERS

INSIDE SPRING SECURITY

eazy  
bytes

- ✓ We can create our own filters by implementing the **Filter** interface from the `jakarta.servlet` package. Post that we need to override the `doFilter()` method to have our own custom logic. This method accepts 3 parameters the `ServletRequest`, `ServletResponse` and `FilterChain`.
  - **ServletRequest**—It represents the HTTP request. We use the `ServletRequest` object to retrieve details about the request from the client.
  - **ServletResponse**—It represents the HTTP response. We use the `ServletResponse` object to modify the response before sending it back to the client or further along the filter chain.
  - **FilterChain**—The filter chain represents a collection of filters with a defined order in which they act. We use the `FilterChain` object to forward the request to the next filter in the chain.

- ✓ You can add a new filter to the spring security chain either before, after, or at the position of a known one. Each position of the filter is an index (a number), and you might find it also referred to as “the order.”
- ✓ Below are the methods available to configure a custom filter in the spring security flow,
  - `addFilterBefore(filter, class)` – adds a filter before the position of the specified filter class
  - `addFilterAfter(filter, class)` – adds a filter after the position of the specified filter class
  - `addFilterAt(filter, class)` – adds a filter at the location of the specified filter class

Bên trong Spring Security framework, chúng ta có ba phương thức quan trọng. Các phương thức là addFilterBefore, addFilterAfter và addFilterAt. Sử dụng các bộ lọc này, chúng tôi có thể đưa Custom Filter của riêng mình vào FilterChain nội bộ của Spring Security. Nếu bạn nhìn vào các phương thức này, các phương thức này đang chấp nhận hai tham số. Một là đối tượng của lớp filter tùy chỉnh của bạn là gì? Và tham số thứ hai là cung cấp cho tôi tên filter tích hợp sẵn bên trong Spring Boot và dựa trên tên filter tích hợp này, tôi có thể thêm ngay trước đó hoặc ngay sau đó hoặc cùng một vị trí của filter tích hợp này.

Bây giờ, hãy thử kiểm tra interface filter này bên trong các thư viện Java của chúng tôi để nó trở nên cực kỳ rõ ràng đối với bạn.

```

package jakarta.servlet;

import java.io.IOException;

public interface Filter {
    default void init(FilterConfig filterConfig) throws ServletException {
    }

    void doFilter(ServletRequest var1, ServletResponse var2, FilterChain
var3) throws IOException, ServletException;

    default void destroy() {
    }
}

```

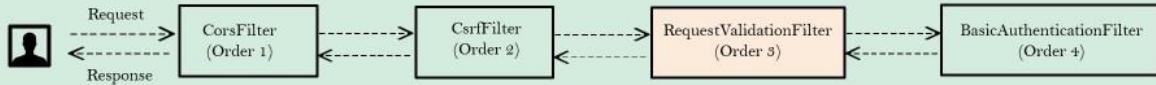
Điều này có trong thư viện Java của chúng tôi và tên package là jakarta.servlet. Nhưng package này, đó là jakarta.servlet, bạn chỉ có thể nhìn thấy bên trong các phiên bản Java gần đây chứ không phải các phiên bản cũ hơn. Trong các phiên bản cũ hơn, bạn sẽ thấy tên gói javax.servlet. Lý do tại sao nó được đổi tên này, Oracle đã bàn giao Java Enterprise Edition cho cộng đồng mã nguồn mở với tên jakarta. Đó là lý do tại sao họ đã đổi tên tất cả các tên package.

Bên trong bộ lọc này, bạn có thể thấy có ba phương pháp. Một là **init**, một phương thức **default**. Theo mặc định, nó trống. Và cái cuối cùng là phương thức **destroy**. Đây cũng là một phương thức **default** và nó trống. Chúng tôi có thể viết một số business logic nếu bạn muốn bên trong init và **destroy**. Vì vậy, về cơ bản, bất kỳ logic nào bạn viết bên trong init sẽ được thực thi bất cứ khi nào chúng tôi cố gắng khởi tạo filter này và tương tự, bất cứ khi nào bạn viết một số logic bên trong hàm **destroy**, thì điều tương tự sẽ được thực thi bất cứ khi nào filter cụ thể này bị hủy bên trong bộ chứa servlet. Nhưng theo mặc định, điều này có thể trống. Không có lo lắng cho chúng tôi để ghi đè lên chúng. Đến với phương thức chính, **doFilter**. Vì vậy, đây là phương pháp mà bạn cần ghi đè và triển khai tất cả business logic của mình bất cứ khi nào bạn muốn xác định filter tùy chỉnh của riêng mình.

### 3. Adding a custom filter using addFilterBefore() method

Với sự trợ giúp của phương pháp **addFilterBefore**. Như bạn có thể thấy ở đây, tôi đã xác định một số security filter sẵn có bên trong Spring Security. Hãy nghĩ rằng bộ lọc đầu tiên mà nó sẽ thực thi cors filter là **CORS filter**. Và sau những bộ lọc này, chúng tôi cũng có một bộ lọc sẵn có với **BasicAuthenticationFilter**. Nếu yêu cầu của tôi là thực thi một số logic ngay trước khi authentication thì tôi cần tạo một filter tùy chỉnh và cùng một filter tùy chỉnh mà tôi phải định cấu hình ngay trước **BasicAuthenticationFilter**. Chúng tôi biết bên trong **BasicAuthenticationFilter**, việc trích xuất thông tin đăng nhập sẽ diễn ra và dựa trên những thông tin đăng nhập này, việc authentication thực tế của user sẽ diễn ra bên trong Spring Security. Đó là lý do vì mục tiêu của tôi là viết một số logic ngay trước khi authentication, tôi sẽ tạo một filter tùy chỉnh mới với tên **RequestValidationFilter** và filter tùy chỉnh này tôi sẽ định cấu hình ngay trước **BasicAuthenticationFilter**.

addFilterBefore(filter, class) – It will add a filter before the position of the specified filter class.



Here we add a filter just before authentication to write our own custom validation where the input email provided should not have the string 'test' inside it.

Bất kể các số nào khác mà tôi đã xác định ở đây như một, hai, ba, bốn khác, những số này tôi chỉ trình bày cho mục đích hiểu biết của chúng tôi, nhưng bên trong **Spring Security Filter**, các số có thể khác nhau. Vì vậy, tôi đã đề cập đến những con số này chỉ cho mục đích hiểu biết của chúng tôi. Logic mà tôi sẽ viết bên trong yêu cầu này tới Authentication Filter là nếu ai đó đang cố đăng nhập bằng tên người dùng bao gồm kiểm tra giá trị "TEST". Tôi sẽ báo lỗi lại cho user nói rằng đó là lỗi yêu cầu không hợp lệ có nghĩa là bất kỳ ai đang cố đăng nhập bằng tên người dùng "TEST" họ sẽ không thể đăng nhập vào ứng dụng của tôi vì **authentication** sẽ không bao giờ xảy ra. Và với filter tùy chỉnh của mình, tôi sẽ từ chối tất cả các yêu cầu tìm kiếm.

### Create my own custom filter

Tạo **RequestValidationBeforeFilter** bên trong package **filter**

```

public class RequestValidationBeforeFilter implements Filter {

    public static final String AUTHENTICATION_SCHEME_BASIC = "Basic";
    private Charset credentialsCharset = StandardCharsets.UTF_8;

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
        throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;
        String header = req.getHeader(AUTHORIZATION);
        if (header != null) {
            header = header.trim();
            if (StringUtils.startsWithIgnoreCase(header,
AUTHENTICATION_SCHEME_BASIC)) {
                byte[] base64Token =
  
```

```

        header.substring(6).getBytes(StandardCharsets.UTF_8);
        byte[] decoded;
        try {
            decoded = Base64.getDecoder().decode(base64Token);
            String token = new String(decoded, credentialsCharset);
            int delim = token.indexOf(":");
            if (delim == -1) {
                throw new BadCredentialsException("Invalid basic
authentication token");
            }
            String email = token.substring(0, delim);
            if (email.toLowerCase().contains("test")) {
                res.setStatus(HttpServletRequest.SC_BAD_REQUEST);
                return;
            }
        } catch (IllegalArgumentException e) {
            throw new BadCredentialsException("Failed to decode basic
authentication token");
        }
    }
    chain.doFilter(request, response);
}
}

```

Bạn có thể thấy tôi có **doFilter** method bộ lọc chấp nhận ba tham số đầu vào là **ServletRequest**, **ServletResponse** và **FilterChain**. Bên trong **doFilter** method, chúng ta cần viết một số logic mà chúng ta muốn được thực thi như một phần của bộ lọc này.

Tôi đang cố gắng ép kiểu request và response này vào một **HttpServletRequest** và **HttpServletResponse**. Từ request, Tôi đang cố lấy **authorization header** sẽ được gửi bởi angular UI application. Khi chúng tôi có **authorization header**, tôi sẽ cắt nó để không có khoảng trống bên trong giá trị **header** của tôi. Nếu bạn kiểm tra bên trong UI application, bạn sẽ thấy chúng tôi đang gửi một header ủy quyền với một giá trị cơ bản theo sau là khoảng trắng.

```

-- app.request.interceptor.ts
httpHeaders = httpHeaders.append('Authorization', 'Basic ' +
window.btoa(this.user.email + ':' + this.user.password));

```

Và chúng tôi đang cố gắng gửi base 64 value của email và mật khẩu của tôi được phân tách bằng dấu hai chấm phân cách. Chúng tôi sẽ sử dụng một chuỗi con và chúng tôi đang cố trích xuất giá trị đầu tiên bên trong **authorization header**, đó là username. Quan tâm hơn đến việc xác thực username hoặc email nên đang trích xuất email và khi đã có email trong tay, sẽ cố gắng xác thực xem email có chứa bất kỳ giá trị "test" nào không. Nếu có, tôi sẽ trả lại trạng thái yêu cầu không hợp lệ với lỗi 400.

Mặt khác, nếu không có vấn đề gì trong request, tôi chỉ cần gọi **doFilter**. Vì vậy, đối tượng chuỗi này sẽ nhận được dưới dạng tham số thứ ba cho **doFilter method**.

Bây giờ là bước tiếp theo, chúng ta cần thêm **custom filter** này ngay trước **BasicAuthenticationFilter**. Tương tự, chúng ta cần vào **ProjectSecurityConfig**. Bên trong lớp này, bây giờ chúng ta có thể định cấu hình bộ lọc tùy chỉnh của riêng mình để được thực thi ngay trước **BasicAuthentication Filter**.

```
// ProjectSecurityConfig

.addFilterAfter(new CsrfCookieFilter(), BasicAuthenticationFilter.class)
```

Giống như bạn có thể thấy ở đây, chúng tôi đã cấu hình một custom filter, đó là CSRF cookie filter. Tương tự như vậy, chúng ta cũng có thể cấu hình custom filter của riêng mình.

```
public class ProjectSecurityConfig {

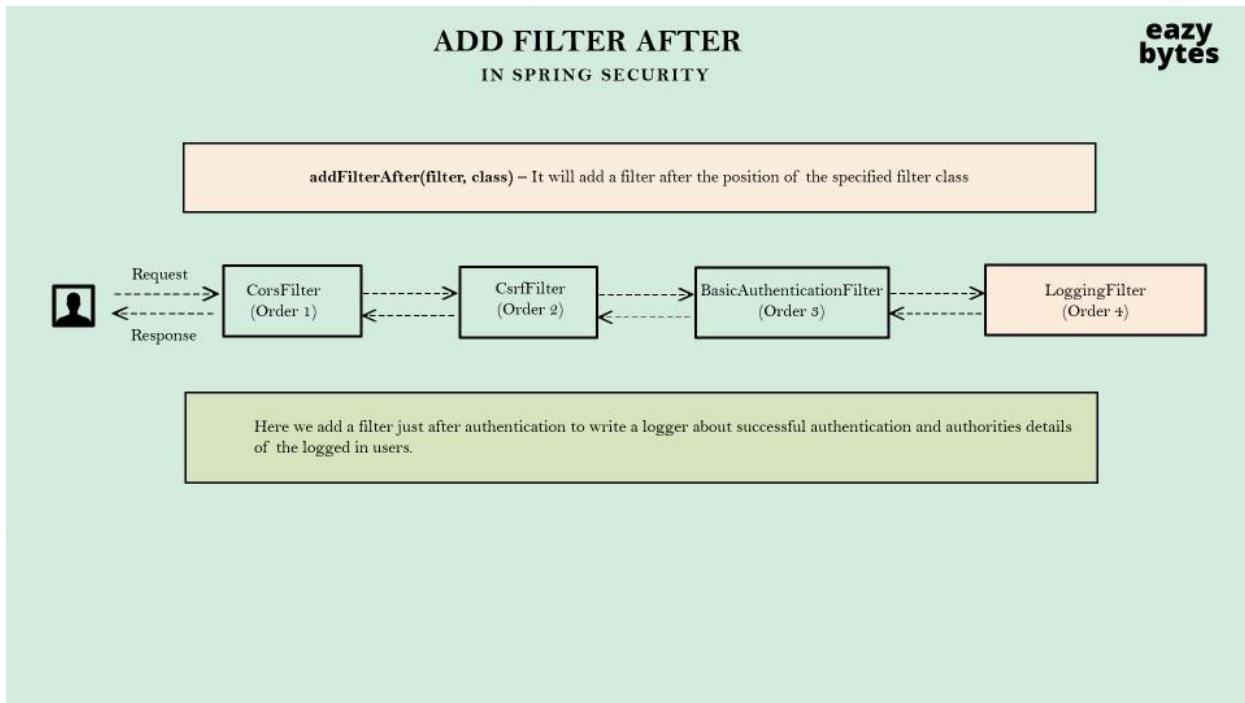
    @Bean
    SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {
        CsrfTokenRequestAttributeHandler requestHandler = new CsrfTokenRequestAttributeHandler();
        requestHandler.setCsrfRequestAttributeName("_csrf");
        http.securityContext().requireExplicitSave(false) SecurityContextConfigurer<HttpSecurity>
            .and().sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.ALWAYS)) HttpSecurity
            .cors().configurationSource(new CorsConfigurationSource());

        no usages
        @Override
        public CorsConfiguration getCorsConfiguration(HttpServletRequest request) {
            CorsConfiguration config = new CorsConfiguration();
            config.setAllowedOrigins(Collections.singletonList("http://localhost:4200"));
            config.setAllowedMethods(Collections.singletonList("*"));
            config.setAllowCredentials(true);
            config.setAllowedHeaders(Collections.singletonList("*"));
            config.setMaxAge(3600L);
            return config;
        }
    }.and().csrf((csrf) -> csrf.csrfTokenHandler(requestHandler).ignoringRequestMatchers(
        ...patterns: "/contact", "/register"
        .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
        .addFilterAfter(new CsrfCookieFilter(), BasicAuthenticationFilter.class)
        .addFilterBefore(new RequestValidationBeforeFilter(), BasicAuthenticationFilter.class)
        .addFilterAfter(new AuthoritiesLoggingAfterFilter(), BasicAuthenticationFilter.class)
        .addFilterAfter(new AuthoritiesLoggingAfterFilter(), BasicAuthenticationFilter.class)
        .authorizeHttpRequests() AuthorizationManagerRequestMat...
            .requestMatchers(PathVariable"/myAccount").hasRole("USER")
            .requestMatchers(PathVariable"/myBalance").hasAnyRole( ...roles: "USER", "ADMIN")
            .requestMatchers(PathVariable"/myLoans").hasRole("USER")
            .requestMatchers(PathVariable"/myCards").hasRole("USER")
            .requestMatchers(PathVariable"/user").authenticated()
            .requestMatchers(PathVariable"/notices", PathVariable"/contact", PathVariable"/register").permitAll()
        .and().formLogin() FormLoginConfigurer<HttpSecurity>
        .and().httpBasic();
    ).return http.build();
}

@Bean
public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
}
```

Khi hoàn thành, chúng ta có thể truy cập ứng dụng UI và kiểm tra quy trình đăng nhập. Bằng cách này, bạn có thể có bất kỳ custom logic nào mà bạn muốn được thực thi ngay trước khi **authentication** và bạn luôn có thể viết một custom filter và định cấu hình nó với sự trợ giúp của phương thức **addFilterBefore**. Nhưng hãy đảm bảo rằng bên trong bất kỳ bộ lọc tùy chỉnh nào, bạn không bao giờ được viết bất kỳ logic nào sẽ mất nhiều thời gian, đặc biệt là các truy vấn DB và hàng nghìn hàng nghìn dòng mã vì điều đó sẽ có một số tác động đến hiệu suất đối với ứng dụng của bạn. Lý do là mỗi và mọi yêu cầu đến với ứng dụng web của bạn sẽ bị chặn bởi bộ lọc tùy chỉnh này và điều tương tự sẽ được thực hiện cho tất cả các yêu cầu khác của bạn.

#### 4. Adding a custom filter using addFilterAfter() method



Tôi muốn một số business logic được thực thi ngay sau khi quá trình **authentication** của tôi hoàn tất. Đối với loại kịch bản này, chúng ta cũng có thể chọn BasicAuthenticationFilter. Nếu chúng tôi có thể định cấu hình **custom authentication filter** của mình để được thực thi ngay sau BasicAuthenticationFilter, thì yêu cầu của chúng tôi sẽ được đáp ứng. Những gì tôi sẽ viết bên trong **custom authentication filter** của mình trong tình huống này là tôi sẽ thêm một trình ghi nhật ký vào bên trong custom filter của mình nói rằng như vậy và do đó, xác thực người dùng thành công và anh ta có quyền tương tự. Để định cấu hình **filter** ngay sau một trong các **filter** tích hợp sẵn của **Spring Security**, chúng ta có thể sử dụng phương thức **addFilterAfter**. Sử dụng phương pháp này, bạn có thể truyền hai tham số. Tham số đầu tiên là đối tượng của **custom filter** của bạn và tham số thứ hai là class name của filter tích hợp Spring Security của bạn.

Tạo custom filter mới là AuthoritiesLoggingAfterFilter

```
public class AuthoritiesLoggingAfterFilter implements Filter {  
  
    private final Logger LOG =  
        Logger.getLogger(AuthoritiesLoggingAfterFilter.class.getName());  
  
    @Override  
    public void doFilter(ServletRequest request, ServletResponse response,  
    FilterChain chain)  
        throws IOException, ServletException {  
  
        Authentication authentication =  
SecurityContextHolder.getContext().getAuthentication();  
        if (null != authentication) {  
            LOG.info("User " + authentication.getName() + " is successfully  
authenticated and "  
            + "has the authorities " +
```

```
        authentication.getAuthorities().toString());
    }
    chain.doFilter(request, response);
}

}
```

Sau đó, cần truy cập lớp ProjectSecurityConfig và ở đây, ngay sau khi cấu hình **addFilterBefore**, tôi có thể gọi một phương thức có tên là **addFilterAfter**.

```
.csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse()))
    .addFilterAfter(new CsrfCookieFilter(), BasicAuthenticationFilter.class)
    .addFilterBefore(new RequestValidationBeforeFilter(), BasicAuthenticationFilter.class)
    .addFilterAfter(new AuthoritiesLoggingAfterFilter(), BasicAuthenticationFilter.class)
```

## 5. Adding a custom filter using addFilterAt() method

Hãy khám phá một phương thức khác mà chúng ta có, đó là **addFilterAt**. Sử dụng phương pháp này, bạn có thể định cấu hình chính xác **custom filter** của riêng mình vào một vị trí cụ thể trong chuỗi bộ lọc của Spring Security.

Các bộ lọc trong Spring Security được sắp xếp theo một thứ tự cụ thể để thực hiện các chức năng bảo mật như xác thực người dùng, phân quyền và xử lý các yêu cầu HTTP. Bằng cách sử dụng **addFilterAt()**, bạn có thể chèn một bộ lọc tùy chỉnh vào vị trí mong muốn trong chuỗi bộ lọc này.

Việc thêm một bộ lọc tùy chỉnh vào một vị trí cụ thể có thể hữu ích trong các trường hợp sau:

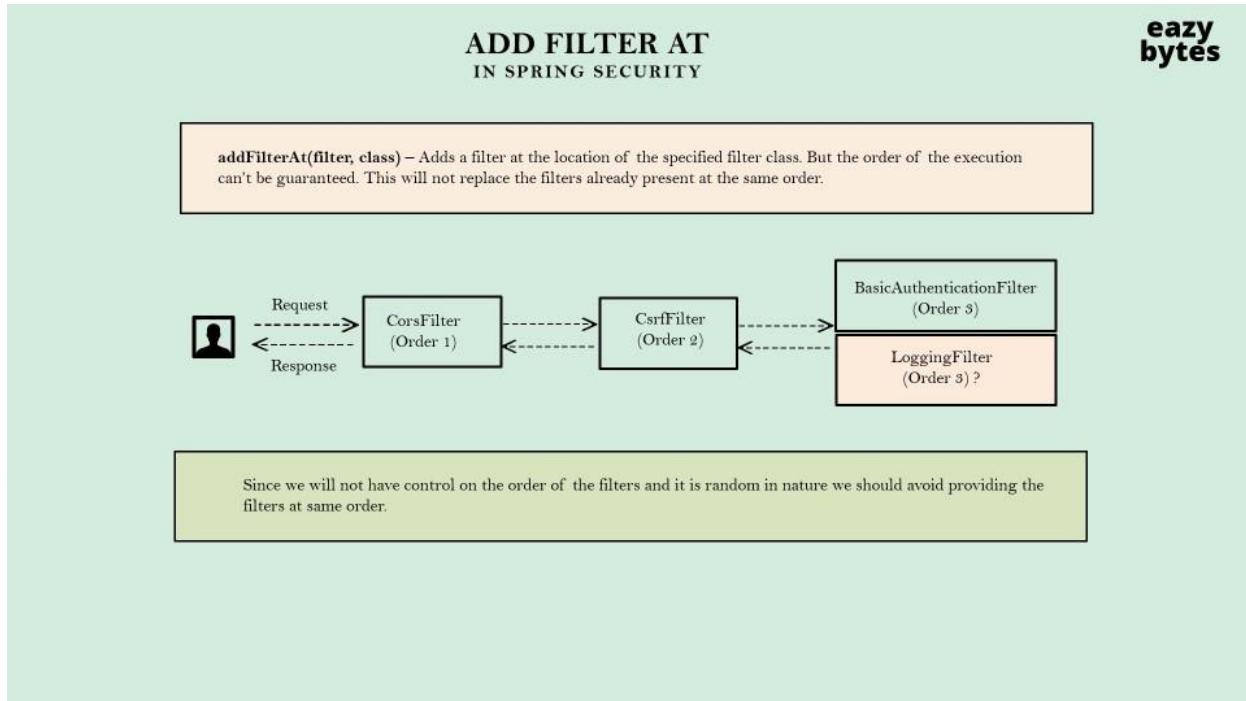
1. Mở rộng chức năng bảo mật: Bạn muốn thêm một bộ lọc tùy chỉnh để xử lý một phần tử bảo mật cụ thể, chẳng hạn như xử lý các thông tin đăng nhập từ một nguồn dữ liệu không phải chuẩn hoặc thực hiện một xác thực người dùng bổ sung.
2. Thay đổi thứ tự thực hiện bộ lọc: Bạn muốn đảm bảo rằng một bộ lọc tùy chỉnh được thực hiện trước hoặc sau một bộ lọc cụ thể trong chuỗi bộ lọc, để tùy chỉnh hành vi bảo mật của ứng dụng.
3. Kích hoạt tính năng tùy chỉnh: Bạn muốn kích hoạt một tính năng tùy chỉnh trong quá trình xử lý bảo mật bằng cách thêm một bộ lọc tùy chỉnh vào vị trí thích hợp.

Với **addFilterAt()**, bạn cung cấp đối tượng bộ lọc tùy chỉnh của mình và lớp của bộ lọc cụ thể mà bạn muốn chèn nó vào. Spring Security sẽ chèn bộ lọc tùy chỉnh vào vị trí được chỉ định trong chuỗi bộ lọc, đảm bảo rằng nó được thực hiện theo đúng thứ tự mong muốn.

Lưu ý rằng **addFilterAt()** chỉ có sẵn trong phiên bản Spring Security 4.1 trở lên.

Bộ lọc nội bộ sẽ được thực thi bên trong chuỗi nhưng điều này sẽ hơi phức tạp. Ở đây, bạn có thể thắc mắc nếu tôi định cấu hình hai **filter** ở cùng một phần, đây là ba filter, filter nào sẽ được thực hiện trước? Trong nội bộ, Spring Security sẽ thực hiện ngẫu nhiên một trong số chúng. Đó là lý do tại sao bất cứ khi nào bạn sử dụng phương pháp này, đó là **addFilterAt**, hãy cẩn thận với logic của bạn. Bạn phải

luôn viết logic của mình theo cách mà bất kể bộ lọc nào sẽ được thực thi, sẽ không có bất kỳ tác dụng phụ nào đối với business logic của bạn.



Để giới thiệu cho bạn mục đích của phương pháp này, tôi sẽ tạo một bộ lọc tùy chỉnh mới có tên LoggingFilter, và bên trong bộ lọc này, tôi sẽ ghi lại một câu lệnh mới nói rằng quá trình xác thực đang được tiến hành. Có thể trong các ứng dụng, có thể không có bất kỳ tình huống nào mà bạn muốn sử dụng addFilterAt. Dự án thực tế, luôn sử dụng **addFilterBefore** hoặc **addFilterAfter**. Rất hiếm khi bạn thấy ai đó sử dụng phương pháp addFilterAt này. Có thể ai đó có yêu cầu nói rằng trong quá trình xác thực, họ muốn gửi email đến user nói rằng quá trình xác thực của bạn đang được tiến hành. Hoặc có thể họ muốn gửi thông báo tới một trong các ứng dụng nội bộ của họ nói rằng xác thực thành công.

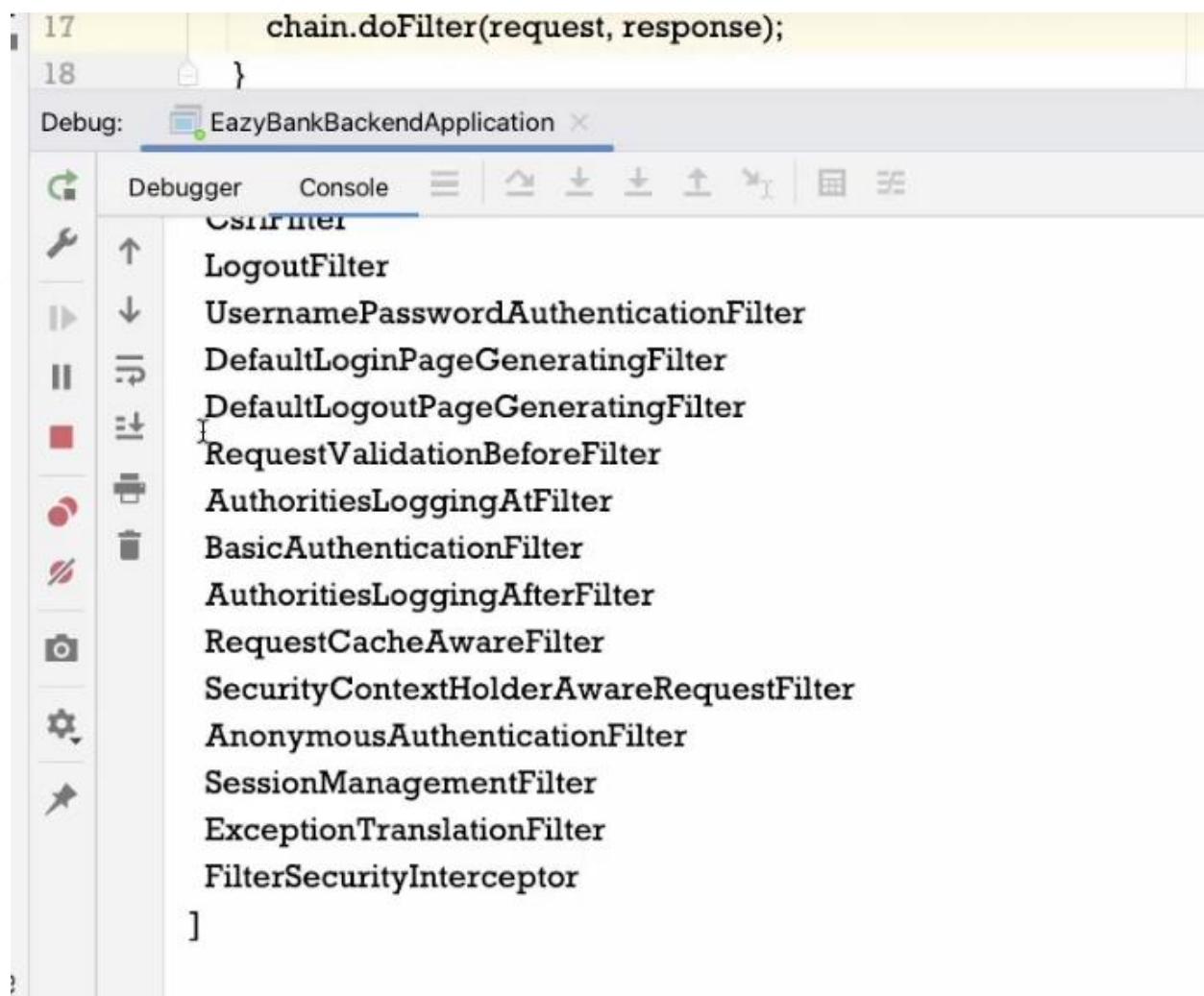
### Tạo custom filter mới là AuthoritiesLoggingAtFilter

```
public class AuthoritiesLoggingAtFilter implements Filter {  
  
    private final Logger LOG =  
        Logger.getLogger(AuthoritiesLoggingAtFilter.class.getName());  
  
    @Override  
    public void doFilter(ServletRequest request, ServletResponse response,  
    FilterChain chain)  
        throws IOException, ServletException {  
        LOG.info("Authentication Validation is in progress");  
        chain.doFilter(request, response);  
    }  
}
```

Sau đó, cần truy cập lớp ProjectSecurityConfig và ở đây

```
.csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse()))
    .addFilterAfter(new CsrfCookieFilter(), BasicAuthenticationFilter.class)
    .addFilterBefore(new RequestValidationBeforeFilter(), BasicAuthenticationFilter.class)
    .addFilterAt(new AuthoritiesLoggingAtFilter(), BasicAuthenticationFilter.class)
    .addFilterAfter(new AuthoritiesLoggingAfterFilter(), BasicAuthenticationFilter.class)
```

Khi hoàn thành, chúng ta có thể truy cập ứng dụng UI và kiểm tra quy trình đăng nhập.



Bạn có thể thấy các filter được thực hiện xung quanh BasicAuthenticationFilter. Filter được thực thi trước và sau BasicAuthenticationFilter này. Spring Security sẽ đảm bảo cho bạn về filter trước và sau. Nhưng liên quan đến filter cat, nó sẽ được thực thi trong bất kỳ filter nào khác. Trong một số trường hợp, BasicAuthenticationFilter sẽ được thực thi trước và trong một số trường hợp, bộ lọc tùy chỉnh của riêng bạn sẽ được thực thi trước. Vì vậy, nó là hoàn toàn ngẫu nhiên. Đó là lý do tại sao hãy hết sức cẩn thận về hành vi này.

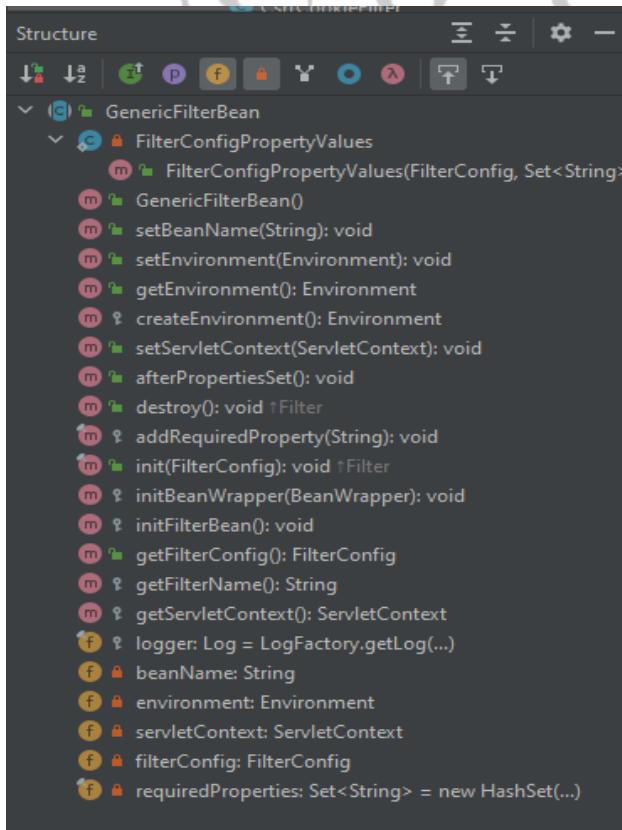
## 6. Details about GenericFilterBean and OncePerRequestFilter

Chúng ta đang cố triển khai một interface có tên là **Filter** và bắt cứ khi nào tôi đang cố implement filter đó, tôi chỉ cần đảm bảo rằng mình đang viết tất cả business logic của mình bằng cách ghi đè một phương thức có tên là `doFilter`. Nhưng ngoài interface **Filter**, còn có các tùy chọn nâng cao khác mà chúng tôi có bắt cứ khi nào chúng tôi sử dụng Spring Security. Tùy chọn đầu tiên là **GenericFilterBean**. Vì vậy, đây là một lớp bean trừu tượng và tùy chọn thứ hai mà chúng tôi có là **OncePerRequestFilter**. Hãy cố gắng hiểu những tình huống nào chúng ta có thể tận dụng các lớp này bắt cứ khi nào chúng ta muốn viết một custom filter.

Trước tiên tôi đã mở **GenericFilterBean**, một abstract class có sẵn bên trong Spring Security. Nếu bạn thấy lớp trừu tượng này cũng implement khai filter.

```
public abstract class GenericFilterBean implements Filter, BeanNameAware, EnvironmentAware, EnvironmentCapable, ServletContextAware, InitializingBean, DisposableBean
```

Nhưng lợi thế mà tôi sẽ nhận được bắt cứ khi nào tôi cố gắng tận dụng GenericFilterBean này bên trong custom filter của mình là gì? Đây là một triển khai cơ bản của **Filter** và nó là một class siêu hạng tiện dụng cho bất kỳ loại **filter** nào. Lợi thế mà chúng tôi sẽ nhận được bắt cứ khi nào chúng tôi cố gắng tận dụng GenericFilterBean là nó sẽ cung cấp tất cả các chi tiết về **config parameters, init parameters, and servlet context parameters** mà bạn đã định cấu hình bên trong web.xml hoặc bên trong bộ mô tả triển khai của mình. Vì vậy, nếu bạn có một tình huống trong đó bên trong custom filter của mình, bạn muốn truy cập tất cả các **config parameters, init parameters, and servlet context parameters** này thì chắc chắn bạn có thể sử dụng lớp trừu tượng này.



Tất cả những chi tiết này chúng tôi thường xác định bên trong bộ mô tả triển khai của mình để những chi tiết đó sẽ sẵn có bên trong abstract class này. Bằng cách đó, bạn không cần phải viết tất cả business logic để đọc các thông số này. Vì vậy, đó là lý do tại sao vui lòng tận dụng GenericFilterBean này bất cứ khi nào bạn gặp những tình huống như vậy.

Và lớp trừu tượng thứ hai mà tôi luôn khuyên bạn nên sử dụng là **OncePerRequestFilter**. Giống như bất cứ khi nào bạn tạo custom filter và cố gắng định cấu hình filter đó bên trong spring security filter chain. Theo mặc định, **Spring Security không thể đảm bảo với bạn rằng nó sẽ chỉ được thực thi một lần cho mỗi yêu cầu**. Có thể có các tình huống trong đó servlet container của bạn có thể gọi filter của bạn nhiều lần vì nhiều lý do khác nhau, nhưng nếu bạn muốn **đảm bảo filter của mình chỉ cần được thực thi một lần cho mỗi yêu cầu bằng mọi giá thì bạn nên xác định custom filter của mình bằng cách mở rộng lớp trừu tượng này đó là OncePerRequestFilter**. Tên này cũng là **OncePerRequestFilter**, có nghĩa là filter này sẽ đảm bảo rằng filter của bạn sẽ chỉ được thực thi một lần cho mỗi yêu cầu. Nó sẽ hoạt động như thế nào, nếu bạn đi và kiểm tra bên trong phương thức doFilter có sẵn bên trong OncePerRequestFilter này, nó có một lượng logic tốt để xác định xem filter của tôi đã được thực thi hay chưa. Nếu nó đã được thực thi, nó chỉ cần tiếp tục mà không cần gọi filter. Nó đang cố gắng tiếp tục mà không cần gọi custom filter của tôi. Vì vậy, tất cả logic bỏ qua và đảm bảo filter sẽ chỉ được thực thi một lần được xác định bên trong phương thức doFilter này.

Nếu OncePerRequestFilter của tôi đã triển khai phương thức doFilter bằng cách ghi đè nó thì tôi có thể xác định business logic của riêng mình ở đâu? Trước đây, chúng tôi đang viết tất cả business logic của mình bên trong phương thức doFilter. Nếu bạn đi và kiểm tra bất kỳ Filter nào mà chúng tôi đã xác định, chúng tôi đang viết tất cả business logic của mình bên trong phương thức doFilter. Nhưng trong kịch bản của OncePerRequestFilter, câu chuyện sẽ khác. Bạn chỉ cần đảm bảo rằng bạn đang viết tất cả business logic của mình bên trong một phương thức có tên là **doFilterInternal**. Vì vậy, **doFilterInternal** này là một abstract method. Bạn có thể ghi đè lên điều này và viết tất cả logic của bạn. Trong nội bộ, OncePerRequestFilter của tôi sẽ cố gắng thực hiện việc gọi phương thức này chỉ một lần cho mỗi yêu cầu. Và bộ lọc này cũng có các loại phương pháp khác cực kỳ hữu ích. Giống như nếu bạn hỏi tôi một ví dụ, có một phương thức gọi là **shouldNotFilter**. Nếu bạn không muốn bộ lọc này được thực thi cho một số API web hoặc một số đường dẫn API còn lại, bạn luôn có thể xác định các chi tiết đó và thay vì trả về giá trị false, bạn có thể trả về giá trị true dựa trên các điều kiện mà bạn có. Nếu bạn trả về false, đây là hành vi mặc định, thì custom filter mà bạn sắp triển khai sẽ được gọi một lần cho mỗi yêu cầu. Vì vậy, các loại **phương pháp trung thực - fidelity method** này có sẵn bên trong OncePerRequestFilter này, rất hữu ích nếu bạn có loại tình huống lọc đặc biệt này.

Structure

OncePerRequestFilter

- OncePerRequestFilter()
- doFilter(ServletRequest, ServletResponse, FilterChain): void  $\uparrow$ Filter
- skipDispatch(HttpServletRequest): boolean
- isAsyncDispatch(HttpServletRequest): boolean
- isAsyncStarted(HttpServletRequest): boolean
- getAlreadyFilteredAttributeName(): String
- shouldNotFilter(HttpServletRequest): boolean
- shouldNotFilterAsyncDispatch(): boolean
- shouldNotFilterErrorDispatch(): boolean
- doFilterInternal(HttpServletRequest, HttpServletResponse, FilterChain): void
- doFilterNestedErrorDispatch(HttpServletRequest, HttpServletResponse, FilterChain): void

ALREADY\_FILTERED\_SUFFIX: String = ".FILTERED"



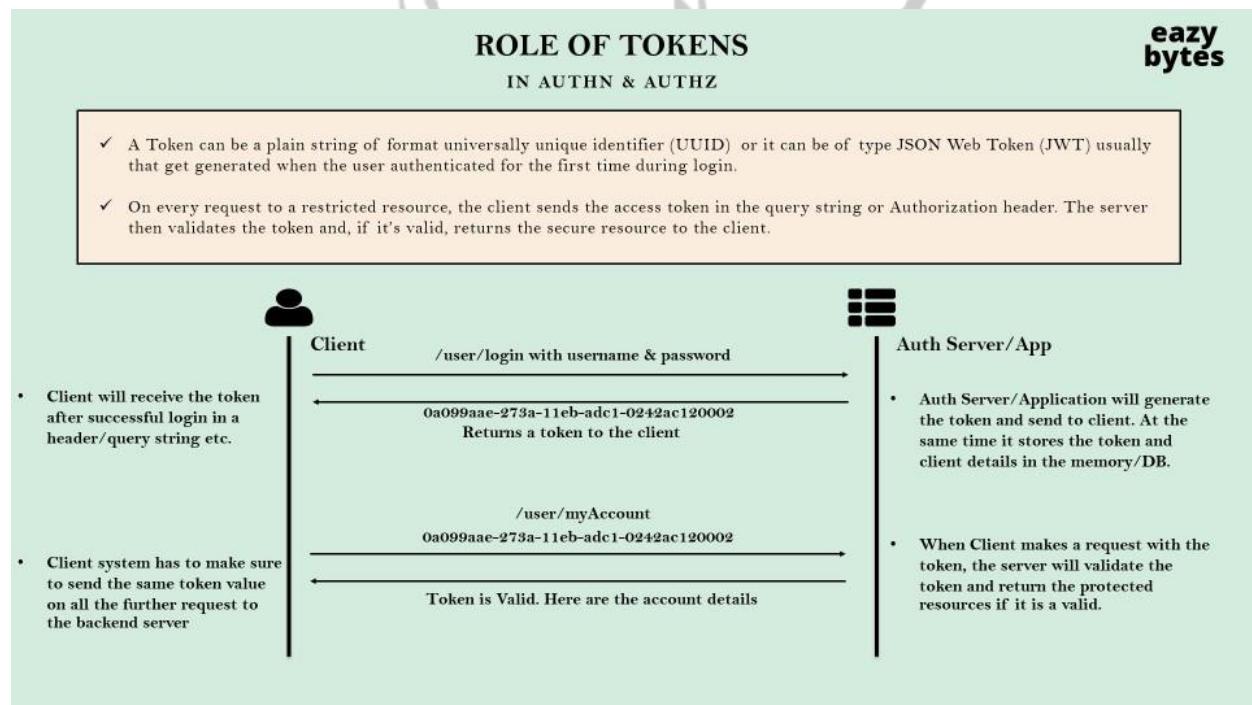
## Section 9: Token based Authentication using JSON Web Token (JWT)

### 1. Advantages of Token based Authentication

**Token** có thể là một chuỗi định dạng đơn giản có định dạng duy nhất trên toàn cầu hoặc nó có thể là loại JWT token. JWT, có nghĩa là JSON WEB Token. Và những Token này sẽ được tạo thường ngay khi quá trình xác thực user hoàn tất, ngay lần đầu tiên trong quá trình đăng nhập. Và một khi Token đó được tạo trong quá trình đăng nhập, Token tương tự có thể được các client application tận dụng để gửi tới backend system bất cứ khi nào chúng cố gắng truy cập tài nguyên được bảo vệ. Và nếu Token này được gửi bởi client application của tôi là hợp lệ thì backend server của tôi sẽ phản hồi bằng phản hồi thích hợp.

Trong bước đầu tiên, sẽ có một client application và sẽ có một backend application hoặc sẽ có một Authorization Server chịu trách nhiệm tạo Token. Ở bước đầu tiên, user của tôi sẽ cố gắng đăng nhập vào backend application của tôi bằng tên người dùng và mật khẩu của chính họ. Nếu các thông tin đăng nhập này hợp lệ, thì backend application hoặc authorization server của tôi sẽ tạo Token có bản chất ngẫu nhiên và Token tương tự sẽ được gửi lại cho user.

Bây giờ, trong bước tiếp theo, bất cứ khi nào khách hàng của tôi muốn truy cập một trong các API bảo mật như /user/myAccount, client application phải đảm bảo rằng nó đang gửi cùng một Token tới backend server.



Và lần này backend server của tôi đủ thông minh để hiểu vì người bạn này gửi Token, sử dụng Token này, nó phải authorization và nếu Token hợp lệ, nó phải gửi phản hồi thích hợp. Vì vậy, đây là trách nhiệm của backend server của tôi bất cứ khi nào nó nhận được Token hợp lệ. Vì Token của tôi hợp lệ nên nó sẽ phản hồi với chi tiết tài khoản của user của tôi.

Và đây là câu chuyện về những gì sẽ xảy ra bên trong bốn bước mà chúng ta đã thảo luận. Nếu bạn thấy ở đây, chúng tôi đang tận dụng Token bên trong authentication và authorization của mình. Và

tại sao tôi nên sử dụng Token, tôi không thể tận dụng thông tin đăng nhập của người dùng và gửi đi gửi lại cùng một thông tin đăng nhập cho mỗi yêu cầu? Giờ đây, chúng ta đang ở trong một thế giới tiên tiến, nơi có rất nhiều lỗ hổng bảo mật đang bị phơi bày hàng ngày. Đó là lý do tại sao chúng ta nên tận dụng Token. Hãy cố gắng hiểu tại sao chúng ta nên sử dụng Token và những lợi thế mà chúng ta nhận được bất cứ khi nào chúng ta tận dụng Token.

## ADVANTAGES OF TOKENS

eazy  
bytes

- ★ Token helps us not to share the credentials for every request. It is a security risk to send credentials over the network frequently.
- ★ Tokens can be invalidated during any suspicious activities without invalidating the user credentials.
- ★ Tokens can be created with a short life span.
- ★ Tokens can be used to store the user related information like roles/authorities etc.
- ★ Reusability - We can have many separate servers, running on multiple platforms and domains, reusing the same token for authenticating the user.
- ★ Stateless, easier to scale. The token contains all the information to identify the user, eliminating the need for the session state. If we use a load balancer, we can pass the user to any server, instead of being bound to the same server we logged in on.
- ★ We already used tokens in the previous sections in the form of CSRF and JSESSIONID tokens.
  - CSRF Token protected our application from CSRF attacks.
  - JSESSIONID is the default token generated by the Spring Security which helped us not to share the credentials to the backend every time.

**Lợi thế đầu tiên là**, bất cứ khi nào bạn sử dụng token, bạn phải chia sẻ thông tin đăng nhập thực tế của mình với backend application chỉ trong quá trình đăng nhập. Bằng cách sử dụng client application của mình, bạn có thể gọi hàng trăm API REST khác nhau trong khoảng thời gian 30 phút hoặc một giờ. Và trong tất cả các yêu cầu tiếp theo, bạn không phải chia sẻ thông tin đăng nhập thực tế của mình và bạn có thể làm việc với backend server bằng cách chia sẻ token. Vì vậy, đây là một trong những lợi thế chính và chúng tôi đang lưu thông tin đăng nhập của mình để không bị lộ một cách không cần thiết. Và backend application của tôi, bạn cũng không cần phải xác thực lại nhiều lần. Nếu có token truy cập hợp lệ, nó có thể xác thực xem token đó có hợp lệ hay không. Và theo đó, nó có thể cung cấp cho tôi một phản hồi.

**Và lợi thế thứ hai mà chúng tôi có là**, hãy nghĩ về một tình huống, một trong những tin tức, anh ta đã biết về các token có trong ứng dụng web của bạn. Và nếu chúng tôi biết rằng ai đó đã hack tất cả các token của chúng tôi thì chúng tôi chỉ cần vô hiệu hóa các token này. Trong khi đó, nếu bạn nghĩ kịch bản tương tự mà không có token, nếu tôi tiếp tục chia sẻ thông tin đăng nhập của mình với backend server thì nếu một trong những tin tức đánh cắp tất cả thông tin đăng nhập của tôi hoặc mạng, thì tùy chọn duy nhất mà chúng tôi có ở đây là, chúng tôi cần yêu cầu tất cả user của chúng tôi thay đổi mật khẩu hoặc email hoặc tên người dùng của họ, điều này sẽ mang lại nhiều tiếng xấu cho tổ chức của bạn. Đó là lý do tại sao token sẽ lưu trong loại tình huống bảo mật này.

Đồng thời, các token có thể được tạo với thời gian tồn tại rất ngắn. Hầu hết các trang web, họ tạo token có tuổi thọ một ngày hoặc một giờ. Vì vậy, dựa trên yêu cầu kinh doanh của bạn, bạn có thể

tạo token và bạn có thể xác định tuổi thọ cho token đó. Nếu ứng dụng của bạn là một ứng dụng blog đơn giản và không có gì để đánh cắp từ ứng dụng của bạn thì chắc chắn bạn có thể tạo token có giá trị trong một năm. Nhưng đối với các ứng dụng nhạy cảm khác như ứng dụng ngân hàng, thì chắc chắn họ sẽ tạo token truy cập có tuổi thọ ngắn. Và bất cứ khi nào người dùng của tôi đăng nhập lại, anh ấy sẽ nhận được token mới.

Và bằng cách sử dụng token, chúng tôi cũng có thể lưu trữ một số thông tin người dùng hoặc thông tin vai trò của user của tôi. Đôi khi, client application là các dịch vụ siêu nhỏ khác, bất kỳ ai đang làm việc với API REST của bạn, họ muốn biết thông tin chi tiết về user như địa chỉ email, vai trò và quyền hạn của anh ấy là gì, vì vậy loại thông tin như vậy chúng tôi có thể nhúng vào bên trong token. Hiện tại, bên trong JSESSIONID token, được tạo bởi Spring Security theo mặc định, nó không mang lại sự linh hoạt đó. Đó là lý do tại sao chúng ta sẽ thảo luận về các phương pháp nâng cao mà chúng ta có với sự trợ giúp của JWT token.

**Và lợi thế tiếp theo mà chúng tôi có ở đây, với các token, khả năng tái sử dụng.** Có nghĩa là, giả sử nghĩ về một Google. Bên trong Google có rất nhiều ứng dụng nội bộ như Gmail, Google Photos, Maps, Google Drive nên đây đều là những ứng dụng thuộc sở hữu của cùng một tổ chức. Nếu user của tôi đang cố điều hướng từ Gmail sang Maps hoặc Gmail sang Google Drive, nếu Google hỏi đi hỏi lại thông tin đăng nhập, thì điều đó sẽ không thân thiện với người dùng. Thay vào đó, những gì họ sẽ làm là, trong nội bộ, họ sẽ chuyển token từ Gmail sang Google Drive và token tương tự sẽ được Google Drive tận dụng và nếu Authorization Server đồng ý với token, thì nó sẽ nhận được một thông báo thích hợp. phản ứng. Vì vậy, tại đây, bạn có thể thấy chúng tôi có thể sử dụng lại token ở nhiều nơi và token là thứ sẽ giúp chúng tôi đạt được SSO, đó là Đăng nhập một lần. Nhiều lần trong tổ chức của bạn, bạn có thể đã thấy, bạn không cần phải đăng nhập nhiều lần nếu bạn đang cố gắng chuyển đổi các ứng dụng khác nhau trong cùng một tổ chức. Điều đó đang xảy ra như thế nào, với sự trợ giúp của các token.

Và những token này cũng giúp chúng tôi không trạng thái, đặc biệt là trong môi trường microservice. **Stateless - Không trạng thái** ở đây có nghĩa là, bên trong một số môi trường cụm nơi chúng tôi có nhiều phiên bản của cùng một ứng dụng web trong quá trình đăng nhập, yêu cầu có thể đã chuyển đến phiên bản một và tất cả các yêu cầu tiếp theo không phải chỉ chuyển đến chính phiên bản đó. Nó có thể chuyển sang phiên bản hai hoặc phiên bản ba hoặc phiên bản bốn hoặc bất kỳ phiên bản nào bên trong cụm. Và cách những người dùng khác sẽ biết về người dùng đã đăng nhập của bạn, với sự trợ giúp của token của bạn. Và những trường hợp này, họ không cần phải nhớ bất cứ điều gì về user trong các phiên. Vì vậy, đó là những gì không trạng thái ở đây, các phiên bản của bạn là các microservice.

Họ không phải lưu trữ bất kỳ thứ gì bên trong session của họ về user. Vì token của chúng tôi sẽ có thông tin của người dùng nên chúng tôi có thể ở trạng thái không trạng thái và với sự trợ giúp của token, chúng tôi có thể cố gắng hiểu thông tin chi tiết của user, chẳng hạn như vai trò, email hoặc quyền hạn của anh ấy. Chúng tôi đã sử dụng token bên trong ứng dụng web của mình. Hai token như vậy là CSRF token và JSESSIONID token.

## 2. Deep dive about JWT Tokens

JSESSIONID là token mặc định được tạo bởi Spring Security do các nhược điểm của nó. Chúng tôi cần tìm kiếm các phương pháp nâng cao như JWT token. Vì vậy, để triển khai JWT token bên trong ứng dụng web của chúng tôi, trước tiên, chúng tôi cần hiểu chúng. Vậy JWT token là gì? JWT có nghĩa là JSON Web Token bởi vì bên trong, những backend application này, chúng duy trì dữ liệu ở định dạng JSON. Và chúng được thiết kế để sử dụng cho các yêu cầu web, đặc biệt là bất cứ khi nào bạn đang cố gắng giao tiếp với backend application của mình từ client application giao diện người dùng dưới dạng JSON với sự trợ giúp của các dịch vụ REST. Và JWT là loại token được yêu thích và sử dụng phổ biến nhất do các tính năng và ưu điểm đặc biệt mà nó mang lại.

The slide has a light green background with a white header bar. The header bar contains the text 'JWT TOKENS' on the left and the 'eazy bytes' logo on the right. The main content area has a light pink background. It contains a list of three bullet points:

- ✓ JWT means JSON Web Token. It is a token implementation which will be in the JSON format and designed to use for the web requests.
- ✓ JWT is the most common and favorite token type that many systems use these days due to its special features and advantages.
- ✓ JWT tokens can be used both in the scenarios of Authorization/Authentication along with Information exchange which means you can share certain user related data in the token itself which will reduce the burden of maintaining such details in the sessions on the server side.

Below this list, there is a yellow box containing text and a sample token with its parts labeled.

A JWT token has 3 parts each separated by a period(.) Below is a sample JWT token,

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9IiIwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c

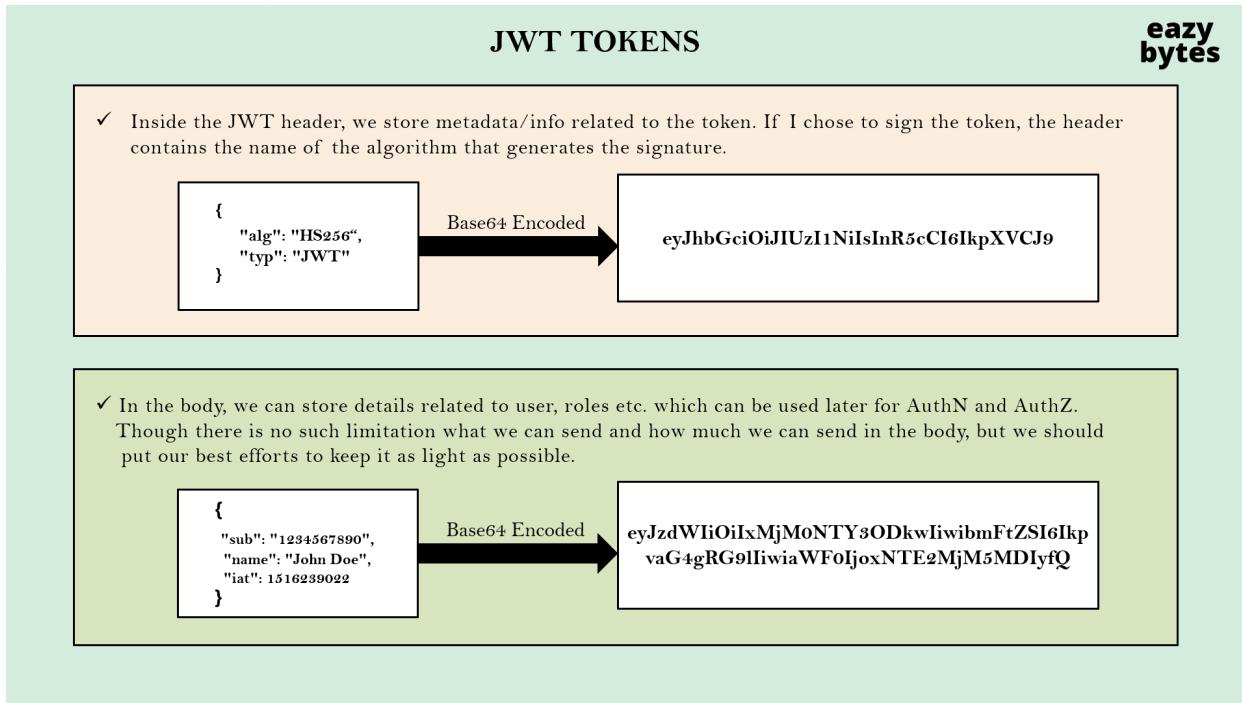
1. Header  
2. Payload  
3. Signature (Optional)

Và giống như bất kỳ token nào, JWT token cũng có thể được sử dụng trong các tình huống authentication và authorization. Nhưng nếu bạn hỏi tôi điểm đặc biệt hay ưu điểm của JWT token là gì, thì câu trả lời của tôi là những JWT token này, chúng cũng giúp bạn lưu trữ và chia sẻ dữ liệu liên quan đến người dùng bên trong chính backend application đó. Điều này sẽ giảm gánh nặng trong việc duy trì thông tin chi tiết của người dùng bên trong các session ở phía máy chủ và điều này sẽ cực kỳ hữu ích trong môi trường microservice nơi nhiều dịch vụ đang cố gắng giao tiếp với nhau và chúng không cần phải ghi nhớ bất cứ điều gì. Chúng có thể hoàn toàn không trạng thái vì bản thân JWT token của tôi sẽ có tất cả các chi tiết mà tôi cần liên quan đến người dùng.

Để hiểu cách JWT token này sẽ giúp chúng tôi lưu trữ và chia sẻ thông tin chi tiết của người dùng, trước tiên, chúng tôi cần hiểu định dạng và cấu trúc của JWT token. Thông thường, JWT token sẽ có ba phần, được phân tách bằng dấu chấm. Nhưng nếu bạn nhìn vào backend application, có hai dấu chấm bên trong backend application này. Bất cứ điều gì bạn thấy được đánh dấu bằng màu vàng, phần đó, chúng tôi gọi nó là phần header của JWT token. Và chúng tôi có một văn bản được tô sáng màu xanh lá cây và phần này chúng tôi gọi nó là payload hoặc body của backend application. Và cuối cùng, chúng tôi cũng có signature, về bản chất là tùy chọn. Vì vậy, tổng cộng, có hai phần bắt buộc bên trong JWT

token. Một là header và cái thứ hai là payload và cái thứ ba là signature là một phần tùy chọn bên trong JWT token.

Bên trong JWT header, chúng tôi lưu trữ siêu dữ liệu hoặc thông tin liên quan đến token. Thông thường bên trong header này, chúng tôi lưu trữ một số thông tin như thuật toán là gì, loại token đó là gì, định dạng token mà tôi đã sử dụng trong khi tạo JWT token này là gì. Vì vậy, header sẽ chỉ có thông tin siêu dữ liệu và tất cả thông tin bên trong JWT token sẽ không được gửi ở định dạng văn bản thuần túy, như bạn có thể thấy ở đây ở phía bên trái.



Bên trong header của tôi, siêu dữ liệu mà tôi đang cố lưu trữ là thuật toán, là HS256 và loại là JWT. Thay vì gửi trực tiếp thông tin này cho khách hàng bên trong JWT token, điều mà JWT token khuyên chúng tôi nên làm là chúng tôi cần mã hóa base64 giá trị siêu dữ liệu của bạn để nó sẽ chuyển đổi, như bạn có thể thấy ở bên tay phải hộp bên.

Tương tự như vậy, nếu bạn đến với phần thứ hai của JWT token, đó là body hoặc payload, bên trong phần này, chúng tôi có thể lưu trữ tất cả các chi tiết người dùng mà chúng tôi muốn, chẳng hạn như tên của anh ấy là gì? Email của anh ấy là gì? Role của anh ấy là gì? Khi token được phát hành, thời gian hết hạn của token là gì? Ai đã phát hành token? Ai đã ký token. Vì vậy, tất cả các loại thông tin như vậy chúng tôi có thể lưu trữ bên trong phần thân hoặc trọng tải của JWT token. Và ở đây, thay vì gửi thông tin này ở định dạng văn bản thuần túy, chúng tôi sẽ chuyển đổi văn bản thuần túy thành giá trị được mã hóa Base64. Khi thao tác đăng nhập thành công, backend application của tôi, nó có thể tạo JWT token và ở đây JWT token sẽ có header và payload là một phần bắt buộc. Và cùng một token mà tôi sẽ mong đợi từ user và nếu token giống như những gì tôi đã tạo ban đầu, thì tất cả các hoạt động tiếp theo sẽ thành công. Và nếu có bất kỳ sự không phù hợp nào giữa những gì tôi tạo ban đầu và những gì tôi đang nhận được ngay bây giờ, thì chắc chắn backend server của tôi sẽ báo lỗi.

Backend server của bạn, cách tiếp cận cơ bản nhất mà nó có thể tuân theo là lưu trữ JWT token này bên trong cơ sở dữ liệu hoặc bên trong bộ đệm bất cứ khi nào nó được tạo lần đầu tiên. Và trong tất cả các yêu cầu tiếp theo đến từ client, bất cứ khi nào nó được sử dụng một JWT token cụ thể, nó sẽ xác thực nếu JWT token, nó sẽ nhận được giống như token mà nó đã được lưu trữ bên trong cơ sở dữ liệu hoặc bên trong bộ đệm. Vì vậy, đó là cách tiếp cận cơ bản nhất. Nhưng với sự trợ giúp của JWT token signature, chúng tôi thậm chí không phải lưu trữ nó ở bất cứ đâu. Những gì backend server của tôi có thể làm là bất cứ khi nào nó tạo JWT token với các giá trị bên trong header và payload, nó có thể ký điện tử token đó để trong tương lai, nếu ai đó cố gắng giả mạo token, nó có thể dễ dàng phát hiện ra điều đó. Vì vậy, hãy cố gắng hiểu cách signature của JWT token sẽ giúp chúng tôi xác thực JWT token nhiều lần mà không cần lưu trữ bên trong hệ thống lưu trữ hoặc bên trong bộ đệm.

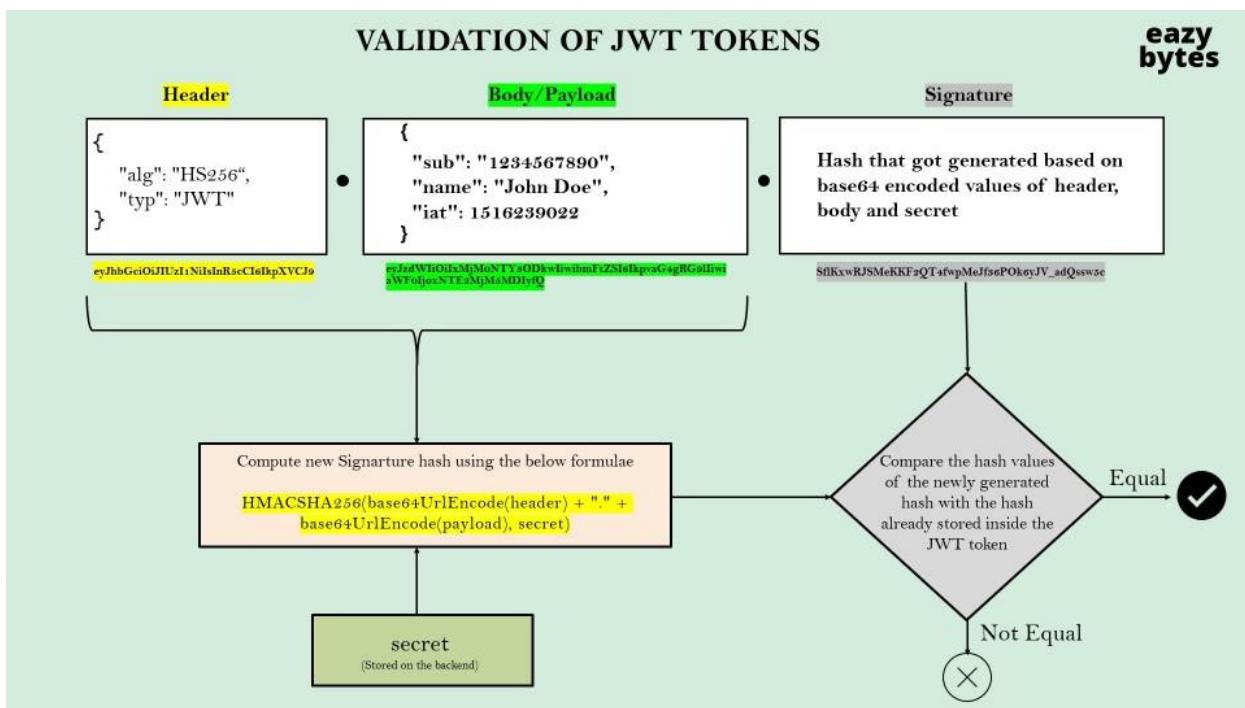
Nhưng nếu bạn đang chia sẻ JWT Token này, được tạo trong quá trình đăng nhập vào các client application qua internet, mã này sẽ được user và khách hàng của bạn sử dụng và cùng một token sẽ được di chuyển bên trong mạng internet, thì chắc chắn bạn cần đảm bảo rằng không ai thay đổi giá trị header và nội dung như quyền hạn và tên người dùng.

Giống như, hãy nghĩ về kịch bản một trong những user có vai trò là “user”. Anh ấy đã đăng nhập vào ứng dụng và nhận được JWT Token. Và nếu anh ấy thông minh và nếu anh ấy biết tất cả những điều này về cách thức hoạt động của JWT Token này, nếu anh ấy là một hacker, thì anh ấy có thể chỉ cần thay đổi thông tin vai trò bên trong token từ người dùng thành quản trị viên. Điều đó có nghĩa là anh ấy có thể nâng cao thông tin về vai trò của mình và sau khi nâng cao thông tin về vai trò đó, anh ấy có thể chuyển đổi lại văn bản thuần túy đó sang bộ mã hóa base64. Bởi vì encoding và decoding là một quá trình đơn giản nên nó không liên quan đến bất kỳ khóa keys hay hashing nào. Vì nó cực kỳ đơn giản, một số hacker có quyền truy cập vào JWT Token, họ luôn có thể giả mạo các token này. Vậy làm cách nào chúng tôi có thể đảm bảo rằng không ai giả mạo JWT Token hoặc mạng hoặc từ các client application hoặc bởi user của bạn? Vì vậy, chúng tôi có phần signature bên trong JWT Token. Backend của tôi phải làm là bất cứ khi nào nó cố gắng tạo JWT Token mới, sau khi xác thực thành công, nó phải ký điện tử token.

Nó sẽ làm điều này như thế nào, nó sẽ nhận được sự trợ giúp từ một trong các thuật toán, chẳng hạn như thuật toán SHA-256. Vì vậy, thuật toán này sẽ chuyển thông tin về header, payload của bạn, cùng với signature mà chỉ backend của bạn mới biết. Vì vậy, nếu bạn có thể thấy ở đây, đầu vào của thuật toán là giá trị base64 encoded của header, theo sau là dấu chấm và sau đó là giá trị được base64 encoded của payload hoặc body của bạn và tham số cuối cùng là giá trị secret. Vì vậy, giá trị secret này sẽ chỉ được biết bởi backend của bạn nơi nó đang phát hành JWT Token. Dầu ra từ thuật toán này, dựa trên dữ liệu mà bạn chuyển, là chuỗi băm ngẫu nhiên này. Và chuỗi băm này bạn sẽ gửi bên trong phần signature của JWT Token của mình. Và với ba phần này như header, payload và signature, bạn có thể gửi lại cho client và client phải gửi cùng một token. Và vì một số lý do, nếu bất kỳ user hoặc client application nào của bạn hoặc một trong những hacker giả mạo token của bạn thì họ có thể giả mạo header hoặc họ có thể giả mạo payload hoặc họ có thể giả mạo giá trị signature, bất kể họ là gì, backend của bạn sẽ dễ dàng biết về nó và nó sẽ làm mất hiệu lực phiên đó.

## VALIDATION OF JWT TOKENS

eazy  
bytes



Chúng tôi có phần header và chúng tôi có phần payload. Và từ payload header và body, cùng với giá trị secret, backend của tôi có thể đã tạo một phần signature và phần signature này là một chuỗi băm. Bên trong nó chứa một giá trị băm và cùng một token sẽ gửi lại cho client application và nghĩ về một kịch bản, client application sẽ gửi lại cho chúng tôi trong yêu cầu tiếp theo bên trong backend. Nay giờ, khi nó nhận được JWT token đó từ client application, nó sẽ cố gắng tính toán hàm băm signature mới bằng cách sử dụng cùng một công thức, cùng một thuật toán, cùng một bộ mã hóa URL base64 của header, theo sau là dấu chấm, tiếp theo là bộ mã hóa URL base64 của payload và cùng với secret tương tự mà nó đã sử dụng ban đầu. Vì vậy, đầu ra của tính toán này sẽ lại là một hàm băm signature mới. Và những gì backend của tôi sẽ làm là, nó sẽ cố gắng so sánh các giá trị băm của hàm băm mới được tạo với giá trị băm đã được tạo sẵn có sẵn bên trong chính JWT token đó.

Hãy nghĩ về một kịch bản, nếu ai đó giả mạo header hoặc payload, thì chắc chắn khi tôi cố gắng tính toán một signature mới, giá trị băm sẽ khác với giá trị đã được lưu trữ bên trong phần signature. Và trong các tình huống mà ai đó trực tiếp giả mạo signature, thì rõ ràng trong tình huống đó, giá trị băm sẽ không bao giờ khớp. Vì vậy, đây là một cách tiếp cận rất tốt mà không cần lưu trữ JWT token bên trong hệ thống lưu trữ, chúng tôi có thể xác thực JWT token, nếu chúng bị giả mạo hay không. Và điều này sẽ mang lại sự tự tin 100% cho tất cả mọi người như client application và backend, rằng không ai sẽ giả mạo JWT token của chúng tôi. Nếu bất kỳ ai cố gắng giả mạo, thì toàn bộ phiên sẽ bị vô hiệu.

### 3. Configuring filters to generate the JWT tokens

Để định cấu hình tạo mã thông báo JWT trong Spring Boot với Spring Security, bạn cần làm theo các bước sau:

Bước 1: Thêm phụ thuộc Đảm bảo rằng bạn có các phụ thuộc cần thiết trong tệp dự án của mình: `pom.xml`

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.5</version>
</dependency>

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.5</version>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.11.5</version>
    <scope>runtime</scope>
</dependency>
```

Bước 2: Tạo Filter `JWTTokenGeneratorFilter`

- Nhiệm vụ của filter này là sinh và gắn JWT (JSON Web Token) vào header của response để xác thực và ủy quyền người dùng.

```
public class JWTTokenGeneratorFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
    HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {
        Authentication authentication =
        SecurityContextHolder.getContext().getAuthentication();
        if (null != authentication) {
            SecretKey key =
        Keys.hmacShaKeyFor(SecurityConstants.JWT_KEY.getBytes(StandardCharsets.UTF_8));
            String jwt = Jwts.builder().setIssuer("Eazy
        Bank").setSubject("JWT Token")
                .claim("username", authentication.getName())
                .claim("authorities",
```

```

        populateAuthorities(authentication.getAuthorities())
            .setIssuedAt(new Date())
            .setExpiration(new Date((new Date()).getTime() +
30000000))
            .signWith(key).compact();
        response.setHeader(SecurityConstants.JWT_HEADER, jwt);
    }

    filterChain.doFilter(request, response);
}

@Override
protected boolean shouldNotFilter(HttpServletRequest request) {
    return !request.getServletPath().equals("/user");
}

private String populateAuthorities(Collection<? extends GrantedAuthority>
collection) {
    Set<String> authoritiesSet = new HashSet<>();
    for (GrantedAuthority authority : collection) {
        authoritiesSet.add(authority.getAuthority());
    }
    return String.join(", ", authoritiesSet);
}

}

```

Trong SecurityConstants để lưu các giá trị **KEY** và **HEADER**:

```

public interface SecurityConstants {

    public static final String JWT_KEY = "jxgEQeXHuPq8VdbYFNkANDudQ53YUn4";
    public static final String JWT_HEADER = "Authorization";
}

```

Ở đây chúng tôi chỉ đề cập trực tiếp bên trong constants file, nhưng trong môi trường product bạn nên yêu cầu nhóm DevOps của mình đưa giá trị này dưới dạng biến môi trường bằng các công cụ CICD như GitHub hoặc Jenkins hoặc bạn cũng có thể cấu hình giá trị này như một biến môi trường bên trong product sercer của bạn và có thể truy cập biến đó từ code của bạn.

Lớp **JWTTokenGeneratorFilter** kế thừa từ **OncePerRequestFilter**, một lớp cơ bản trong Spring Security để đảm bảo filter chỉ được thực thi một lần cho mỗi request.

Phương thức **doFilterInternal** được ghi đè (**@Override**) từ lớp cha. Nó được gọi khi một request đi qua filter này.

Authentication **authentication** = **SecurityContextHolder.getContext().getAuthentication()**; lấy đối tượng Authentication từ **SecurityContextHolder**, đại diện cho thông tin xác thực của người dùng.

Kiểm tra xem **authentication** có khác **null** hay không. Nếu khác **null**, tức là người dùng đã được xác thực, tiếp tục xử lý.

Tạo một `SecretKey` từ `SecurityConstants.JWT_KEY` để sử dụng cho việc ký và xác minh chữ ký của JWT.

Sử dụng `Jwts.builder()` để tạo JWT. Trong ví dụ này, JWT chứa các thông tin sau:

- Issuer (người phát hành): "Eazy Bank"
  - Subject (chủ đề): "JWT Token"
  - Claims (thông tin): "username" là tên đăng nhập của người dùng và "authorities" là danh sách các quyền được lấy từ `Authentication`.
  - Issued At (thời điểm phát hành): là thời điểm hiện tại.
  - Expiration (thời gian hết hạn): là thời điểm hiện tại cộng thêm 30000000 milliseconds (tương đương khoảng 5 phút).
  - Sử dụng `key` để ký JWT.
1. Thiết lập header của response với tên là `SecurityConstants.JWT_HEADER` và giá trị là JWT vừa tạo.
  2. `filterChain.doFilter(request, response)`; chuyển tiếp request và response đến filter tiếp theo trong chuỗi.

Phương thức `shouldNotFilter` là một phương thức ghi đè khác. Nó kiểm tra đường dẫn (servlet path) của request và trả về kết quả phản chiếu của việc so sánh với `"/user"`. Điều này đảm bảo rằng filter này sẽ không được áp dụng cho request có đường dẫn là `"/user"`.

Phương thức `populateAuthorities` được sử dụng để chuyển đổi danh sách quyền (`collection`) thành một chuỗi dạng "authority1,authority2,authority3" để đưa vào claim `"authorities"` của JWT. Vì vậy, ở đây yêu cầu của tôi là bộ lọc tạo mã thông báo JWT này chỉ được thực thi trong quá trình đăng nhập vì tôi không muốn mã thông báo được tạo lặp đi lặp lại cho tất cả các yêu cầu tiếp theo. Tôi muốn điều này chỉ xảy ra trong quá trình đăng nhập.

Bước 3: Và sau khi tạo bộ lọc này, chúng ta cần cấu hình hoặc đưa filter này vào `ProjectSecurityConfig`. Đối với chuỗi, chúng ta sẽ gọi một phương thức `addFilterAfter`. Lý do tại sao tôi muốn gọi phương thức này là vì chúng tôi muốn bộ lọc của mình được thực thi sau khi xác thực thành công.

```
// ProjectSecurityConfig
    .addFilterAfter(new AuthoritiesLoggingAfterFilter(), BasicAuthenticationFilter.class)
    .addFilterAfter(new JWTTOKENGeneratorFilter(), BasicAuthenticationFilter.class)
```

#### 4. Configuring filters to validate JWT tokens

Tạo Filter `JWTTOKENValidatorFilter` Thực hiện một số xử lý để xác thực và ủy quyền người dùng sử dụng JSON Web Token (JWT) trong một ứng dụng web.

```
public class JWTTOKENValidatorFilter extends OncePerRequestFilter {
    @Override
    protected void doFilterInternal(HttpServletRequest request,
    HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {
```

```

String jwt = request.getHeader(SecurityConstants.JWT_HEADER);
if (null != jwt) {
    try {
        SecretKey key = Keys.hmacShaKeyFor(
            SecurityConstants.JWT_KEY.getBytes(StandardCharsets.UTF_8));
        Claims claims = Jwts.parserBuilder()
            .setSigningKey(key)
            .build()
            .parseClaimsJws(jwt)
            .getBody();
        String username = String.valueOf(claims.get("username"));
        String authorities = (String) claims.get("authorities");
        Authentication auth = new
        UsernamePasswordAuthenticationToken(username, null,
            AuthorityUtils.commaSeparatedStringToAuthorityList(authorities));
        SecurityContextHolder.getContext().setAuthentication(auth);
    } catch (Exception e) {
        throw new BadCredentialsException("Invalid Token received!");
    }
}
filterChain.doFilter(request, response);
}

@Override
protected boolean shouldNotFilter(HttpServletRequest request) {
    return request.getServletPath().equals("/user");
}
}

```

1. Phương thức `doFilterInternal` được gọi khi một request đi qua filter này. Nó nhận vào các tham số `HttpServletRequest`, `HttpServletResponse` và `FilterChain`.
2. Dòng `String jwt = request.getHeader(SecurityConstants.JWT_HEADER)`; lấy giá trị của header có tên `SecurityConstants.JWT_HEADER` từ request, trong đó có chứa JWT.
3. Kiểm tra xem `jwt` có khác `null` hay không. Nếu không phải `null`, tức là JWT tồn tại trong header, tiếp tục xử lý.
4. Trong khối `try`, đoạn mã này sẽ giải mã JWT bằng cách sử dụng `Jwts.parserBuilder()` và cung cấp khóa bí mật (`SecretKey`) để xác minh chữ ký của JWT.
5. `Claims claims = Jwts.parserBuilder()...` lấy ra các thông tin (claims) được đính kèm trong JWT (ví dụ: `username`, `authorities`).
6. Từ `claims`, trích xuất giá trị của `"username"` và `"authorities"` để sử dụng cho việc xác thực và ủy quyền.
7. Tạo một đối tượng `Authentication` với thông tin vừa lấy được và đặt nó vào `SecurityContextHolder` để xác thực người dùng.
8. Nếu xảy ra lỗi trong quá trình giải mã hoặc xác thực, ngoại lệ `BadCredentialsException` sẽ được ném.
9. Sau khi hoàn thành xử lý, `filterChain.doFilter(request, response)`; chuyển request và response đến filter tiếp theo trong chuỗi.

Phương thức **shouldNotFilter** là một phương thức khác được ghi đè. Nó kiểm tra xem request có đường dẫn (servlet path) là "/user" hay không. Nếu đúng, filter này sẽ không được áp dụng (không thực hiện **doFilterInternal**) và chuyển trực tiếp đến filter tiếp theo trong chuỗi.

Bước 3: Và sau khi tạo bộ lọc này, chúng ta cần cấu hình hoặc đưa filter này vào ProjectSecurityConfig. Đối với chuỗi, chúng ta sẽ gọi một phương thức **addFilterAfter**. Với cấu hình này, những gì chúng tôi đang nói với Spring Security là custom filter của tôi là JWT token validation filter phải được thực thi trước basic authentication filter, điều đó có nghĩa là custom filter của tôi sẽ được thực thi trước khi authentication vào Spring Security.

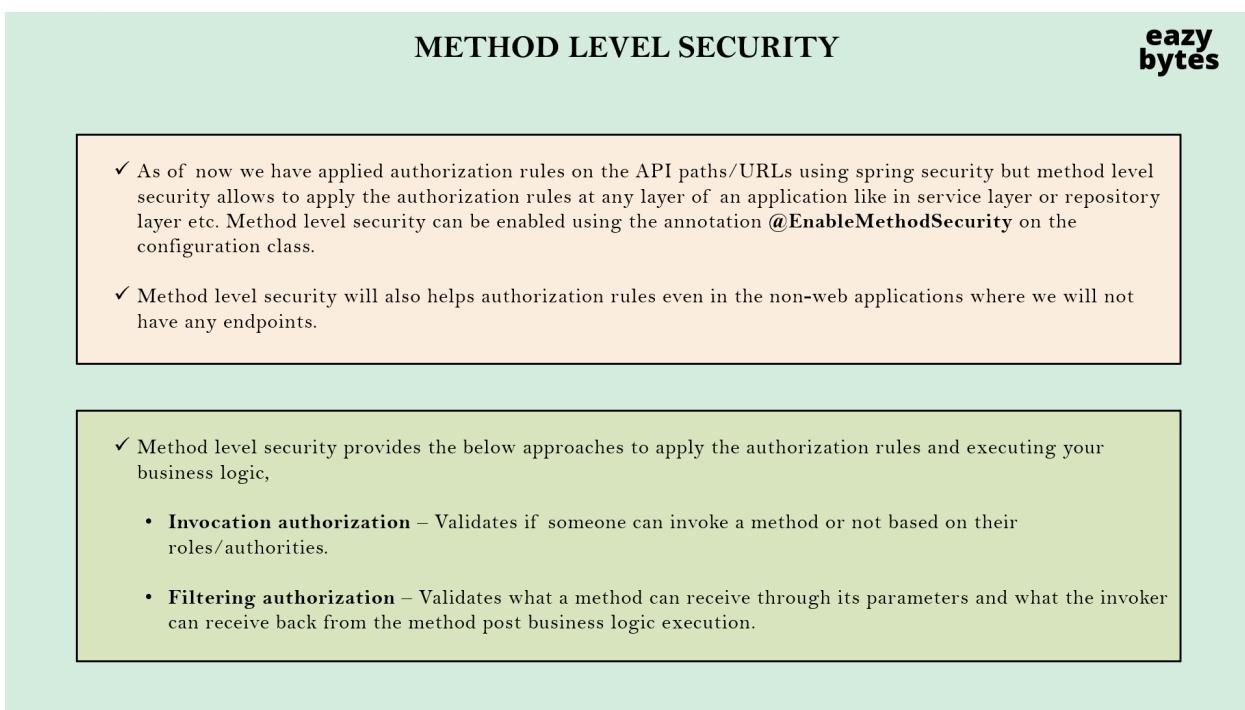
```
// ProjectSecurityConfig
    .addFilterAfter(new AuthoritiesLoggingAfterFilter(), BasicAuthenticationFilter.class)
    .addFilterAfter(new JWTTOKENGeneratorFilter(), BasicAuthenticationFilter.class)
    .addFilterBefore(new JWTTOKENValidatorFilter(), BasicAuthenticationFilter.class)
```



## Section 10: Method Level Security

### 1. Introduction to method level security in Spring Security

Bên trong Java, mọi thứ chúng tôi viết đều có sự trợ giúp của các method. Tất cả logic nghiệp vụ của chúng tôi nằm trong các phương thức Java. Đó là lý do tại sao với sự trợ giúp của **method-level security - bảo mật cấp phương thức** này có sẵn bên trong Spring Security, chúng tôi có thể thực thi các quy tắc authorization trên các phương thức Java của bạn bất kể phương thức của bạn có hiện diện bên trong lớp controller layer, service layer hoặc repository layer hay không. Nhưng theo mặc định, method-level security này không được bật. Vì vậy, để kích hoạt tính năng method-level security này bên trong ứng dụng web của bạn, nơi bạn có Spring Security làm phụ thuộc, bạn phải đảm bảo rằng bạn đang đề cập đến annotation `EnableMethodSecurity`.



The slide has a light green background. At the top center, it says 'METHOD LEVEL SECURITY'. In the top right corner, there is a logo for 'eazy bytes'. The slide contains two main sections of text, each preceded by a checkmark.

**Section 1:**

- ✓ As of now we have applied authorization rules on the API paths/URLs using spring security but method level security allows to apply the authorization rules at any layer of an application like in service layer or repository layer etc. Method level security can be enabled using the annotation `@EnableMethodSecurity` on the configuration class.
- ✓ Method level security will also helps authorization rules even in the non-web applications where we will not have any endpoints.

**Section 2:**

- ✓ Method level security provides the below approaches to apply the authorization rules and executing your business logic,
  - **Invocation authorization** – Validates if someone can invoke a method or not based on their roles/authorities.
  - **Filtering authorization** – Validates what a method can receive through its parameters and what the invoker can receive back from the method post business logic execution.

Vì vậy, annotation này, bạn cần đặt lên Spring Boot application main class hoặc bất kỳ configuration class. Và với sự trợ giúp của method-level security, chúng tôi cũng có thể thực thi các quy tắc authorization ngay cả trong các ứng dụng không phải web.

Kịch bản đầu tiên mà bạn có thể xem xét bảo mật ở mức phương thức là invocation authorization - authorization yêu cầu. Bất cứ khi nào bạn đang sử dụng method-level security trên bất kỳ phương thức Java nào, để gọi phương thức cụ thể đó, người dùng đã đăng nhập hoặc user phải quyền hạn hoặc vai trò. Nếu không, họ sẽ không thể gọi các phương thức của bạn. Vì vậy, điều này sẽ luôn hoạt động như một biện pháp bảo mật hai bên trong các ứng dụng web của bạn.

Ngoài việc thực thi authorization và authentication trên các đường dẫn API và URL của bạn, bạn luôn có thể coi các method-level security này là cấp bảo mật thứ hai. Với hai cấp độ bảo mật, ứng dụng của bạn sẽ trở nên an toàn hơn và không ai có thể hack được ứng dụng của bạn. Và kịch bản thứ hai mà chúng ta có thể sử dụng method-level security là filtering authorization. Như bạn đã biết, các phương

thức Java của chúng tôi, chúng chấp nhận rất nhiều dữ liệu đầu vào. Và sau khi xử lý đầu vào và thực thi logic nghiệp vụ, các phương thức Java của chúng tôi cũng đã ghi rất nhiều dữ liệu. Với sự trợ giúp của filtering authorization, bạn có thể authentication loại dữ liệu nào bạn muốn chấp nhận và loại dữ liệu nào bạn muốn trả lại cho user dựa trên các tiêu chí lọc khác nhau hoặc dựa trên quyền hạn hoặc quy tắc của họ. Vì vậy, đây là hai tình huống mà bạn có thể tận dụng method-level security bên trong ấn phẩm web của mình. Và ở đây, chỉ đề cập đến annotation `EnableMethodSecurity` là chưa đủ. Vì vậy, đó là cấu hình đầu tiên mà bạn cần thực hiện để kích hoạt tính năng này bên trong web application của mình. Sau khi bạn thực hiện những thay đổi này, như bạn có thể thấy, có rất nhiều annotation có sẵn để thực thi method-level security bên trong Spring Security.

**METHOD LEVEL SECURITY**

**eazy  
bytes**

- ✓ Spring security will use the aspects from the AOP module and have the interceptors in between the method invocation to apply the authorization rules configured.
- ✓ Method level security offers below 3 different styles for configuring the authorization rules on top of the methods,
  - The `prePostEnabled` property enables Spring Security `@PreAuthorize` & `@PostAuthorize` annotations
  - The `securedEnabled` property enables `@Secured` annotation
  - The `jsr250Enabled` property enables `@RoleAllowed` annotation

```
@Configuration  
@EnableMethodSecurity(prePostEnabled = true, securedEnabled = true, jsr250Enabled = true)  
public class ProjectSecurityConfig {  
    ...  
}
```

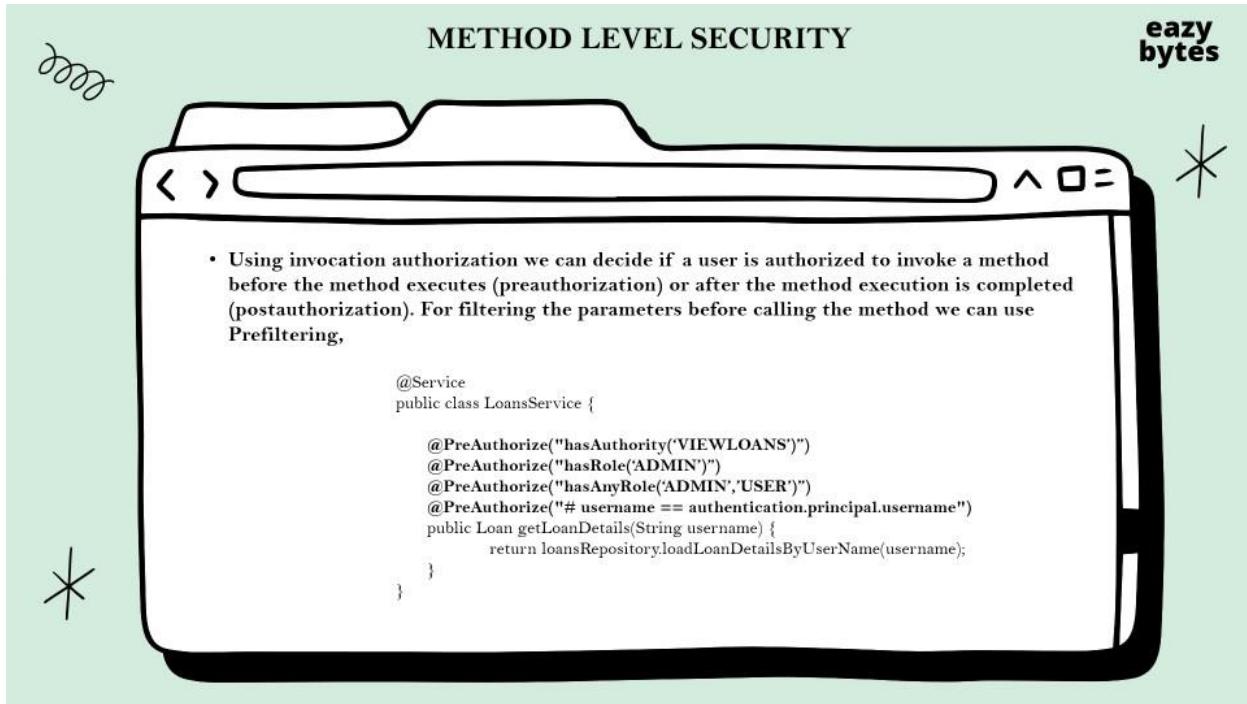
- ✓ `@Secured` and `@RoleAllowed` are less powerful compared to `@PreAuthorize` and `@PostAuthorize`

Tập hợp annotation đầu tiên PreAuthorize và PostAuthorize. Bất cứ khi nào bạn muốn tận dụng các annotation PreAuthorize và PostAuthorize này, bạn cũng nên kích hoạt chúng một cách cụ thể bằng cách đề cập đến một tham số có tên `prePostEnabled` là đúng. Và rất tương tự, chúng tôi có các loại annotation khác như ở mức độ Security và ở mức độ Role được phép. Vì vậy, đây là các annotation khác nhau mà chúng ta có thể tận dụng để thực thi bảo mật cấp phương thức. Nhưng nếu bạn hỏi tôi sự khác biệt giữa tất cả các annotation này là gì, thì tôi sẽ nói rằng những annotation này đến từ các thông số kỹ thuật hoặc thư viện khác nhau. Đó là lý do tại sao bất cứ khi nào bạn muốn tận dụng các annotation này, bạn phải bật chúng một cách cụ thể bằng cách chuyển các tham số này như `prePostEnabled`, `secureEnabled` hoặc `jsr250Enabled` thành `true` so với (không rõ ràng) ở annotation `EnableMethodSecurity`.

Tôi luôn khuyên bạn nên tận dụng annotation PreAuthorize và PostAuthorize vì chúng rất mạnh và chúng cho phép chúng tôi tận dụng Spring Expression Language bất cứ khi nào chúng ta sử dụng các annotation này. Cùng với đó, có nhiều khả năng của các quy tắc ủy quyền mà chúng ta có thể định cấu hình với sự trợ giúp của annotation PreAuthorize và PostAuthorize. Trong khi ở Secured và ở `RoleAllowed` kém hiệu quả hơn.

## 2. Details about method invocation authorization in method level security

Chúng ta sẽ sử dụng hai annotation. Cái đầu tiên là annotation `@PreAuthorize` và cái thứ hai là annotation `@PostAuthorize`. Annotation đầu tiên mà chúng ta có thể sử dụng là annotation `@PreAuthorized`. Bất cứ khi nào tôi sử dụng annotation này và nếu tôi xác định một số quy tắc authorization hoặc bất kỳ yêu cầu bảo mật nào khác, thì Spring Security framework sẽ đảm bảo rằng phương thức sẽ chỉ được gọi nếu các yêu cầu bảo mật của tôi được đáp ứng dựa trên thông tin chi tiết về user.



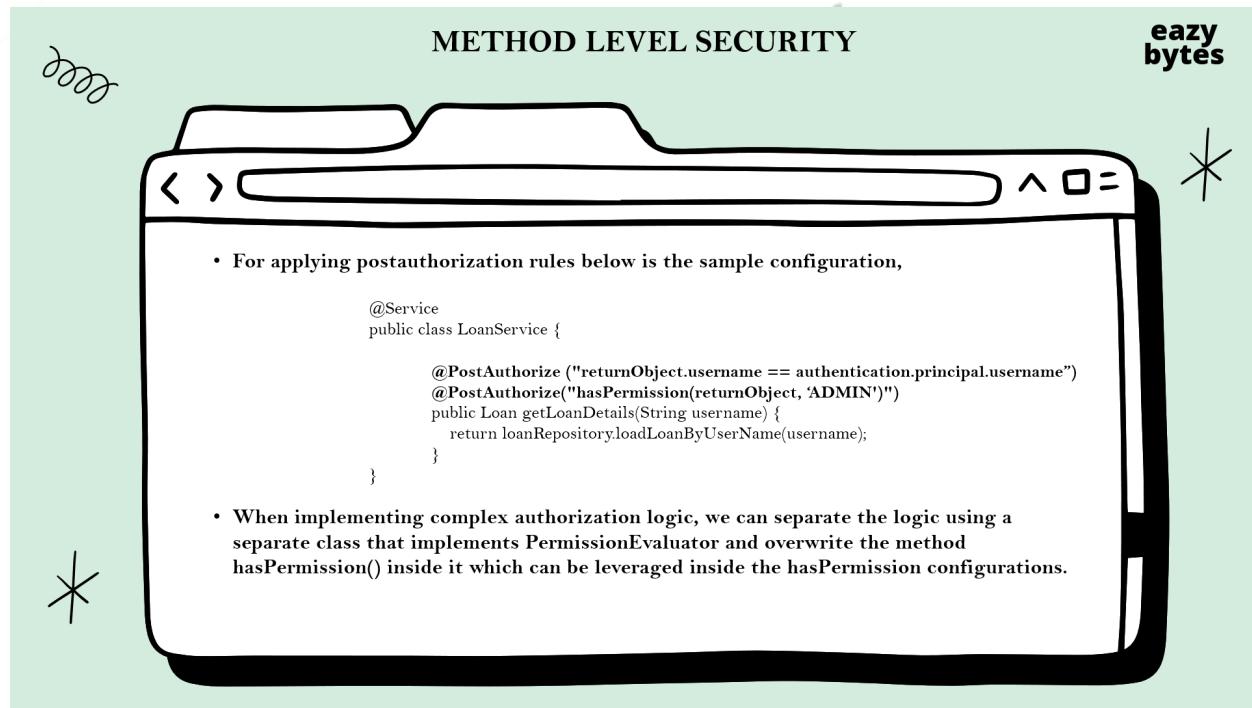
Giống như bạn có thể thấy ở đây, tôi đã đề cập đến nhiều cách tiếp cận khác nhau về cách chúng ta có thể sử dụng annotation `@PreAuthorized`. Vì vậy, có một phương thức Java được gọi là `getLoanDetails` và phương thức này sẽ chấp nhận `username` làm đầu vào và dựa trên `username`, nó sẽ tải tất cả các chi tiết khoản vay cho người dùng cụ thể đó và ghi lại các chi tiết khoản vay đó vào UI application. Nhưng thay vì thực hiện việc gọi phương thức Java này khá đơn giản và dễ dàng, chúng ta có thể kết thúc một số quy tắc authorization như bên trong annotation `PreAuthorized` của tôi, tôi có thể sử dụng phương thức **hasAuthority**, **hasRole** để kiểm tra xem người dùng đã đăng nhập của tôi có quyền hạn như vậy hay không, v.v. vai trò. Và rất tương tự, chúng ta cũng có thể sử dụng **hasAnyRole** hoặc **hasAnyAuthority**.

Và ngoài việc tận dụng các phương thức liên quan đến **hasAuthority**, **hasRole** này. Chúng tôi cũng có thể sử dụng Spring Expression Languages để xác định các yêu cầu bảo mật. Vì vậy, đây là lợi thế mà các annotation `PreAuthorize` và `PostAuthorize` này mang lại cho các nhà phát triển so với các annotation khác như `@Secured` và `@Role`. Bạn có thể thấy ở đây cấu hình cuối cùng mà tôi đã thực hiện với sự trợ giúp của Spring Expression Language.

Với Spring Expression Language này, những gì tôi đang làm là tôi chỉ đang cố lấy tham số đầu vào là tên người dùng và đảm bảo liệu tên người dùng đã đăng nhập và đầu vào nhận được cho tên người dùng phương thức này có giống nhau hay không. Vì vậy, làm cách nào tôi biết tên người dùng đã

đăng nhập là gì? Bằng cách nhận trợ giúp từ Spring Security và xem xét Authentication object có sẵn bên trong Spring Security. Vì vậy, bên trong authentication object này, có một principal và đối với principal đó, chúng ta có thể gọi username. Khi tôi nhận được username của người dùng đã đăng nhập, tôi có thể so sánh với tham số đầu vào username mà tôi đã nhận được cho phương thức này. Vì một số lý do, nếu chúng khác nhau, thì tôi không muốn thực hiện phương pháp đó vì ai đó đang cố lấy thông tin chi tiết về khoản vay của người khác. Có thể họ đang cố hack hệ thống và họ đang cố lấy thông tin chi tiết về khoản vay của những người dùng khác. Vì vậy, những loại tình huống đó, chúng ta có thể dễ dàng xử lý với sự trợ giúp của method level security.

Và cũng rất tương tự chúng ta cũng có một annotation khác đó là PostAuthorize annotation. Không giống như annotation PreAuthorized, PostAuthorize sau sẽ không dừng việc gọi phương thức. Thay vào đó, nó sẽ thực hiện lời gọi phương thức Java mà không có bất kỳ quy tắc ủy quyền nào. Nhưng bất cứ khi nào bạn sử dụng PostAuthorize annotation, nó sẽ đảm bảo rằng nó đang xác thực những gì bạn đang gửi lại.



Giống như với sự trợ giúp của PostAuthorized annotation, đối tượng trả về, tôi có thể kiểm tra xem đối tượng trả về có liên quan đến một người dùng cụ thể hay không, tôi có thể có trường tên người dùng. Vì vậy, đó là lý do tại sao tôi phải sử dụng `returnObject.username` nếu nó khớp với người dùng đã đăng nhập với sự trợ giúp của `authentication.principal.username`. Và rất tương tự, giống như PreAuthorize bên trong PostAuthorize, chúng ta có thể sử dụng `hasAnyRole`, `hasAnyAuthority`, `hasAuthority`, `hasRole`. Tất cả những phương pháp đó chúng ta cũng có thể sử dụng ở đây.

Nhưng như tôi đã nói, việc post authorization sẽ không dừng việc thực thi phương thức. Nó sẽ làm cho việc thực thi hoàn tất và sau khi quá trình thực thi hoàn tất, nó sẽ quyết định xem nó có phải trả lại kết quả đầu ra cho người dùng hay không dựa trên các quy tắc ủy quyền mà chúng tôi đã xác định. Vì một số lý do, nếu yêu cầu bảo mật hoặc yêu cầu ủy quyền của bạn, bạn không thể đạt được với sự trợ giúp của tất cả các phương pháp mà chúng ta đã thảo luận như với sự trợ giúp của `hasAuthority`,

hasRole và Spring Expression Language. Nếu yêu cầu của bạn cực kỳ phức tạp thì bạn luôn có thể ghi đè và interface có tên PermissionEvaluate. Bạn có thể xác định lớp của riêng mình, bạn có thể triển khai interface PermissionEvaluator này và sau khi bạn triển khai PermissionEvaluator này, interface này có tên phương thức như hasPermission và bạn có thể viết tất cả logic phức tạp của mình bên trong đó. Và từ hasPermission này, bạn có thể trả về giá trị boolean đúng hoặc sai và theo đó, Spring Security của bạn có thể hiểu liệu lời gọi phương thức có thể được cho phép hay không và để gọi tất cả logic nghiệp vụ mà bạn đã viết bên trong PermissionEvaluator implementation class, bạn phải chuyển từ khóa hasPermission cho các annotation PreAuthorize hoặc PostAuthorize của mình. Chúng ta cũng có thể nhanh chóng kiểm tra interface này.

```
no usages 1 implementation
public interface PermissionEvaluator extends AopInfrastructureBean {
    no usages 1 implementation
    boolean hasPermission(Authentication authentication, Object targetDomainObject, Object permission);

    no usages 1 implementation
    boolean hasPermission(Authentication authentication, Serializable targetId, String targetType, Object permission);
}
```

Ở đây bạn có thể thấy tôi đã mở một PermissionEvaluator interface có sẵn bên trong Spring Security framework. Bất cứ khi nào bạn triển khai PermissionEvaluator này, bạn có thể ghi đè các phương thức như hasPermission. Vì vậy, phương pháp này sẽ chấp nhận một vài tham số đầu vào như chi tiết người dùng đã đăng nhập là gì. Và tham số thứ hai mà chúng ta có là targetDomainObject. Vì vậy, bạn luôn có thể chuyển một đối tượng như đối tượng mà bạn đang cố trả lại cho user hoặc đối tượng mà bạn nhận được từ user. Vì vậy, đối tượng mục tiêu đó bạn có thể chuyển thành tham số thứ hai. Và tham số thứ ba là PermissionExpression mà bạn muốn đánh giá bên trong phương thức hasPermission này là gì. Vì vậy, ở đây bạn cần đề cập đến object ID hay the serialization ID. Vì vậy, thay vì đề cập đến serialization ID và target type, bạn luôn có thể chuyển chính đối tượng đó với sự trợ giúp của targetDomainObject này. Vì vậy, đây là hai phương thức bạn có thể ghi đè và dựa trên những gì bạn trả về từ phương thức này, chẳng hạn như true hoặc false, theo đó, Postauthorization và PreAuthorization sẽ xảy ra. Và để gọi logic mà bạn đã trả về bên trong phương thức hasPermission này, bạn luôn có thể chuyển hasPermission cho các annotation của mình như PreAuthorize hoặc PostAuthorize.

Và ở đây bạn có thể có một câu hỏi như, Khi nào tôi có thể sử dụng PreAuthorize và khi nào tôi có thể sử dụng PostAuthorize? Trong Spring Security, PreAuthorize và PostAuthorize là hai chú thích (annotations) được sử dụng để áp dụng kiểm soát truy cập dựa trên phương pháp (method-level access control) trong ứng dụng của bạn.

**Bạn có thể sử dụng PreAuthorize khi bạn muốn áp dụng kiểm soát truy cập trước khi phương thức được thực thi.** PreAuthorize cho phép bạn đặt điều kiện cho phép truy cập dựa trên các biểu thức SpEL (Spring Expression Language). Ví dụ, bạn có thể kiểm tra vai trò của người dùng hoặc các thuộc tính khác của đối tượng trong biểu thức SpEL để xác định liệu phương thức có được thực thi hay không.

Ví dụ sử dụng PreAuthorize:

```
@PreAuthorize("hasRole('ROLE_ADMIN')")
```

```
public void deleteProduct(int productId) {  
    // Xóa sản phẩm  
}
```

Trong ví dụ trên, phương thức **deleteProduct** chỉ có thể được thực thi nếu người dùng có vai trò "ROLE\_ADMIN".

PostAuthorize, theo mặc định, được sử dụng sau khi phương thức đã được thực thi và được sử dụng để kiểm tra kết quả trả về từ phương thức. Bạn có thể sử dụng PostAuthorize để kiểm tra các điều kiện trên kết quả trả về và quyết định liệu kết quả có được trả về cho người dùng hay không. Các biểu thức SpEL cũng được sử dụng để kiểm tra các điều kiện.

Ví dụ sử dụng PostAuthorize:

```
@PostAuthorize("returnObject.owner == authentication.name")  
public Product getProduct(int productId) {  
    // Lấy thông tin sản phẩm từ cơ sở dữ liệu  
    return product;  
}
```

Trong ví dụ trên, phương thức **getProduct** chỉ trả về kết quả nếu người dùng xác thực (authenticated) trùng khớp với chủ sở hữu (owner) của sản phẩm.

Thông tin cuối cùng mà tôi muốn chia sẻ ở đây là điều kỳ diệu này xảy ra như thế nào. Giống như, nếu tôi vừa đề cập đến các annotation, Spring Security framework có thể thực thi tất cả các biện pháp bảo mật như thế nào? Câu trả lời là Spring AOP. Bằng cách tận dụng Spring AOP, Spring Security của tôi khi chạy, nó có thể chặn lời gọi phương thức và trước khi gọi phương thức, nó có thể thực thi tất cả các quy tắc authorization mà chúng tôi đã định cấu hình với sự trợ giúp của PreAuthorized annotation. Và bất cứ khi nào bạn có annotation PostAuthorize, thì trong trường hợp đó, ngay trước khi đạt được đầu ra phương thức của tôi, Spring Security của tôi sẽ chặn lời gọi phương thức với sự trợ giúp của các khái niệm Spring AOP. Hầu hết những điều kỳ diệu luôn xảy ra bên trong bất kỳ dự án Spring nào, bạn luôn có thể thấy bên trong chúng tận dụng khái niệm Spring AOP. Spring AOP là một khái niệm rất thú vị và mạnh mẽ.

### 3. Details about filtering authorization in method level security

Trong Spring Security, PreFilter và PostFilter là hai tính năng mạnh mẽ được sử dụng để áp dụng lọc ẩn quyền trước và sau khi một tập hợp kết quả được trả về từ một phương thức được bảo vệ.

1. PreFilter: PreFilter (bộ lọc trước) là giai đoạn đầu tiên trong quy trình xử lý dữ liệu. Nhiệm vụ chính của PreFilter là lọc và tiền xử lý dữ liệu đầu vào để chuẩn bị cho quá trình xử lý tiếp theo. Một số công việc thường được thực hiện trong PreFilter bao gồm:

- Loại bỏ dữ liệu nhiễu: Loại bỏ các giá trị nhiễu hoặc dữ liệu không hợp lệ từ đầu vào.

- Chuẩn hóa dữ liệu: Chuyển đổi dữ liệu về định dạng hoặc đơn vị chung để đảm bảo tính nhất quán và dễ dàng xử lý.
- Lựa chọn đặc trưng: Chọn các đặc trưng (feature) quan trọng hoặc phù hợp để đưa vào quá trình xử lý tiếp theo.
- Giảm số chiều dữ liệu: Thực hiện các phương pháp giảm số chiều dữ liệu, như PCA (Phân tích thành phần chính), để giảm kích thước dữ liệu và tăng tốc độ xử lý.

Ví dụ sử dụng PreFilter:

```
@PreFilter("filterObject.contactName != 'Test' ")
public List<Contact> saveContactInquiryDetails (@RequestBody List< Contact> contacts) {
    // Lấy danh sách sản phẩm từ cơ sở dữ liệu dựa trên productIds
    return contacts;
}
```

Trong ví dụ trên, phương thức `saveContactInquiryDetails` lấy danh sách liên hệ dựa trên danh sách `contacts`. Annotation `@PreFilter` được sử dụng để kiểm tra và lọc các liên hệ trong danh sách liên hệ dựa trên `contactName` khác "`'Test'`".

2. PostFilter: PostFilter (bộ lọc sau) là giai đoạn thứ hai trong quy trình xử lý dữ liệu, xảy ra sau khi dữ liệu đã được xử lý qua các bước tiền xử lý và phân tích. Nhiệm vụ chính của PostFilter là lọc và xử lý kết quả đầu ra để đảm bảo tính chính xác và đáng tin cậy của dữ liệu. Một số công việc thường được thực hiện trong PostFilter bao gồm:

- Loại bỏ dữ liệu nhiễu: Phát hiện và loại bỏ dữ liệu nhiễu hoặc kết quả không chính xác từ đầu ra của quá trình xử lý trước đó.
- Sửa chữa hoặc đánh giá lại dữ liệu: Kiểm tra và sửa chữa các giá trị hoặc kết quả không hợp lệ trong dữ liệu đầu ra, đồng thời đánh giá lại tính chính xác của chúng.
- Đánh giá và chọn lọc kết quả: Đánh giá độ tin cậy của kết quả và loại bỏ các kết quả không mong muốn hoặc không chính xác.
- Xử lý dữ liệu đầu ra: Áp dụng các biện pháp xử lý bổ sung hoặc biến đổi dữ liệu đầu ra để đáp ứng yêu cầu cụ thể hoặc chuẩn hóa định dạng dữ liệu.
- Tích hợp kiểm tra và giám sát: Thực hiện các phương pháp kiểm tra và giám sát kết quả đầu ra để đảm bảo tính nhất quán và chất lượng của dữ liệu.

Qua đó, giai đoạn PostFilter giúp đảm bảo rằng dữ liệu đầu ra cuối cùng sau quá trình xử lý đã được lọc và tinh chỉnh để đáp ứng

Ví dụ sử dụng PostFilter:

```
@ PostFilter ("filterObject.contactName != 'Test' ")
public List<Contact> saveContactInquiryDetails (@RequestBody List< Contact> contacts) {
    // Lấy danh sách sản phẩm từ cơ sở dữ liệu dựa trên productIds
    return contacts;
}
```

Trong ví dụ này, chúng ta có một phương thức `saveContactInquiryDetails` được chú thích bằng `@PostFilter`. Đây là một annotation của Spring Security, cho phép áp dụng một điều kiện lọc trên danh sách `Contact` trước khi trả về kết quả.

Điều kiện lọc được xác định trong câu lệnh `"filterObject.contactName != 'Test'"`. Nghĩa là chỉ những đối tượng `Contact` trong danh sách mà có trường `contactName` khác "Test" mới được bảo lưu trong danh sách kết quả. Các đối tượng có `contactName` là "Test" sẽ bị loại bỏ khỏi danh sách trước khi nó được trả về.

Khi phương thức `saveContactInquiryDetails` được gọi, Spring Security sẽ tự động áp dụng quy trình lọc này trên danh sách `contacts` và chỉ trả về các đối tượng `Contact` thỏa mãn điều kiện lọc.

Lưu ý rằng cú pháp `"filterObject.contactName"` đại diện cho trường `contactName` trong mỗi đối tượng `Contact` trong danh sách.



## Section 11: Deep dive of OAuth2 & OpenID Connect

### 1. Problems that OAuth2 trying to solve

Hiện tại, bên trong ứng dụng web của chúng tôi, chúng tôi đã viết tất cả logic nghiệp vụ, logic bảo mật, yêu cầu xác thực và ủy quyền, mọi thứ chúng tôi gộp vào một ứng dụng web duy nhất. Và điều này có thể hoạt động với phần lớn các ứng dụng hoặc tổ chức nơi họ có một ứng dụng web duy nhất, nhưng nếu tổ chức của bạn có nhiều ứng dụng web hoặc ứng dụng di động hoặc vi dịch vụ, thì chắc chắn bạn nên luôn tách biệt logic bảo mật, xác thực của mình và logic ủy quyền thành một thành phần riêng biệt, chẳng hạn như nếu bạn có thể coi ngân hàng là một ví dụ thời gian thực, ngân hàng có thể có ứng dụng web ngân hàng trực tuyến, ứng dụng di động và ngân hàng cũng có thể có một số microservices, ứng dụng web nội bộ để xử lý khoản vay, phê duyệt khoản vay, vì vậy việc duy trì các yêu cầu xác thực và ủy quyền bên trong mỗi ứng dụng web và ứng dụng di động này là một công việc rất tẻ nhạt. Bạn phải luôn chuyển logic chống bảo mật này vào thành phần chung của mình để tất cả các ứng dụng web của bạn, bao gồm cả ứng dụng di động, microservices, chúng luôn có thể hoạt động với thành phần chung này để xác thực và ủy quyền cho user.

Để đạt được điều tương tự, chúng ta nên tìm kiếm các tùy chọn nâng cao mà chúng ta có, bất cứ khi nào chúng ta nói về authentication và authorization. Tương tự, chúng ta sẽ tìm hiểu mọi thứ về OAuth2. OAuth2 là một tiêu chuẩn. Bất cứ khi nào bạn nói về bảo mật, authentication và authorization, nhiều tổ chức doanh nghiệp, như Google, Facebook, GitHub, Netflix. Vì vậy, rất nhiều tổ chức lớn, họ tận dụng OAuth2 bên trong tổ chức của mình để triển khai authentication và authorization bên trong các ứng dụng web của họ. Vì vậy, đó là lý do tại sao bên trong section này, chúng ta sẽ chỉ nói về OAuth2 framework.

Vấn đề đầu tiên mà OAuth2 giải quyết trong ngành, tôi muốn giải thích bằng một ví dụ. Tương tự, tôi cho rằng có một ứng dụng web TweetAnalyzer. Đây là một ứng dụng hư cấu, mà tôi đang giả định. Hãy nghĩ về một kịch bản. Bạn đang sử dụng Twitter. Sử dụng Twitter, bạn thường xuyên tạo một số tweet, bạn tạo một số tweet lại và bạn cũng có lượng người theo dõi tốt và họ thích các tweet của bạn. Là người dùng thường xuyên của ứng dụng Twitter, bạn muốn phân tích thông tin về các tweet của mình, chẳng hạn như tần suất bạn tạo tweet, bạn tạo bao nhiêu tweet hàng ngày, hàng tuần và bạn nhận được bao nhiêu lượt thích, bao nhiêu lượt retweet. Bạn đang lấy. Vì vậy, tất cả những thông tin mà bạn muốn hiểu. Cũng vì lý do đó mà ứng dụng web có tên trang web TweetAnalyzer. Vì vậy, trang web này sẽ phân tích dữ liệu các tweet của bạn và tạo các số liệu từ dữ liệu đó cũng như các số liệu tương tự mà bạn có thể sử dụng để hiểu dữ liệu các tweet của mình. Vì vậy, kịch bản rất đơn giản. Một người dùng Twitter muốn sử dụng trang web của bên thứ ba có tên là TweetAnalyzer để nhận một số thông tin chi tiết về các tweet của anh ấy, trang web này hiện diện bên trong ứng dụng Twitter. Trong trường hợp này, chắc chắn là TweetAnalyzer của tôi, để phân tích dữ liệu các tweet của tôi, nó cần tất cả thông tin từ ứng dụng Twitter vì ứng dụng Twitter là ứng dụng lưu trữ tất cả thông tin về các tweet của tôi bao nhiêu tweet tôi đang tạo, bao nhiêu lượt thích tôi đã nhận. Tất cả những thông tin này nằm trong ứng dụng Twitter. Nếu TweetAnalyzer của tôi hỏi ứng dụng Twitter của tôi, hãy cung cấp thông tin người dùng Twitter như vậy, chắc chắn ứng dụng Twitter của tôi sẽ không đợi.

## INTRO TO OAUTH2

PROBLEM THAT OAUTH2 SOLVES

easy  
bytes



Twitter App



Twitter user



TweetAnalyzer website that analyzes user tweets data and generates metrics from it

✓ **Scenario :** The twitter user want to use an third party website called TweetAnalyzer, to get some insights about his tweets data present inside Twitter App.

- **With Out OAUTH2 :** Twitter user has to share his twitter account credentials to the TweetAnalyzer website. Using user credentials, the TweetAnalyzer website will invoke the APIs of Twitter app to fetch the tweet details and post that generates a report for the end user.

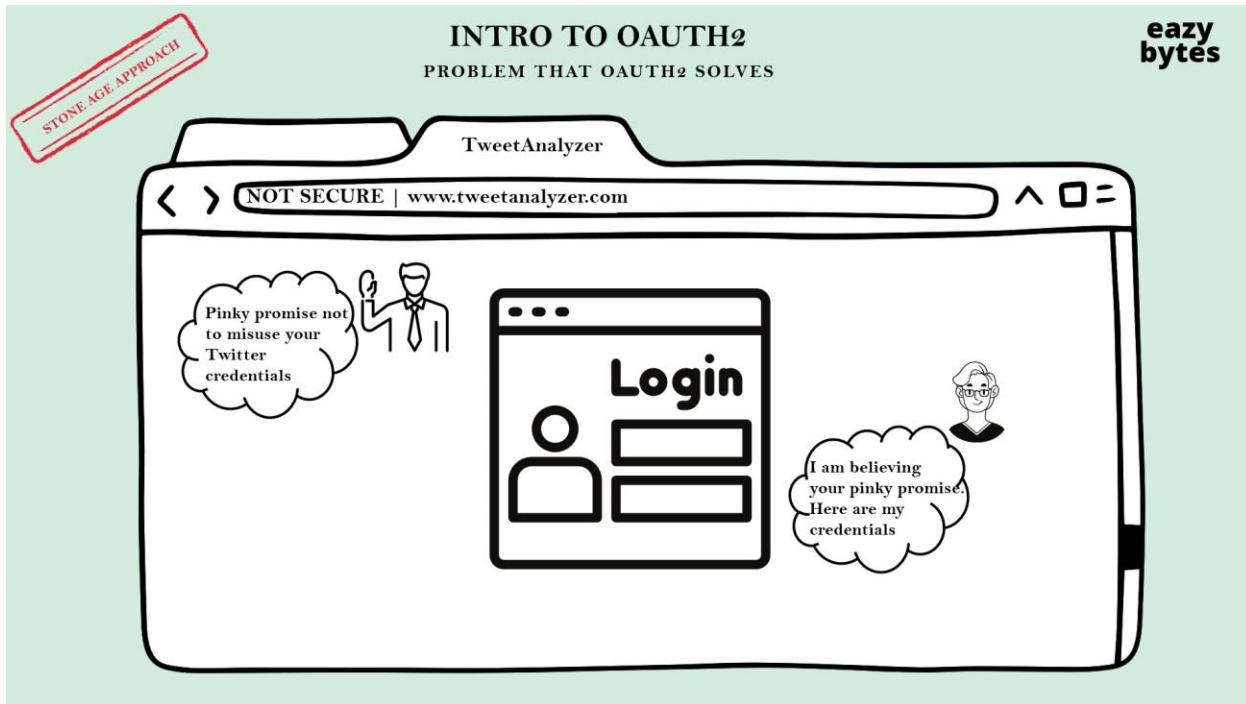
But it has a bigger disadvantage, the TweetAnalyzer can go fraud and make another operations on your behalf like change password, change email, make a rogue tweet etc.

- **With OAUTH2 :** Twitter user doesn't have to share his twitter account credentials to the TweetAnalyzer website. Instead he will let Twitter App to give a temporary access token to TweetAnalyzer with limited access like it can only read the tweets data.

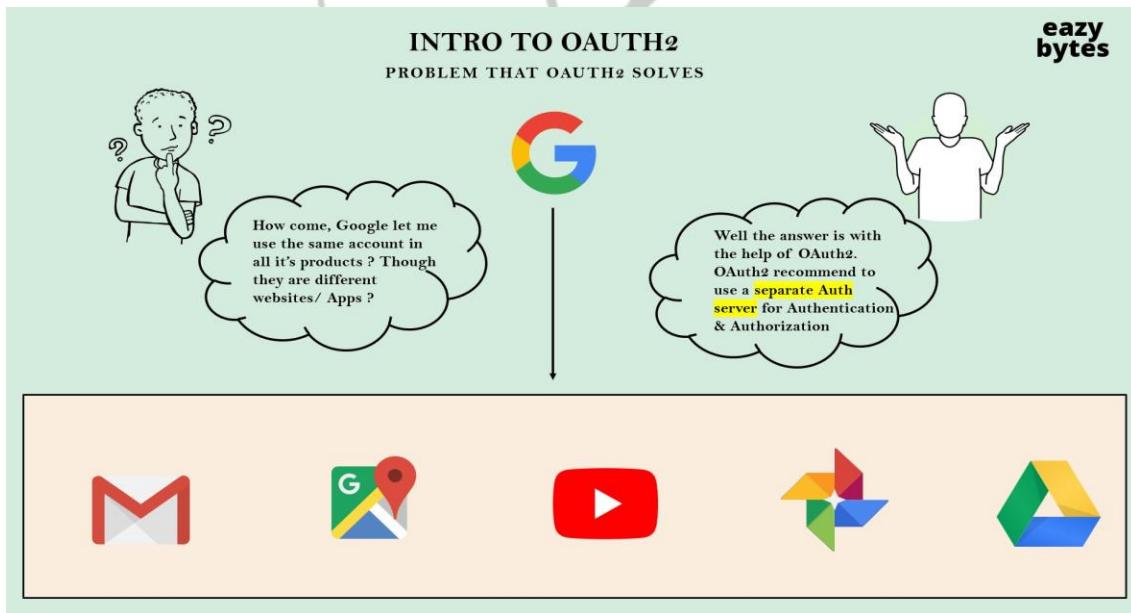
With this approach, the TweetAnalyzer can only read the tweets data and it can't perform any other operation.

Để thực hiện công việc này, chắc chắn TweetAnalyzer của tôi sẽ hỏi thông tin đăng nhập của người dùng Twitter của tôi. Vì vậy, trang web bên thứ ba của tôi, đó là TweetAnalyzer, có thể yêu cầu, vui lòng chia sẻ thông tin đăng nhập Twitter của bạn. Sử dụng cùng thông tin đăng nhập, tôi có thể thử phân tích dữ liệu của bạn bằng cách đăng nhập vào tài khoản của bạn. Trước khi phát minh ra OAuth2 framework, đây là cách nó từng xảy ra, giống như mọi user, họ phải chia sẻ thông tin xác thực của riêng mình với các ấn phẩm web của bên thứ ba, như TweetAnalyzer, nếu họ gặp phải những tình huống như vậy. Vì vậy, ngay sau khi bạn cung cấp thông tin đăng nhập, thì chắc chắn là TweetAnalyzer, họ có thể sử dụng thông tin đăng nhập của bạn, họ có thể phân tích dữ liệu của bạn và họ có thể tạo báo cáo cho mục đích hiểu biết của bạn. Nhưng nó có một nhược điểm rất lớn vì bạn đang cung cấp thông tin đăng nhập của riêng mình cho ứng dụng web của bên thứ ba và nó có thể trở thành gian lận hoặc nó có thể thực hiện bất kỳ hoạt động nào khác thay mặt bạn, chẳng hạn như thay vì nghiên cứu các tweet của bạn, họ có thể thay đổi mật khẩu của bạn, họ có thể thay đổi email của bạn hoặc họ có thể tạo một dòng tweet lừa đảo. Vì vậy, tất cả điều này có thể xảy ra nếu bạn cố tin vào ứng dụng web của bên thứ ba.

Vì vậy, nếu bạn nhìn vào cách tiếp cận của Thời kỳ đồ đá, như trước OAuth2 framework, cách mọi người sử dụng để xử lý cùng một kịch bản TweetAnalyzer, bạn có thể thấy có một trang web TweetAnalyzer đang đưa ra một lời hứa màu hồng rằng vui lòng chia sẻ thông tin đăng nhập tài khoản Twitter của bạn, tôi sẽ không lạm dụng chúng. Và bạn với tư cách là user, bạn không có bất kỳ lựa chọn nào khác. Bạn tin vào lời hứa màu hồng của ấn phẩm web bên thứ ba và bạn chia sẻ thông tin đăng nhập thực tế của tài khoản Twitter với trang web này và vì bạn đang xử lý thông tin đăng nhập chính của mình nên rất có thể ứng dụng web của bên thứ ba có thể sử dụng sai thông tin đăng nhập của bạn.

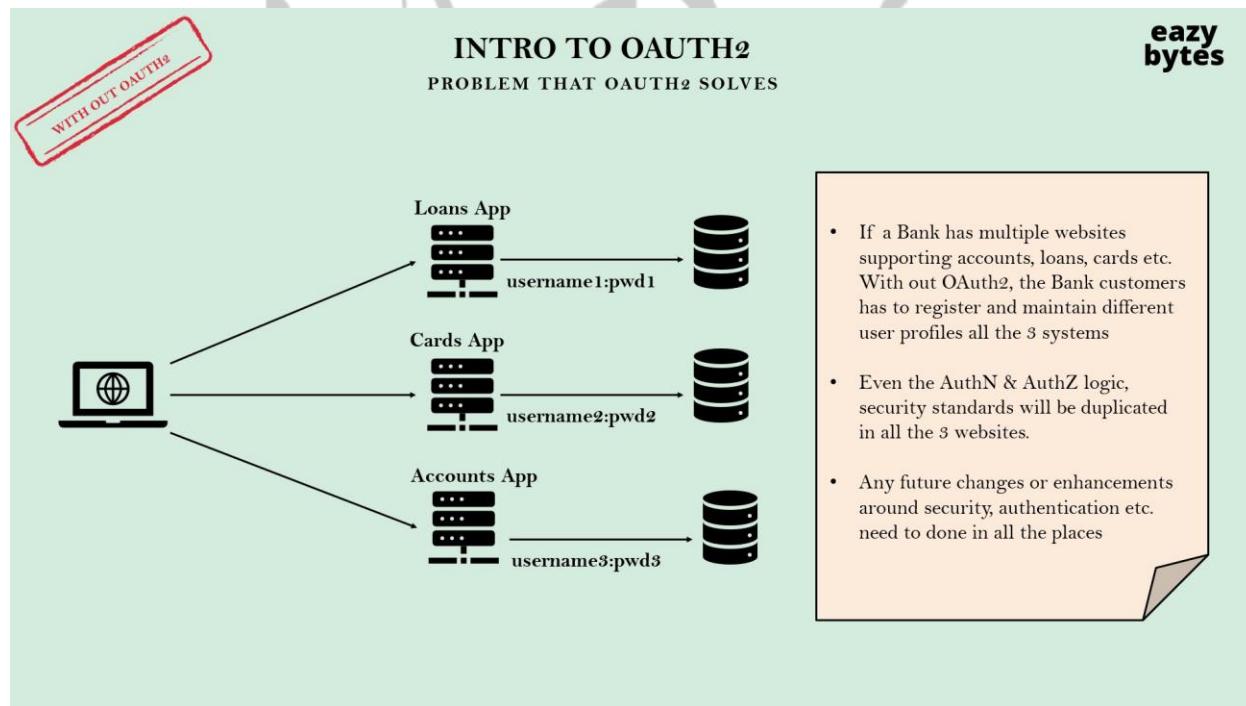


Vì vậy, đây từng là cách tiếp cận thời kỳ đồ đá, trước khi phát minh ra OAuth2. Cách tiếp cận này sẽ không hoạt động nữa. OAuth2 sẽ trợ giúp như thế nào trong các loại tình huống này? Chắc chắn rằng chúng ta sẽ thảo luận trong các bài giảng sắp tới. Bây giờ, hãy cố gắng hiểu các sự cố khác mà OAuth2 cố gắng giải quyết tại đây. Bạn có nghĩ bất cứ lúc nào, chẳng hạn như cách Google cho phép chúng tôi sử dụng đi sử dụng lại cùng một thông tin đăng nhập trong tất cả các ứng dụng web của mình, trong tất cả các sản phẩm của mình không? Giống như, Google là một tổ chức rất lớn và nó có rất nhiều sản phẩm và ứng dụng web, ứng dụng di động, chẳng hạn như họ có Gmail, họ có Google Maps, họ có YouTube, họ có Google Photos, họ có Google Drive.



Đây là tất cả các ứng dụng web hoặc ứng dụng di động khác nhau, nhưng tất cả chúng đều cho phép cùng một thông tin xác thực. Tôi không cần phải sử dụng các thông tin đăng nhập khác nhau, đăng ký thông tin khác nhau bằng các tài khoản khác nhau trong tất cả các ứng dụng web này. Làm thế nào điều đó có thể thực hiện được với sự trợ giúp của OAuth2. OAuth2 luôn khuyên bạn nên duy trì một authorization server riêng sẽ có logic authentication và authorization. Vì vậy, trong nội bộ, những gì Google có thể đang làm là bắt cứ khi nào ai đó cố gắng đăng nhập vào bất kỳ sản phẩm nào trong số này, họ sẽ chấp nhận tên người dùng và mật khẩu từ user và họ sẽ gửi những thông tin chi tiết đó đến một trong những thành phần khác đó là authorization server. Và vì tất cả chúng đều trở đến cùng một authorization server, nên các thông tin đăng nhập giống nhau có thể hoạt động và trong tương lai, nếu Google muốn thay đổi một số logic authorization và authentication, thì chỉ có một nơi duy nhất mà họ phải thực hiện thay đổi. Vì vậy, đó là một lợi thế nữa với OAuth2.

Và một ưu điểm nữa mà đôi khi bạn có thể đã thấy, khi đăng nhập vào Gmail, bạn không phải đăng nhập lại vào YouTube. Ngay khi bạn mở YouTube bên trong một tab khác, chi tiết đăng nhập của bạn sẽ được các trang web khác như YouTube, Google Photos tự động phát hiện và tất cả những điều này đều có thể thực hiện được nhờ sự trợ giúp của OAuth2. Bởi vì chúng tôi có thể chia sẻ token truy cập với tất cả các trang web này và vì tất cả chúng đều trở đến cùng một authorization server ở hậu trường nên chúng sẽ không gặp bất kỳ sự cố nào, do đó, đó là sức mạnh của OAuth2. Vì vậy, điểm chính mà bạn phải nhớ ở đây là OAuth2 khuyến khích chúng tôi sử dụng một authorization server riêng, chịu trách nhiệm thực hiện authorization và authentication.

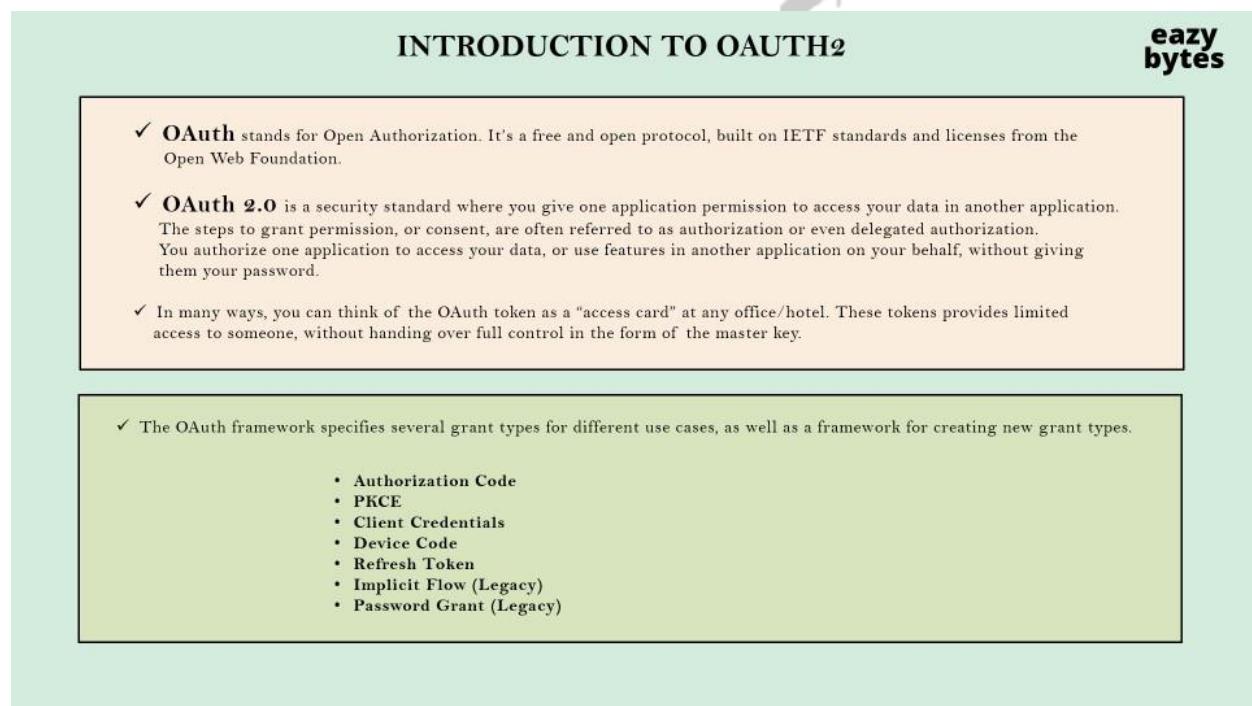


Và vấn đề tương tự tiếp theo mà OAuth2 giải quyết là nghĩ về một tình huống trong đó có một tổ chức ngân hàng và tổ chức ngân hàng này có nhiều ứng dụng khác nhau để xử lý khoản vay, xử lý thẻ và xử lý thông tin tài khoản. Nếu với tư cách là user, nếu ứng dụng ngân hàng của tôi hỏi tất cả user của tôi thì đây là những ứng dụng khác nhau, vui lòng đăng ký chúng riêng lẻ và duy trì các tên người dùng và mật khẩu khác nhau, điều đó sẽ không có ý nghĩa gì vì theo user của tôi, bạn là người dùng duy nhất. Đó là nơi OAuth2 xuất hiện vì với sự trợ giúp của OAuth2, tất cả các ứng dụng web này, chúng có thể trả

đến một authorization server duy nhất và bắt đầu duy trì thông tin chi tiết của người dùng trong các cơ sở dữ liệu khác nhau. Chúng có thể có một thành phần chung riêng biệt là authorization server. Vì vậy, đó là lý do tại sao bất cứ thứ gì bạn nhìn thấy trên màn hình, chẳng hạn như vị trí các tài khoản khác nhau cho các ứng dụng khác nhau được lưu trữ bên trong cơ sở dữ liệu khác nhau. Đây từng là một tình huống không có OAuth2, nhưng khi OAuth2 xuất hiện, mọi người hiện đang theo dõi một thành phần chung riêng biệt với authorization server tên xử lý xác thực và ủy quyền cho tất cả các ứng dụng trong tổ chức của tôi. Vì vậy, đây là tất cả một số vấn đề phổ biến mà OAuth2 giải quyết trong product của bạn.

## 2. Introduction to Oauth2

OAuth 2.0 là một giao thức nguồn mở và miễn phí được xây dựng bởi chính cộng đồng nguồn mở. Nhưng xin lưu ý rằng OAuth 2.0 chỉ là một thông số kỹ thuật. Nó xác định các tiêu chuẩn mà chúng tôi cần tuân theo bất cứ khi nào chúng tôi cố gắng đạt được authentication và authorization bên trong bất kỳ ứng dụng web nào. Nó không phải là một sản phẩm, nó không phải là một máy chủ, hoặc nó không phải là một framework triển khai mà chúng ta có thể tận dụng trực tiếp. Vì vậy, chỉ là một tiêu chuẩn của đặc điểm kỹ thuật.



**INTRODUCTION TO OAUTH2**

**eazy bytes**

- ✓ OAuth stands for Open Authorization. It's a free and open protocol, built on IETF standards and licenses from the Open Web Foundation.
- ✓ OAuth 2.0 is a security standard where you give one application permission to access your data in another application. The steps to grant permission, or consent, are often referred to as authorization or even delegated authorization. You authorize one application to access your data, or use features in another application on your behalf, without giving them your password.
- ✓ In many ways, you can think of the OAuth token as a "access card" at any office/hotel. These tokens provides limited access to someone, without handing over full control in the form of the master key.

- ✓ The OAuth framework specifies several grant types for different use cases, as well as a framework for creating new grant types.
  - Authorization Code
  - PKCE
  - Client Credentials
  - Device Code
  - Refresh Token
  - Implicit Flow (Legacy)
  - Password Grant (Legacy)

Dựa trên các tiêu chuẩn và đặc điểm kỹ thuật này, có nhiều sản phẩm và khuôn khổ được phát triển trong ngành mà chúng ta sẽ thảo luận trong các bài giảng sắp tới. Nhưng hiện tại, xin lưu ý rằng OAuth 2.0 chỉ là một thông số kỹ thuật hoặc giao diện hoặc giao thức chỉ xác định các tiêu chuẩn và không có mã triển khai nào được viết như một phần của OAuth 2.0 này.

Giống như chúng ta đang thảo luận trước đây, OAuth 2.0 là một tiêu chuẩn bảo mật nơi bạn có thể cấp cho một ứng dụng quyền truy cập dữ liệu của mình bên trong một ứng dụng khác, đặc biệt, nếu ứng dụng web của bên thứ ba đang cố truy cập dữ liệu của bạn bên trong một ứng dụng web khác, thì OAuth 2.0 là một lựa chọn rất tốt mà bạn có thể xem xét. Giống như nghĩ về một tình huống, một trong những trang web chỉnh sửa ảnh, họ muốn truy cập ảnh của bạn có sẵn trong Google photos hoặc trong Google Drive của bạn để họ có thể tải tất cả ảnh của bạn trên trang web của họ và trên hết, bạn có thể

làm việc chỉnh sửa ảnh. Vì vậy, ở đây chắc chắn, có một ứng dụng web của bên thứ ba có liên quan. Chúng tôi không nên cung cấp mật khẩu Google, mật khẩu Gmail, thông tin đăng nhập của chúng tôi cho ứng dụng web của bên thứ ba để tải tất cả ảnh của chúng tôi. Thay vào đó, với sự trợ giúp của framework OAuth 2.0 hoặc với sự trợ giúp của giao thức OAuth 2.0, chúng tôi sẽ đảm bảo rằng chúng tôi đang cấp quyền rất hạn chế cho các ứng dụng web của bên thứ ba này. Và sử dụng quyền hạn chế này đối với ấn phẩm bên thứ ba của tôi, nó chỉ có thể đọc ảnh và không thể thực hiện bất kỳ hành động nào khác. Vì vậy, đó là điểm mạnh của framework OAuth 2.0.

Một lần nữa, chúng ta sẽ cấp quyền như thế nào, với sự trợ giúp của một token có tên là access token. Access token là một thành phần rất quan trọng bên trong framework OAuth 2.0 vì access token này chỉ quyết định những ứng dụng bên thứ ba có thể thực hiện, chúng có thẩm quyền gì hoặc chúng có những quyền gì. Theo nhiều cách, bạn luôn có thể coi access token OAuth 2.0 là thẻ truy cập tại bất kỳ văn phòng hoặc khách sạn nào. Với sự trợ giúp của thẻ ra vào bên trong văn phòng của chúng tôi, các hạn chế sẽ được áp dụng. Giống như, bạn không thể đi lang thang khắp văn phòng của mình, bạn sẽ chỉ được cấp quyền truy cập vào một khu vực phòng ban hạn chế hoặc một khu vực hạn chế. Rất giống với cách sử dụng access token OAuth 2.0, các hạn chế sẽ được thực thi.

Và một ưu điểm khác của framework OAuth 2.0 là nó có nhiều loại cấp phép khác nhau cho các trường hợp sử dụng khác nhau. Grant type - Loại cấp là authentication và authorization flow user hoặc ứng dụng. Chỉ sử dụng các grant type này, access token sẽ được cấp. Giống như, nếu bạn có một kịch bản liên quan đến user thì chắc chắn chúng ta cần tuân theo authorization code grant type. Và tương tự, chúng tôi cũng có grant type PKCE có thể được sử dụng bên trong các ứng dụng JavaScript hoặc bên trong các ứng dụng PACE đơn lẻ.

Đồng thời, nếu hai dịch vụ microservices hoặc nếu hai backend applications đang cố gắng giao tiếp với nhau, chúng tôi có client credentials grant type và bất cứ khi nào chúng tôi có các thiết bị liên quan như Apple TV hoặc Android TV, chúng tôi có thể theo dõi quy trình cấp device code grant flow .

Và rất giống nhau, chúng ta có refresh token grant flow, implicit flow, and password grant flow.. Vì vậy, chúng ta sẽ thảo luận chi tiết về tất cả các luồng tài trợ này khi sử dụng chúng và cách chúng hoạt động. Nhưng vui lòng lưu ý rằng , implicit flow, và password grant flow, chúng không được dùng nữa và đang bị xóa như một phần của thông số kỹ thuật OAuth 2.1.

### 3. OAuth2 terminologies or jargons – Các thuật ngữ trong OAuth2

#### OAUTH2 TERMINOLOGY

 **Resource owner** – It is you the end user. In the scenario of **TweetAnalyzer**, the end user who want to use the **TweetAnalyzer** website to get insights about this tweets. In other words, the end user owns the resources (Tweets), that why we call him as Resource owner

 **Client** – The TweetAnalyzer website is the client here as it is the one which interacts with Twitter after taking permission from the resource owner/end user.

 **Authorization Server** – This is the server which knows about resource owner. In other words, resource owner should have an account in this server. In the scenario of **TweetAnalyzer**, the Twitter server which has authorization logic acts as Authorization server.

 **Resource Server** – This is the server where the APIs, services that client want to consume are hosted. In the scenario of **TweetAnalyzer**, the Twitter server which has APIs like `/getTweets` etc. logic implemented. In smaller organizations, a single server can acts as both resource server and auth server.

 **Scopes** – These are the granular permissions the Client wants, such as access to data or to perform certain actions. In the scenario of **TweetAnalyzer**, the Auth server can issue an access token to client with the scope of only READ TWEETS.

OAuth2 (Open Authorization 2.0) là một giao thức ủy quyền mở được sử dụng rộng rãi để cho phép ứng dụng bên thứ ba truy cập vào dữ liệu của người dùng mà không cần chia sẻ mật khẩu.

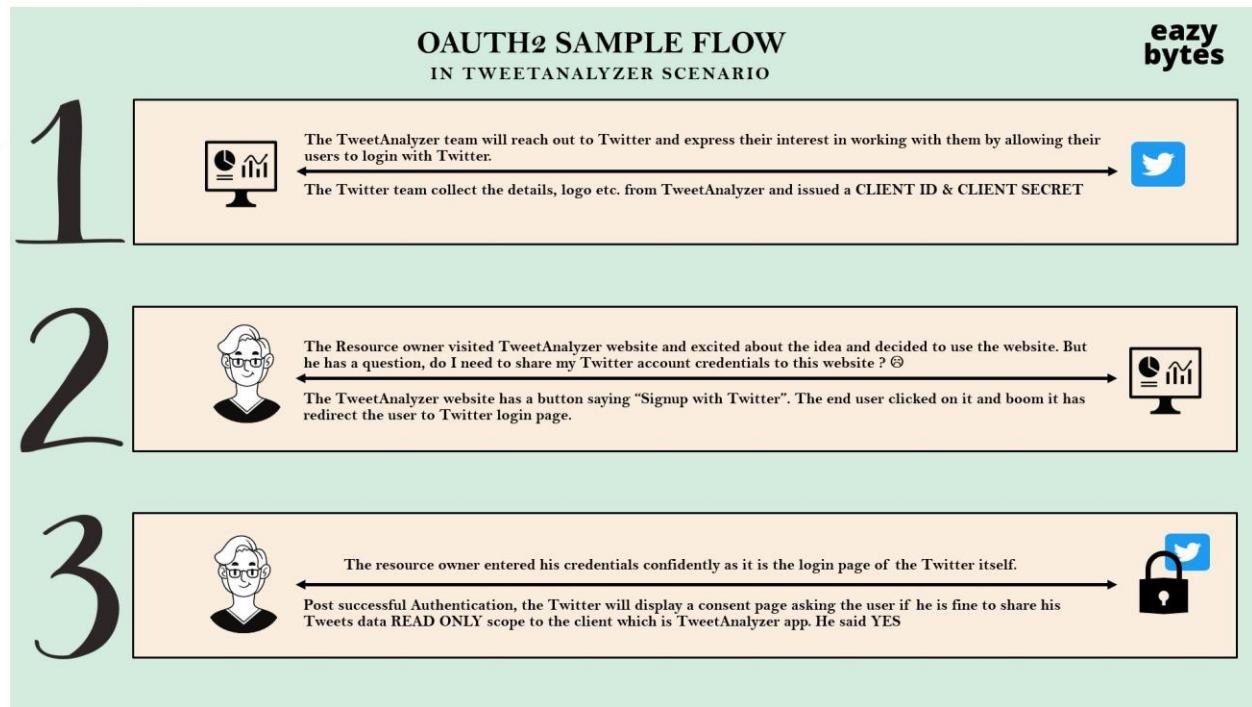
Dưới đây là một số thuật ngữ quan trọng trong OAuth2:

1. **Resource Owner (Chủ sở hữu tài nguyên)**: Là người dùng cuối cùng, người sở hữu tài nguyên mà ứng dụng muốn truy cập.
2. **Client (Ứng dụng)**: Là ứng dụng bên thứ ba muốn truy cập tài nguyên của người dùng. Nó có thể là ứng dụng di động, trang web hoặc dịch vụ web.
3. **Authorization Server (Máy chủ ủy quyền)**: Là máy chủ quản lý quyền truy cập và xác thực người dùng. Nó phát hành các mã ủy quyền cho Client sau khi người dùng đã đăng nhập thành công.
4. **Resource Server (Máy chủ tài nguyên)**: Là máy chủ chứa dữ liệu người dùng và kiểm soát quyền truy cập vào tài nguyên. Nó sẽ yêu cầu mã ủy quyền từ Client để cung cấp dữ liệu cho Client.
5. **Access Token (Mã truy cập)**: Là một mã thông báo (token) được Authorization Server cấp phát cho Client sau khi Client đã hoàn tất quá trình xác thực. Mã truy cập này được sử dụng để gọi các API của Resource Server để truy cập vào dữ liệu của người dùng.
6. **Refresh Token (Mã làm mới)**: Là một mã thông báo (token) được Authorization Server cấp phát cùng với Access Token. Refresh Token được sử dụng để lấy mã truy cập mới sau khi mã truy cập hiện tại hết hạn.
7. **Grant Type (Loại ủy quyền)**: Là các phương thức xác thực khác nhau được sử dụng trong OAuth2. Một số Grant Type phổ biến bao gồm Authorization Code, Implicit, Client Credentials và Refresh Token.

8. Scope (Phạm vi): Là quyền truy cập cụ thể mà người dùng phải cho phép cho ứng dụng. Phạm vi xác định những dữ liệu nào mà Client có thể truy cập.
9. Redirect URI (URI chuyển hướng): Là URI mà Authorization Server sẽ chuyển hướng người dùng sau khi đã xác thực thành công hoặc từ chối quyền truy cập.

### 3. OAuth2 Sample flow

Đầu tiên, bạn cần liên hệ với nhóm Twitter đó. Vì vậy, bạn liên hệ với nhóm Twitter và nói rằng tôi có rất nhiều ý tưởng và nhóm Twitter cũng thích ý tưởng của bạn và bạn hỏi họ như: chúng tôi có bất kỳ API REST nào được hiển thị để lấy thông tin tweet của một người dùng cụ thể không? Và nhóm Twitter đã trả lời rằng, tất nhiên là chúng tôi có, nhưng bạn cần tuân theo khuôn khổ OAuth2 để nhận access token từ user.



Và cùng access token mà bạn cần chuyển cho tôi. Và dựa trên access token và các quyền mà bạn có, bạn có thể gọi các API REST bên trong resource server của tôi

**Trước tiên**, bạn cần tự tin mình là khách hàng của ứng dụng Twitter. Vì vậy, đó là lý do tại sao bạn lại liên hệ với nhóm Twitter đó và họ sẽ hỏi tất cả các chi tiết cơ bản của bạn. Trang web của bạn là gì? Tên thực thể của bạn là gì, logo của bạn là gì? Và dựa trên tất cả quá trình xác minh hoàn tất, họ sẽ cấp cho bạn client ID và client secret. Client secret giống như mật khẩu mà bạn phải lưu trữ an toàn. Vì vậy, bây giờ bạn đã sẵn sàng. Bạn có Client ID hợp lệ và client secret.

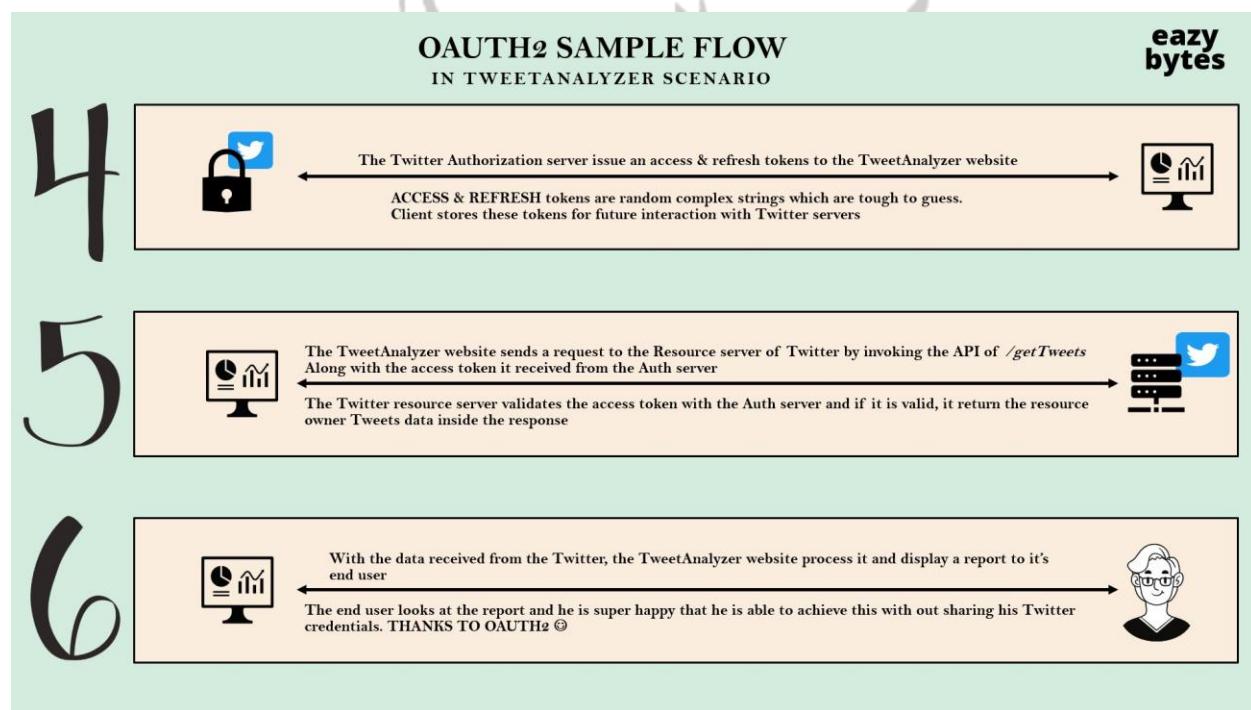
**Trong bước tiếp theo**, rõ ràng là bạn sẽ thực hiện tiếp thị trên web nói rằng bạn có thể phân tích bất kỳ thông tin tweet nào của người dùng và tạo báo cáo. Và một trong những người dùng mà chúng tôi có thể gọi họ là resource owner đã truy cập trang web của bạn và anh ấy rất hào hứng với ý tưởng này và quyết định sử dụng trang web của bạn. Nhưng anh ấy có một câu hỏi, "Tôi có cần chia sẻ chi tiết tài khoản Twitter của mình với trang web này không?" Dĩ nhiên là không. Trang web

TweetAnalyzer sẽ có một nút nói rằng hãy đăng ký bằng Twitter. Ngay khi user nhấp vào đó, một điều kỳ diệu đã xảy ra. Anh chuyển hướng đến trang đăng nhập Twitter.

Với tư cách là một client application, bạn cần cẩn thận như người dùng đang được chuyển hướng đến trang đăng nhập thực tế của authorization Server. Ở đây, authorization Server là authorization Server của tổ chức Twitter. Vì vậy, anh ta sẽ được chuyển hướng và đến trang đăng nhập của ứng dụng Twitter. Và giờ đây, Resource Owner hoặc người dùng cuối của tôi rất tin tưởng vào các ứng dụng web của tôi vì anh ấy không chia sẻ thông tin đăng nhập của mình với trang web TweetAnalyzer của tôi. Thay vào đó, anh ấy đang trình bày thông tin chi tiết của mình bên trong trang đăng nhập Twitter đó.

Vì vậy, **trong bước ba**, Resource Owner đã nhập thông tin đăng nhập của mình vào trang đăng nhập của chính Twitter. Sau khi xác thực thành công, ứng dụng Twitter sẽ hiển thị trang đồng ý hỏi người dùng xem liệu anh ấy có đồng ý chia sẻ dữ liệu tweet của mình ở định dạng chỉ đọc cho ứng dụng này hay ứng dụng khác, đó là TweetAnalyzer, bằng cách hiển thị các chi tiết cơ bản của chúng. Và sau khi thấy điều đó, chắc chắn vì tôi muốn sử dụng ứng dụng TweetAnalyzer, tôi có thể nói đồng ý bên trong trang đồng ý.

Sau khi tôi đồng ý với Authorization Server của ứng dụng Twitter, **trong bước bốn**, Authorization Server sẽ cấp access và refresh token cho client application.



Chúng ta sẽ nói về refresh tokens là gì và access tokens là gì. Tuy nhiên, hiện tại, vui lòng lưu ý rằng, bằng cách sử dụng access token, client application có thể gọi các API REST được lưu trữ trên Resource Server để có thể nhận được phản hồi thích hợp. Vì vậy, các access token và refresh tokens này là các chuỗi phức tạp được tạo ngẫu nhiên, rất khó đoán. Vì vậy, khách hàng của tôi cũng có tùy chọn lưu trữ các access token này vì Authorization Server của tôi sẽ phát hành các token với thời hạn 24 giờ hoặc 7 ngày hoặc không giới hạn thời gian. Vì vậy, tùy thuộc vào Authorization Server, thời gian hết hạn

mà nó muốn đặt là bao nhiêu. Nếu client application lưu trữ những chi tiết này, thì nó không phải yêu cầu user hoặc Resource Owner nhập đi nhập lại thông tin đăng nhập của mình bên trong trang đăng nhập Twitter. Vì vậy, đó là lý do tại sao nó phải lưu trữ các token này mà nó nhận được từ Authorization Server.

Sau khi client application nhận được access token và refresh token **trong bước năm**, trang web TweetAnalyzer của tôi sẽ gửi yêu cầu tới Resource Server của Twitter bằng cách gọi API, đó là /getTweets. Nhưng bất cứ khi nào nó đang cố gọi một API trên Resource Server, nó cũng sẽ chuyển access token mà nó nhận được từ Authorization Server. Ngay sau khi client application gửi tất cả các chi tiết này, Resource Server của tôi sẽ xác thực access token với máy chủ Auth vì cả hai đều thuộc cùng một tổ chức, đó là Twitter trong tình huống này. Nếu access token hợp lệ, thì TweetAnalyzer của tôi, nó sẽ chỉ nhận thông tin về các tweet. Cùng với đó, ứng dụng web bên thứ ba của tôi có quyền truy cập rất hạn chế. Nó chỉ có thể luôn đọc thông tin tweet đó. Nó không thể tạo một tweet mới, nó không thể cập nhật hồ sơ của tôi, nó không thể cập nhật email hoặc mật khẩu của tôi. Vì vậy, đó là điểm mạnh của access token và framework Oauth2 ở đây.

**Ở bước cuối cùng**, như bạn có thể mong đợi, client application đã nhận được tất cả các tweet. Đằng sau hậu trường, nó thực thi business logic của riêng mình và tạo báo cáo cho user hoặc Resource Owner của tôi. User xem báo cáo và anh ấy vô cùng hạnh phúc khi có thể đạt được điều này mà không cần chia sẻ thông tin đăng nhập Twitter của mình nhờ framework Oauth2. Đây là luồng Oauth2 mẫu mà bạn có thể tưởng tượng.

#### 4. Deep dive on Authorization code grant type flow in OAUTH2

Trước tiên, bạn cần hiểu các grant type flows này là gì, sự khác biệt giữa chúng là gì và sau đó chúng ta cần sử dụng các kịch bản nào. Vì vậy, đó là lý do tại sao chúng ta hãy cố gắng thảo luận từng grant flows này, ngoại trừ device code grant type. Lý do tại sao tôi bỏ qua device code grant type là vì loại device code grant type được sử dụng cho các thiết bị như Apple TV, Android TV, nơi sẽ không có trình duyệt hoặc bàn phím liên quan đến nhập thông tin đăng nhập của người dùng. Và tôi chắc chắn rằng bạn sẽ không xử lý các loại tình huống như vậy bên trong các ứng dụng web của mình. Vì vậy, với lý do đó, chúng ta có thể bỏ qua luồng device code grant type đó và chúng ta có thể thảo luận về tất cả các loại cấp phép còn lại có sẵn trong framework OAuth 2.0.

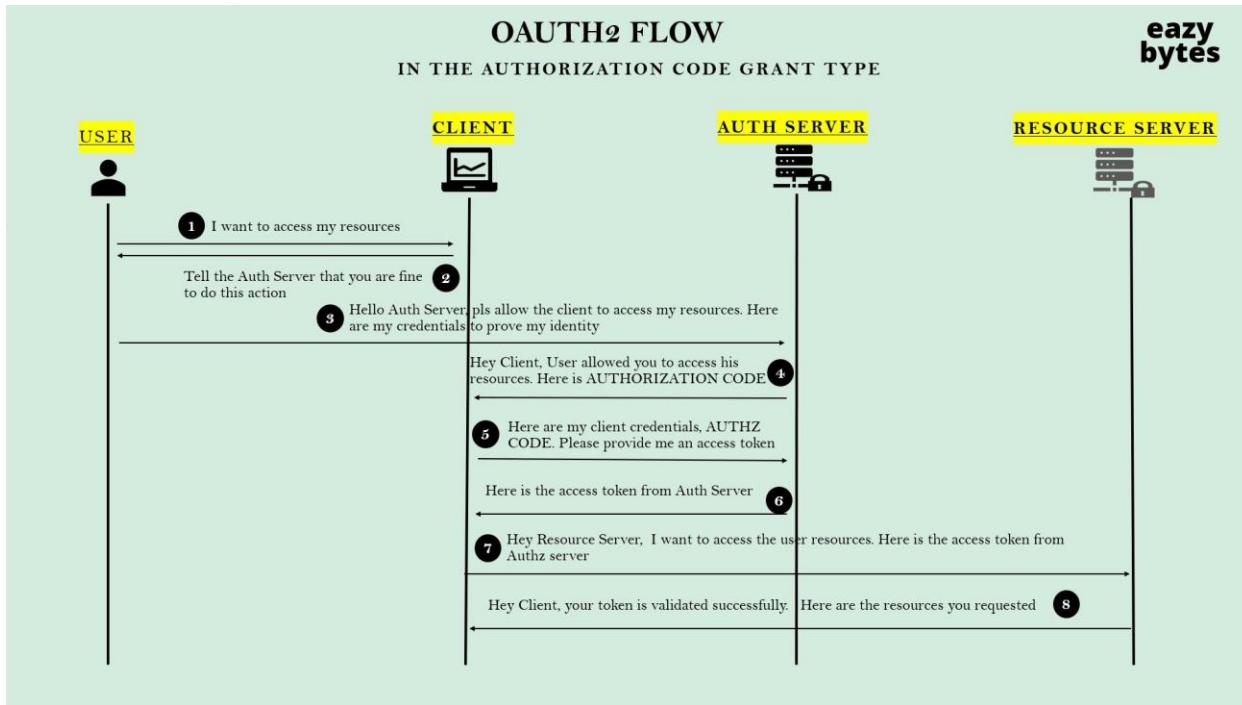
Grant type mà chúng ta có thể thảo luận là authorization code. Quy tắc ngón tay cái cơ bản đầu tiên mà bạn cần nhớ bất cứ khi nào bạn đang cố gắng sử dụng **authorization code grant type flow** là bạn nên sử dụng flow này bất cứ khi nào **có sự tham gia của user và hai ứng dụng khác nhau đang cố gắng giao tiếp với nhau**. Vì vậy, nếu bạn có thể thấy ở đây, có một người dùng là Resource Owner và có một client application, đồng thời có một Authorization Server và Resource Server.

**Ở bước đầu tiên**, điều sẽ xảy ra là Resource Owner của tôi hoặc người dùng của tôi sẽ liên hệ với client application nói rằng tôi muốn truy cập tài nguyên của mình. Trong kịch bản của TweetAnalyzer, tài nguyên là các tweet nằm bên trong Resource Server và nếu bạn lấy kịch bản của Google Photos, thì ảnh sẽ là tài nguyên. Ở đây, người dùng của tôi đang nói với client application, tôi muốn truy cập tài nguyên của mình từ Resource Server của mình.

**Sau bước đầu tiên này**, client application của tôi sẽ thông báo cho user, Resource Server của chúng tôi nói rằng vui lòng thông báo cho Authorization Server rằng bạn có thể thực hiện hành động này vì những tài nguyên này mà Resource Owner của tôi đang cố truy cập, chúng được lưu trữ bên trong

Resource Server và bắt cứ khi nào tôi muốn nói chuyện với Resource Server, trước tiên, **tôi cần nói chuyện với Authorization Server và lấy quyền truy cập**. Đó là lý do tại sao client application của tôi sẽ thông báo cho người dùng vui lòng làm việc với Authorization Server và cố gắng gửi thông tin đăng nhập của bạn bên trong Authorization Server và cung cấp sự đồng ý của bạn.

Đó là những gì tôi đang cố gắng truyền đạt tới user trong bước thứ hai và sau khi hoàn thành bước thứ hai đó, user của tôi sẽ được chuyển hướng đến trang đăng nhập Authorization Server và tại đây, user của tôi, Resource Owner của chúng tôi, anh ấy có thể nói với Authorization Server, giống như đây là thông tin đăng nhập của tôi.

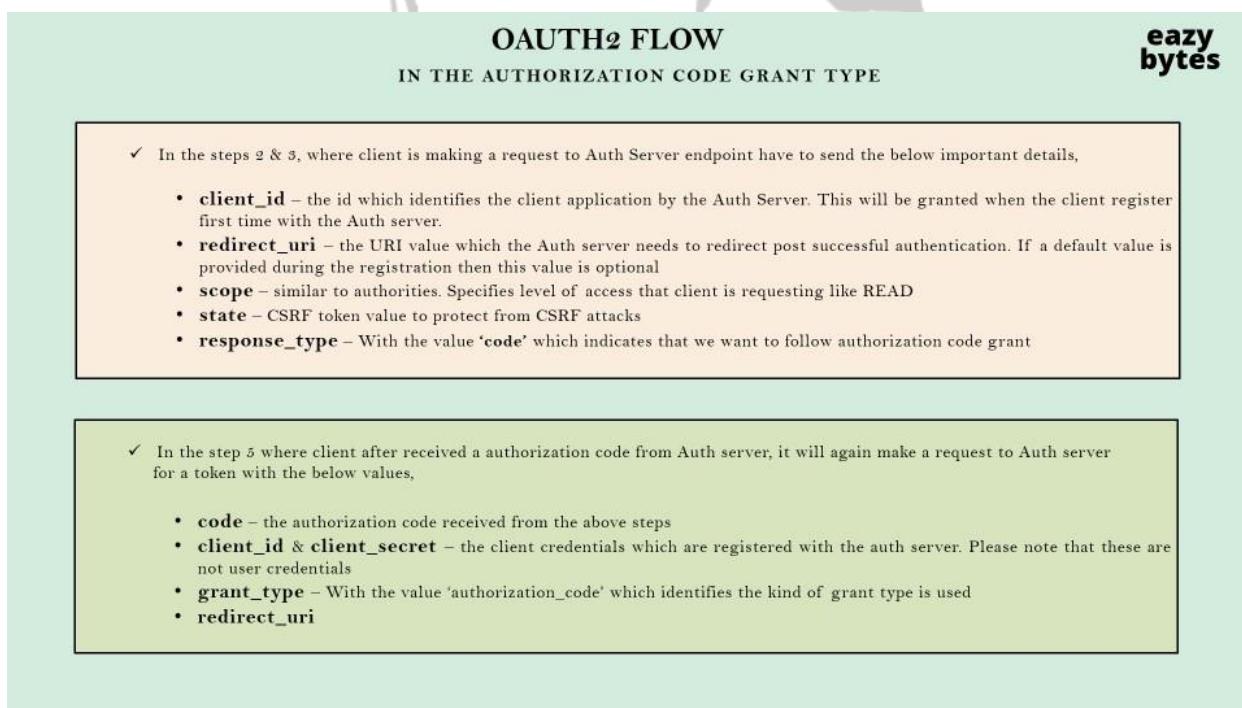


Vui lòng cho phép khách hàng như vậy truy cập tài nguyên của tôi. Và sau khi Authorization Server nhận được thông tin đăng nhập của anh ấy và sau khi Resource Owner của tôi chứng minh danh tính của anh ấy, Authorization Server của tôi sẽ cấp temporary authorization code. Nó sẽ không trực tiếp phát hành access token vì trong các bước đầu tiên của một, hai, ba, bốn, Authorization Server của tôi sẽ chỉ cố gắng xác thực danh tính user bằng cách lấy thông tin đăng nhập của anh ta. Sau khi xác thực Resource Owner thành công, Authorization Server của tôi sẽ gửi authorization code đến client application và authorization code này sẽ tồn tại trong một khoảng thời gian rất ngắn, có thể trong vài giây. Và ngay sau khi client application của tôi nhận được authorization code, nó phải thực hiện thêm một yêu cầu tới Authorization Server nói rằng tôi đã authorization code, code này xác nhận rằng xác thực người dùng thành công và anh ấy đồng ý cho tôi truy cập vào resources. Vì vậy, bây giờ tôi muốn access token.

Để cung cấp access token, Authorization Server của tôi sẽ yêu cầu thông tin đăng nhập của khách hàng, chẳng hạn như client ID, client secret là gì và authorization code mà máy chủ nhận được trong bước bốn là gì? Sau khi tất cả authorization code, client ID, client secret này, mọi thứ đều hợp lệ, thì Authorization Server của tôi sẽ cấp access token cho client application của tôi.

Bây giờ, client application của tôi sẽ nói hoan hô, tôi đã nhận được access token. Bây giờ tôi có thể truy cập Resource Server và tôi có thể thử truy cập vào các tài nguyên được bảo vệ của Resource Owner của mình và với access token, nó sẽ đưa ra yêu cầu tới Resource Server nói rằng đây là access token. Tôi muốn truy cập các tài nguyên như vậy của Resource Owner như vậy. Sau khi Resource Server của tôi hài lòng rằng access token đã cho là hợp lệ, đồng thời, client application của tôi có đủ phạm vi hoặc đủ quyền để truy cập vào tài nguyên cụ thể, thì Resource Server của tôi sẽ trả lại tất cả các tài nguyên mà client application của tôi đã yêu cầu bên trong phản hồi như một phần của bước tám. Vì vậy, đây là **quy trình hoàn chỉnh của authorization code grant type**.

Bạn có thể thấy có một xác minh hai bước. Một là ở bước đầu tiên, Authorization Server sẽ xác thực xem Resource Owner của tôi có hợp lệ hay không bằng cách hỏi thông tin đăng nhập của anh ấy. Ở cấp độ bảo mật thứ hai, Authorization Server sẽ mong đợi **Authorization Code** mà nó tạm thời nhận được ở bước bốn, cùng với client ID và client secret từ client application. Nếu tất cả những điều này đều thỏa mãn, Authorization Server sẽ cấp access token. Vì vậy, ở đây, bạn có thể có các câu hỏi như làm thế nào Authorization Server sẽ biết nơi nó phải chuyển hướng trở lại sau khi xác thực thành công? Tất cả những chi tiết đó, client application sẽ gửi, như bạn có thể thấy trong bước hai và ba, khi nó đang cố lấy authorization code, client application của tôi sẽ chỉ gửi client\_id. Nó sẽ không gửi client\_secret vì client\_secret phải được gửi trong bước tiếp theo khi nó cố lấy access token.



Vì vậy, client\_id đầu tiên sẽ được gửi và redirect\_uri, chi tiết URI nơi Authorization Server cần chuyển hướng đăng nhập xác thực thành công Resource Owner của tôi. Và trong scope, chúng tôi cần cung cấp các quyền mà tôi đang yêu cầu để các chi tiết scope tương tự sẽ được hiển thị cho Resource Owner, chẳng hạn như client application này đang cố truy cập email, ảnh hoặc tweet của bạn. Vì vậy, tất cả các chi tiết scope đó, nó sẽ hiển thị cho user và nó sẽ cố gắng lấy sự đồng ý từ user. Vì vậy, đó là lý do tại sao client application gửi chi tiết scope là rất quan trọng.

Và khách hàng cũng nên gửi một giá trị state. Giá trị state sẽ hoạt động như một CSRF token để bảo vệ khỏi các cuộc tấn công CSRF. Bất kể giá trị state nào mà client application của tôi đang gửi đến Authorization Server, thì giá trị trạng thái tương tự phải được Authorization Server gửi lại cho client application để đảm bảo không có ai ở giữa đã giả mạo yêu cầu của tôi và đây là giá trị thực và chính hãng phản hồi mà Authorization Server đang gửi. Vì vậy, đó là lý do tại sao state sẽ hoạt động như một CSRF token bên trong authorization code grant type flow. Và cuối cùng, chúng ta cũng nên gửi response\_type vì chúng ta chỉ muốn nó nhận authorization code như một phần của bước hai và ba. Chúng ta nên gửi response\_type dưới dạng code.

Khi client application của tôi nhận được authorization code bên trong tệp bước, lần này, nó phải gửi code, là authorization code mà nó nhận được từ các bước trên. Và cùng với code, nó cũng phải chứng minh danh tính của chính nó. Vì vậy, đó là lý do tại sao client application của tôi, cùng với client\_id, lần này nó cũng nên chia sẻ client\_secret. Vì vậy, những client\_id, client\_secret, thường thì client application của tôi sẽ nhận được trong quá trình đăng ký mà nó đã thực hiện với Authorization Server. Nhưng vui lòng đảm bảo rằng bạn hiểu rõ rằng client\_id, client\_secret này không phải là thông tin đăng nhập của người dùng. Xác thực người dùng đã được tính toán trong các bước trên. Và tham số tiếp theo mà nó phải vượt qua là grant\_type là gì? Lần này, grant\_type mà chúng tôi cần gửi là authorization\_code, có nghĩa là tôi đang sử dụng authorization code grant type flow. Vui lòng cấp cho tôi access token. Vì vậy, đó là những gì chúng tôi đang cố gắng truyền đạt với sự trợ giúp của grant\_type.

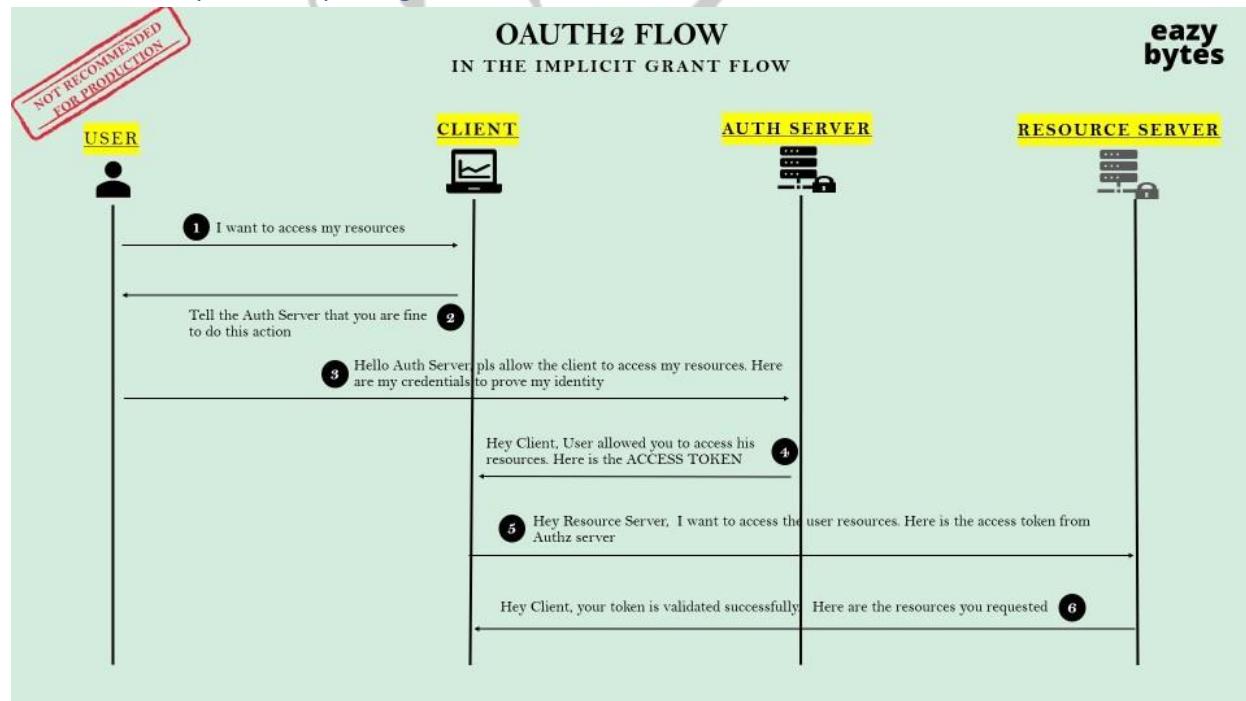
Và cuối cùng, chúng ta cũng nên gửi redirect\_uri. Trang nơi Authorization Server phải chuyển hướng phản hồi trở lại cùng với access token. Vì vậy, ở đây bạn có thể có một câu hỏi như bên trong authorization code grant type flow, tại sao khách hàng thực hiện yêu cầu hai lần? Một là authorization code và lần thứ hai là access token. Như tôi đã nói trước đây, trong bước đầu tiên, Authorization Server sẽ đảm bảo người dùng tương tác trực tiếp với nó, cùng với thông tin đăng nhập. Nếu thông tin chi tiết là chính xác và nếu Resource Owner hợp lệ, client application của tôi sẽ nhận được authorization code tạm thời. Và trong bước thứ hai, bản thân khách hàng của tôi phải chứng minh danh tính của mình và đó là lý do tại sao lần này, cùng với authorization code, khách hàng cũng phải client secret, client ID để lấy access token. Vậy tại sao hai bước khác nhau có nghĩa là chỉ để bảo mật tốt hơn, vì vậy nếu bạn chỉ có một yêu cầu đi từ máy khách đến Authorization Server, thì mọi người rất dễ dàng hack luồng. Đó là lý do tại sao với hai bước, chúng tôi đang đảm bảo flow của mình tốt hơn rất nhiều.

- ✓ We may wonder that why in the Authorization Code grant type client is making request 2 times to Auth server for authorization code and access token.
- In the first step, authorization server will make sure that user directly interacted with it along with the credentials. If the details are correct, auth server send the authorization code to client
  - Once it receives the authorization code, in this step client has to prove it's identity along with the authorization code & client credentials to get the access token.

- ✓ Well you may ask why can't Auth server directly club both the steps together and provide the token in a single step. The answer is that we used to have that grant type as well which is called as '**implicit grant type**'. But this grant type is not recommended to use due to it's less secure.

Và tất nhiên, bạn có thể có một câu hỏi như Authorization Server không thể kết hợp trực tiếp cả hai bước này với nhau và cung cấp token trong một bước duy nhất? Vì vậy, câu trả lời là chúng tôi có một loại grant type hỗ trợ loại hành vi này. Grant type này, chúng tôi gọi nó là implicit grant type. Nhưng loại cấp phép này không được khuyến nghị sử dụng vì nó kém an toàn hơn và không được dùng nữa, đồng thời loại grant type này đang bị xóa khỏi phiên bản OAuth 2.1.

## 5. Deep dive implicit grant flow in OAUTH2



**Implicit grant flow** không còn được dùng từ lâu vì nó rất kém an toàn và mọi người được khuyến nghị sử dụng authorization code grant type flow bất cứ khi nào có user tham gia. Và điều này sẽ bị xóa hoàn toàn khỏi phiên bản OAUTH2.1. Chúng ta hãy cố gắng tìm hiểu Implicit grant flow để bạn có thể nhận được sự khác biệt giữa authorization code grant type flow và implicit grant flow. Ở bước đầu tiên, user hoặc Resource Owner của tôi sẽ gửi yêu cầu tới client application nói rằng, "Tôi muốn truy cập tài nguyên của mình như ảnh hoặc tweet của tôi từ Resource Server ." Và như một phản hồi, client application của tôi sẽ nói với user của tôi, "Được rồi, tôi sẽ chuyển hướng bạn đến Authorization Server. Vui lòng nhập thông tin đăng nhập của bạn. Khi quá trình xác thực của bạn hoàn tất, vui lòng đồng ý rằng bạn ổn để làm động tác này." Vì vậy, bây giờ user của tôi sẽ được chuyển hướng đến trang đăng nhập Authorization Server. Và tại đây, user hoặc Resource Owner của tôi, anh ấy có thể nhập thông tin đăng nhập của mình để chứng minh danh tính của mình và anh ấy có thể đưa ra sự đồng ý của mình. Sau khi xác thực thành công, Authorization Server của tôi sẽ trực tiếp cấp access token. Không có Authorization Code ở giữa. Trong một bước rất đơn giản, client của tôi sẽ nhận được access token. Và sử dụng access token này, client application của tôi có thể gửi yêu cầu tới Resource Server. Và nếu access token hợp lệ, Resource Server của tôi sẽ phản hồi bằng phản hồi thích hợp.

Và lý do tại sao nó rất kém an toàn là nếu bạn cố gắng hiểu danh sách chi tiết mà chúng tôi cần gửi đến Authorization Server trong implicit grant flow, chúng tôi chỉ nên gửi client ID và the redirect URI, scope và state and response type. Giá trị loại phản hồi phải là token.

**OAUTH2 FLOW**  
IN THE RESOURCE OWNER CREDENTIALS GRANT TYPE



✓ In the step 2, where client is making a request to Auth Server endpoint have to send the below important details,

- **client\_id & client\_secret** – the credentials of the client to authenticate itself.
- **scope** – similar to authorities. Specifies level of access that client is requesting like READ
- **username & password** – Credentials provided by the user in the login flow
- **grant\_type** – With the value 'password' which indicates that we want to follow password grant type

✓ We use this authentication flow only if the client, authorization server and resource servers are maintained by the same organization.

✓ This flow will be usually followed by the enterprise applications who want to separate the Auth flow and business flow. Once the Auth flow is separated different applications in the same organization can leverage it.

Bởi vì đây là một request, không có nghĩa là gửi client secret đến Authorization Server bởi vì bất cứ khi nào bạn gửi client secret trong request, mọi thứ sẽ bị lộ bên trong URL của bạn. Đó là lý do tại sao bên trong implicit grant flow, client application của tôi không phải chia sẻ client secret. Cùng với đó, có một vấn đề nghiêm trọng. Bất kỳ ai cũng có thể bắt chước như thể tôi là một ứng dụng phân tích tweet và gửi yêu cầu tới Authorization Server. Vì vậy, đó là một nhược điểm rất đầu tiên.

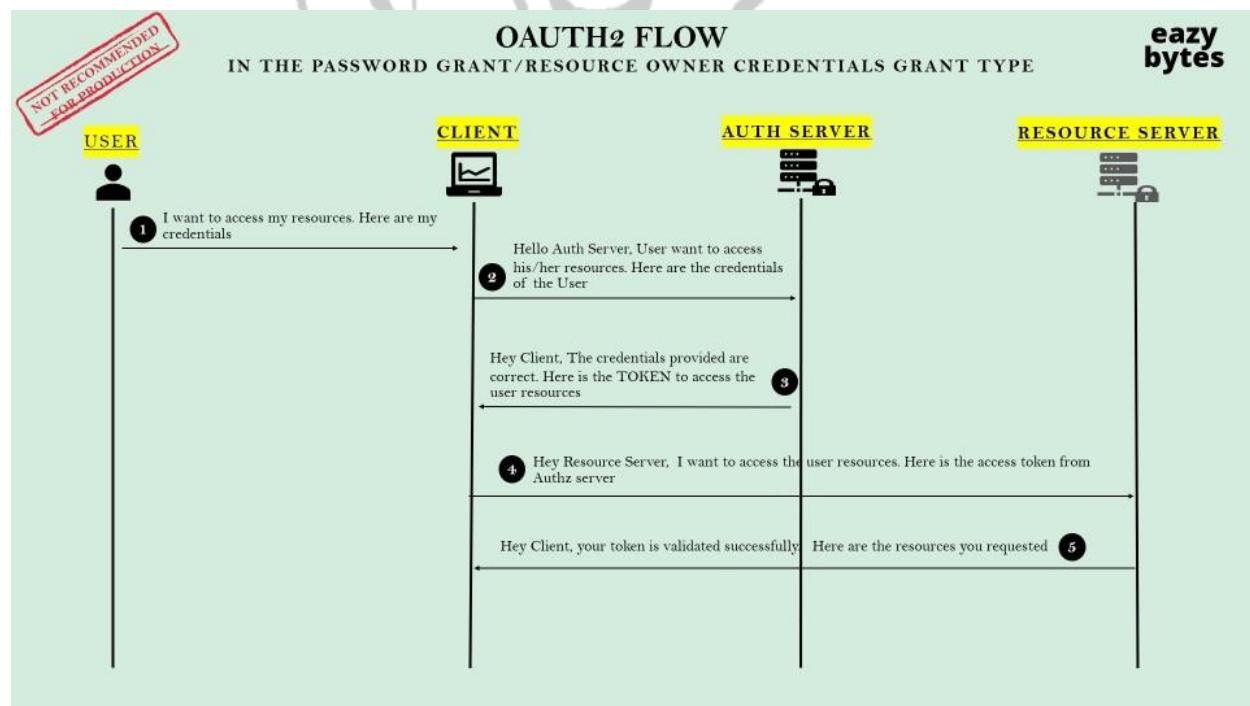
Và nhược điểm thứ hai mà chúng tôi gặp phải là vì chúng tôi đang thực hiện một request, nên chắc chắn phản hồi từ Authorization Server mà chúng tôi cũng sẽ nhận được dưới dạng response. Và access token, bất kể Authorization Server nào đang cấp lại cho client application, nó sẽ trở thành một phần của chính URL, điều này lại rất không an toàn. Bất kỳ tin tức nào, anh ta có thể lấy access token hoặc anh ta có thể cố gắng chỉnh sửa access token mà không thay đổi giá trị CSRF token. Vì nó có rất nhiều nhược điểm và không có cách nào bên trong luồng loại cấp phép OAUTH2 để biến implicit grant flow này thành một luồng bảo mật. Đó là lý do tại sao có một luồng mới được phát minh, đó là authorization code grant type flow.

## 6. Deep dive of password grant type flow in OAUTH2

Một số người, họ gọi luồng này là luồng flow as resource owner credentials grant flow và một số ít người, họ gọi nó là **password grant type flow**. Cả hai đều đề cập đến cùng một flow. Bên trong flow này cũng có user tham gia và client application sẽ tham gia cùng với Authorization Server và Resource Server.

Ở bước đầu tiên, những gì user hoặc resource owner của tôi sẽ làm là anh ấy sẽ truy cập client application và anh ấy sẽ thông báo rằng, "Tôi muốn truy cập tài nguyên này và tài nguyên khác" và anh ấy sẽ trực tiếp cung cấp thông tin đăng nhập của mình cho client application. Có thể client application có thể là một số loại UI application chấp nhận thông tin đăng nhập thực tế của Resource Owner của tôi mà anh ấy đã đăng ký với Authorization Server. Đây là điểm rất quan trọng mà bạn cần phải hiểu.

Ngay sau khi bạn nhận ra rằng user đang thực sự chia sẻ thông tin xác thực của chính họ về Authorization Server, thì bạn có thể dễ dàng xác định rằng luồng OAUTH2 là loại **password grant type flow**. Khi client application của tôi nhận được thông tin đăng nhập thực tế từ user của tôi, nó sẽ gửi yêu cầu có thể ở dạng yêu cầu tới Authorization Server, nói rằng user muốn truy cập tài nguyên của họ.



Đây là thông tin đăng nhập của user và cùng với thông tin đăng nhập của user, client application cũng sẽ chuyển client ID client secret của riêng mình. Sau khi tất cả các chi tiết này được Authorization Server của tôi xác thực, nó sẽ trực tiếp phát hành access token và sau khi client application của tôi nhận được access token, token tương tự sẽ được gửi đến Resource Server. Và nếu access token hợp lệ, Resource Server của tôi sẽ gửi phản hồi trả lại client application của tôi. Ở đây bên trong flow này, nhược điểm chính là Resource Owner của tôi. Anh ấy đang chia sẻ thông tin đăng nhập thực tế của mình cho chính client application. Và vì lý do đó, điều này không được khuyến nghị cho các ứng dụng product.

Hãy cố gắng hiểu chi tiết yêu cầu mà client application của tôi sẽ gửi đến Authorization Server bên trong bước hai để nhận access token là gì.

**OAUTH2 FLOW**  
IN THE RESOURCE OWNER CREDENTIALS GRANT TYPE

**eazy bytes**

✓ In the step 2, where client is making a request to Auth Server endpoint have to send the below important details,

- **client\_id** & **client\_secret** – the credentials of the client to authenticate itself.
- **scope** – similar to authorities. Specifies level of access that client is requesting like READ
- **username & password** – Credentials provided by the user in the login flow
- **grant\_type** – With the value 'password' which indicates that we want to follow password grant type

✓ We use this authentication flow only if the client, authorization server and resource servers are maintained by the same organization.

✓ This flow will be usually followed by the enterprise applications who want to separate the Auth flow and business flow. Once the Auth flow is separated different applications in the same organization can leverage it.

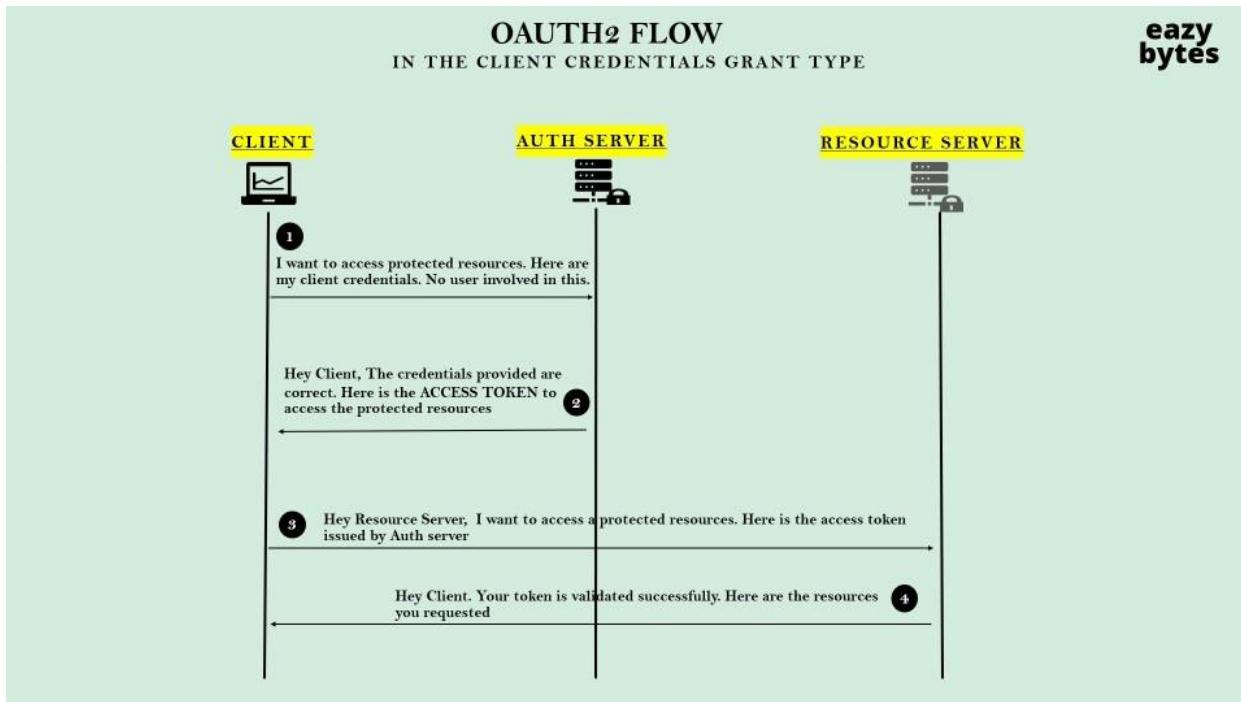
Bạn có thể thấy ở đây trong bước hai, Client Application của tôi sẽ client ID, client secret, scope và cùng với các chi tiết này, nó cũng sẽ chuyển tên người dùng và mật khẩu của user của tôi là gì. Và tên người dùng và mật khẩu này có thể đã được user của tôi cung cấp như một phần của quy trình đăng nhập vào Client Application của tôi. Và grant type mà bạn cần đề cập trong trường hợp này là password. Tôi hy vọng bạn cũng nhận ra rằng dựa trên grant type này, Authorization Server của tôi sẽ cố gắng hiểu luồng loại cấp phép mà Client Application của tôi muốn xem xét là gì. Vì vậy, trong trường hợp này, nếu Authorization Server của tôi nhận được loại cấp dưới dạng mật khẩu, thì nó sẽ cho rằng Client Application muốn tuân theo loại cấp thông tin xác thực của resource owner.

Mặc dù tôi đã nói rằng resource order credentials grant type này không được khuyến nghị cho các ứng dụng product, nhưng đôi khi bạn vẫn sẽ thấy chúng được sử dụng trong một số tổ chức. Kịch bản duy nhất mà bạn thấy một số tổ chức đang sử dụng grant type flow này là tất cả Client Application, Authorization Server và Resource Server đều thuộc về cùng một tổ chức. Vì vậy, nếu Authorization Server của tôi có thể tin vào Client Application của tôi, điều đó có nghĩa là họ sẽ không lạm dụng thông tin đăng nhập của Resource Owner của tôi, thì họ có thể sử dụng loại resource owner credentials grant type này vì họ cũng có một điểm hợp lệ, nói rằng Client Application cũng chỉ thuộc về tổ chức của tôi.

## 7. Deep dive of client credentials grant type flow in OAUTH2

Chúng ta có thể sử dụng **grant type** này bất cứ khi **nào không có user tham gia và trong các tình huống là hai back-end application hoặc hai microservices application đang cố gắng giao tiếp với nhau.**

Có thể back-end application hoặc microservice application khác nhau này có thể thuộc về các tổ chức khác nhau và chúng muốn giao tiếp với nhau. Vì vậy, ở đây, bạn có thể nghĩ như client là một microservice application, thuộc về tổ chức A và microservice client của tổ chức A này muốn giao tiếp với các microservices được triển khai bên trong Resource Server của tổ chức B. Nếu chúng muốn giao tiếp với nhau, thì chắc chắn là microservice client của tôi hoặc client service của tôi, nó phải kết nối với Auth Server, nó phải lấy access token thì mới kết nối được với Resource Server của tổ chức tôi B. Lý do là cả Auth Server và Resource Server thuộc cùng một tổ chức B



Ở bước đầu tiên, client application sẽ gửi thông tin chi tiết đến Auth Server, nói rằng, "Đây là thông tin đăng nhập khách hàng của tôi, không có user nào tham gia. Vui lòng xác thực thông tin đăng nhập khách hàng của tôi và cấp access token của tôi." Khi thông tin đăng nhập của client application được xác thực, Auth Server của tôi sẽ cấp access token cho client application. Và khi client application phát hành access token, nó sẽ gọi trực tiếp dịch vụ có sẵn bên trong Resource Server. Nếu access token hợp lệ, Resource Server của tôi sẽ đưa ra phản hồi thích hợp cho client application. Đây là một luồng OAuth2 rất đơn giản, có thể được tận dụng trong các tình huống mà user không tham gia. Bây giờ, hãy cố gắng hiểu yêu cầu mà client application phải gửi đến Auth Server là gì, chứ không phải để nhận access token.

- ✓ In the step 1, where client is making a request to Auth Server endpoint, have to send the below important details,

- **client\_id & client\_secret** – the credentials of the client to authenticate itself.
- **scope** – similar to authorities. Specifies level of access that client is requesting like READ
- **grant\_type** – With the value '**client\_credentials**' which indicates that we want to follow client credentials grant type

- ✓ This is the most simplest grant type flow in OAUT<sup>H</sup>2.
- ✓ We use this authentication flow only if there is no user and UI involved. Like in the scenarios where 2 different applications want to share data between them using backend APIs.

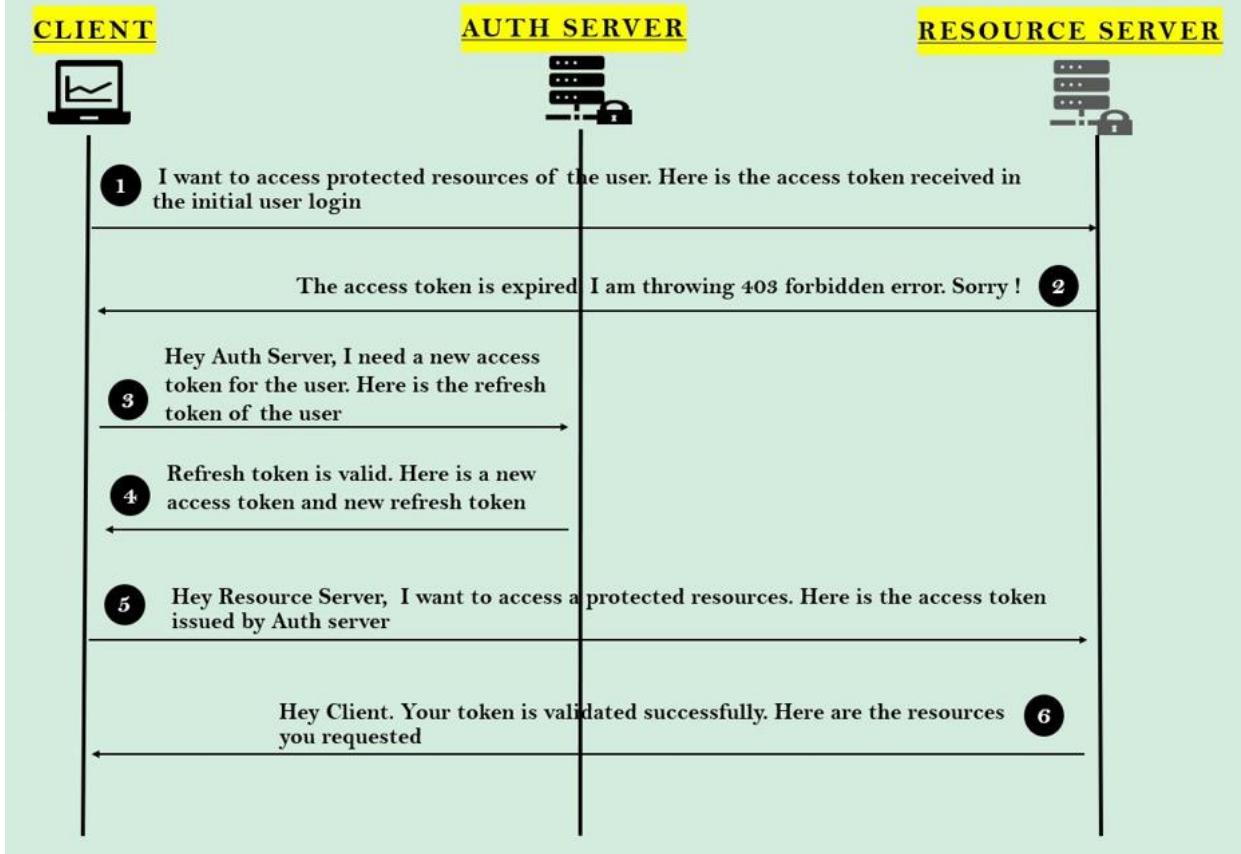
Trong bước đầu tiên, chính client application, khi đến lúc gửi yêu cầu tới Auth Server để lấy access token, nó phải gửi client\_id, client\_secret, scope là gì, cho dù nó muốn đến, cố gắng lấy quyền READ hoặc quyền WRITE và sau giá trị scope, nó sẽ gửi grant\_type dưới dạng client\_credentials. Vì vậy, bất cứ khi nào Auth Server nhận được loại cấp phép này là client\_credentials, nó sẽ biết rằng không có user nào tham gia. Nếu client\_id và client\_secret hợp lệ, nó sẽ trực tiếp gửi access token trả lại client server. Và ở đây, không có (không rõ ràng) liên quan, không có giá trị state nào liên quan để CSRF token, bởi vì tất cả giao tiếp này sẽ không xảy ra thông qua trình duyệt và bất cứ khi nào trình duyệt không ở đó, sẽ không có khả năng xảy ra CSRF các cuộc tấn công.

## 8. Deep dive of refresh token grant type flow in OAUT<sup>H</sup>2

Khi chúng tôi kiểm tra authorization code grant type flow bên trong trang web, phản hồi mà Auth Server của tôi gửi lại cho client application, chúng tôi đã nhận được hai loại token. Cái chính là access\_token và cái thứ hai là refresh\_token, và cũng có một thông tin như access token này sẽ hết hạn sau vài giây. Vì vậy, ở đây 86.400 giây có nghĩa là nó sẽ hết hạn sau 24 giờ, nghĩa là sau một ngày. Vì vậy, bất cứ khi nào access token của tôi hết hạn, client application của tôi không phải yêu cầu user hoặc Resource Owner nhập lại thông tin đăng nhập của họ hoặc yêu cầu lại để bắt đầu quy trình đăng nhập với sự trợ giúp của authorization code grant type flow.

## OAUTH2 FLOW

### IN THE REFRESH TOKEN GRANT TYPE



Thay vào đó, nó có thể tận dụng refresh token này mà nó nhận được ban đầu từ Authorization Server. Với refresh token mà nó nhận được, nó có thể bắt đầu một luồng loại cấp phép khác với Authorization Server. Và nếu refresh token hợp lệ, Authorization Server của tôi sẽ trả lại access token mới với thời gian hết hạn mới. Cùng với đó, nó cũng sẽ phát hành một refresh token mới. Vì vậy, đó là mục đích của refresh token này. Tôi hy vọng bạn hiểu rõ, chẳng hạn như thay vì yêu cầu người dùng cuối hoặc Resource Owner bắt đầu lại thao tác đăng nhập sau 24 giờ, client application của tôi, họ luôn có thể sử dụng refresh token này và đăng sau hậu trường, bất cứ khi nào họ nhận được phản hồi từ Auth Server thông báo rằng access token cụ thể đã hết hạn, trong nội bộ, họ có thể gửi thêm một yêu cầu đến Auth Server dưới dạng refresh token grant type flow.

Trong refresh token grant type flow, không có người dùng cuối nào tham gia vì client application của tôi, ngay khi nhận được thông báo rằng một access token cụ thể đã hết hạn, nó sẽ không liên quan đến user của tôi. Nó chỉ đơn giản là tận dụng refresh token mà nó nhận được trong hoạt động đăng nhập ban đầu. Và thông thường, các refresh token này sẽ không có thời gian hết hạn. Hãy cố gắng hiểu cách thức hoạt động của luồng cấp loại refresh token này.

Ở bước đầu tiên, client application của tôi, như thường lệ, gửi yêu cầu đến Resource Server, tôi muốn truy cập các tài nguyên tương tự. Đây là access token. Nó không bao giờ biết rằng một access

token cụ thể đã hết hạn. Nhưng vì access token đã hết hạn, Resource Server của tôi sẽ gửi lỗi 403 bị cấm, nói rằng access token đã hết hạn, xin lỗi. Cùng với đó, client application của tôi biết phải làm gì. Nó sẽ gửi một yêu cầu mới đến Auth Server, nói rằng tôi cần một access token mới cho người dùng tương tự. Đây là refresh token mà bạn đã cấp trong thao tác đăng nhập ban đầu. Nếu refresh token hợp lệ, Auth Server của tôi sẽ cấp cho tôi access token mới và refresh token mới. Và refresh token mới, client application của tôi phải lưu trữ nó ở đâu đó, bên trong trình duyệt hoặc bên trong cơ sở dữ liệu, bất cứ nơi nào nó muốn.

Và client application của tôi sử dụng access token mới mà nó nhận được, nó sẽ gửi yêu cầu tới Resource Server. Và Resource Server của tôi, vì access token lần này là hợp lệ, nên nó sẽ gửi phản hồi thích hợp trả lại client application của tôi. Một lần nữa, như bạn có thể thấy, client application có trách nhiệm phát hiện trường hợp access token hết hạn đó và bắt đầu tất cả các loại cấp refresh token này ở phía sau hậu trường. Nó không bao giờ liên quan đến user và user của tôi sẽ không bao giờ cảm thấy như có điều gì đó đang xảy ra đăng sau hậu trường.

Hãy thử xem yêu cầu mà client application của tôi phải gửi đến Auth Server bất cứ khi nào nó đang cố bắt đầu refresh token grant type. Giống như bạn có thể thấy ở đây, trong bước ba, client application của tôi, nó phải gửi `client_id` và `client_secret` đến Auth Server.

**OAUTH2 FLOW**  
IN THE REFRESH TOKEN GRANT TYPE

**eazy bytes**

✓ In the step 3, where client is making a request to Auth Server endpoint have to send the below important details,

- `client_id` & `client_secret` – the credentials of the client to authenticate itself.
- `refresh_token` – the value of the refresh token received initially
- `scope` – similar to authorities. Specifies level of access that client is requesting like READ
- `grant_type` – With the value 'refresh\_token' which indicates that we want to follow refresh token grant type

- This flow will be used in the scenarios where the access token of the user is expired. Instead of asking the user to login again and again, we can use the refresh token which originally provided by the Authz server to reauthenticate the user.
- Though we can make our access tokens to never expire but it is not recommended considering scenarios where the tokens can be stole if we always use the same token
- Even in the resource owner credentials grant types we should not store the user credentials for reauthentication purpose instead we should reply on the refresh tokens.

Nó cũng sẽ gửi `refresh_token` mà nó nhận được ban đầu là gì. Và sau `refresh_token`, nó phải gửi chi tiết `scope`. Và thông tin quan trọng cuối cùng mà client application của tôi phải gửi là `Grant_type`. Vì vậy, đối với grant type này, giá trị mà nó phải gửi là `refresh token` và `the scope`. Vì vậy, dựa trên giá trị này, Auth Server của tôi biết rằng nó phải lấy `refresh token` và cấp `access token` mới và `refresh token` mới.

Ở đây bạn có thể có một câu hỏi rằng tại sao chúng ta không thể tạo `access token` của mình cũng không bao giờ hết hạn để chúng ta không bao giờ phải phụ thuộc vào `refresh token` này? Nhưng đó

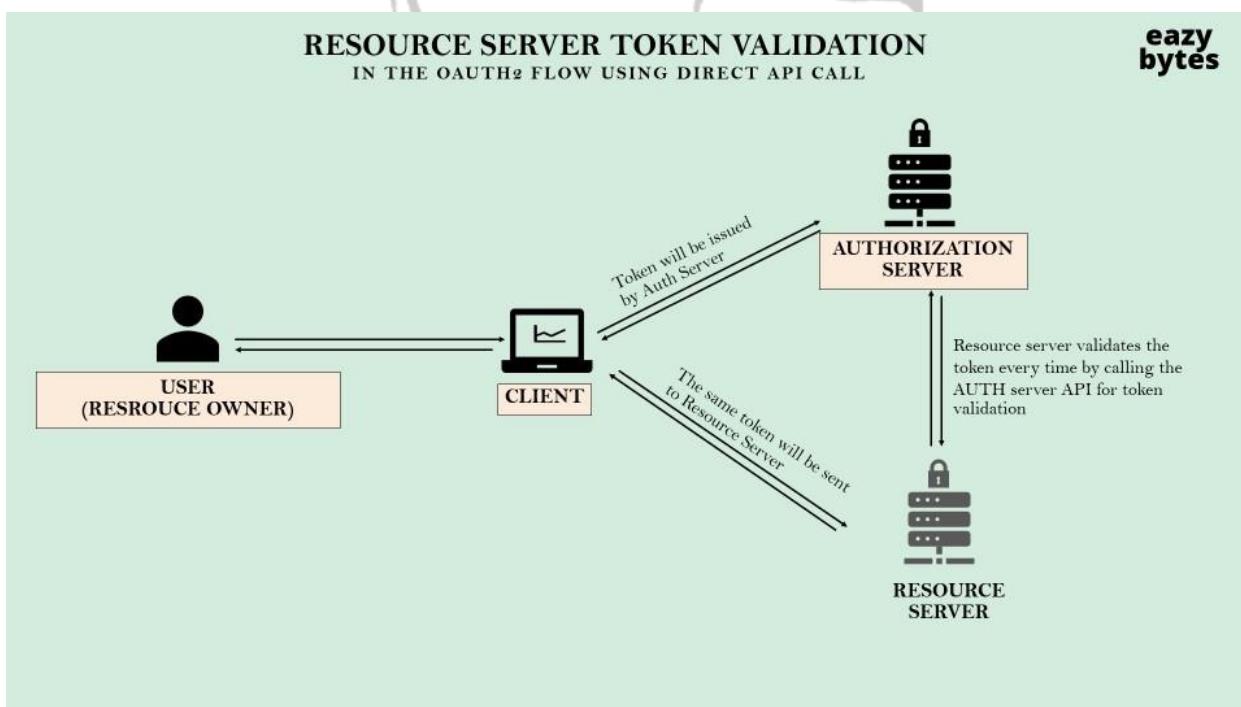
không phải là một cách tiếp cận được khuyến nghị. Lý do là nếu access token của bạn sẽ không hết hạn bất cứ lúc nào và nếu ai đó có quyền truy cập vào access token này, thì nguy hiểm nhất là họ giữ tên người dùng và mật khẩu thực của bạn. Đó là lý do tại sao hầu hết các Auth Server từ các tổ chức lớn hơn, họ sẽ không bao giờ cấp access token với thời gian không giới hạn hoặc không bao giờ hết hạn. Họ sẽ luôn đảm bảo 24 giờ hoặc 7 ngày dựa trên yêu cầu kinh doanh của họ.

Một lần trong quá trình đăng nhập, nếu nhận được access token và refresh token, chúng ta không nên hỏi đi hỏi lại thông tin đăng nhập của resource owner. Thay vào đó, client application có thể tận dụng refresh token và cố gắng lấy access token mới.

## 9. How resource server validates the tokens issued by Auth server

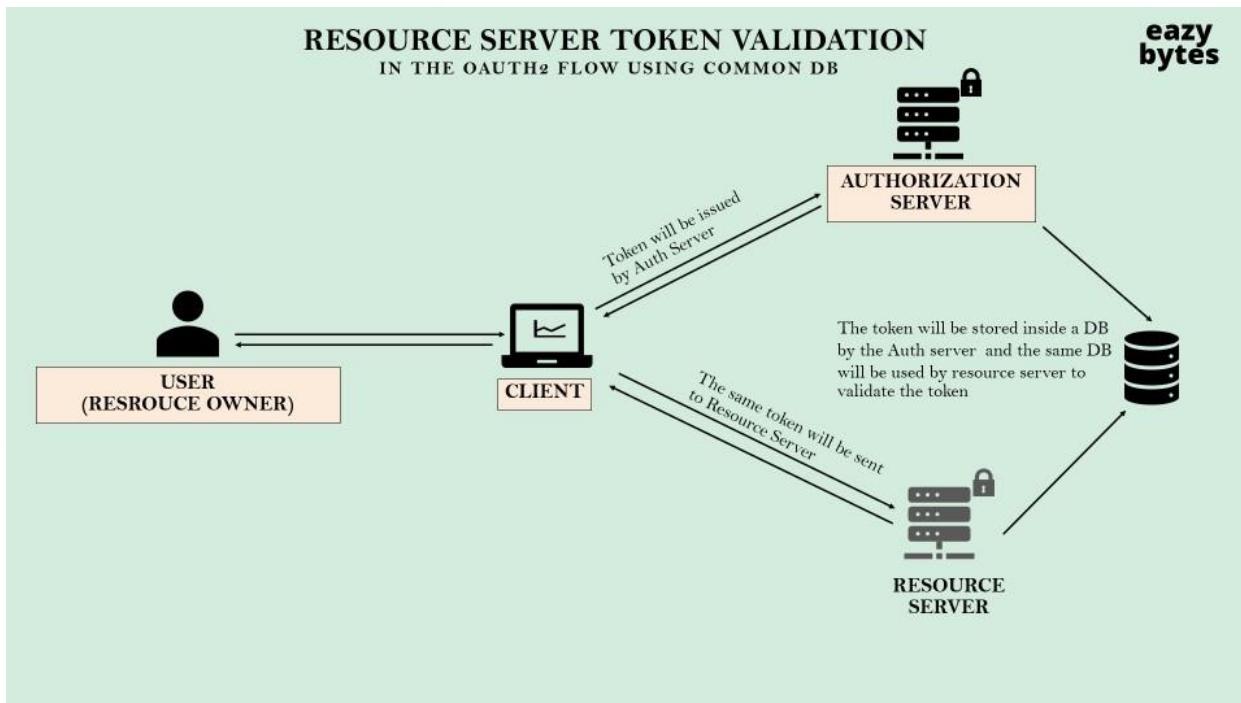
Trong tất cả các OAuth 2 grant flows mà chúng ta đã thảo luận trước đây, Auth Server của tôi luôn chịu trách nhiệm cấp access token cho client application và cùng một access token mà client application của tôi có thể gửi đến resource server. Nhưng câu hỏi ở đây là làm thế nào resource server của tôi biết liệu access token do khách hàng của tôi cung cấp có hợp lệ hay không, làm cách nào để kiểm tra điều tương tự với Auth Server?

Chúng tôi có nhiều cách tiếp cận khác nhau. Cách tiếp cận đầu tiên là sẽ có sự tương tác trực tiếp tới API giữa Auth Server và resource server.



Bất cứ khi nào resource server của tôi nhận được access token, nó sẽ gọi một trong các API được hiển thị bởi Auth Server của tôi. Và nếu Auth Server của tôi xác nhận rằng access token như vậy là hợp lệ, thì resource server của tôi sẽ gửi phản hồi thành công trả lại client application của tôi. Vì vậy, đây là cách tiếp cận rất cơ bản, nhưng giống như bạn có thể thấy ở đây đối với mọi yêu cầu mà bạn nhận được từ client application có access token, resource server của tôi phải gọi Auth Server, do đó, có các vấn đề về hiệu suất hoặc lưu lượng truy cập không cần thiết với phương pháp này.

Cách tiếp cận thứ hai là cả Auth Server và resource server của tôi, chúng có thể sử dụng một cơ sở dữ liệu chung. Những gì Auth Server của tôi có thể làm là bất cứ khi nào nó đang cấp access token cho client application, nó có thể tạo một mục bên trong cơ sở dữ liệu này.

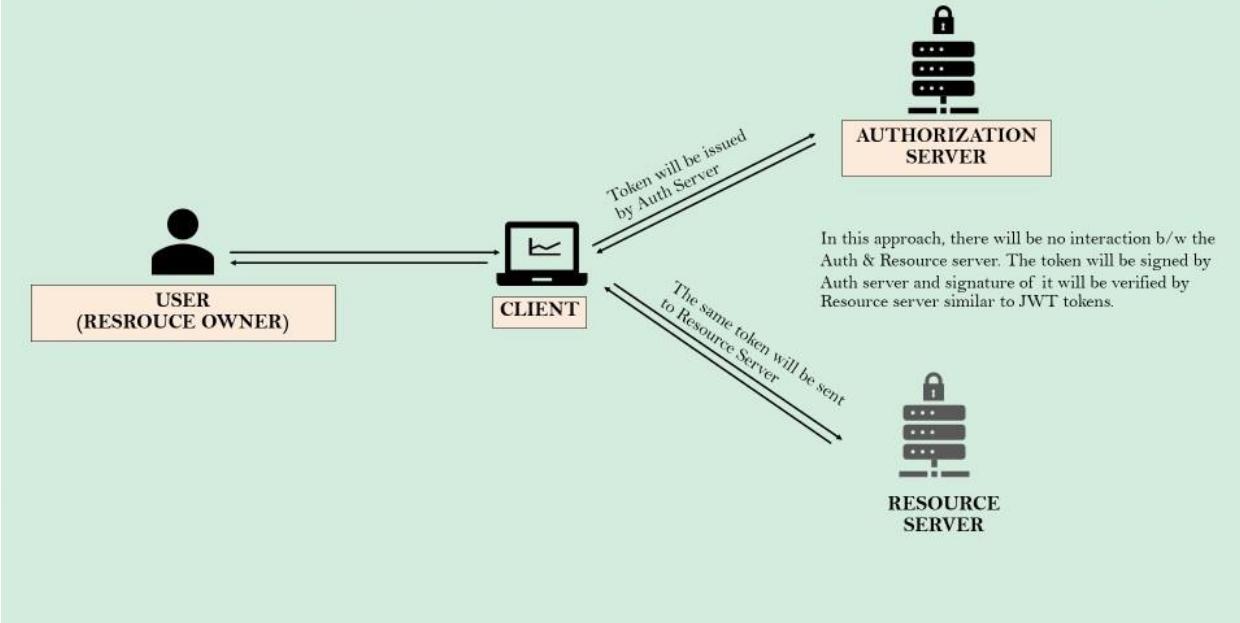


Và khi cùng một access token được client application của tôi gửi đến resource server, resource server của tôi có thể truy vấn cơ sở dữ liệu chung và nếu access token đã cho có trong cơ sở dữ liệu chung ở trạng thái hoạt động, thì nó có thể gửi phản hồi thành công tới client application của tôi.

Và phương pháp cuối cùng, là phương pháp được đề xuất mà chúng ta có thể làm theo, là resource server của tôi có thể kết nối với Auth Server, có thể trong quá trình khởi động ứng dụng web và nó có thể lấy public certificate từ authorization server của tôi. Vì vậy, bằng cách sử dụng cùng một public certificate bất cứ khi nào nó nhận được public certificate do Auth Server của tôi cấp, nó có thể xác minh xem access token có bị giả mạo hay không hoặc liệu nó có hợp lệ hay không bằng cách sử dụng khóa chung.

## RESOURCE SERVER TOKEN VALIDATION IN THE OAUTH2 FLOW USING CERTIFICATES

eazy  
bytes



Điều này hoạt động rất giống với sự trợ giúp của chữ ký điện tử mà chúng ta đã thảo luận trong phần JWT token. Trong phương pháp này, sẽ không có bất kỳ giao tiếp nào cần diễn ra liên tục giữa Auth Server và resource server của bạn, chỉ lần đầu tiên resource server của tôi sẽ kết nối với Auth Server để lấy public certificate từ Auth Server. Đó là lý do tại sao đây là phương pháp được đề xuất nhiều nhất cho bất kỳ ai và bạn sẽ thấy rất nhiều phương pháp này được sử dụng trong ngành. Với điều này, tôi cho rằng bạn đã hiểu rõ về các tùy chọn khác nhau mà chúng tôi có bất cứ khi nào resource server của tôi cỗ xác thực access token bằng cách kết nối với Auth Server.

## 10. Introduction to OpenID Connect

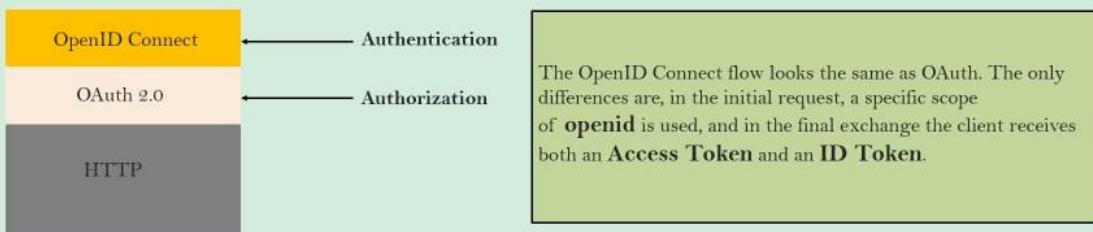
Đôi khi, bạn có thể nghe rất nhiều trong các blog hoặc từ trang web nói rằng OpenID tốt hơn framework OAuth 2.0. Vì vậy, ngày nay mọi người đang nói rất nhiều về OpenID Connect ở dạng ngắn gọn mà chúng ta cũng có thể gọi nó là OIDC. Mọi người bắt đầu đề xuất bạn sử dụng OpenID thay vì OAuth 2.0. Cùng với đó, bạn có thể có một câu hỏi như sự khác biệt giữa OAuth 2.0 và OpenID Connect. Có hai framework khác nhau không? Tôi nên sử dụng cái nào? Tại sao người bạn này lại dạy tôi tất cả về khuôn khổ OAuth 2.0 nếu OpenID là một khuôn khổ tốt hơn? Tôi biết bạn có thể có tất cả các loại câu hỏi.

Và bất cứ khi nào bạn đang sử dụng framework OAuth 2.0, bạn sẽ nhận được , bạn sẽ nhận được phạm vi mà access token đó được cấp. Với các access token và scope này, bạn có thể thực thi ủy quyền. Bạn có thể cố gắng hiểu các quyền của user của tôi tại client application nhưng vì phần lớn các ứng dụng web như ứng dụng blog hoặc ứng dụng web tĩnh nên họ nhận được địa chỉ email đã xác minh là chi tiết tài khoản đã xác minh từ Google, Facebook, GitHub, Twitter với đó là lý do họ bắt đầu sử dụng framework OAuth 2.0 ngay cả để xác thực. Giống như chúng ta đã thảo luận trước đây khi chúng ta thảo luận về trang web Slack, nơi tôi đã đưa ra bản demo về cách luồng OAuth 2.0 hoạt động với sự trợ giúp của Google đăng nhập ở đó với cùng một OAuth 2.0 mà chúng tôi đang thực hiện cả authentication và authorization nhưng từ framework thay đổi chúng tôi chỉ nhận được access token và chi tiết scope và không có cách nào với framework OAuth 2.0 để biết ai là user của tôi, thông tin chi tiết của anh ấy là gì, chẳng hạn như email của anh ấy là gì, chi tiết địa chỉ của anh ấy là gì. Không có cách nào để client application của tôi hiểu thông tin này bất cứ khi nào chúng tôi triển khai framework OAuth 2.0.

Vì lý do đó, nhiều tổ chức đang sử dụng rộng rãi OAuth 2.0 hoặc ES, họ cũng muốn biết chi tiết nhận dạng của user hoặc resource owner. Với yêu cầu đó, mọi người bắt đầu sử dụng phong cách riêng của họ để xác định hoặc gửi thông tin chi tiết người dùng đã đăng nhập tới các client application từ ủy quyền công việc(authorizations of work). Và vì mọi người bắt đầu sử dụng phong cách riêng của họ nên không có sự đồng nhất. Đó là nơi mà nhiều người đã thảo luận về vấn đề này và họ giới thiệu một khái niệm mới gọi là OpenID Connect.

### What is OpenID Connect?

- OpenID Connect is a protocol that sits on top of the OAuth 2.0 framework. While OAuth 2.0 provides authorization via an access token containing scopes, OpenID Connect provides authentication by introducing a new ID token which contains a new set of information and claims specifically for identity.
- With the ID token, OpenID Connect brings standards around sharing identity details among the applications.

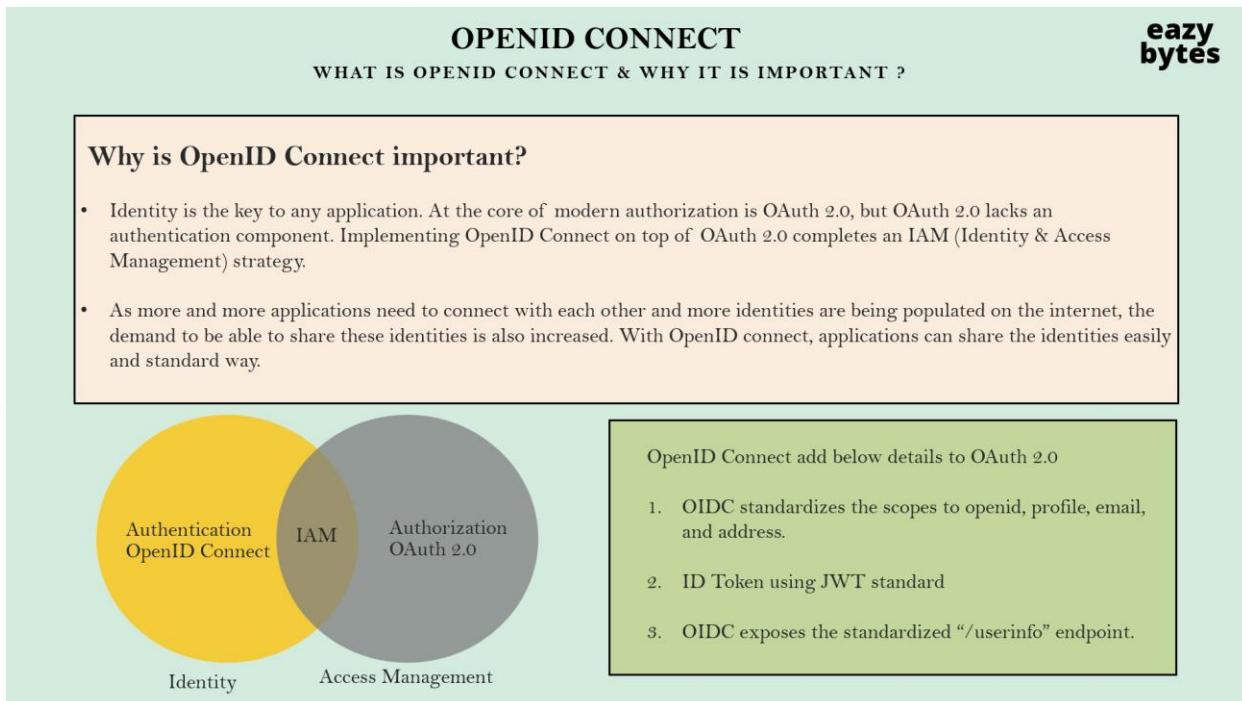


OpenID là gì? Nó cũng là một giao thức nằm trên OAuth 2.0. Đó là một trình bao bọc nhỏ nằm trên OAuth 2.0 của bạn nếu không sử dụng OAuth 2.0 hoặc bạn không biết về OAuth 2.0 thì không có cách nào để bạn sử dụng OpenID Connect đó. Bạn có thể thấy ở đây OAuth 2.0 được xây dựng dựa trên giao thức HTTP và điều này nhằm mục đích authorization với sự trợ giúp của access token. Trên hết, có một lớp nhỏ hoặc giao thức bao bọc được xây dựng là OpenID Connect. Vì vậy, bất cứ khi nào bạn theo dõi OpenID Connect, bạn sẽ nhận được cả authentication và authorization. Và ở đây bạn có thể có một câu hỏi như OpenID Connect của tôi sẽ giúp tôi như thế nào với các chi tiết xác thực hoặc chi tiết người dùng. Vì vậy, bất cứ khi nào chúng tôi triển khai OpenID Connect bên trong authorization server của mình, nó sẽ hiển thị hai loại token. Một là access token và một là ID token. Tất nhiên sẽ luôn có refresh token nhưng nếu bạn bỏ qua refresh token liên quan đến user thì sẽ có hai loại token. Một là access token xử lý authorization và mã thứ hai là ID token xử lý chi tiết người dùng về chi tiết authorization.

Và câu hỏi tiếp theo mà bạn có thể có ở đây là làm thế nào tôi có thể bắt đầu OpenID Connect để truyền với authorization server. Những gì chúng tôi phải làm là bên trong scope, chúng tôi nên gửi lệnh gọi giá trị OpenID và bất cứ khi nào chúng tôi gửi scope cụ thể này bên trong yêu cầu tới authorization server, authorization server của tôi đủ thông minh để xác định điều này và gửi cả access token và mã thông báo ID. Và với hai token, điều gì sẽ xảy ra sau đó một khái niệm mới ra đời, đó là IAM identity và access management.

Với hai loại token này, tôi có thể xác định chi tiết nhận dạng - identity details của user của mình và tôi cũng có thể kết thúc việc thực thi access management bằng sự kết hợp giữa access management và identity này. Giống như bạn có thể thấy, có một khái niệm gọi là IAM đã xuất hiện mà bạn có thể đã nghe thấy rất nhiều trong Authorization server của ngành như Okta, Keycloak, tất cả họ đều triển khai khái niệm IAM này và họ tuân theo OpenID Connect và framework OR two. Họ cung cấp tất cả các tính năng này và việc chúng tôi chỉ muốn sử dụng access token hay cả Access Token và ID token là tùy thuộc vào chúng tôi.

Và với việc phát minh ra OpenID Connect giờ đây tất cả các ứng dụng đều có cách tiêu chuẩn để chia sẻ chi tiết người dùng dưới dạng mã thông báo ID. Cùng với đó, hạn chế mà chúng tôi từng có trước khi OpenID Connect được mọi người sử dụng và đó là lý do tại sao mọi người bắt đầu sử dụng OpenID Connect.



Và lợi thế khác mà chúng tôi có bất cứ khi nào chúng tôi sử dụng OpenID Connect là bên trong OpenID Connect, chúng tôi có các scope như OpenID profile, email và địa chỉ giúp chúng tôi hiểu chi tiết về user của mình như chi tiết hồ sơ, chi tiết email của anh ấy là gì, chi tiết địa chỉ, identity detail mà chúng tôi có thể dễ dàng tìm nạp với các scope được hỗ trợ bên trong OpenID Connect và ID token mà chúng tôi nhận được bất cứ khi nào chúng tôi theo dõi OpenID Connect. Nó tuân theo tiêu chuẩn JWT token, nghĩa là chúng tôi có thể lưu trữ bất kỳ loại thông tin bên trong phần thân của JWT token. Đồng thời, tất cả các Authorization Server tuân theo OpenID Connect đều hiển thị một điểm cuối chuyên dụng với thông tin người dùng có dấu gạch chéo tên mà client application của tôi có thể gọi vào bất kỳ thời điểm nào để hiểu thêm chi tiết về người dùng đã đăng nhập. Vì vậy, đó là một lợi thế nữa mà chúng tôi có với OpenID Connect. Tôi hy vọng bạn hiểu rõ về sự khác biệt giữa OpenID Connect và OR two. Tôi luôn khuyên bạn nên sử dụng OpenID Connect vì điều đó cung cấp cả access token và ID token. Trong khi nếu bạn chỉ sử dụng OR two token thì bạn sẽ chỉ nhận được access token. Nhưng xin lưu ý rằng sẽ không có OpenID Connect nếu không có OR two, OpenID Connect được xây dựng dựa trên OR two đó dưới dạng một trình bao bọc nhỏ có thể cung cấp ID token.

## Section 12: Implementing OAuth2 using spring security

### 1. Registering the client details with the GitHub to use its OAuth2 Auth server

Để đăng ký thông tin khách hàng của bạn với GitHub để sử dụng Auth Server OAuth2 của nó, bạn cần tạo một ứng dụng OAuth mới trên GitHub. Hãy làm theo các bước sau:

1. Đăng nhập vào tài khoản GitHub của bạn.
2. Truy cập vào cài đặt tài khoản của bạn bằng cách nhấp vào hình đại diện của bạn ở góc phải trên cùng và chọn "Settings" từ menu thả xuống.
3. Trong thanh bên trái, nhấp vào "Developer settings".
4. Trên trang Cài đặt nhà phát triển, chọn "OAuth Apps" từ thanh bên.

Bây giờ, bạn cần cung cấp các chi tiết yêu cầu cho ứng dụng OAuth của bạn:

6. Nhập "Application name" cho ứng dụng của bạn. Đây là tên sẽ hiển thị cho người dùng khi họ ủy quyền ứng dụng của bạn.
7. Trong trường "Homepage URL", nhập URL nơi người dùng có thể tìm hiểu thêm về ứng dụng hoặc tổ chức của bạn.
8. Trong trường "Authorization callback URL", nhập URL mà GitHub sẽ chuyển hướng người dùng sau khi họ ủy quyền ứng dụng của bạn. Thông thường, đây là URL của trang trên trang web của bạn xử lý phản hồi OAuth.
9. Tuỳ chọn, bạn có thể điền vào trường "Application description" để cung cấp thêm thông tin về ứng dụng của bạn.
10. Dưới "Tùy chọn nâng cao", bạn có thể chỉ định các cài đặt bổ sung nếu cần, chẳng hạn như các "Callback URL(s)" và "Authorization scopes" được phép cho ứng dụng của bạn.
11. Nhấp vào nút "Register application" để tạo ứng dụng OAuth của bạn.

Sau khi bạn đã đăng ký ứng dụng OAuth của mình, GitHub sẽ cung cấp cho bạn "Client ID" và "Client Secret". Đây là thông tin đăng nhập quan trọng mà bạn sẽ cần sử dụng trong mã ứng dụng của mình để xác thực và tương tác với máy chủ OAuth2 của GitHub.

111. Registering the client details with the GitHub to use it's OAUTH2 Auth server

github.com/settings/applications/new

Search or jump to... Pull requests Issues Marketplace Explore

Register a new OAuth application

Application name \* EazyBytes  
Something users will recognize and trust.

Homepage URL \* http://localhost:8080  
The full URL to your application homepage.

Application description EazyBytes Client Application  
This is displayed to all users of your application.

Authorization callback URL \* http://localhost:8080  
Your application's callback URL. Read our OAuth documentation for more information.

Enable Device Flow  
Allow this OAuth App to authorize users via the Device Flow.  
Read the Device Flow documentation for more information.

**Register application** Cancel

© 2022 GitHub, Inc. Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About

Settings / Developer settings / EazyBytes

General

**EazyBytes**

eazybytes owns this application. Transfer ownership

You can list your application in the GitHub Marketplace so that other users can discover it. List this application in the Marketplace

**0 users** Revoke all user tokens

Client ID 8cf67ab304dc500092e3

Client secrets Generate a new client secret  
You need a client secret to authenticate as the application to the API.

Application logo Upload new logo  
You can also drag and drop a picture from your computer.

Application name \* EazyBytes

## 2. Building a springboot application that uses GitHub Auth server during OAuth2

Để xây dựng một ứng dụng Spring Boot sử dụng Auth Server OAuth2 của GitHub, bạn có thể làm theo các bước sau:

Bước 1: Tạo ứng dụng OAuth trên GitHub Trước tiên, bạn cần tạo một ứng dụng OAuth trên GitHub. Đăng nhập vào tài khoản GitHub của bạn, truy cập vào "Settings", sau đó chọn "Developer settings" và điều hướng đến trang "OAuth Apps". Tại đây, bạn có thể tạo một ứng dụng mới và lấy client ID và client secret của ứng dụng.

Bước 2: Cấu hình ứng dụng Spring Boot Thêm các phụ thuộc cần thiết vào file pom.xml hoặc build.gradle của dự án Spring Boot để sử dụng Spring Security và OAuth2.

Bước 3: Cấu hình thông tin xác thực OAuth2 Trong file application.properties (hoặc application.yml) của ứng dụng Spring Boot, cấu hình các thông tin xác thực OAuth2 từ GitHub:

```
spring.security.oauth2.client.registration.github.client-id=YOUR_CLIENT_ID
spring.security.oauth2.client.registration.github.client-secret=YOUR_CLIENT_SECRET
spring.security.oauth2.client.registration.github.redirect-uri=YOUR_REDIRECT_URI
spring.security.oauth2.client.registration.github.scope=user:email
spring.security.oauth2.client.provider.github.authorization-
uri=https://github.com/login/oauth/authorize
spring.security.oauth2.client.provider.github.token-uri=https://github.com/login/oauth/access_token
spring.security.oauth2.client.provider.github.user-info-uri=https://api.github.com/user
```

Thay thế **YOUR\_CLIENT\_ID**, **YOUR\_CLIENT\_SECRET**, và **YOUR\_REDIRECT\_URI** bằng giá trị tương ứng từ ứng dụng OAuth2 của GitHub bạn đã tạo.

Bước 4: Triển khai điểm cuối OAuth2 callback Tạo một controller để xử lý điểm cuối callback OAuth2 từ GitHub:

```
@Controller
public class OAuth2Controller {
    public String main(OAuth2AuthenticationToken token) {
        System.out.println(token.getPrincipal());
        return "secure.html";
    }
}
```

Trong controller này, bạn có một phương thức để chuyển hướng người dùng đến trang xác thực OAuth2 từ GitHub và một phương thức để xử lý mã code từ GitHub sau khi người dùng đã xác thực thành công. Trong phương thức callback, bạn có thể triển khai logic để lấy access token từ GitHub và thực hiện các tác vụ cần thiết trong ứng dụng của bạn.

Bước 5: Cấu hình bảo mật và truy cập

Trong bước này, bạn cần cấu hình bảo mật và quyền truy cập trong ứng dụng của mình. Điều này bao gồm xác thực người dùng và xác định vai trò người dùng để quyết định quyền truy cập vào các tài nguyên.

```
@Configuration
public class SpringSecOAuth2GitHubConfig {

    @Bean
    SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws
Exception {

    http.authorizeHttpRequests().anyRequest().authenticated().and().oauth2Login()
    ;
    return http.build();
}

/*@Bean
public ClientRegistrationRepository clientRepository() {
    ClientRegistration clientReg = clientRegistration();
    return new InMemoryClientRegistrationRepository(clientReg);
}

private ClientRegistration clientRegistration() {
    return
CommonOAuth2Provider.GITHUB.getBuilder("github").clientId("8cf67ab304dc500092
e3")

.clientSecret("6e6f91851c864684af2f91eaa08fb5041162768e").build();
} */

}
```

Bên trong class này tôi đã định nghĩa tất cả các cấu hình liên quan đến Spring Security. Trước tiên, bạn cần đảm bảo rằng bạn đang đặt annotation `@Configuration` và annotation đó như bình thường. Tôi đang tạo Bean của security filter chain. và bên trong phương thức này, sử dụng mẫu http này, tôi chỉ đề cập đến bất kỳ yêu cầu nào đến với ứng dụng Spring Boot của tôi đều phải được xác thực(authenticated).

Và lần này tôi không muốn sử dụng đăng nhập biểu mẫu hoặc HTTP cơ bản. Tôi muốn sử dụng kiểu đăng nhập OAUTH2. Đó là lý do tại sao chúng ta cần thực hiện phương pháp đăng nhập OAUTH2 này. Sau khi tôi cung cấp các chi tiết này bên trong Spring Boot application của mình, ứng dụng của tôi sẽ hoạt động như một client application bên trong luồng lớn OAUTH2. Nhưng ở đây, client application của tôi không biết giá trị clientID, clientsecret là gì.

Tương tự, chúng ta cần tạo một Bean ClientRegistrationRepository. Bên trong phương thức này, tôi chỉ đang tạo một đối tượng mới của InMemoryClientRegistrationRepository. Và tương tự, tôi đang liên hệ lại để Spring Security của tôi biết nơi tôi đã lưu trữ thông tin chi tiết về khách hàng của mình. Vì vậy, hiện tại, chúng tôi đang lưu trữ thông tin chi tiết về khách hàng bên trong bộ nhớ. Vì vậy, tất cả những chi tiết đăng ký khách hàng mà tôi đã đề cập bên trong phương pháp này, đó là client registration. Vì vậy, đầu ra từ phương thức này là một đối tượng client registration, mà chúng ta có thể chuyển đến kho lưu trữ InMemoryClientRegistrationRepository này. Ở đây, bên trong phương thức clientRepository này, bạn có thể thấy tôi đang tận dụng một trong các lớp enum có sẵn bên trong Spring

Security framework. Lớp enum này là CommonOAuth2Provider. Bên trong CommonOAuth2Provider này, có nhiều giá trị enum, chẳng hạn như nếu bạn muốn sử dụng Google Art Server, bạn có thể sử dụng giá trị enum này, đó là Google. Và rất giống nhau, chúng tôi cũng có GitHub và đăng nhập nội dung đó. Chúng tôi cũng có Facebook và trong class, chúng tôi cũng có OKTA.

Lý do tại sao chúng ta phải sử dụng lớp enum này là Spring Security. Những gì họ đã làm là họ đã cung cấp một số phương thức trợ giúp bên trong enums này với tên getBuilder. Bên trong getBuilder này, bạn có thể thấy nó đang chuyển tất cả các chi tiết scope, authorizationURL của Google là gì? TokenURL là gì? certificateURL là gì? issuerURL là gì? userinfoURL là gì? Nếu không, tôi phải chuyển tất cả các chi tiết này theo cách thủ công. Rất tương tự, đối với GitHub cũng vậy, nó đang điền vào tất cả authorization URL, token URL, and cũng tương tự đối với Facebook and OKTA. Nếu tôi không có lớp enum này bên trong Spring Security, thì tôi phải thực hiện nhiều nghiên cứu, hiểu authorization URL là gì? Token URL là gì? Nhưng với enum này, nó sẽ trở nên cực kỳ đơn giản. Chúng ta chỉ cần biết giá trị enum là gì? Trong trường hợp của chúng tôi, chúng tôi muốn sử dụng máy chủ github. Đó là lý do tại sao tôi đang làm việc trên githu.

Chúng ta cần gọi phương thức getBuilder này và với phương thức getBuilder này, chúng ta phải chuyển một giá trị github và post giá trị đó. Chúng ta cần gọi các phương thức này như clientid và clientsecret để truyền các giá trị này mà chúng ta đã nhận được trong quá trình đăng ký. Và ở class, chúng ta nên làm việc với phương thức Builder này với client application của tôi, đó là Spring Boot application. Nó biết tất cả các chi tiết như Auth Server mà nó phải kết nối là gì? Trong trường hợp này, đó là github.

Và clientId là gì? clientSecret mà nó phải xem xét là gì? Tất nhiên, chúng ta không nên mã hóa trực tiếp các giá trị này ở đây. Chúng ta nên cố gắng đọc chúng từ các tệp thuộc tính nhất định, bằng cách tuân theo các tiêu chuẩn product. Nhưng hiện tại, vì đây là một ứng dụng web đơn giản nên điều đó sẽ ổn thôi. Vì vậy, đây là một trong những quy trình dài mà bạn có thể làm theo nhưng chúng tôi không phải làm theo quy trình này.

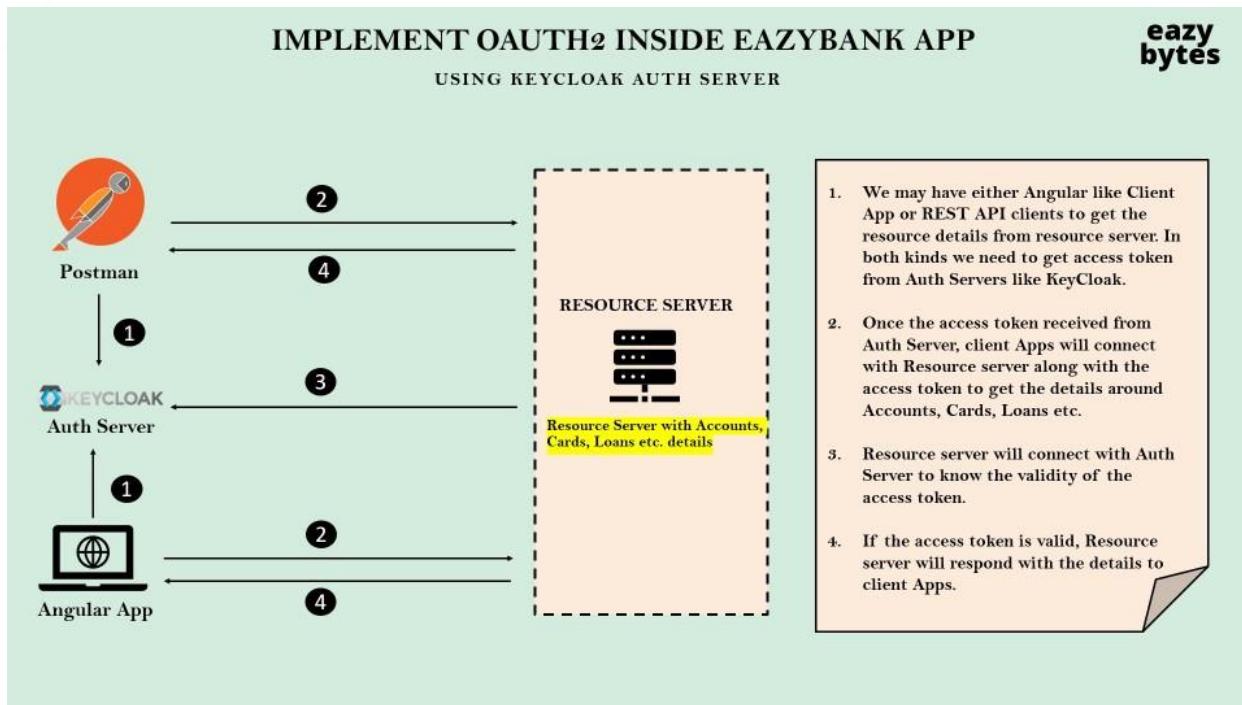
Ngoài ra còn có một cách tiếp cận ngắn khác mà chúng ta có thể tận dụng. Trước khi tôi giải thích điều đó, tôi sẽ cố gắng đưa vào các mã này để bạn luôn có thể tham khảo chúng để bạn tham khảo. Vì vậy, tôi đang lưu những chi tiết này. Cách tiếp cận ngắn hơn mà tôi đang nói là nếu bạn có thể mở application.properties bên trong ứng dụng khởi động Spring này, chúng ta có thể định cấu hình clientID và clientsecret này với các thuộc tính. Và rất tương tự, chúng ta cũng có thể cấu hình clientsecret.

```
spring.security.oauth2.client.registration.github.client-id=8cf67ab304dc500092e3
spring.security.oauth2.client.registration.github.client-
secret=6e6f91851c864684af2f91eaa08fb5041162768e
```

## Section 13: Implementing OAuth2 style login inside EazyBank using KeyCloak

### 1. Introduction to OAuth2 flow inside EazyBank web App

Chúng tôi sẽ tạo Auth Server của riêng mình, Server này có thể được tận dụng bởi tất cả các ứng dụng đang được phát triển và duy trì bởi ngân hàng của tôi, đó là EazyBank. Cho đến bây giờ chúng tôi chỉ có một ứng dụng web EazyBank. Nhưng hãy nghĩ đến một tình huống nếu ứng dụng EazyBank của tôi phát triển thành nhiều dạng như ứng dụng di động, microservices, thì trong tất cả các tình huống đó, việc có một Auth Server riêng do chính EazyBank sở hữu là một giải pháp phù hợp. Trước tiên chúng tôi sẽ thiết lập Auth Server của riêng mình với sự trợ giúp của **KeyCloak** và backend application mà chúng tôi có đang lưu trữ tất cả chi tiết tài khoản, chi tiết thẻ, chi tiết khoản vay. Chúng tôi sẽ chuyển đổi nó thành một resource server. Khi tôi có các authorization server and resource server, này, chúng ta sẽ khám phá hai loại authentication khác nhau.



Một là bất cứ khi nào giao tiếp API với API diễn ra. Giống như nếu một trong các ứng dụng của bên thứ ba đang cố sử dụng các dịch vụ còn lại bên trong resource server của tôi. Trong trường hợp đó, chúng tôi phải sử dụng client credentials grant flow.

Và trường hợp khác là một trong những UI client application như ứng dụng Angular mà chúng tôi có thể kết nối với resource server của tôi để tìm nạp các chi tiết từ phần back-end và các chi tiết tương tự mà nó sẽ hiển thị trên trang UI. Vì vậy, đây là hai trường hợp mà chúng ta sẽ khám phá nhưng kịch bản UI vì sẽ có user tham gia nên chúng ta cần sử dụng authorization code grant type flow.

## 2. Introduction to KeyCloak Auth Server

Khi chúng ta nói về các giao thức OAuth 2.0 và Open ID, tôi đã nói với bạn rằng đây chỉ là các giao thức hoặc thông số kỹ thuật và chúng không cung cấp bất kỳ triển khai nào. Nhưng các tổ chức lớn hơn như Google, Microsoft, GitHub, Twitter, Netflix, tất cả các tổ chức này bằng cách sử dụng các thông số kỹ thuật OAuth 2.0 và Open ID này, họ đã bắt đầu xây dựng Auth Server của riêng mình để triển khai tất cả các thông số kỹ thuật được xác định trong các giao thức này. Nhưng điều này cũng mở ra một thị trường rộng lớn bên trong ngành công nghiệp phần mềm, nơi mọi người bắt đầu tìm kiếm các Auth Server được tạo sẵn mà họ có thể tận dụng trong tổ chức của mình, bởi vì không phải mọi công ty hoặc không phải mọi tổ chức đều có đủ khả năng xây dựng các Auth Server của riêng họ như Google và Facebook. Để kết thúc thị trường kiếm tiền đó, có rất nhiều thị trường Auth Server đã hình thành. Giống như bạn có thể thấy ở đây Keycloak là một trong những Auth Server như vậy.

Và rất tương tự, chúng ta cũng có những lựa chọn nổi tiếng khác trong ngành như Okta, ForgeRock, thậm chí Amazon cũng có sản phẩm của riêng họ với cái tên Amazon Cognito. Vì vậy, tất cả các công ty này đang cố gắng bán các Auth Server này cho các tổ chức nhỏ hơn hoặc cho các tổ chức mà họ không quan tâm đến việc xây dựng Auth Server của riêng họ từ đầu. Nhưng xin lưu ý rằng tôi không nói rằng chúng ta chỉ có bốn lựa chọn trong số này. Có rất nhiều lựa chọn trong ngành, nhưng theo hiểu biết của tôi, đây là những sản phẩm Auth Server được sử dụng nhiều nhất.

Chúng ta sẽ tận dụng Keycloak, vì hai lý do. Một là nó là một mã nguồn mở. Không có tiền liên quan. Bạn luôn có thể tải xuống và thiết lập trên máy chủ và máy tính xách tay của riêng mình và bắt đầu sử dụng ngay lập tức. Trong khi tất cả các máy chủ Auth khác như Okta, ForgeRock, Amazon Cognito, chúng đều là sản phẩm thương mại. Và một lý do khác mà tôi chọn Keycloak là, mặc dù nó là một sản phẩm nguồn mở, nhưng nó là một sản phẩm rất ổn định và nhóm Keycloak, họ luôn làm rất tốt trong việc cập nhật Keycloak Auth server thường xuyên dựa trên OAuth 2.0 và các tiêu chuẩn bảo mật khác mà chúng tôi có trong ngành. Trên hết, Keycloak này được tài trợ và duy trì bởi không ai khác ngoài Red Hat. Với tất cả những lý do này, Keycloak là một lựa chọn rất tốt để sử dụng làm Auth Server.

Nếu bạn xem trang web Keycloak.org này, bạn có thể thấy ở cấp độ cao các tính năng được cung cấp bởi Keycloak là gì. Bất cứ khi nào bạn sử dụng Keycloak làm Auth Server, bạn sẽ có toàn quyền kiểm soát. Bạn có thể lấy sản phẩm này và thiết lập bên trong các máy chủ của riêng bạn hoặc bên cloud của riêng bạn hoặc bên trong local system của bạn và bạn sẽ có đầy đủ các đặc quyền của quản trị viên. Và với các đặc quyền quản trị viên này, bạn có thể tạo bất kỳ số lượng quản trị viên hoặc người dùng nào.

Và một tính năng nữa của Keycloak là nó cũng có nhiều dịch vụ rủi ro, bằng cách sử dụng chúng tôi có thể tạo người dùng, chúng tôi có thể tạo quản trị viên, chúng tôi có thể tạo vai trò-role, chúng tôi có thể nhận access token, chúng tôi có thể xác thực access token. Có nhiều dịch vụ có sẵn trong sản phẩm Keycloak. Nếu bạn quan tâm, bạn luôn có thể khám phá theo tài liệu chính thức của Keycloak. Và bạn có thể thấy nó hỗ trợ **đăng nhập một lần(single sign-on)**, điều đó có nghĩa là nếu bạn có nhiều ứng dụng trong tổ chức của mình và nếu tất cả chúng đều trỏ đến cùng một Auth Server, đó là Keycloak, thì user của tôi chỉ có thể đăng nhập vào một trong các ứng dụng. Và cùng một access token có thể được tận dụng trong tất cả các ứng dụng khác mà không cần phải đăng nhập lại nhiều lần trong các ứng dụng khác nhau.

Và nó cũng hỗ trợ Open ID connect, OAuth 2.0, nó cũng cung cấp quản lý trung tâm để tạo quản trị viên và người dùng. Nó cũng cung cấp cho bạn thông tin social login, nghĩa là bạn cũng có thể thiết

lập đăng nhập Github, đăng nhập Facebook với sự trợ giúp của Keycloak. Và bạn cũng có thể đặt Keycloak trở đến hệ thống lưu trữ của mình như LDAP, thư mục hoạt động hoặc bất kỳ cơ sở dữ liệu nào bạn muốn để tất cả thông tin chi tiết về người dùng của bạn cùng với mật khẩu của họ sẽ được lưu trữ bên trong các hệ thống lưu trữ này. Với điều này, tôi cho rằng bạn đã bị thuyết phục sử dụng Keycloak.

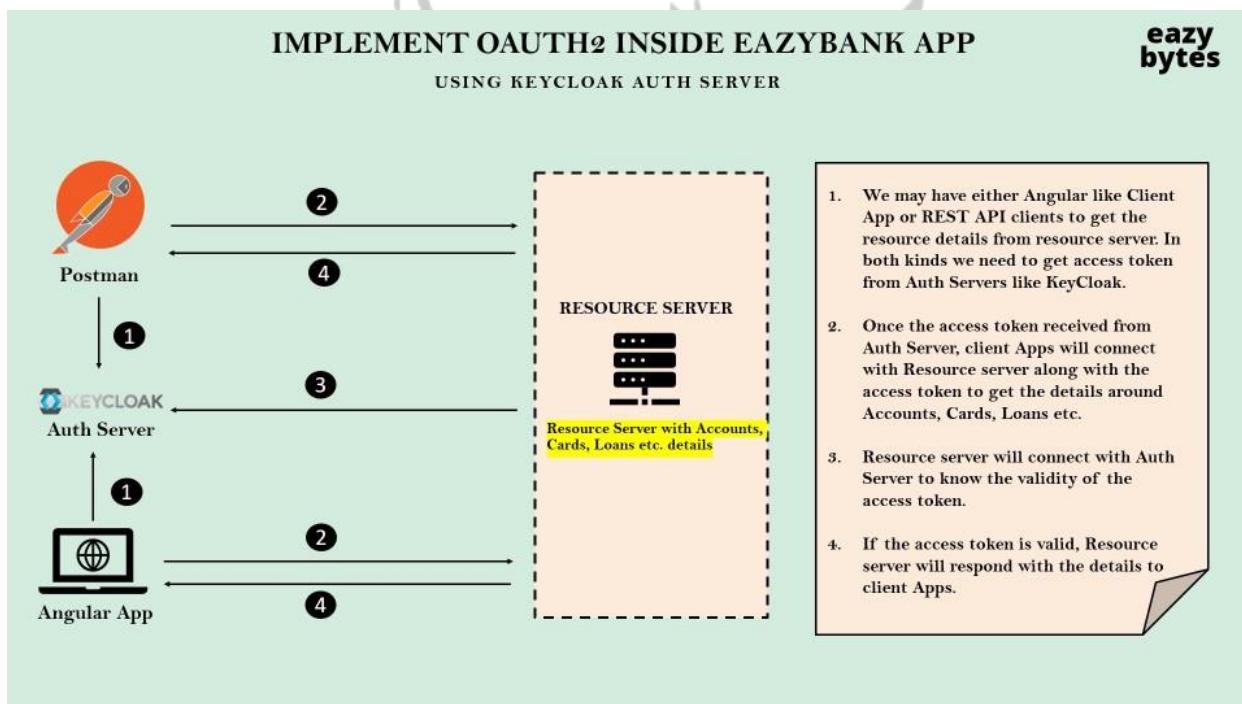
Để bắt đầu với Keycloak này, chúng ta cần truy cập trang tải xuống này. Bạn có thể thấy trên trang này có một bảng máy chủ. Dưới tiêu đề máy chủ này, có một hàng đầu tiên có tên Keycloak. Nếu bạn nhìn vào phía bên tay phải, có một tùy chọn tải xuống. Tại đây, bạn có thể tải xuống máy chủ Keycloak này bằng cách nhấp vào máy chủ này (không rõ ràng). Thao tác này sẽ bắt đầu tải máy chủ Keycloak vào hệ thống cục bộ của tôi.

### 3. Installation of Keycloak server & setup admin account

<https://www.keycloak.org/getting-started/getting-started-zip>

### 4. Creating Client Credentials inside Keycloak for API-API secured invocations

Bên trong kịch bản giao tiếp API với API này, chúng ta sẽ thực hiện các API REST có sẵn bên trong Resource Server với sự trợ giúp của Postman. Bằng cách này, chúng tôi đang cố gắng bắt chước hai ứng dụng khác nhau đang cố gắng nói chuyện với nhau thông qua giao tiếp API REST.



Giống như bạn có thể thấy ở đây, trong bước đầu tiên Postman của tôi phải lấy access token từ Keycloak auth server. Sau khi nhận được access token, nó có thể gửi mã tương tự đến Resource Server và trong bước thứ ba, Resource Server của tôi sẽ xác thực, xem access token có hợp lệ hay không với máy chủ Keycloak của tôi. Và nếu access token hợp lệ, Resource Server của tôi sẽ gửi phản hồi thành công tới ứng dụng Postman.

Trong trường hợp này, như chúng ta đã thảo luận trước đây, bất cứ khi nào không có user tham gia và bất cứ khi nào giao tiếp diễn ra giữa hai ứng dụng khác nhau với sự trợ giúp của giao tiếp API, thì chúng ta phải sử dụng grant type flow là client credentials grant type. Nhưng để tận dụng client credentials grant flow, trước tiên tôi cần lấy **client ID**, **client secret** từ Auth Server của mình để ứng dụng Postman của tôi có thể tận dụng luồng tương tự để nhận access token.

Vì vậy, hãy thử tạo các client details này bên trong Keycloak server. Đây là bảng điều khiển quản trị keycloak của tôi. Tại đây, vui lòng đảm bảo rằng bạn đã chọn đúng realm đó là easybankdev. Khi bạn chọn lĩnh vực này, bạn có thể nhấp vào tab clients này mà chúng tôi có.

Client ID	Type	Description	Home URL
account	OpenID Connect	—	http://localhost:8180/realms/easybankdev/account/
account-console	OpenID Connect	—	http://localhost:8180/realms/easybankdev/account/
admin-cli	OpenID Connect	—	—
broker	OpenID Connect	—	—
realm-management	OpenID Connect	—	—
security-admin-console	OpenID Connect	—	http://localhost:8180/admin/easybankdev/console/

Tại đây, chúng ta có thể tạo bất kỳ số lượng client nào. Hiện tại, tôi đang tạo client này theo cách thủ công bằng thông tin đăng nhập quản trị viên của mình. Nhưng trong thế giới thực, với ứng dụng bên thứ ba của chúng tôi đang cố gắng sử dụng Resource Server của bạn, họ phải liên hệ với tổ chức của bạn và họ sẽ trải qua quá trình xem xét thích hợp và khi mọi người hài lòng, các nhà lãnh đạo dự án của tôi, họ sẽ gửi một lưu ý cho quản trị viên Keycloak để tạo client details về ứng dụng khách bên trong máy chủ keycloaks của tôi.

Và sau khi các chi tiết này được tạo, client ID và client secret có thể được cung cấp cho ứng dụng của bên thứ ba, bằng cách sử dụng chúng, họ có thể bắt đầu giao tiếp với ứng dụng của bạn. Vì vậy, ở đây, chúng tôi có một tùy chọn để tạo **client** bằng cách nhấp vào nút “**Create client**”. Vì vậy, hãy để tôi nhấp vào đó. Client type mà tôi muốn tạo là OpenID connect. Vì chúng tôi muốn tận dụng cả OAuth2 và OpenID, nên chúng tôi có thể chọn kết nối OpenID này và tại đây, chúng tôi có thể chọn **client ID** của riêng mình. **Client ID** mà tôi muốn sử dụng là easybankapi. Khi chúng tôi đã cung cấp **Client ID** này, chúng tôi có thể nhấp vào phần tiếp theo này.

Clients > Create client

### Create client

Clients are applications and services that can request authentication of a user.

1 General Settings

Client type: OpenID Connect

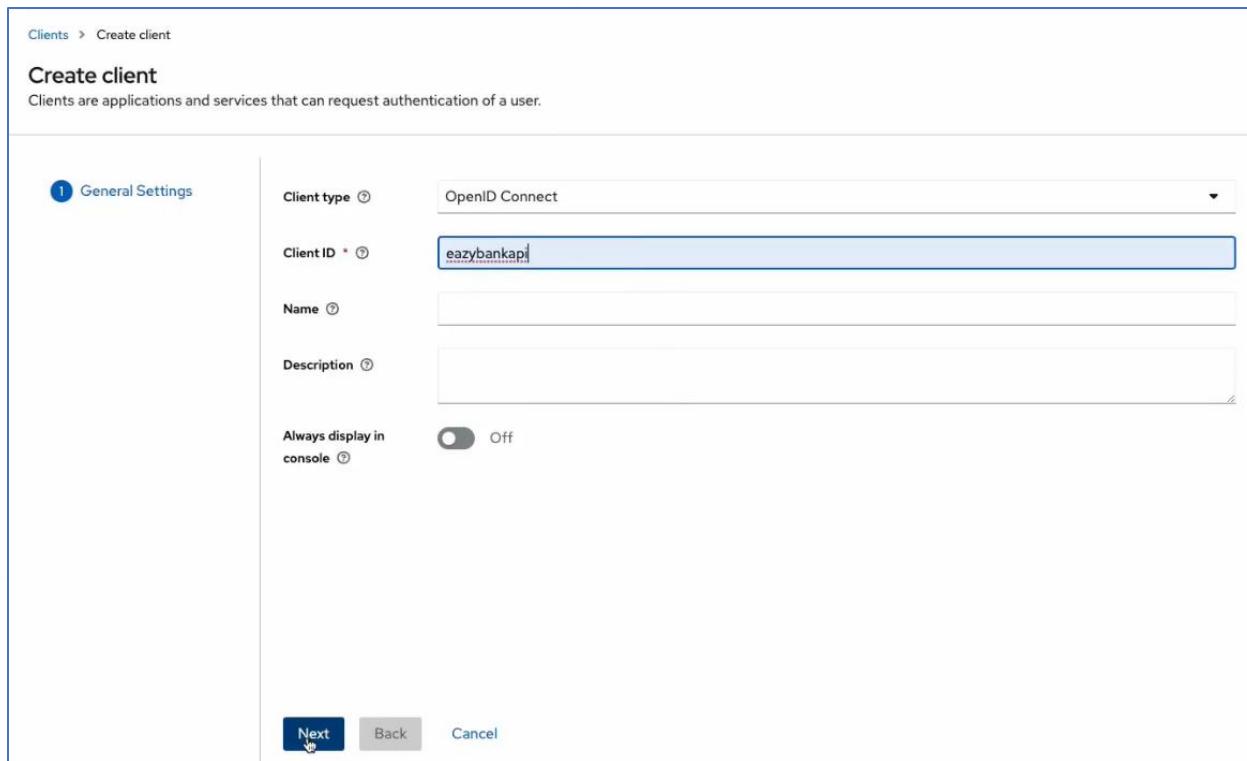
Client ID: eazybankapi

Name:

Description:

Always display in console: Off

Next Back Cancel



Ở đây bên trong trang này, trước tiên tôi cần cho biết liệu client application của mình có cần bất kỳ xác thực nào để nhận access token hay không. Tất nhiên, chúng tôi muốn cung cấp bí client secret cho ứng dụng của bên thứ ba bằng cách sử dụng ứng dụng này, họ có thể tự xác thực. Đó là lý do tại sao chúng tôi cần bật client authentication này và đăng bên trong OAuth authentication flows, chúng tôi cần chọn grand type flows mà ứng dụng khách của tôi sẽ gọi là gì. Vì vậy, chúng tôi cần vô hiệu hóa standard flow vì standard flow biểu thị authorization code flow, nơi có liên quan đến user, hiện tại chúng tôi không cần điều này. Chúng ta có thể vô hiệu hóa điều này. Và tương tự như vậy, direct access grant, resource owner, credentials grant type.

Trong loại cấp phép này, user cũng sẽ tham gia, vì vậy tôi không muốn điều này, hãy để tôi tắt tính năng này. Vì chúng tôi muốn client credentials grant type flow, chúng tôi cần chọn vai trò tài khoản dịch vụ này. Nếu bạn có thể nhấp vào dấu hỏi này, bạn có thể thấy, bất cứ khi nào bạn chọn dấu chấm hỏi này, client của bạn sẽ tuân theo quy trình client credentials grant flow.

The screenshot shows the Keycloak 'Create client' interface. On the left, a sidebar menu is open with 'Clients' selected. The main area is titled 'Create client' with the sub-section 'General Settings'. Under 'Client authentication', the 'On' toggle is selected. In the 'Authentication flow' section, the 'Service accounts roles' checkbox is checked and highlighted with a tooltip: 'Allows you to authenticate this client to Keycloak and retrieve access token dedicated to this client. In terms of OAuth2 specification, this enables support of 'Client Credentials Grant' for this client.' Below the form are 'Save', 'Back', and 'Cancel' buttons.

Lý do tại sao họ chọn service accounts role này là vì hai dịch vụ khác nhau đang cố gắng liên lạc với nhau, đó là lý do tại sao họ được cấp service account roles này. Vì vậy, khi bạn đã cung cấp các chi tiết này, bạn có thể nhấn vào "Save".

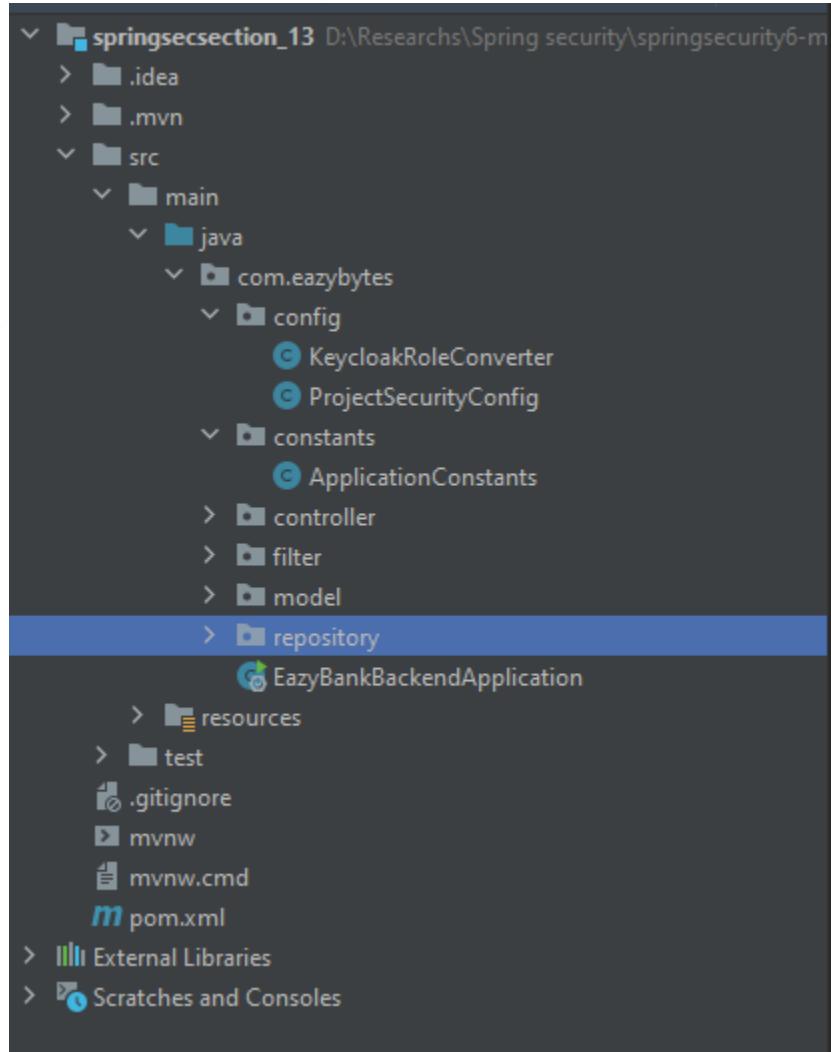
Và đây, bây giờ bạn có thể thấy, nếu bạn có thể chuyển đến tab "**Credential**", ở đây bạn có thể thấy đây là **client secret** g và **client ID** là cái này, đó là easybankapi.

The screenshot shows the 'Client details' page for 'easybankapi'. The 'Credentials' tab is active. The 'Client ID' field contains 'easybankapi'. Other tabs include 'Settings', 'Keys', 'Roles', 'Client scopes', 'Service accounts roles', 'Sessions', and 'Advanced'. A success message 'Client created successfully' is displayed. On the right, a sidebar shows 'Jump to section' with options: General Settings, Access settings, Capability config, Login settings, and Logout settings. At the bottom are 'Save' and 'Revert' buttons.

Và nếu bạn muốn chỉ định một số role, bạn có thể chỉ định ở đây. Nhưng ở cấp độ cao, chúng tôi đã tạo thành công một client bên trong Keycloak Server. Nếu tôi thử quay lại các client, bạn có thể thấy đã có một số client mặc định được tạo. Nhưng client mà tôi đã tạo là easybankapi. Và bạn có thể thấy đây là client ID và điều này sẽ hỗ trợ OpenID Connect và thông tin credentials này.

Client ID	Type	Description	Home URL
account	OpenID Connect	—	<a href="http://localhost:8180/realm/eazybankdev/account">http://localhost:8180/realm/eazybankdev/account/</a>
account-console	OpenID Connect	—	<a href="http://localhost:8180/realm/eazybankdev/account">http://localhost:8180/realm/eazybankdev/account/</a>
admin-cli	OpenID Connect	—	—
broker	OpenID Connect	—	—
eazybankapi	OpenID Connect	—	—
realm-management	OpenID Connect	—	—
security-admin-console	OpenID Connect	—	<a href="http://localhost:8180/admin/eazybankdev/console">http://localhost:8180/admin/eazybankdev/console/</a>

## 5. Setup of EazyBank Resource Server



Và tên lớp sẽ là KeycloakRoleConverter. Logic mà tôi sẽ viết bên trong KeycloakRoleConverter này là Keycloak auth server của tôi sẽ trả lại access token. Bên trong access token này sẽ có thông tin chi tiết về vai trò của client application hoặc roles trò của user của tôi. Tôi có trách nhiệm trích xuất thông tin về role hoặc thông tin về authorities đó từ token và chuyển đổi chúng thành định dạng mà Spring Security của tôi có thể hiểu được. Vì vậy, đó là lý do tại sao tôi phải tạo KeycloakRoleConverter này.

Bên trong lớp này, tôi cần triển khai một interface có tên **Converter**. Như bạn có thể thấy ở đây, **Converter** này sẽ chấp nhận JWT token nhận được từ Keycloak Server của tôi và nó sẽ trích xuất thông tin roles và gửi lại danh sách GrantedAuthority vì Spring Security của tôi chỉ có thể hiểu thông tin roles hoặc thông tin authorities nếu tôi gửi ở dạng đối tượng GrantedAuthority. Vì vậy, đó là những gì **Converter** này sẽ làm. Bất cứ khi nào chúng tôi triển khai interface này, chúng tôi phải ghi đè một phương thức.

Như bạn có thể thấy, tên phương thức là convert và nó sẽ chấp nhận mã thông báo JWT.

```

public class KeycloakRoleConverter implements Converter<Jwt,
Collection<GrantedAuthority>> {

    @Override
    public Collection<GrantedAuthority> convert(Jwt jwt) {
        Map<String, Object> realmAccess = (Map<String, Object>)
        jwt.getClaims().get("realm_access");

        if (realmAccess == null || realmAccess.isEmpty()) {
            return new ArrayList<>();
        }

        Collection<GrantedAuthority> returnValue = ((List<String>)
realmAccess.get("roles"))
            .stream().map(roleName -> "ROLE_" + roleName)
            .map(SimpleGrantedAuthority::new)
            .collect(Collectors.toList());

        return returnValue;
    }
}

```

Và từ JWT token này, trước tiên tôi đang cố trích xuất một đối tượng Map có tên khóa là `real_access`. Vì vậy, từ JWT, tôi sẽ gọi `getClaims` và bên trong đó `claims` rằng tôi sẽ có một đối tượng Map với tên khóa là `real_access`. Và đối tượng Map mà tôi nhận được từ `getClaims` này sẽ được gán cho map `realmAccess` này. Nếu Map `realmAccess` của tôi trống, tôi chỉ viết một `ArrayList` trống. Nếu không, trước tiên bạn có thể thấy tôi đang cố lấy tất cả thông tin về role có sẵn bên trong map `realmAccess` này bằng cách chuyển `roleName`. Và điều này sẽ cung cấp danh sách chi tiết về quyền hạn hoặc danh sách các vai trò có trong JWT token. Khi tôi có điều đó, tôi chỉ cần gọi phương thức `stream`. Và sau phương thức `stream`, tôi đang gọi phương thức `map`. Phương pháp `map` này sẽ cho phép tôi viết một biểu thức Lambda bằng cách sử dụng biểu thức này để tôi có thể thực thi logic nghiệp vụ của riêng mình trên từng role có trong danh sách `stream` này.

Ở đây, những gì tôi đang làm là trước tiên tôi lấy từng tên role mà tôi đã nhận được và tôi thêm tiền tố vào giá trị `ROLE_` vì Spring Security của tôi sẽ mong đợi giá trị tiền tố này bất cứ khi nào chúng tôi sử dụng các role. Và một khi tôi đã thực thi logic này, tôi sẽ gọi lại phương thức `map` một lần nữa vì dựa trên giá trị tên role mới nhất, tôi muốn chuyển đổi chúng thành một đối tượng của `SimpleGrantedAuthority`. Khi tôi đã tạo một `SimpleGrantedAuthority`, tôi muốn thu thập tất cả các `SimpleGrantedAuthority` đó vào một danh sách. Và tôi cũng sẽ quay trở lại từ phương pháp này. Vì vậy, tôi biết điều này có vẻ rất phức tạp với bạn.

Cùng với đó, bây giờ chúng ta có lớp role **Converter**. Bây giờ chúng ta có trách nhiệm định cấu hình `KeycloakRoleConverter` này bên trong lớp `ProjectSecurityConfig`. Tương tự như vậy bên trong phương thức `defaultSecurityFilterChain` này, ngay sau những thay đổi liên quan đến `CSRF` này, tôi sẽ dán hai dòng mã.

```

JwtAuthenticationConverter jwtAuthenticationConverter = new JwtAuthenticationConverter();
jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter(new KeycloakRoleConverter());

```

Giống như bạn có thể thấy ở đây, tôi đang cố gắng tạo một đối tượng của jwtAuthenticationConverter và đối tượng này có sẵn bên trong chính Spring Security. Sử dụng đối tượng của jwtAuthenticationConverter này, tôi gọi phương thức setJwtGrantedAuthoritiesConverter và với phương thức này, tôi đang chuyển đổi đối tượng của KeycloakRoleConverter của mình. Sử dụng phương pháp này, chúng tôi truyền đạt rằng tôi đã trả về một logic tùy chỉnh bên trong trình chuyển đổi này để đọc JWT access token mà tôi đã nhận được từ Keycloak. Và tôi cũng sẽ chuyển đổi sang các granted authorities.

Hiện tại tôi đang hỗ trợ formLogin và httpBasic. Chúng tôi sẽ không hỗ trợ điều này nữa vì chúng tôi sẽ tận dụng framework OAuth2. Chúng ta cần gọi phương thức là oauth2ResourceServer vì từ giờ trở đi, Spring Boot web application của tôi sẽ hoạt động như một Resource Server. Và Resource Server này sẽ tận dụng JWT token để thực hiện xác thực. Và sử dụng phương thức JWT này, một lần nữa chúng ta nên gọi một phương thức gọi là jwtAuthenticationConverter. Và với jwtAuthenticationConverter này, chúng ta cần chuyển đổi đối tượng mà chúng ta đã tạo ở trên cùng. Vì vậy, hãy để tôi chuyển cái tương tự ở đây làm đầu vào cho phương thức jwtAuthenticationConverter. Với điều này, chúng tôi đang nói với Spring Security rằng Spring Boot web application của tôi sẽ hoạt động như một oauth2ResourceServer và nó sẽ tận dụng JWT access token của chúng tôi.

```
and().oauth2ResourceServer().jwt().jwtAuthenticationConverter(jwtAuthenticationConverter);
```

```
@Configuration
public class ProjectSecurityConfig {

    @Bean
    SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {
        CsrfTokenRequestAttributeHandler requestHandler = new CsrfTokenRequestAttributeHandler();
        requestHandler.setCsrfRequestAttributeName("_csrf");
        JwtAuthenticationConverter jwtAuthenticationConverter = new JwtAuthenticationConverter();
        jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter(new KeycloakRoleConverter());

        http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and() HttpSecurity
            .cors().configurationSource(new CorsConfigurationSource() {
                no usages
            })
            @Override
            public CorsConfiguration getCorsConfiguration(HttpServletRequest request) {...}
        }).and().csrf((csrf) -> csrf.csrfTokenRequestHandler(requestHandler).ignoringRequestMatchers( ...patterns: "/contact", "/register")
            .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
            .addFilterAfter(new CsrfCookieFilter(), BasicAuthenticationFilter.class)
            .authorizeHttpRequests() AuthorizationManagerRequestMatcher
                .requestMatchers( @"/myAccount").hasRole("USER")
                .requestMatchers( @"/myBalance").hasAnyRole( ...roles: "USER", "ADMIN")
                .requestMatchers( @"/myLoans").authenticated()
                .requestMatchers( @"/myCards").hasRole("USER")
                .requestMatchers( @"/user").authenticated()
                .requestMatchers( @"/notices", @"/contact", @"/register").permitAll()
            .and().oauth2ResourceServer().jwt().jwtAuthenticationConverter(jwtAuthenticationConverter);
        return http.build();
    }
}
```

Và với sự trợ giúp của jwtAuthenticationConverter này, chúng tôi đang cố gắng chuyển đổi các roles có trong access token thành các granted authorities

Và thay đổi cuối cùng mà chúng tôi phải thực hiện là bên trong application.properties của mình, chúng tôi cần thiết lập liên kết giữa Resource Server và Keycloak auth server. Làm thế nào Resource Server của tôi biết chi tiết về Keycloak auth server của tôi là gì ?

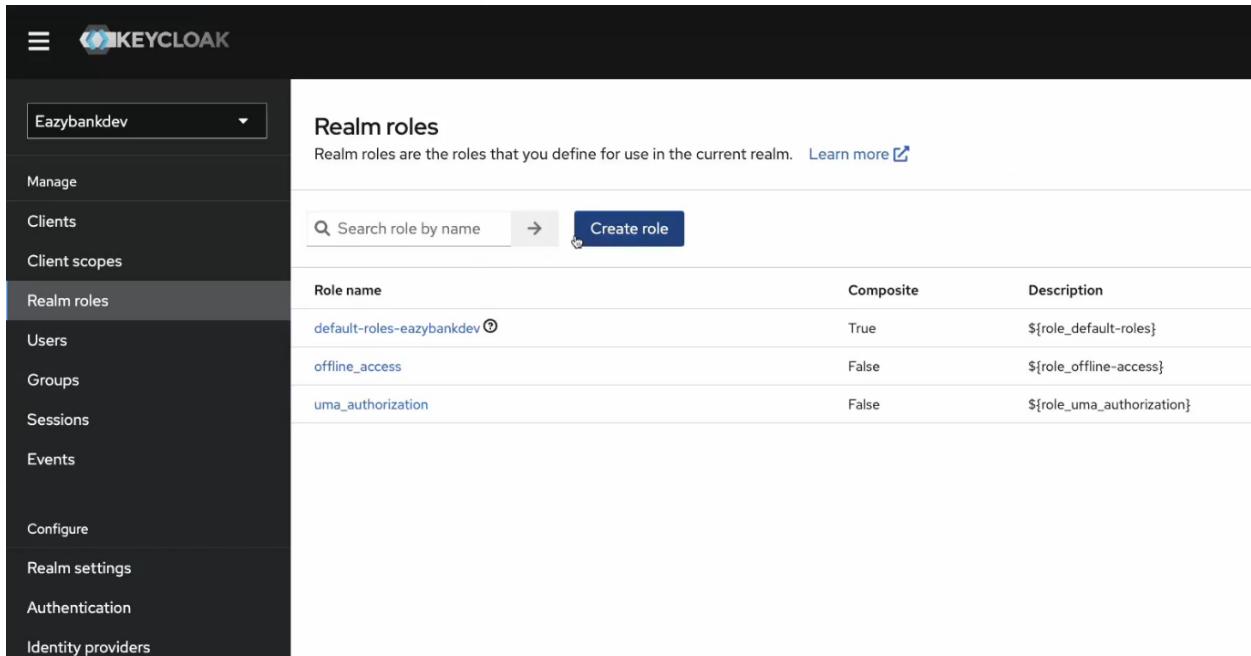
```
spring.security.oauth2.resourceserver.jwt.jwk-set-uri =  
http://localhost:8180/realmseeasybankdev/protocol/openid-connect/certs
```

Giống như bạn có thể thấy ở đây, đây là thuộc tính giống như `spring.security.oauth2.resourceserver.jwt.jwk-set-url`. Vì vậy, đối với thuộc tính này, chúng tôi cần chuyển URL của Keycloak auth server từ nơi Resource Server của tôi có thể tải xuống public certificate. Điều gì sẽ xảy ra, Resource Server của tôi trong quá trình khởi động, nó sẽ tải xuống public certificate bằng cách kết nối với Auth Server. Vì vậy, Auth Server của tôi sẽ có private certificate hay private key mà nó sẽ sử dụng để ký điện tử tất cả access token của tôi, ID tokens. Và Resource Server của tôi, với sự trợ giúp của public certificate, nó có thể xác thực xem token có bị ai đó giả mạo hay không, token có hợp lệ hay không. Bằng cách này, Resource Server của tôi không phải kết nối với Auth Server bất cứ khi nào muốn xác thực Auth Server đã cho.

## 6. Getting Access token from Keycloak using client credentials grant type

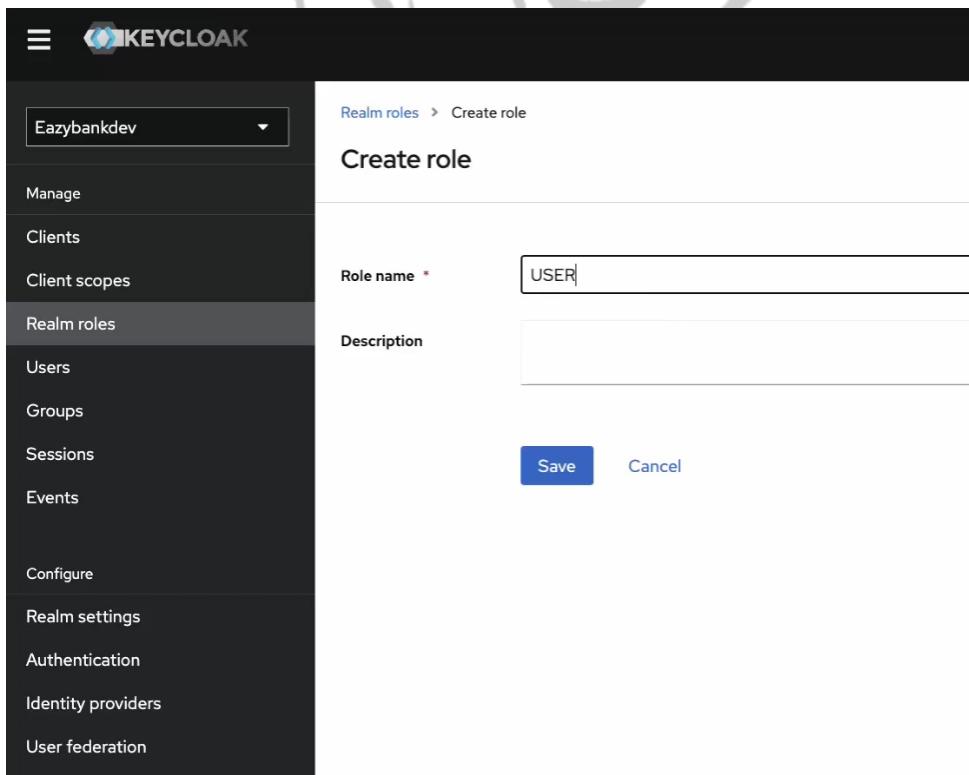
Nếu bạn kiểm tra ProjectSecurityConfig của chúng tôi, chúng tôi định cấu hình mọi thứ, tất cả các điểm cuối của chúng tôi với các vai trò USER and ADMIN. Vì vậy, tương tự, những gì chúng tôi có thể làm là ngay bây giờ, chúng tôi là quản trị viên của Keycloak Auth Server, đó là lý do tại sao chúng tôi có thể tạo các role này và gán cho client application mà chúng tôi đã tạo trước đó.

Tương tự, tôi có thể truy cập bảng điều khiển dành cho quản trị viên Keycloak tại đây, có một tùy chọn Roles trong Realm.



Role name	Composite	Description
default-roles-eazybankdev	True	\${role_default-roles}
offline_access	False	\${role_offline-access}
uma_authorization	False	\${role_uma_authorization}

Và trên trang này, có một nút “Create Role”. Bạn có thể nhấp vào đó và tên vai trò mà tôi muốn tạo là USER. Hãy để tôi lưu cái này. Và sau đó, tôi muốn tạo thêm một role nữa. Vì vậy, hãy để tôi quay lại Realm role và nhấp vào Create role này. Và vai trò thứ hai mà tôi muốn tạo là ADMIN. Vì vậy, chúng tôi có thể lưu các chi tiết này.



Realm roles > Create role

Create role

Role name \*

USER

Description

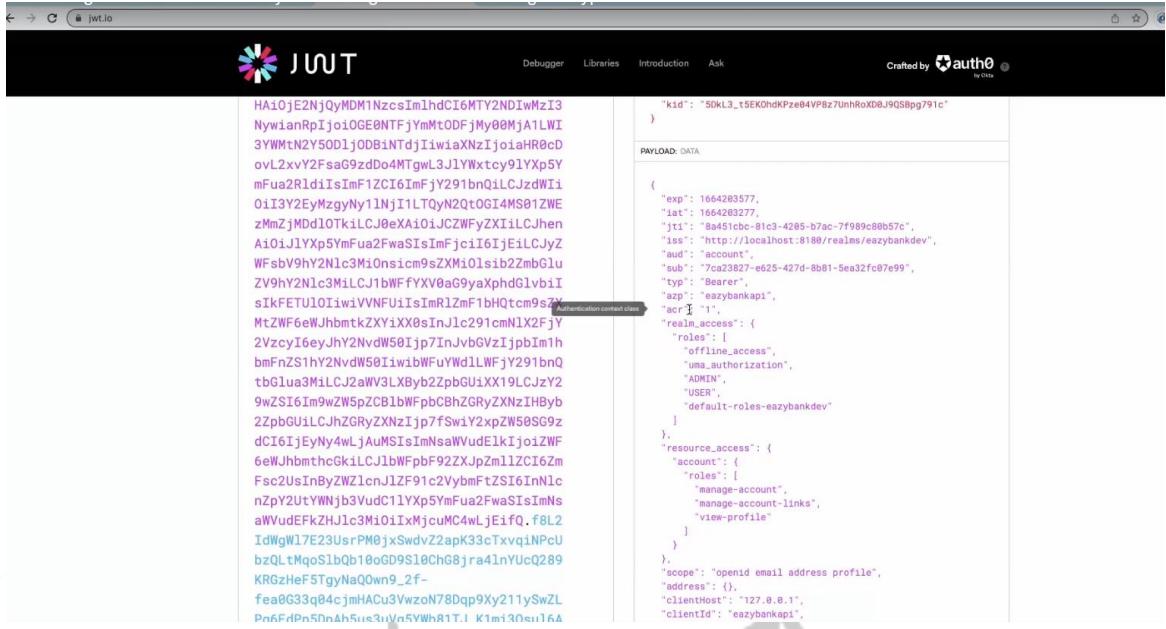
Save Cancel

Bây giờ, tôi mới tạo các role nhưng tôi chưa gán các role này cho client. Tương tự như vậy, tôi có thể truy cập Clients, tôi có thể mở Client của mình, đó là eazybankapi. Và ở đây, có một tab gọi là Service account roles. Nhưng ở đây, client này, là eazybankapi, là tài khoản dịch vụ được sử dụng để liên lạc API với API. Vì vậy, đó là lý do tại sao chúng ta cần nhấp vào Service account roles này. Và ở đây, tôi cần nhấp vào nút “Assign role” này và chọn tất cả các vai trò mà tôi muốn, giống như ADMIN and USER mà tôi đã tạo trước đó.

Name	Inherited	Description
default-roles-eazybankdev	False	\${role_default-roles}

Tôi đang nhấp vào nút Assign này. Cùng với đó, bây giờ client của tôi đã được chỉ định các vai trò này. Bây giờ, nếu tôi thử gọi lại điểm cuối tok này, lần này, tôi sẽ nhận được access token mới.

Hãy để tôi lấy cái này và tôi sẽ truy cập trang web JWT. Và ở đây, tôi sẽ dán nó.



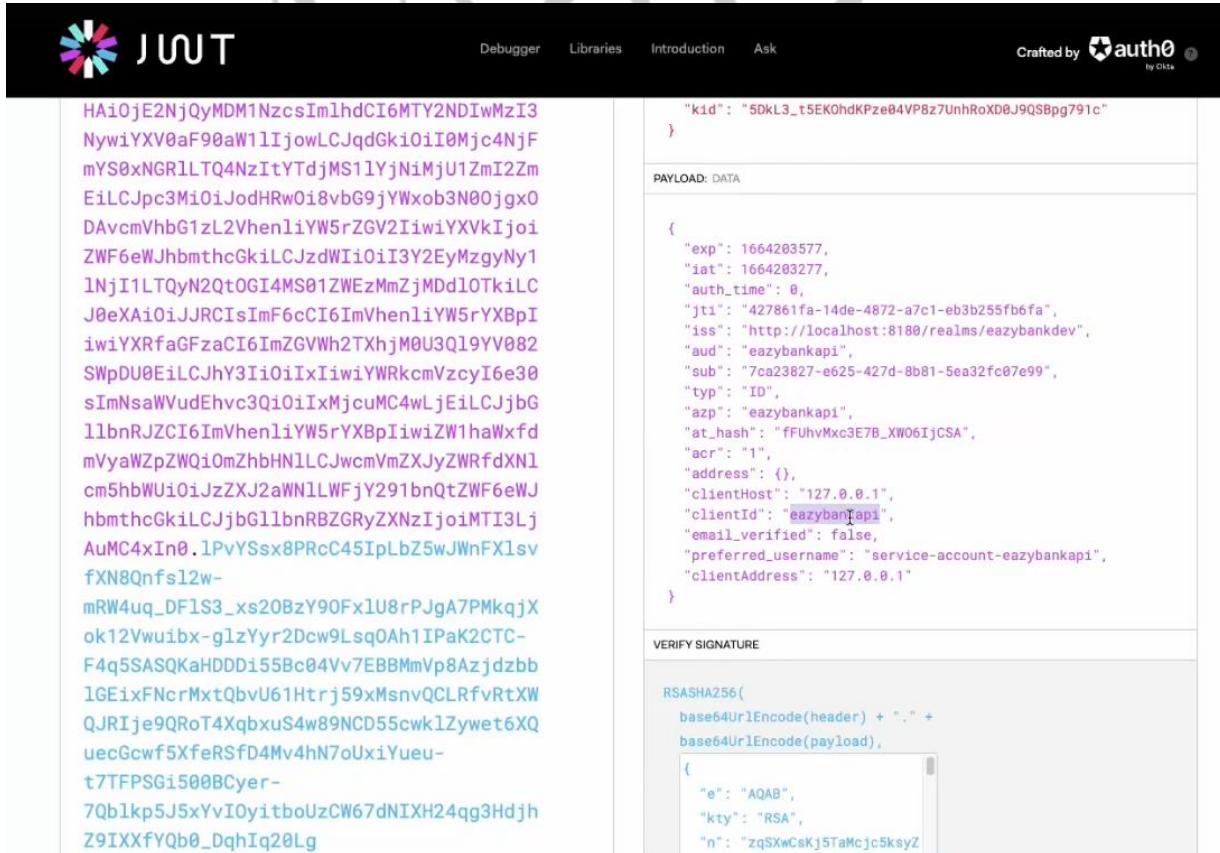
The screenshot shows a JWT token on the jwt.io website. The token is a long string of characters. The payload section is expanded, showing a JSON object with the following structure:

```

{
  "exp": 1664203577,
  "iat": 1664203277,
  "jti": "8a451bc-81c3-4285-b7ac-7f989c88b57c",
  "iss": "http://localhost:8180/realms/eazybankdev",
  "aud": "account",
  "sub": "7ca23827-e625-427d-8b81-5ea32fc07e99",
  "typ": "Bearer",
  "azp": "eazybankapi",
  "acr": "1",
  "realm_access": {
    "roles": [
      "offline_access",
      "uma_authorization",
      "ADMIN",
      "USER",
      "default-roles-eazybankdev"
    ]
  },
  "resource_access": {
    "account": {
      "roles": [
        "manage-account",
        "manage-account-links",
        "view-profile"
      ]
    }
  },
  "scope": "openid email address profile",
  "address": {},
  "clientHost": "127.0.0.1",
  "clientID": "eazybankapi"
}

```

Và lần này, bạn có thể thấy bên dưới `realm_access` và các vai trò chúng ta có các vai trò như `ADMIN` và `USER`. Bằng cách này, chúng tôi cũng đã chỉ định thành công các vai trò cho client mà chúng tôi đã tạo trước đó. Tương tự như vậy, hãy xác thực ID token mà chúng tôi nhận được. Vì vậy, hãy để tôi lấy ID token này và tôi đang dán ID token đó vào bên trong trang web JWT.



The screenshot shows a second JWT token on the jwt.io website, identical to the first one. The payload section is expanded, showing the same JSON object with the following structure:

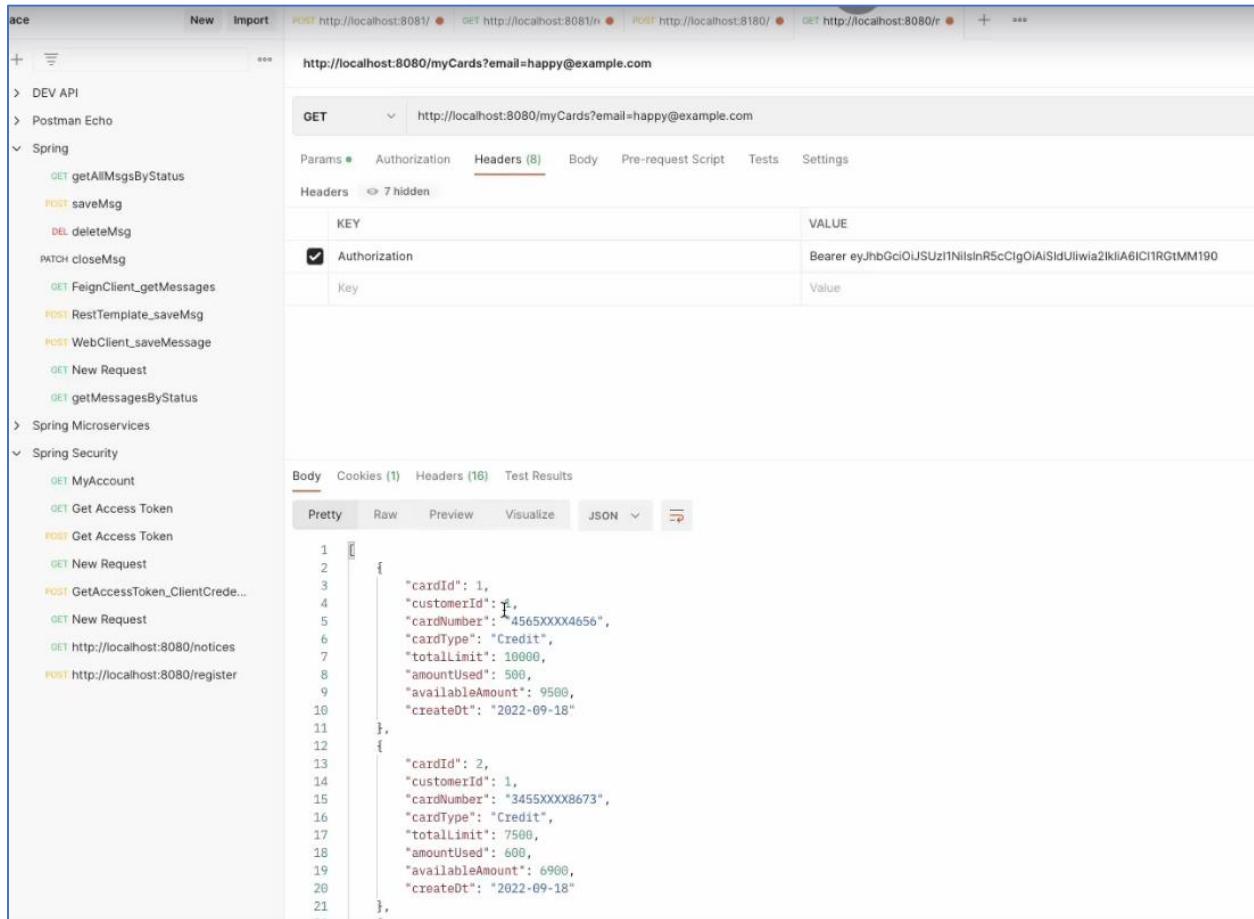
```

{
  "exp": 1664203577,
  "iat": 1664203277,
  "jti": "8a451bc-81c3-4285-b7ac-7f989c88b57c",
  "iss": "http://localhost:8180/realms/eazybankdev",
  "aud": "account",
  "sub": "7ca23827-e625-427d-8b81-5ea32fc07e99",
  "typ": "Bearer",
  "azp": "eazybankapi",
  "acr": "1",
  "realm_access": {
    "roles": [
      "offline_access",
      "uma_authorization",
      "ADMIN",
      "USER",
      "default-roles-eazybankdev"
    ]
  },
  "resource_access": {
    "account": {
      "roles": [
        "manage-account",
        "manage-account-links",
        "view-profile"
      ]
    }
  },
  "scope": "openid email address profile",
  "address": {},
  "clientHost": "127.0.0.1",
  "clientID": "eazybankapi"
}

```

## 7. Passing Access token to Resource server for response through Postman

### Lấy token từ access token



ace

New Import POST http://localhost:8081/ GET http://localhost:8081/r POST http://localhost:8180/ GET http://localhost:8080/r + \*\*\*

http://localhost:8080/myCards?email=happy@example.com

GET http://localhost:8080/myCards?email=happy@example.com

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Headers (7 hidden)

KEY	VALUE
Authorization	Bearer eyJhbGciOiJSUzI1NiIsInR5cC1gOjAiSldUliwia2IkIA6ICl1RGtMM190
Key	Value

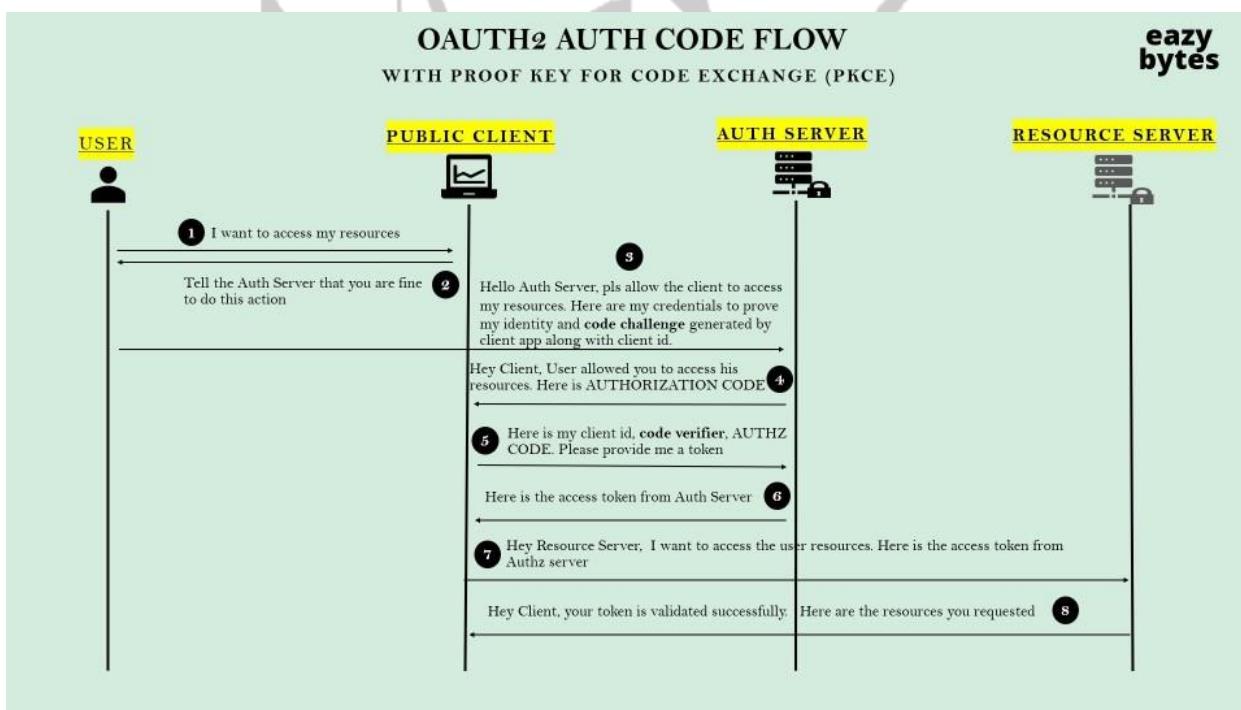
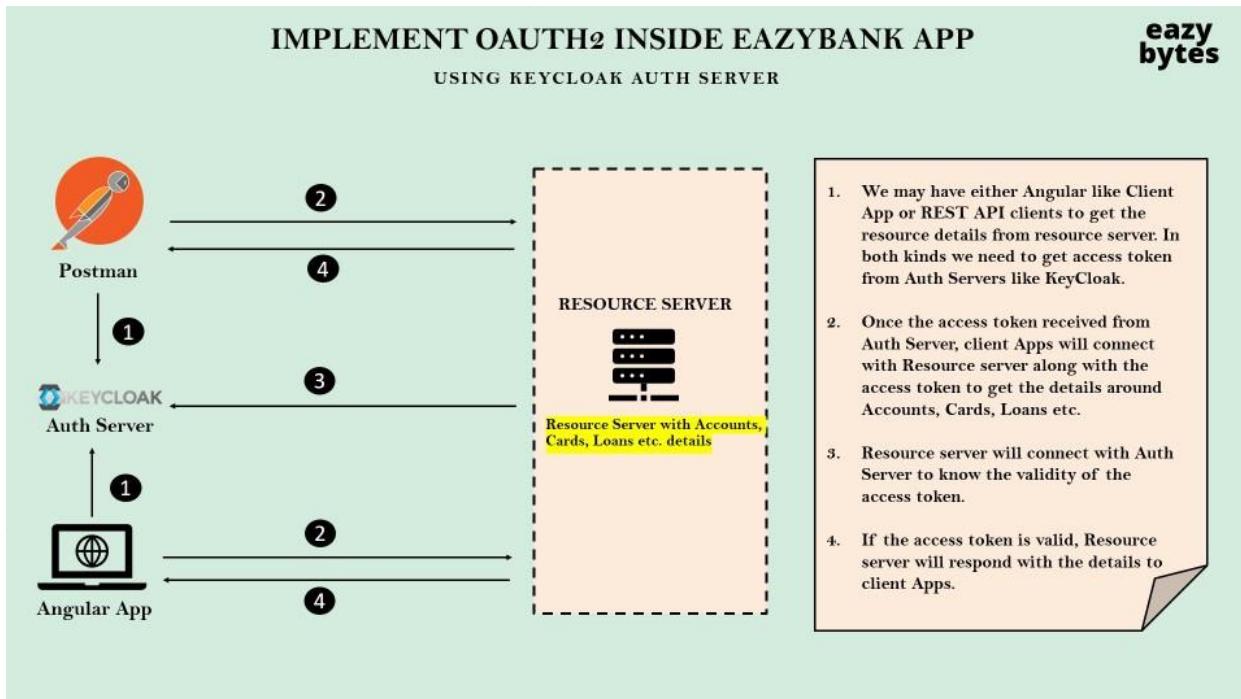
Body Cookies (1) Headers (16) Test Results

Pretty Raw Preview Visualize JSON ↻

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

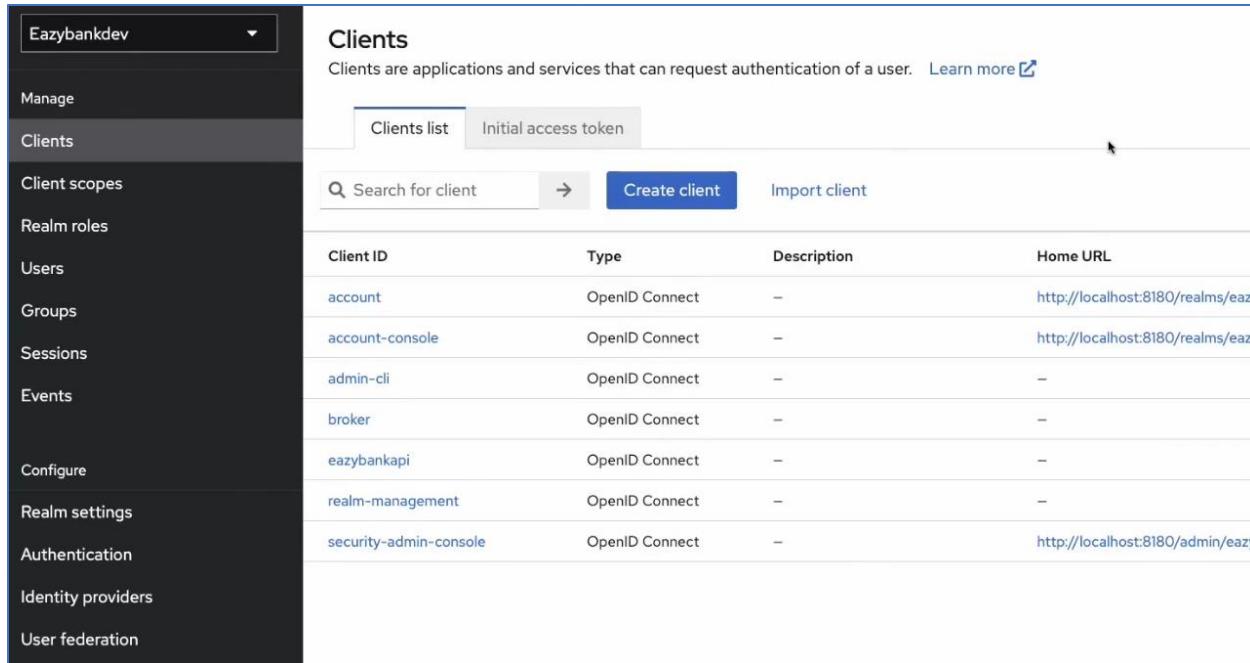
```
[{"cardId": 1, "customerId": 1, "cardNumber": "4565XXXX4656", "cardType": "Credit", "totalLimit": 10000, "amountUsed": 500, "availableAmount": 9500, "createDt": "2022-09-18"}, {"cardId": 2, "customerId": 1, "cardNumber": "3455XXXX8673", "cardType": "Credit", "totalLimit": 7500, "amountUsed": 600, "availableAmount": 6900, "createDt": "2022-09-18"}]
```

## 8. Understanding Authorization code grant type for EazyBank App



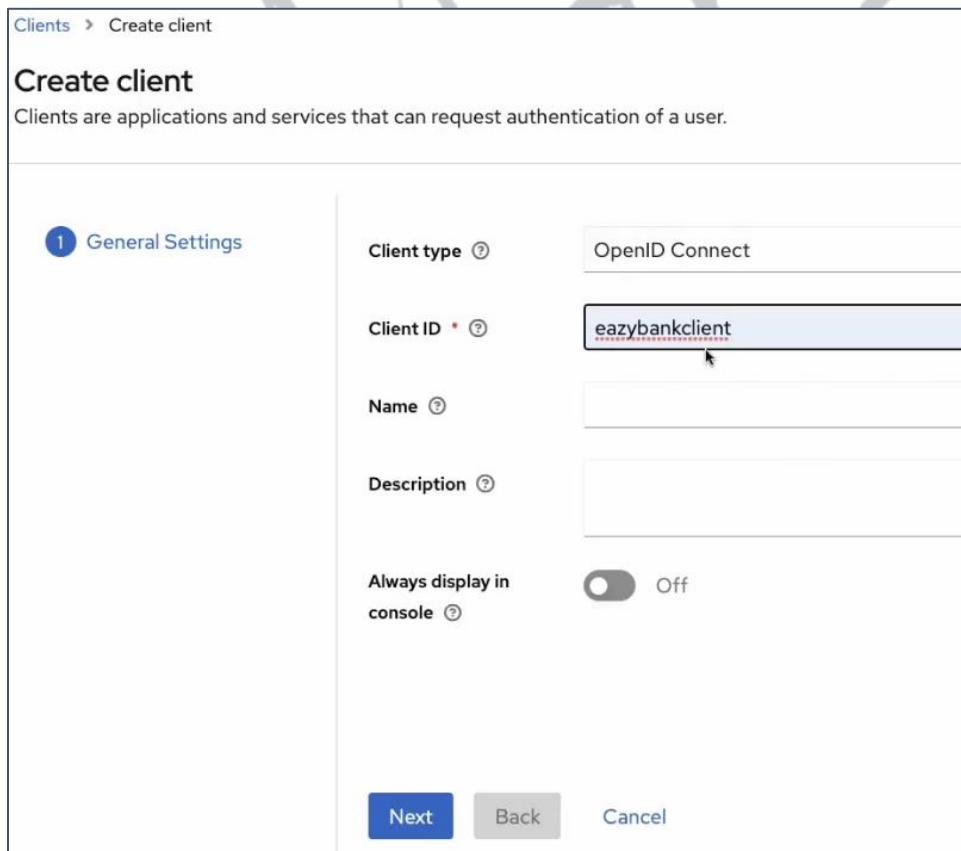
## 9. Creating Client and User details inside Keycloak for Auth code grant flow

Ngay bây giờ, tôi đang ở trong bảng điều khiển quản trị của Keycloak. Tôi sẽ nhấp vào "Clients", chúng ta có thể nhấp vào "Create Client" này.



Client ID	Type	Description	Home URL
account	OpenID Connect	–	http://localhost:8180/realms/eazy
account-console	OpenID Connect	–	http://localhost:8180/realms/eazy
admin-cli	OpenID Connect	–	–
broker	OpenID Connect	–	–
eazybankapi	OpenID Connect	–	–
realm-management	OpenID Connect	–	–
security-admin-console	OpenID Connect	–	http://localhost:8180/admin/eazy

Và lúc này Client ID mà tôi muốn tạo là eazybankclient. Vì vậy, đây là Client ID mà tôi muốn cung cấp và client type phải là chính OpenID và tôi sẽ nhấp vào nút tiếp theo này.



1 General Settings

Client type ⓘ OpenID Connect

Client ID \* ⓘ eazybankclient

Name ⓘ

Description ⓘ

Always display in console ⓘ  Off

Next Back Cancel

Và bên trong trang này, chúng tôi cần kích hoạt client authentication này, bởi vì, trước tiên, ứng dụng khách của tôi phải chứng minh danh tính của chính nó để access token. Đó là lý do tại sao chúng tôi cần đảm bảo rằng chúng tôi đang kích hoạt tính năng này và đến với quy trình xác thực, chúng tôi chỉ muốn xem xét standard flow, tức là, nếu chúng tôi có thể nhấp vào dấu chấm hỏi này authorization code flow.

Clients > Create client

## Create client

Clients are applications and services that can request authentication.

1 General Settings

2 Capability config

Client authentication

Authorization

This enables standard OpenID Connect redirect based authentication with authorization code. In terms of OpenID Connect or OAuth2 specifications, this enables support of 'Authorization Code Flow' for this client.

Authentication flow

Standard flow

Direct access grants

Implicit flow

Service accounts roles

OAuth 2.0 Device Authorization Grant

OIDC CIBA Grant

Vì vậy, đó là lý do tại sao chúng tôi cần đảm bảo rằng chúng tôi đang bật tính năng này và chúng tôi có thể tắt tính năng này vì chúng tôi không muốn tuân theo cấp resource owner password credentials grant.

Cùng với đó, tôi có thể nhấp vào nút lưu và bên trong trang này, nếu chúng tôi có thể nhấp vào credentials, bạn sẽ có thể xem client secret. Bây giờ, quay lại settings, bên trong trang này, nếu chúng ta có thể cuộn xuống, sẽ có một valid redirects URIs.

Client ID: eazybankclient

Name:

Description:

Always display in console:  Off

**Access settings**

Root URL:

Home URL:

Valid redirect URIs:  -

[+ Add valid redirect URIs](#)

Valid post logout redirect URIs:  -

[+ Add valid post logout redirect URIs](#)

Vì vậy, đây là nơi chúng tôi có thể cung cấp chi tiết URL mà Auth Server của tôi cần để chuyển hướng user đăng xác thực thành công. Vì vậy, tôi sẽ sử dụng một số URL giả, bởi vì hiện tại chúng tôi không có bất kỳ client UI application nào. Đó là lý do tại sao tôi đưa ra một số cổng ngẫu nhiên và đường dẫn ngẫu nhiên. Nhưng trong các tình huống thời gian thực, bạn cần cung cấp một URL thích hợp của UI application của mình mà user phải được chuyển hướng sau khi xác thực thành công. Hy vọng điều này là rõ ràng.

Root URL:

Home URL:

Valid redirect URIs:  http://localhost:7080/sample -

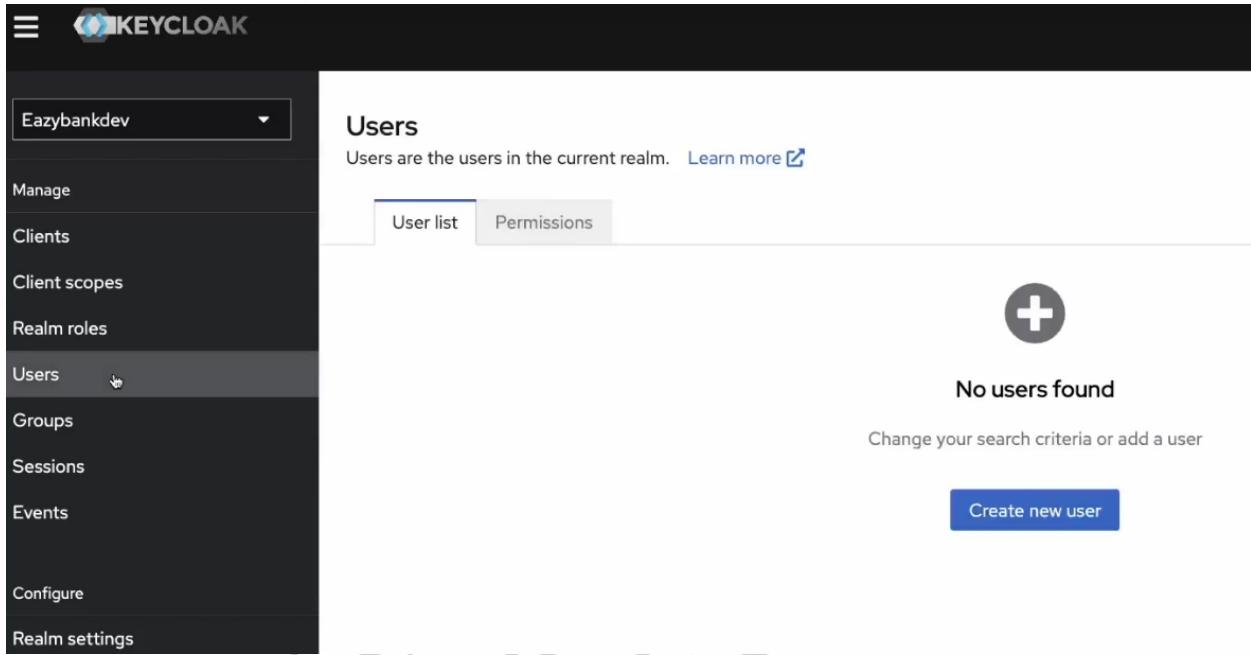
[+ Add valid redirect URIs](#)

Valid post logout redirect URIs:  -

[+ Add valid post logout redirect URIs](#)

Vì vậy, khi chúng tôi xác định tất cả các chi tiết này, chúng tôi có thể nhấp vào nút “Save”. Bây giờ, tôi đã tạo thành công client đó. Tôi có thể sử dụng ứng dụng này để kiểm tra authorization code grant type flow.

Và ở bước tiếp theo, chúng ta cũng nên tạo thông tin chi tiết về user, vì không có user thì sẽ không authorization code grant type flow. Tương tự như vậy, hãy để tôi nhấp vào tab “User” này. Và ở đây, tôi sẽ nhấp vào nút “Create User” này.



The screenshot shows the Keycloak administration interface. The left sidebar has a dropdown set to 'Eazybankdev' and a list of management options: Manage, Clients, Client scopes, Realm roles, Users (selected), Groups, Sessions, Events, Configure, and Realm settings. The main content area is titled 'Users' and contains the message 'Users are the users in the current realm.' Below this are two tabs: 'User list' (selected) and 'Permissions'. A large 'No users found' message is centered, with a 'Create new user' button below it. A search bar is also present.

Và tên người dùng mà tôi muốn tạo là [happy@example.com](mailto:happy@example.com). Vì vậy, chúng tôi cũng có thể cung cấp email tương tự, [happy@example.com](mailto:happy@example.com). Và người dùng này phải được kích hoạt, tôi có thể nhấp vào “Create” này.

Eazybankdev

Manage

Clients

Client scopes

Realm roles

**Users**

Groups

Sessions

Events

Configure

Realm settings

Authentication

Identity providers

User federation

Create user

Username \* happy@example.com

Email happy@example.com

Email verified  On

First name Happy

Last name Example

Enabled  On

Required user actions Select action

Groups

Ngay sau khi tôi tạo, tôi có thể truy cập thông tin đăng nhập và tại đây, tôi có thể đặt mật khẩu. Vì vậy, mật khẩu mà tôi muốn đặt ở đây là 12345. Tương tự, tôi sẽ đặt lại dưới phần xác nhận mật khẩu.

Eazybankdev

Manage

Clients

Client scopes

Realm roles

**Users**

Groups

Sessions

Events

Configure

Realm settings

Users > User details

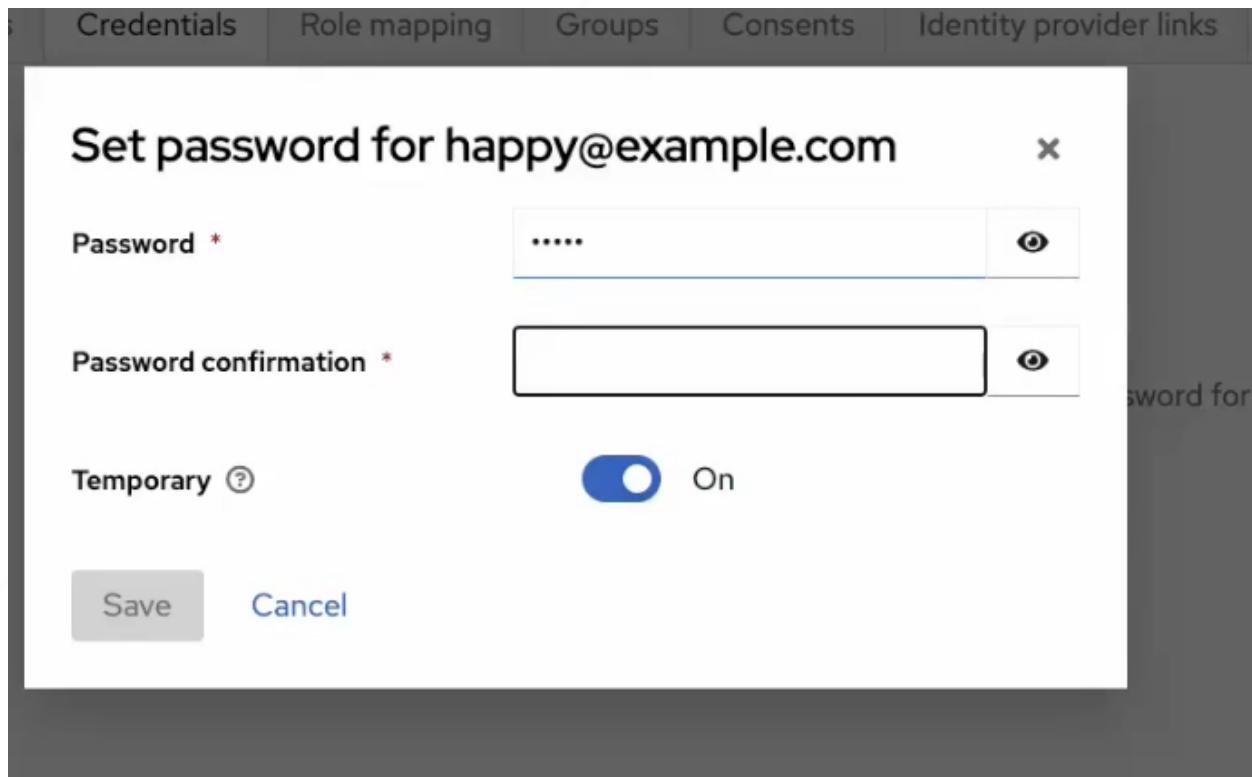
happy@example.com

Details Attributes **Credentials** Role mapping Groups Consents Identity provider links

No credentials

This user does not have any credentials. You can set password for th

Credential Reset



Vui lòng đảm bảo rằng bạn đang tắt temporary này để Keycloak của tôi không buộc user của tôi thay đổi mật khẩu trong lần đăng nhập đầu tiên. Tôi đang lưu mật khẩu này. Cùng với đó, chúng tôi đã tạo thành công một người dùng mới với tên happy@example.com. Vì vậy, ở đây, chúng tôi đã tạo người dùng theo cách thủ công bằng cách đăng nhập với tư cách quản trị viên vào máy chủ Keycloak của tôi.

## 10. Testing Authorization code grant type using Postman App

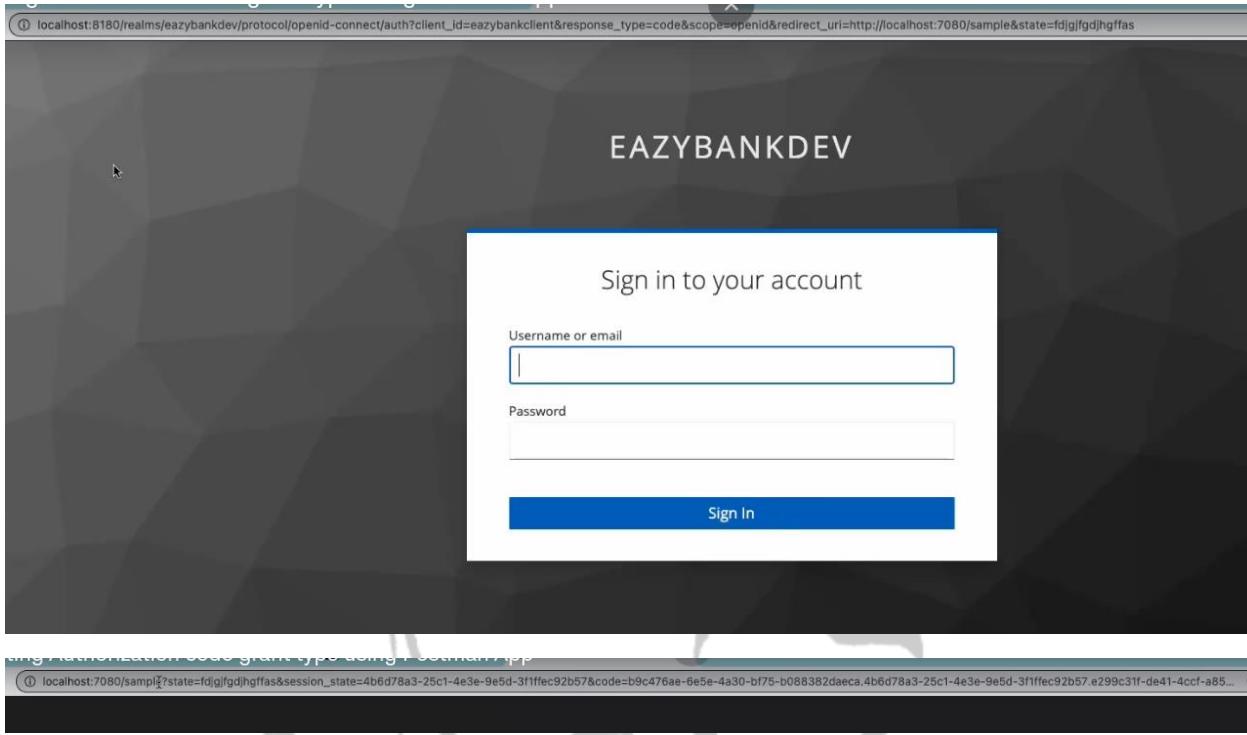
<http://localhost:8180/realm/eazybankdev/protocol/openid-connect/auth>

- Điền các params để lấy link truy cập từ postman

http://localhost:8180/realm/eazybankdev/protocol/openid-connect/auth?client\_id=eazybankclient&response\_type=code&scope=openid&redirect\_uri=http://localhost:7080/sample&state=fdjgjfgdjhgffas

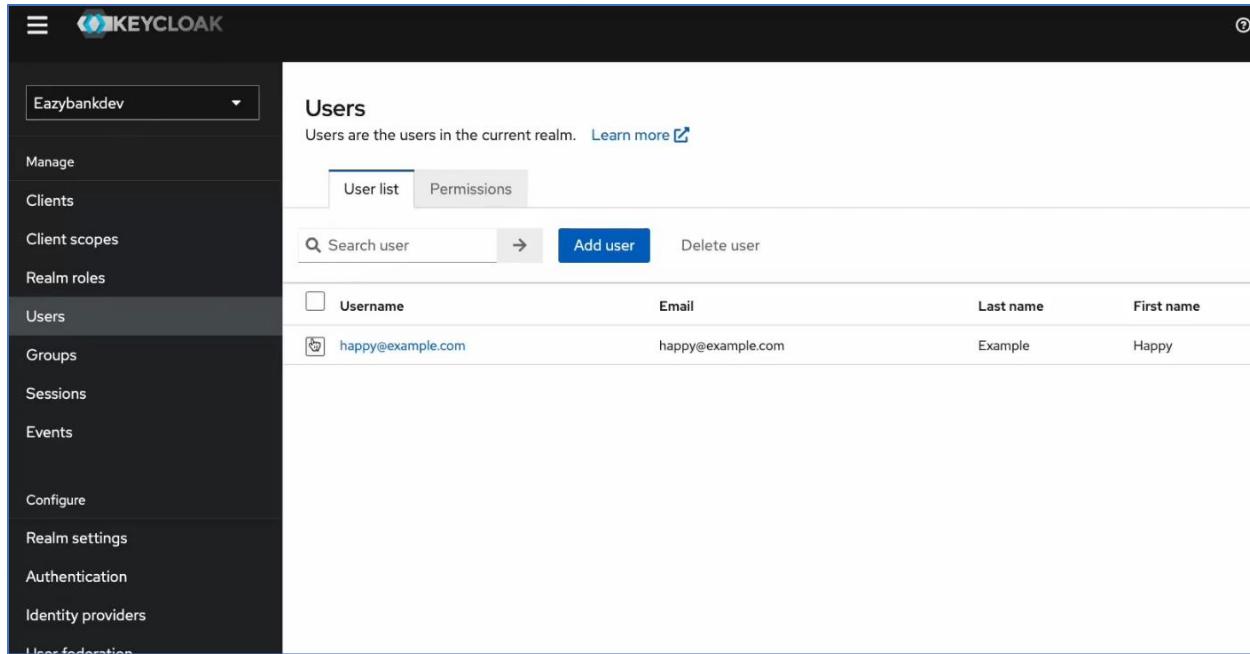
KEY	VALUE	DESCRIPTION
client_id	eazybankclient	
response_type	code	
scope	openid	
redirect_uri	http://localhost:7080/sample	
state	fdjgjfgdjhgffas	
Key	Value	Description

- ## - Đăng nhập



- Đăng nhập thành công sẽ lấy được code và scope dựa vào link
  - Lấy token: <http://localhost:8180/realmseazybankdev/protocol/openid-connect/token>

- Cấp role cho user



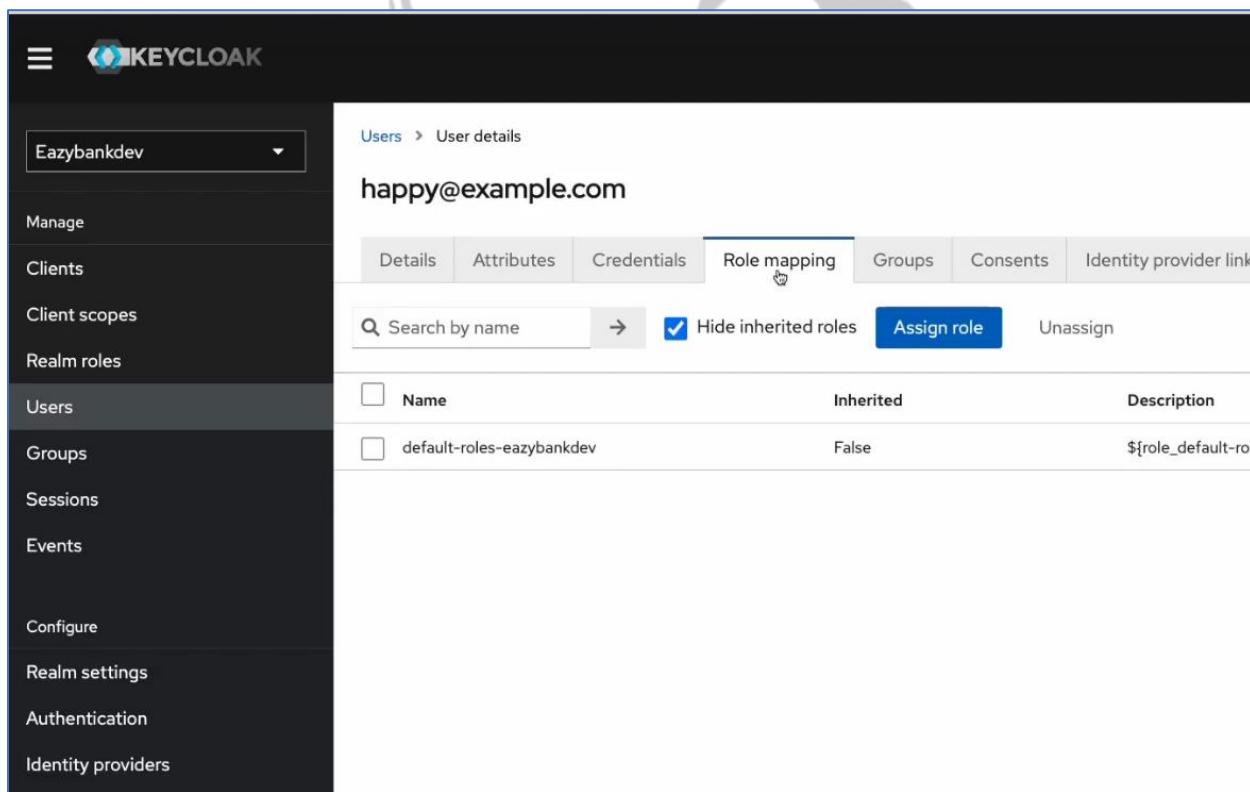
Users

Users are the users in the current realm. [Learn more](#)

User list Permissions

Search user Add user Delete user

Username	Email	Last name	First name
happy@example.com	happy@example.com	Example	Happy



Users > User details

happy@example.com

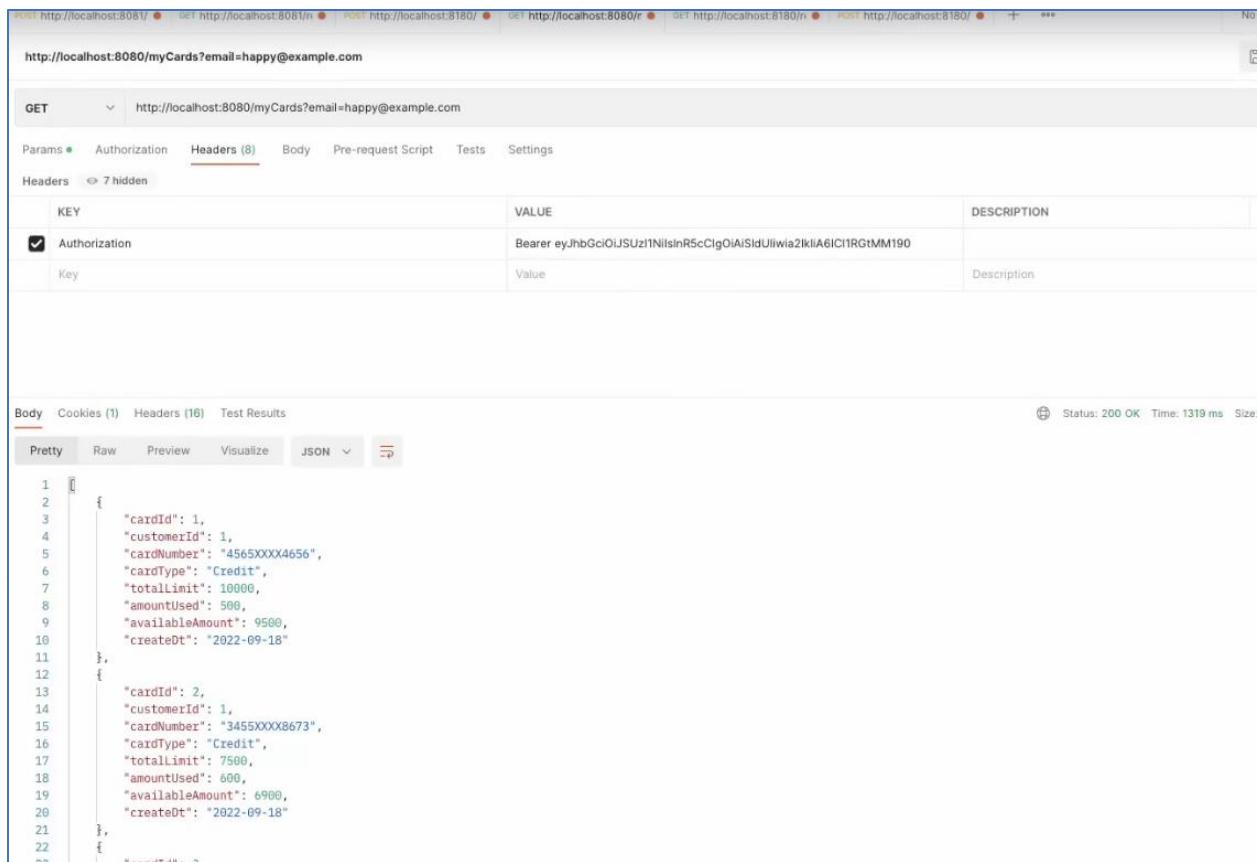
Role mapping

Details Attributes Credentials Groups Consents Identity provider links

Search by name Hide inherited roles Assign role Unassign

Name	Inherited	Description
default-roles-eazybankdev	False	\${role_default-role}

- Thêm các role cho user và call API



http://localhost:8080/myCards?email=happy@example.com

GET http://localhost:8080/myCards?email=happy@example.com

Params • Authorization Headers (8) Body Pre-request Script Tests Settings

Headers 7 hidden

KEY	VALUE	DESCRIPTION
Authorization	Bearer eyJhbGciOiJSUzI1NiIsInR5cC1gOjAiSldUliwa2IkliA6IC1RGtMM190	
Key	Value	Description

Body Cookies (1) Headers (16) Test Results

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "cardId": 1,
4     "customerId": 1,
5     "cardNumber": "4565XXXX4656",
6     "cardType": "Credit",
7     "totalLimit": 10000,
8     "amountUsed": 500,
9     "availableAmount": 9500,
10    "createDt": "2022-09-18"
11  },
12  [
13    {
14      "cardId": 2,
15      "customerId": 1,
16      "cardNumber": "3455XXXX8673",
17      "cardType": "Credit",
18      "totalLimit": 7500,
19      "amountUsed": 600,
20      "availableAmount": 6900,
21      "createDt": "2022-09-18"
22    }
23 ]

```

Status: 200 OK Time: 1319 ms Size

## 11. Deep dive on Authorization code grant type with PKCE

Chúng tôi cần triển khai hoặc tạo framework cho cổng 80 mở của mình bất cứ khi nào angular application của tôi đang cố gắng giao tiếp với spring boot web application của tôi. Đây là cải tiến tiếp theo mà chúng tôi có thể thử bên trong ứng dụng web EazyBank của mình.

Và vì sẽ có user tham gia nên chúng tôi chắc chắn phải tuân authorization code grant type flow của framework OR2. Nhưng tôi có thể triển khai luồng authorization code grant type flow bên trong Angular UI application gốc cạnh của mình không? Tất nhiên, chúng tôi không thể thực hiện. Lý do là đây là một UI application, ứng dụng dựa trên JavaScript của chúng tôi và UI application dùng này không thể ẩn client thành giá trị secret bên trong mã của nó. Bởi vì bất kỳ nhà phát triển hay bất kỳ hacker nào họ có thể dễ dàng nhìn thấy mã JavaScript mà Angular UI application của tôi đang sử dụng bằng cách xem mã nguồn bên trong trình duyệt. Vì vậy, để xử lý các loại tình huống này, chúng tôi có thêm một loại authorization code, đó là PKCE. Vì vậy, hãy để tôi giới thiệu một quy trình PKCE cho bạn.

- ✓ When public clients (e.g., native and single-page applications) request Access Tokens, some additional security concerns are posed that are not mitigated by the Authorization Code Flow alone. This is because public clients cannot securely store a Client Secret.
- ✓ Given these situations, OAuth 2.0 provides a version of the Authorization Code Flow for public client applications which makes use of a Proof Key for Code Exchange (PKCE).

- ✓ The PKCE-enhanced Authorization Code Flow follows below steps,
  - Once user clicks login, client app creates a cryptographically-random **code\_verifier** and from this generates a **code\_challenge**.
  - code challenge is a Base64-URL-encoded string of the SHA256 hash of the code verifier.
  - Redirects the user to the Authorization Server along with the code\_challenge.
  - Authorization Server stores the code\_challenge and redirects the user back to the application with an authorization code, which is good for one use.
  - Client App sends the authorization code and the code\_verifier(created in step 1) to the Authorization Server.
  - Authorization Server verifies the code\_challenge and code\_verifier. If they are valid it respond with ID Token and Access Token (and optionally, a Refresh Token).

Giống như bạn có thể thấy ở đây PKCE có nghĩa là khóa bằng chứng để trao đổi mã. Đó là mô tả đầy đủ về PKCE. Tại sao chúng ta cần phải làm theo những điều này. Một trường hợp khác của authorization code flow cùng với PKCE là các public clients như ứng dụng di động, native application, ứng dụng dựa trên JavaScript, single page application mà chúng có giới hạn là không thể lưu trữ client secret bên trong code base của chúng.

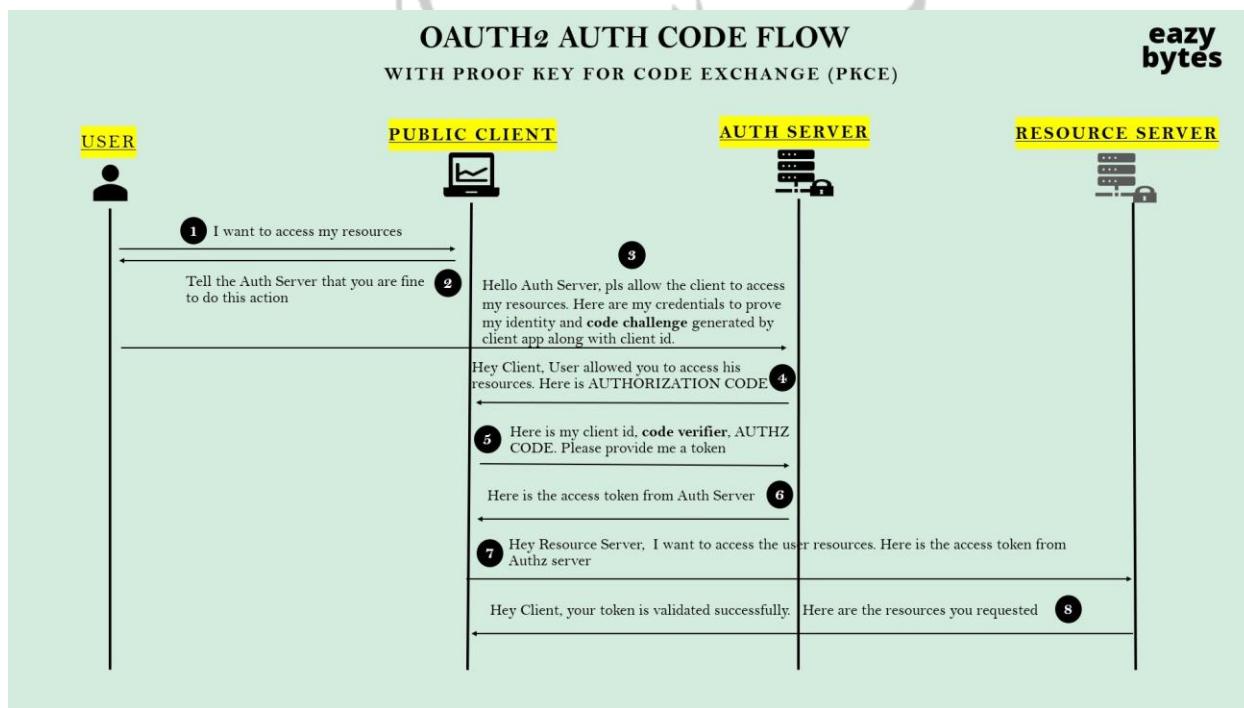
Để vượt qua thử thách này, chúng tôi cần tuân theo authorization code flow cùng với PKCE. Vì vậy, hãy cố gắng hiểu PKCE này sẽ giúp chúng ta vượt qua thử thách này như thế nào. Là một phần của phương pháp PKCE bất cứ khi nào client application của tôi chuyển hướng user của tôi, resource owner của chúng tôi đến trang đăng nhập của Auth Server của tôi. Trong nội bộ, nó sẽ tạo ra một giá trị ngẫu nhiên được gọi là code verifier. Từ code verifier này, nó sẽ tạo ra một code challenge. Để tạo code challenge từ trình code verifier, client application của tôi, trước tiên, nó sẽ áp dụng thuật toán SHA256# trên code verifier của tôi, thuật toán này được tạo ngẫu nhiên. Và một khi nó có giá trị băm của code verifier, client application của tôi có thể chuyển đổi chuỗi băm đó thành mã hóa Base64-URL. Giá trị được mã hóa Base64-URL đó sẽ là code challenge của bạn.

Vì vậy, bây giờ client application của tôi có code verifier và code challenge. Vì vậy, code verifier là một giá trị được tạo ngẫu nhiên bởi client của tôi và từ code verifier. Bằng cách làm theo công thức mà chúng ta đã thảo luận, nó sẽ tạo ra code challenge.

Ở những bước đầu tiên khi client application của tôi đang cố lấy authorization code chứ không phải access token, nó sẽ gửi code challenge đến authorization server. Nó sẽ không gửi cả code verifier và code challenge. Ở bước đầu tiên, nó sẽ chỉ gửi code challenge. Bây giờ Auth Server của tôi sẽ thực hiện code challenge đó và lưu trữ bên trong hệ thống lưu trữ tương ứng với authorization code mà nó sẽ cấp cho client application. Vì vậy, bây giờ client application của tôi sẽ nhận được authorization code. Auth Server của tôi sẽ không gửi code challenge trả lại client application của tôi. Nó sẽ chỉ lưu trữ bên trong hệ thống lưu trữ của nó.

Trong bước tiếp theo, bất cứ khi nào client application của tôi đang cố lấy access token, lần này, nó sẽ gửi giá trị code verifier cùng với giá trị authorization code mà Auth Server của tôi đã cấp trong bước đầu tiên. Sau khi Auth Server của tôi nhận được code verifier cùng với authorization code mà nó đã cấp ban đầu trong bước đầu tiên. Đằng sau hậu trường, Auth Server của tôi sẽ kiểm tra code verifier mà nó nhận được từ client application và nó cũng sẽ tuân theo cùng một hàm băm như SHA256. Và một khi nó có chuỗi băm của code verifier, nó sẽ so sánh với code challenge mà nó đã nhận được trước đó trong bước đầu tiên mà nó lưu trữ dựa trên authorization code. Nếu cả hai đều phù hợp thì chỉ có nó mới cấp access token cho client application.

Vì vậy, nếu bạn thấy ở đây, lợi thế chính của PKCE flow này là bất kỳ client application nào bắt đầu bước đầu tiên, client application đó sẽ nhận được access token. Hãy nghĩ về một tình huống trong đó client application của tôi bắt đầu bước đầu tiên để lấy authorization code cùng với code challenge. Và nếu một hacker nào đó đang cố lấy cắp authorization code mà tôi đã sử dụng từ máy chủ OAuth hoặc nếu anh ta đang cố giả mạo authorization code đó, thì nó sẽ không hoạt động vì bất cứ khi nào anh ta cố lấy quyền truy cập để mở, anh ta cũng nên biết code verifier ban đầu được client application của tôi sử dụng để tính toán code challenge là gì. Vì vậy, nếu không có giá trị xác minh mã đó, hacker của tôi không thể làm bất cứ điều gì với authorization code mà anh ta nhận được. Vì vậy, với code verifier và cơ chế code challenge này, chúng tôi đang gián tiếp mang đến cơ chế bảo mật mặc dù client application của tôi là public client.



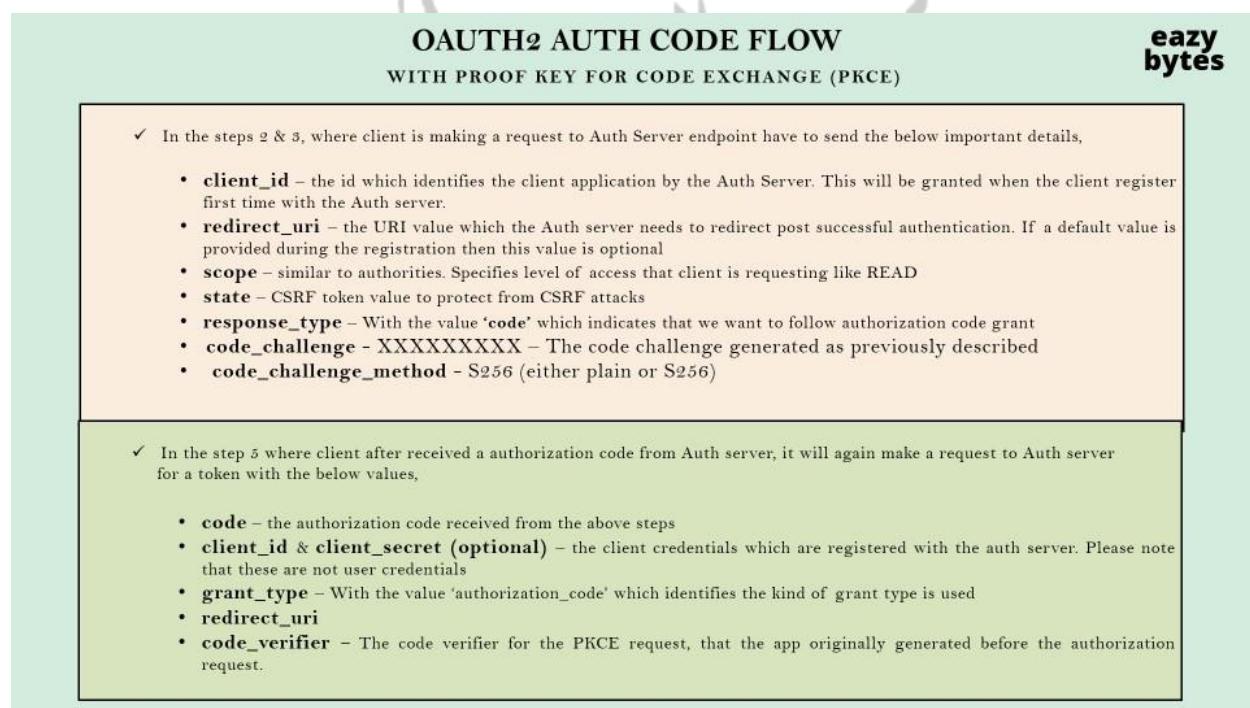
Như thường lệ, user của tôi sẽ liên hệ với public client nói rằng tôi muốn truy cập các tài nguyên như vậy. Bây giờ, public client của tôi sẽ thông báo cho user rằng tôi sẽ chuyển hướng bạn đến trang đăng nhập Auth Server. Hãy chứng minh danh tính của bạn. Tôi có thể cung cấp cho bạn các tài nguyên an toàn mà bạn đã yêu cầu. Vì vậy, đằng sau hậu trường, public client của tôi, khi nó chuyển hướng user đến trang đăng nhập Auth Server, nó sẽ chia sẻ thử thách mã với Auth Server của tôi cùng với client id.

Auth Server của tôi sẽ code challenge đó bên trong hệ thống lưu trữ dựa trên authorization code mà nó sẽ phát hành trong bước bốn.

Sau khi client application của tôi nhận được authorization code. Lần này, nó sẽ thực hiện thêm một yêu cầu tới Auth Server để nhận access token. Vì vậy, để nhận được access token thành công, client application của tôi phải gửi client ID. Và code verifier mà nó đã sử dụng ban đầu để tính toán code challenge là gì. Và cùng với code verifier, nó cũng phải chuyển authorization code mà nó đã nhận được ở bước bốn.

Vì vậy, sau khi tất cả các chi tiết này được gửi đến Auth Server, từ public client của tôi trong bước năm. Ở hậu trường, Auth Server của tôi sẽ sử dụng code verifier đó và nó sẽ tuân theo cùng một thuật toán băm, cùng một công thức. Và một khi nó có giá trị băm từ code verifier, nó sẽ so sánh với code challenge mà nó đã nhận được ban đầu. Nếu cả hai giá trị băm đều khớp thì chỉ Auth Server của tôi sẽ cấp access token cho public client. Sau đó, câu chuyện cũng tương tự như public client của tôi có thể gửi access token đó đến resource server và nó có thể nhận được phản hồi thích hợp từ resource server của tôi. Nếu bạn có thể thấy ở đây, đây là các thành phần yêu cầu hoặc dữ liệu yêu cầu mà client của tôi phải gửi.

Trong bước hai và ba, nơi nó đang cố lấy authorization code, nó phải client ID, redirect URL, scope là gì? state là gì? Response type là gì và code challenge là gì?



Code challenge bắt nguồn từ code verifier mà client application của tôi tạo ngẫu nhiên. Và cùng với code challenge, chúng ta cũng nên nói với Auth Server challenge method mà chúng ta đang theo dõi là gì. Trong hầu hết các trường hợp, chúng tôi tuân theo thuật toán SHA256# ở dạng ngắn, chúng tôi cần gửi giá trị đó dưới dạng S256. Và khi client application của tôi được sử dụng và authorization code ở bước năm, ứng dụng sẽ gửi authorization code đó dưới dạng đầu vào đầu tiên. Cùng với client ID đó, nó cũng có thể gửi client secret. Vì vậy, client secret là một tùy chọn ở đây. Nếu client application của bạn là

public client application, thì ứng dụng đó không phải gửi client secret vì đây là tùy chọn. Và đến grant type, nó phải gửi authorization code đến redirect URI.

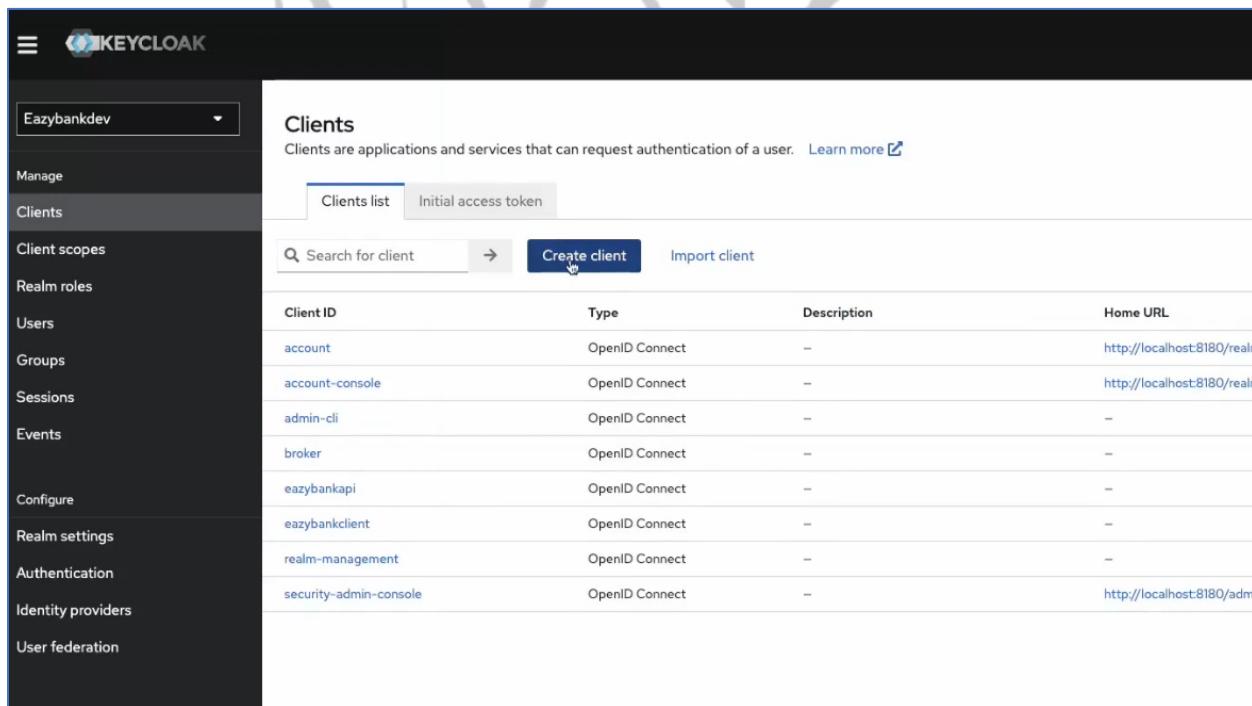
Và cuối cùng, nó cũng sẽ gửi giá trị code verifier mà client application của tôi được sử dụng để tạo giá trị băm code challenge. Vì vậy, đây là những yếu tố đầu vào mà client application của tôi phải chia sẻ với Auth Server.

PKCE ban đầu được thiết kế để bảo vệ luồng authorization code bất cứ khi nào nó được sử dụng bên trong các public facing clients như mobile apps, single page applications, JavaScript applications. Nhưng sau này, nhiều tổ chức trong ngành, họ thấy rất nhiều khả năng hoặc tiềm năng bên trong PKCE này vì với cách tiếp cận này, họ có thể tránh được bất kỳ loại tấn công injection authorization code nào hoặc một số tin tức cố gắng giả mạo authorization code. Đó là lý do tại sao ngày nay khuyến nghị là sử dụng PKCE ngay cả khi client application của bạn có thể lưu trữ client secret một cách an toàn, điều đó sẽ làm cho dòng authorization code của bạn an toàn hơn. Nhưng nếu ứng dụng khách của bạn không thể lưu trữ client secret, thì rõ ràng bạn có thể làm theo phương pháp PKCE mà không cần client secret.

## 12. Creating public facing client details inside Keycloak server

Để bắt đầu với PKCE flow bên trong Angular UI application của chúng tôi, trước tiên chúng tôi cần đăng ký chi tiết Angular application client trong máy chủ Keycloak auth. Tương tự, tôi đã đến bảng điều khiển quản trị của Keycloak.

Để tạo client, chúng tôi cần nhấp vào tab client này mà chúng tôi có và tôi nhấp vào **create client** này.



Client ID	Type	Description	Home URL
account	OpenID Connect	–	http://localhost:8180/realm
account-console	OpenID Connect	–	http://localhost:8180/realm
admin-cli	OpenID Connect	–	–
broker	OpenID Connect	–	–
eazypublicapi	OpenID Connect	–	–
eazypublicclient	OpenID Connect	–	–
realm-management	OpenID Connect	–	–
security-admin-console	OpenID Connect	–	http://localhost:8180/admin
eazypublicclient	OpenID Connect	–	–

Client ID mà tôi muốn cung cấp ở đây là eazypublicclient, vì client application là public-facing UI application. Vì vậy, đó là lý do tại sao tôi đặt tên này là eazypublicclient. Khi bạn đã xác định client ID, bạn có thể nhấp vào nút “Next” này.

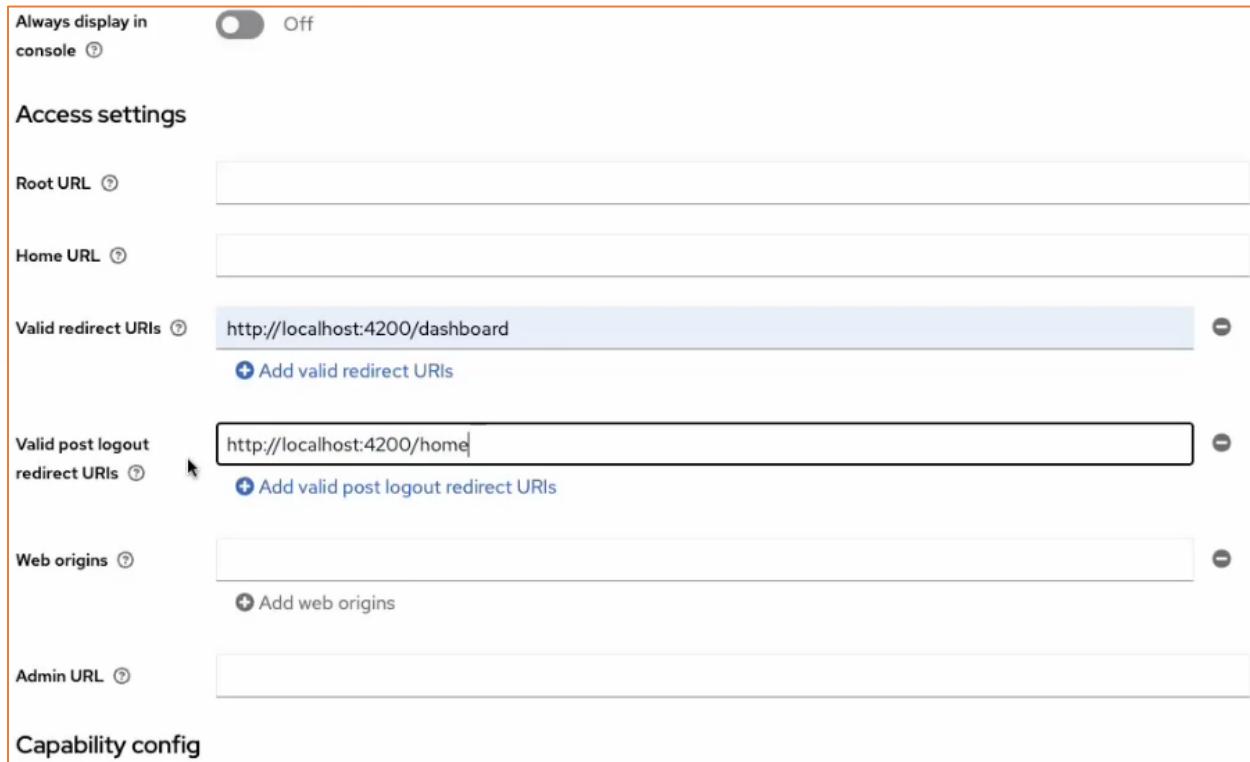
The screenshot shows the 'Create client' page in Keycloak. The left sidebar is titled 'Ezybankdev' and includes 'Manage', 'Clients', 'Client scopes', 'Realm roles', 'Users', 'Groups', 'Sessions', 'Events', 'Configure', 'Realm settings', 'Authentication', 'Identity providers', and 'User federation'. The main content area is titled 'Create client' with the sub-instruction 'Clients are applications and services that can request authentication of a user.' It shows 'General Settings' (Client type: OpenID Connect, Client ID: eazypublicclient, Name: [empty], Description: [empty], Always display in console: Off) and 'Client authentication' (Authorization: [empty], Authentication flow: Standard flow checked, Implicit flow, OAuth 2.0 Device Authorization Grant, Service accounts roles, OIDC CIBA Grant).

Và ở đây, thông tin quan trọng mà chúng ta cần quan tâm là, của chúng ta là một public client, đối với tất cả các public client, client authentication này sẽ bị vô hiệu hóa để chúng ta không bao giờ phải chia sẻ client secret với keycloak auth server. Đó là lý do tại sao tôi không kích hoạt client authentication này. Và chúng tôi có thể xóa direct access grant này. Chúng tôi chỉ muốn tuân theo standard flow, đó là authorization code flow.

The screenshot shows the 'Create client' page in Keycloak, similar to the previous one but with a tooltip. The tooltip for the 'Standard flow' checkbox in the 'Authentication flow' section states: 'This enables standard OpenID Connect redirect based authentication with authorization code. In terms of OpenID Connect or OAuth2 specifications, this enables support of 'Authorization Code Flow' for this client.' The checkbox is checked. Other options like 'Implicit flow', 'OAuth 2.0 Device Authorization Grant', 'Service accounts roles', and 'OIDC CIBA Grant' are unchecked.

Và bây giờ, tôi sẽ nhấp vào nút “Save” này. Cùng với đó, lần này, bạn có thể thấy, không có tab **credentials** nào bên trong **client** của tôi, vì đây là public-facing client. Bây giờ, nếu bạn có thể cuộn

xuống, có một redirect URL mà chúng tôi cần đề cập, chẳng hạn như tôi chỉ muốn chuyển hướng người dùng đến trang dashboard webpage của Angular web application của tôi bất cứ khi nào xác thực thành công. Và rất tương tự, chúng ta cũng có thể định cấu hình logout redirect URI. Bất cứ khi nào user của tôi cố gắng đăng xuất, tôi muốn chuyển hướng anh ta đến trang chủ của Angular application của tôi, đó là lý do tại sao tôi đưa ra điều này.



Always display in  Off  
console ⓘ

**Access settings**

**Root URL** ⓘ

**Home URL** ⓘ

**Valid redirect URLs** ⓘ

**Valid post logout redirect URLs** ⓘ

**Web origins** ⓘ

**Admin URL** ⓘ

**Capability config**

Và khi chúng tôi xác định tất cả các chi tiết này, bạn có thể nhấp vào nút **“Save”** này và sau đó, chúng tôi cần chuyển đến tab **“Advanced”** mà chúng tôi có.

Vì vậy, bên trong tab **“Advanced”** này, nếu bạn có thể tìm kiếm **proof**, sẽ có một danh sách thả xuống mà chúng ta cần chọn để hiểu **PKCE challenge method** mà chúng ta sẽ thực hiện là gì. Chúng tôi sẽ tuân theo S256, có nghĩa là chúng tôi sẽ tuân theo thuật toán băm SHA-256.

Advanced Settings

This section is used to configure advanced settings of this client related to OpenID Connect protocol

Access Token Lifespan: Never expires

OAuth 2.0 Mutual TLS: Off

Challenge Method: S256

Pushed authorization request required: plain

ACR to LoA Mapping: Type a key, Type a value

Default ACR Values: Add

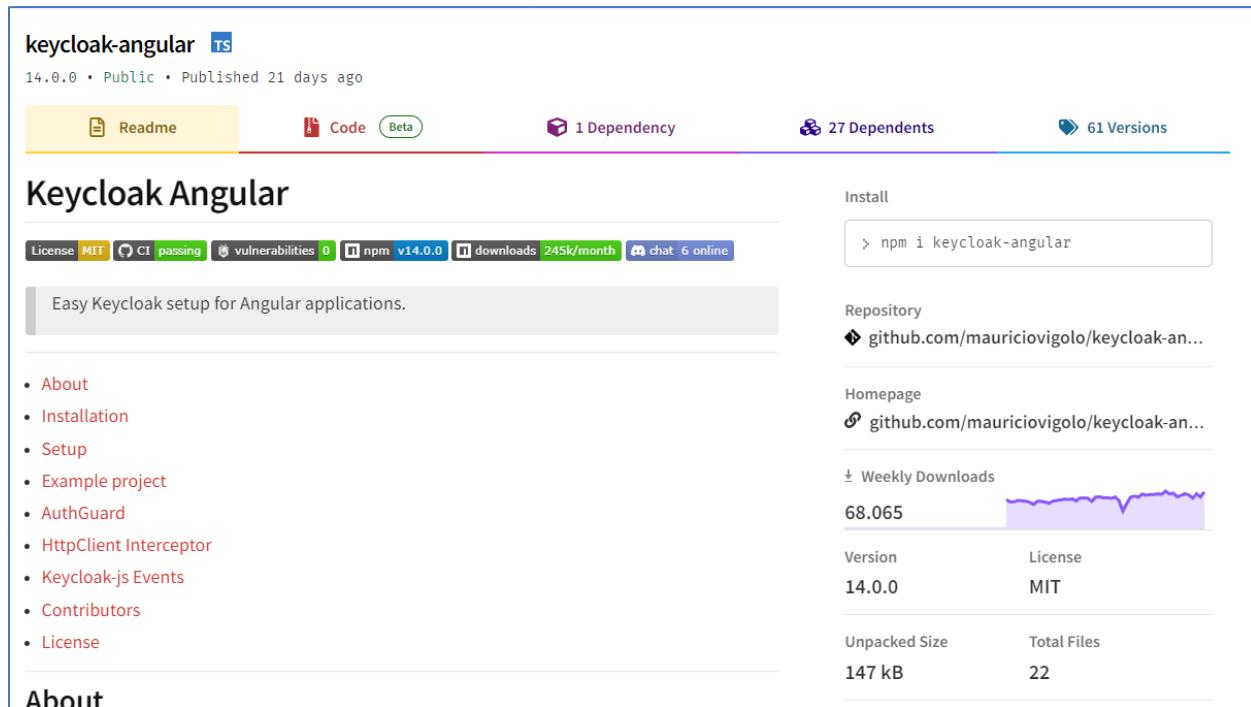
Plain cũng là một trong những tùy chọn, nhưng Plain rất không an toàn. Đó là lý do tại sao chúng tôi cần đảm bảo rằng chúng tôi luôn tuân theo S256 challenging method này. Cùng với đó, public client của tôi đã sẵn sàng bên trong authorization server của tôi.

### 13. Implementing PKCE Authorization code grant type inside Angular UI App

Nhưng tất cả điều này dường như là công việc rất siêu phức tạp, bởi vì client application của tôi là Angular application phải tạo **code verifier**. Từ đó, nó phải tạo ra một **code challenge** và nó sẽ chuyển hướng người dùng đến Keycloak auth server. Nó sẽ cố lấy access token. Vì vậy, tất cả điều này dường như siêu, siêu phức tạp nếu chúng ta cố gắng viết tất cả logic này bằng tay.

Để thực hiện điều này rất đơn giản đối với các Angular application, có một thư viện gọi là "Keycloak Angular". Thư viện Keycloak Angular này là package, nó sẽ giúp cuộc sống của nhà phát triển trở nên cực kỳ dễ dàng khi họ đang cố gắng tích hợp với Keycloak auth server với PKCE authorization grant type flow.

Installation: npm install keycloak-angular keycloak-js



keycloak-angular TS

14.0.0 • Public • Published 21 days ago

Readme Code Beta 1 Dependency 27 Dependents 61 Versions

## Keycloak Angular

License MIT CI passing vulnerabilities 0 npm v14.0.0 downloads 245k/month chat 6 online

Easy Keycloak setup for Angular applications.

- About
- Installation
- Setup
- Example project
- AuthGuard
- HttpClient Interceptor
- Keycloak-js Events
- Contributors
- License

Install

```
> npm i keycloak-angular
```

Repository

Homepage

Weekly Downloads

68.065

Version 14.0.0 License MIT

Unpacked Size 147 kB Total Files 22

### About

Họ đã cung cấp tất cả các bước mà bạn có thể làm theo như cách thiết lập các thư viện này, cách chúng tôi có thể tận dụng chúng để triển khai PKCE và kết nối với Keycloak auth server.

## Setup

In order to make sure Keycloak is initialized when your application is bootstrapped you will have to add an `APP_INITIALIZER` provider to your `AppModule`. This provider will call the `initializeKeycloak` factory function shown below which will set up the Keycloak service so that it can be used in your application.

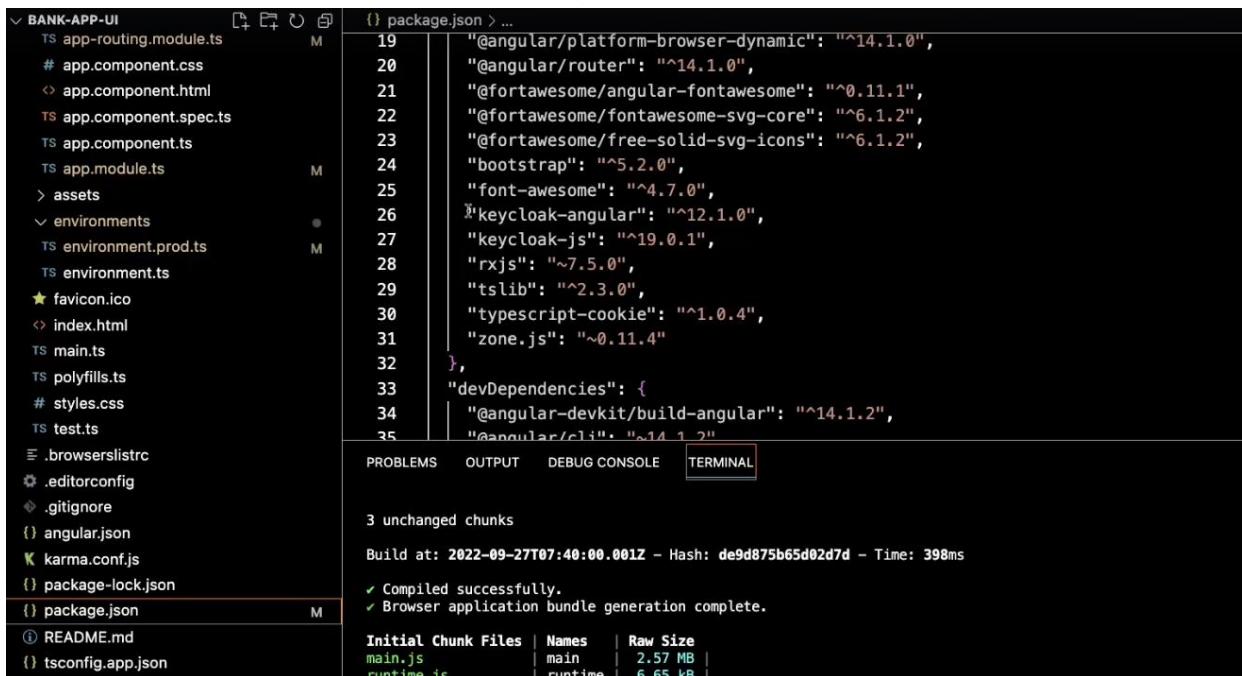
Use the code provided below as an example and implement it's functionality in your application. In this process ensure that the configuration you are providing matches that of your client as configured in Keycloak.

```
import { APP_INITIALIZER, NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { KeycloakAngularModule, KeycloakService } from 'keycloak-angular';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

function initializeKeycloak(keycloak: KeycloakService) {
  return () =>
    keycloak.init({
      config: {
        url: 'http://localhost:8080',
        realm: 'your-realm',
        clientId: 'your-client-id'
      },
      initOptions: {
        onLoad: 'check-sso',
        silentCheckSsoRedirectUri:
          window.location.origin + '/assets/silent-check-sso.html'
      }
    });
}
```

Vì vậy, họ đã đưa ra tất cả các bước này. Tôi đã làm theo tất cả các bước này và tôi cũng đã thực hiện một số thay đổi bên trong ứng dụng Easy Bank Angular UI của mình. Vì vậy, hãy để tôi cố gắng giải thích những thay đổi mà tôi đã thực hiện để bạn hiểu rõ hơn.

Thay đổi đầu tiên mà tôi đã thực hiện là tôi đã thực hiện lệnh cài đặt NPM đó. Cùng với đó, bạn có thể thấy bên trong `pack.json` của tôi, có hai mục được tạo liên quan đến Keycloak Angular và `keycloak.js`. Khi các chi tiết này được thêm vào `package.json`, bạn có thể chia sẻ ứng dụng web này với bất kỳ ai. Và cùng với đó, họ không phải cài đặt các thư viện này một cách thủ công. Bất cứ khi nào họ chạy lệnh cài đặt NPM, các thư viện nhật ký chính này cũng sẽ được tải xuống cùng với các thư viện Angular.



```

BANK-APP-UI
  TS app-routing.module.ts
  # app.component.css
  < app.component.html
  TS app.component.spec.ts
  TS app.component.ts
  TS app.module.ts
  > assets
  environments
    TS environment.prod.ts
    TS environment.ts
  ★ favicon.ico
  <> index.html
  TS main.ts
  TS polyfills.ts
  # styles.css
  TS test.ts
  .browserslistrc
  .editorconfig
  .gitignore
  angular.json
  Karma.conf.js
  package-lock.json
  package.json
  README.md
  tsconfig.app.json

  package.json > ...
  19  "@angular/platform-browser-dynamic": "^14.1.0",
  20  "@angular/router": "14.1.0",
  21  "@fortawesome/angular-fontawesome": "^0.11.1",
  22  "@fortawesome/fontawesome-svg-core": "^6.1.2",
  23  "@fortawesome/free-solid-svg-icons": "^6.1.2",
  24  "bootstrap": "^5.2.0",
  25  "font-awesome": "^4.7.0",
  26  "keycloak-angular": "^12.1.0",
  27  "keycloak-js": "^19.0.1",
  28  "rxjs": "~7.5.0",
  29  "tslib": "^2.3.0",
  30  "typescript-cookie": "^1.0.4",
  31  "zone.js": "~0.11.4"
  32  },
  33  "devDependencies": {
  34    "@angular-devkit/build-angular": "^14.1.2",
  35    "@angular/cli": "^14.1.2"
  36  }

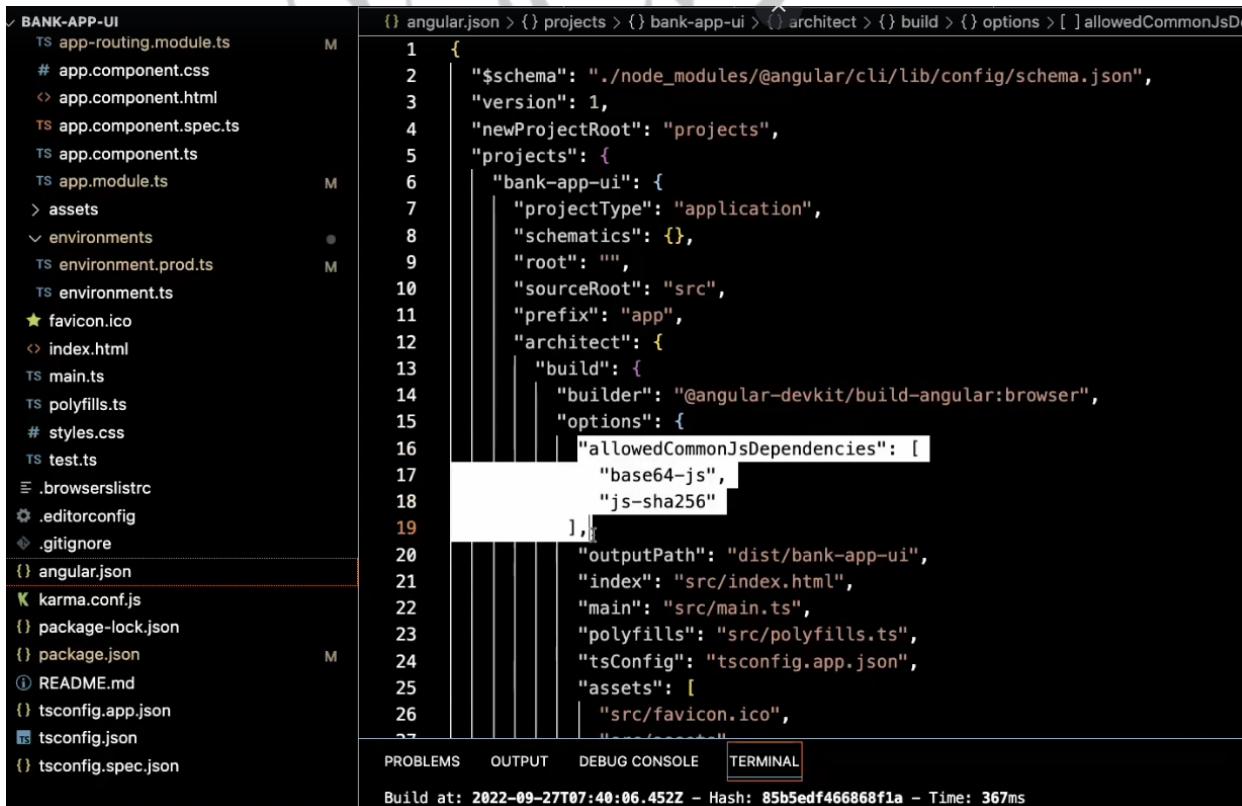
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

3 unchanged chunks
Build at: 2022-09-27T07:40:00.001Z - Hash: de9d875b65d02d7d - Time: 398ms
✓ Compiled successfully.
✓ Browser application bundle generation complete.

Initial Chunk Files | Names | Raw Size
main.js | main | 2.57 MB |
runtime.js | runtime | 6.65 KB |

```

Và thay đổi tiếp theo mà chúng ta cần quan tâm là bên trong Angular.json, tôi đã cho phép các common JS dependencies này là base 64 hyphen JS (Base64-Js) and JavaScript SHA 256 (Js-Sha256). Vì vậy, hai JavaScript dependencies này sẽ cho phép Angular UI client application của tôi để chuyển đổi code verifier của tôi thành giá trị băm SHA 256 và nó có thể áp dụng base 64 URL encoding. Vì vậy, những chi tiết này tôi cũng đã thêm vào bên trong Angular.json dưới dạng common JS dependencies.



```

BANK-APP-UI
  TS app-routing.module.ts
  # app.component.css
  < app.component.html
  TS app.component.spec.ts
  TS app.component.ts
  TS app.module.ts
  > assets
  environments
    TS environment.prod.ts
    TS environment.ts
  ★ favicon.ico
  <> index.html
  TS main.ts
  TS polyfills.ts
  # styles.css
  TS test.ts
  .browserslistrc
  .editorconfig
  .gitignore
  angular.json
  Karma.conf.js
  package-lock.json
  package.json
  README.md
  tsconfig.app.json
  tsconfig.json
  tsconfig.spec.json

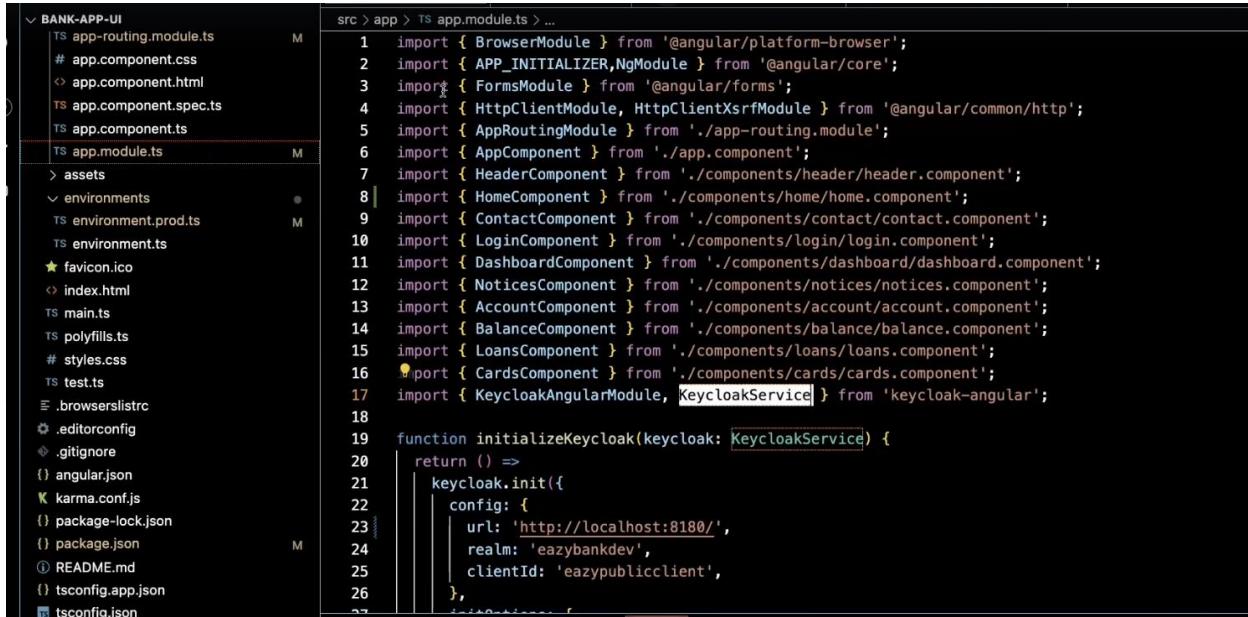
  angular.json > {} projects > {} bank-app-ui > architect > build > options > [ ] allowedCommonJsDependencies
  1  {
  2    "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  3    "version": 1,
  4    "newProjectRoot": "projects",
  5    "projects": {
  6      "bank-app-ui": {
  7        "projectType": "application",
  8        "schematics": {},
  9        "root": "",
  10       "sourceRoot": "src",
  11       "prefix": "app",
  12       "architect": {
  13         "build": {
  14           "builder": "@angular-devkit/build-angular:browser",
  15           "options": {
  16             "allowedCommonJsDependencies": [
  17               "base64-js",
  18               "js-sha256"
  19             ],
  20             "outputPath": "dist/bank-app-ui",
  21             "index": "src/index.html",
  22             "main": "src/main.ts",
  23             "polyfills": "src/polyfills.ts",
  24             "tsConfig": "tsconfig.app.json",
  25             "assets": [
  26               "src/favicon.ico",
  27               "src/assets"
  28             ]
  29           }
  30         }
  31       }
  32     }
  33   }
  34 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Build at: 2022-09-27T07:40:06.452Z - Hash: 85b5edf466868f1a - Time: 367ms

```

Sau khi thay đổi `Angular.json` và `pack.json` này. Thay đổi tiếp theo mà tôi đã thực hiện là bên trong `app.module.ts` Vì vậy, ở đây trước tiên tôi đảm bảo rằng tôi đã nhập `APP_INITIALIZER` này từ core library của Angular và đăng rằng tôi cũng đã nhập hai lớp có tên **KeycloakAngularModule**, **KeycloakService**, từ thư viện Keycloak Angular.



```

src > app > ts app.module.ts > ...
1 import { BrowserModule } from '@angular/platform-browser';
2 import { APP_INITIALIZER, NgModule } from '@angular/core';
3 import { FormsModule } from '@angular/forms';
4 import { HttpClientModule, HttpClientXsrfModule } from '@angular/common/http';
5 import { AppRoutingModule } from './app-routing.module';
6 import { AppComponent } from './app.component';
7 import { HeaderComponent } from './components/header/header.component';
8 import { HomeComponent } from './components/home/home.component';
9 import { ContactComponent } from './components/contact/contact.component';
10 import { LoginComponent } from './components/login/login.component';
11 import { DashboardComponent } from './components/dashboard/dashboard.component';
12 import { NoticesComponent } from './components/notices/notices.component';
13 import { AccountComponent } from './components/account/account.component';
14 import { BalanceComponent } from './components/balance/balance.component';
15 import { LoansComponent } from './components/loans/loans.component';
16 import { CardsComponent } from './components/cards/cards.component';
17 import { KeycloakAngularModule, KeycloakService } from 'keycloak-angular';
18
19 function initializeKeycloak(keycloak: KeycloakService) {
20   return () =>
21     keycloak.init({
22       config: {
23         url: 'http://localhost:8180',
24         realm: 'eazybankdev',
25         clientId: 'eazypublicclient',
26       },
27     });
28 }

```

Và sau đó, bên trong tệp `app.module.ts` của ứng dụng này, tôi đã tạo hàm với tên `initializeKeycloak`. Và đối với chức năng này, chúng tôi sẽ chuyển KeycloakService làm đầu vào. Và bên trong phương thức này, bạn có thể thấy tôi đang xác định rất nhiều tham số khởi tạo liên quan đến Keycloak Server của mình. Đây là các chi tiết cấu hình Keycloak auth server của tôi, giống như đây là URL của Keycloak auth server của tôi, `localhost:8180`. Và real name là gì? Vì vậy, đây là real name là Easy Bank Dev. Và client ID của client application mà chúng tôi đã đăng ký với Keycloak là gì? Vì vậy, đây là client ID mà chúng tôi đã tạo.

```

function initializeKeycloak(keycloak: KeycloakService) {
  return () =>
    keycloak.init({
      config: {
        url: 'http://localhost:8180',
        realm: 'eazybankdev',
        clientId: 'eazypublicclient',
      },
      initOptions: {
        pkceMethod: 'S256',
        redirectUri: 'http://localhost:4200/dashboard',
      },
      loadUserProfileAtStartUp: false
    });
}

```

Và sau các chi tiết cấu hình, chúng ta cũng nên chuyển các **initOptions** này như, **pkceMethod** mà client application của tôi phải tuân theo là gì? Đó là S 256. Và redirect URL là gì?

**redirectUri: 'http://localhost:4200/dashboard'**

Sau đó, chúng tôi chỉ cần xác định tham số **loadUserProfileAtStartUp** là **false**.

Và bên trong NgModule có một vài thay đổi.

```
providers: [
  {
    provide: APP_INITIALIZER,
    useFactory: initializeKeycloak,
    multi: true,
    deps: [KeycloakService],
  }
],
```

Sau những thay đổi này, chúng tôi chỉ có thể chuyển đến tệp header. ([section\\_13\bank-app-ui\src\app\components\header](#))

Nếu bạn có thể mở tệp HTML này. Có một nút đăng nhập mà user của tôi có thể bắt đầu và cũng có một tùy chọn đăng xuất.

```
<li *ngIf="user.authStatus != 'AUTH'" routerLinkActive="active"><a routerLink="/home">Home</a></li>
<li *ngIf="user.authStatus != 'AUTH'" routerLinkActive="active"><a (click)="login()">Login</a></li>
<li *ngIf="user.authStatus == 'AUTH'" routerLinkActive="active"><a (click)="logout()">Logout</a></li>
```

Trước đây, tôi chỉ định tuyến user của mình đến trang đăng nhập bằng cách tận dụng đường dẫn đăng nhập và tương tự bằng cách tận dụng đường dẫn đăng xuất. Nhưng bây giờ những gì tôi sẽ làm là nhấp vào nút đăng nhập và nút đăng xuất này, tôi sẽ kích hoạt chức năng đăng nhập. Vì vậy, chức năng đăng nhập này và chức năng đăng xuất này có sẵn bên trong header.component.js.

Những thay đổi đầu tiên là, trước tiên chúng ta cần nhập các lớp này như **KeycloakService** và **KeycloakProfile**. Sau đó, bên trong header component này, bạn có thể thấy tôi có một phương thức **ngOnInit** sẽ giúp tôi xác định xem user của tôi có đăng nhập hay không. Nếu người dùng cuối đã đăng nhập, tôi đang cố đặt **this.user.authStatus = 'AUTH'**; đồng thời tôi cũng đang tải chi tiết hồ sơ người dùng từ Keycloak và gán cho biến **userProfile** (**this.userProfile = await this.keycloak.loadUserProfile();**). Và **userProfile** này cũng như cách tôi tạo ở đây giống như Keycloak profile.

Vì vậy, từ **userProfile** đó, tôi đang cố tải tên của user để tôi có thể hiển thị bên trong dashboard của mình. Và cuối cùng, tôi cũng đang đặt toàn bộ đối tượng **user** này vào **sessionStorage** với tên **"userdetails"**.

```
window.sessionStorage.setItem("userdetails", JSON.stringify(this.user));
```

Bây giờ, nếu bạn có thể thấy ở đây bên trong chức năng đăng nhập mà chúng tôi đã kích hoạt từ header HTML, chúng tôi chỉ đang gọi phương thức login có sẵn bên trong thư viện Keycloak. Vì vậy, Keycloak này là một tên biến cho Keycloak service. Vì vậy, bên trong lớp dịch vụ Keycloak này có một chức năng gọi là login. Vì vậy, chúng tôi chỉ đang cố gắng gọi điều đó.

```
public login() {  
    this.keycloak.login();  
}
```

Bây giờ tương tự, khi người dùng nhấp vào **logout**, chúng ta cần gọi chức năng **logout** có sẵn bên trong KeycloakService. Tương tự như vậy, chúng tôi cần chuyển URL mà user của tôi phải được chuyển hướng sau khi đăng xuất thành công. Vì vậy, ở đây tôi đang cố gắng đề cập đến localhost:4200 có cùng giá trị mà chúng ta cũng cần đề cập bên trong Keycloak auth server. Vì vậy, nếu bạn thấy ở đây dưới redirect URL đăng xuất hợp lệ, tôi đã đề cập đến cùng một giá trị.

```
public logout() {  
    let redirectURI: string = "http://localhost:4200/home";  
    this.keycloak.logout(redirectURI);  
}
```

Thay đổi tiếp theo mà tôi đã thực hiện bên trong Angular UI application này là, tôi đã xóa interceptor file mà chúng tôi từng có. Vì vậy, đã từng có một interceptor file được sử dụng để giúp chúng tôi điền vào authorization header và the CSRF token. Interceptor đó tôi đã xóa hoàn toàn, vì thư viện Keycloak Angular này, bên trong nó có Interceptor riêng. Và đó là lý do tại sao chúng tôi không cần custom interceptor của riêng mình nữa.

Sau đó, thay đổi quan trọng tiếp theo mà tôi đã thực hiện là bên trong tệp liên quan đến routing. Nếu bạn có thể mở app-routing.module.ts này, bên trong routing này, bạn có thể thấy ở đây trước đây bất cứ khi nào ai đó bắt đầu login path, chúng tôi thường gọi login component chịu trách nhiệm hiển thị trang đăng nhập. Nhưng ngay bây giờ, chúng tôi sẽ không gọi login path. Nếu bạn có thể truy cập tiêu đề component.html thay vì path/login, chúng tôi đang gọi chức năng của Keycloak. Đó là lý do tại sao login page của chúng ta trước đây sẽ không bao giờ được gọi, thay vào đó, chúng ta sẽ thấy Keycloak login page sẽ được kích hoạt từ chức năng đăng nhập này.

Bây giờ, đến với thông tin routing ở đây. Tất cả các routes an toàn như dashboard, my account, my balance, my loans, my cards, tôi đang cố gắng bảo vệ chúng với sự trợ giúp của lớp này, đó là AuthKeycloakGuard. Bên trong lớp này, tôi đã viết một logic.

```
{
  path: 'myAccount', component: AccountComponent, canActivate: [AuthKeyClockGuard], data: {
    roles: ['USER']
  },
  path: 'myBalance', component: BalanceComponent, canActivate: [AuthKeyClockGuard], data: {
    roles: ['USER', 'ADMIN']
  },
  path: 'myLoans', component: LoansComponent, canActivate: [AuthKeyClockGuard], data: {
  },
  path: 'myCards', component: CardsComponent, canActivate: [AuthKeyClockGuard], data: {
    roles: ['USER']
  }
}
```

Nếu bạn có thể thấy ở đây, lớp này mở rộng KeycloakAuthGuard.

```
export class AuthKeyClockGuard extends KeycloakAuthGuard
```

Khi chúng tôi mở rộng KeycloakAuthGuard này, ở đây chúng tôi có thể thấy, chúng tôi đã khai báo các biến như `user` và `userProfile` và cho hàm constructor của lớp này, chúng tôi đang chuyển thông tin route và KeycloakService. Và sau đó, bạn có thể thấy có một phương thức sẽ được gọi bởi Angular, phương thức này rất dễ thực hiện - **isAccessAllowed method**. Vì vậy, đây là logic sẽ được thực thi để hiểu liệu tôi có cần cho phép user của mình truy cập vào secured path hay không.

```
public async isAccessAllowed(
  route: ActivatedRouteSnapshot,
  state: RouterStateSnapshot
) {
  // Force the user to log in if currently unauthenticated.
  if (!this.authenticated) {
    await this.keycloak.login({
      redirectUri: window.location.origin + state.url,
    });
  } else {
    this.userProfile = await this.keycloak.loadUserProfile();
    this.user.authStatus = 'AUTH';
    this.user.name = this.userProfile.firstName || "";
    this.user.email = this.userProfile.email || "";
    window.sessionStorage.setItem("userdetails", JSON.stringify(this.user));
  }

  // Get the roles required from the route.
  const requiredRoles = route.data["roles"];

  // Allow the user to proceed if no additional roles are required to access the route.
  if (!(requiredRoles instanceof Array) || requiredRoles.length === 0) {
    return true;
  }
}
```

```

    // Allow the user to proceed if all the required roles are present.
    return requiredRoles.some((role) => this.roles.includes(role));
}

```

Vì vậy, bên trong phương thức này, bạn có thể thấy có một biến mà chúng ta đã kiểm tra, biến **authenticated**. Biến được authenticated có sẵn bên trong KeycloakAuthGuard. Vì vậy, bất cứ khi nào xác thực thành công, thư viện của tôi sẽ **authenticated = true**. Vì vậy, bất cứ khi nào xác thực hoàn thành, khối khác này sẽ thực thi.

Mặt khác, bên trong khối if của tôi, bạn có thể thấy, người dùng sẽ được chuyển hướng đến trang login. Nếu xác thực của tôi thành công, ở đây bên trong khối khác này, trước tiên, chúng tôi sẽ cố gắng điền vào đối tượng **userProfile** và **authStatus = 'AUTH'**. Và chúng tôi đang cố tìm nạp từ **userProfile.firstName** để có thể hiển thị tên này bên trong dashboard. Và cả email nữa, chúng tôi đang tìm nạp từ **userProfile** mà chúng tôi đã nhận được từ **KeycloakAuth Server**. Vì vậy, tất cả các chi tiết người dùng này, tôi đang đặt bên trong session storage với tên **userdetails**.

Và cuối cùng, bên trong routing module class, bạn có thể thấy, đối với từng đường dẫn được bảo mật như dashboard, myAccount, tôi cũng đang chuyển dữ liệu name role. Vì vậy, bất cứ khi nào dữ liệu đó trống, điều đó có nghĩa là không cần xác thực dựa trên role nào cho secured path cụ thể đó. Đối với trang tổng quan, không có yêu cầu về role. Đến với myAccount, ai đó có role là USER, họ chỉ có thể truy cập. Và rất giống với myBalance, nếu user của tôi có role là USER hay ADMIN, anh ta sẽ có thể truy cập myBalance. Bằng cách này, chúng tôi cũng đang thực thi cơ chế truy cập dựa trên role bên trong chính Angular application của tôi. Tất nhiên, ai đó có thể thay đổi mã JavaScript này và họ có thể tự truy cập các API được bảo mật bằng cách xóa các role này. Tuy nhiên, điều này sẽ hoạt động như một mức độ bảo mật đầu tiên. Và nếu ai đó cố gắng giả mạo mã này thì chắc chắn là đằng sau hậu trường bên trong Spring Boot của chúng tôi, chúng tôi đã định cấu hình các quy tắc ủy quyền tương tự. Và điều đó không thể bị bỏ qua bởi bất kỳ ai vì đó là backend code.

Và thông tin role tương tự, tôi đang tận dụng bên trong auth.route.ts của mình, nơi tôi đã xác định các AuthKeCloakGuard này. Vì vậy, nếu bạn có thể cuộn xuống, sau khi tôi xác định rằng user của mình đã đăng nhập và tôi đã điền những thông tin này, thì bạn có thể thấy, trước tiên, tôi đang cố gắng điền các roles bắt buộc cho một route cụ thể mà user của tôi đang thử truy cập vào.

```

// Get the roles required from the route.
const requiredRoles = route.data["roles"];

// Allow the user to proceed if no additional roles are required to access
// the route.
if (!(requiredRoles instanceof Array) || requiredRoles.length === 0) {
    return true;
}

// Allow the user to proceed if all the required roles are present.
return requiredRoles.some((role) => this.roles.includes(role));

```

Từ route, chúng tôi đang gọi đối tượng dữ liệu này và bên trong đối tượng đó, chúng tôi đang cố đọc các giá trị của role. Nếu role bắt buộc này trống, điều đó có nghĩa là user của tôi không cần bất kỳ role bổ sung nào và anh ấy có thể vui vẻ truy cập vào secure path đó. Đó là lý do tại sao tôi trả lại nó đúng. Tuy nhiên, nếu có một số dữ liệu bên trong các role bắt buộc này thì bạn có thể thấy, tôi đang kiểm tra các role mà anh ấy có với Keycloak server và các role mà chúng tôi đã định cấu hình bên route. Các role sẽ đến từ thư viện Keycloak. Bất kỳ biến role nào mà bạn có thể thấy ở đây mà tôi đã đánh dấu, chúng tôi có các role mà chúng tôi đã định cấu hình bên trong KeycloakAuth server.

Chúng tôi chỉ đang kiểm tra tất cả các role mà user của tôi có bên trong Oauth Server và nếu bất kỳ role nào phù hợp với những gì chúng tôi có bên trong các requiredRoles, thì chúng tôi sẽ trả về đúng, nếu không, anh ta sẽ không thể truy cập vào role được bảo mật secured path. Vì vậy, đó là logic mà chúng tôi đã viết bên trong tất cả routing.module.ts và auth.route.ts này.

Nếu bạn có thể truy **dashboard service.ts**, ngay bây giờ, trước đây, chúng tôi đã từng gửi customer ID làm đầu vào cho backend server, nhưng hiện tại, Spring Boot application của tôi sẽ chỉ chấp nhận email làm đầu vào cho API còn lại của tôi liên quan đến khoản vay, số dư, tài khoản và thẻ.

```

@Injectable({
  providedIn: 'root'
})
export class DashboardService {

  constructor(private http:HttpClient) { }

  getAccountDetails(email: String){
    return this.http.get(environment.rooturl + AppConstants.ACCOUNT_API_URL + "?email="+email,{ observe: 'response',withCredentials: true });
  }

  getAccountTransactions(email: String){
    return this.http.get(environment.rooturl + AppConstants.BALANCE_API_URL+ "?email="+email,{ observe: 'response',withCredentials: true });
  }

  getLoansDetails(email: String){
    return this.http.get(environment.rooturl + AppConstants.LOANS_API_URL+ "?email="+email,{ observe: 'response',withCredentials: true });
  }

  getCardsDetails(email: String){
    return this.http.get(environment.rooturl + AppConstants.CARDS_API_URL+ "?email="+email,{ observe: 'response',withCredentials: true });
  }

  getNoticeDetails(){
    return this.http.get(environment.rooturl + AppConstants.NOTICES_API_URL,{ observe: 'response' });
  }

  saveMessage(contact : Contact){
    var contacts = [];
    contacts.push(contact);
    return this.http.post(environment.rooturl + AppConstants.CONTACT_API_URL,contacts,{ observe: 'response'});
  }
}

```

Đó là lý do tại sao tôi đã thay đổi tên parameter từ ID thành email và tôi đang chuyển email của user. Và bạn có thể thấy, bất cứ khi nào các phương thức này được gọi từ component, chẳng hạn như **getAccountDetails**, component sẽ gọi điều này.

Vì vậy, hãy để tôi mở component của account, nếu bạn có thể vào thư mục accounts. Ở đây, nếu tôi có thể mở accounts.component.ts, bạn có thể thấy, trước khi tôi cố gọi getAccountDetails này của dashboardService, tôi sẽ chuyển email mà tôi có bên trong đối tượng user.

```

export class AccountComponent implements OnInit {
  user = new User();
  account = new Account();
  constructor(private dashboardService: DashboardService) { }

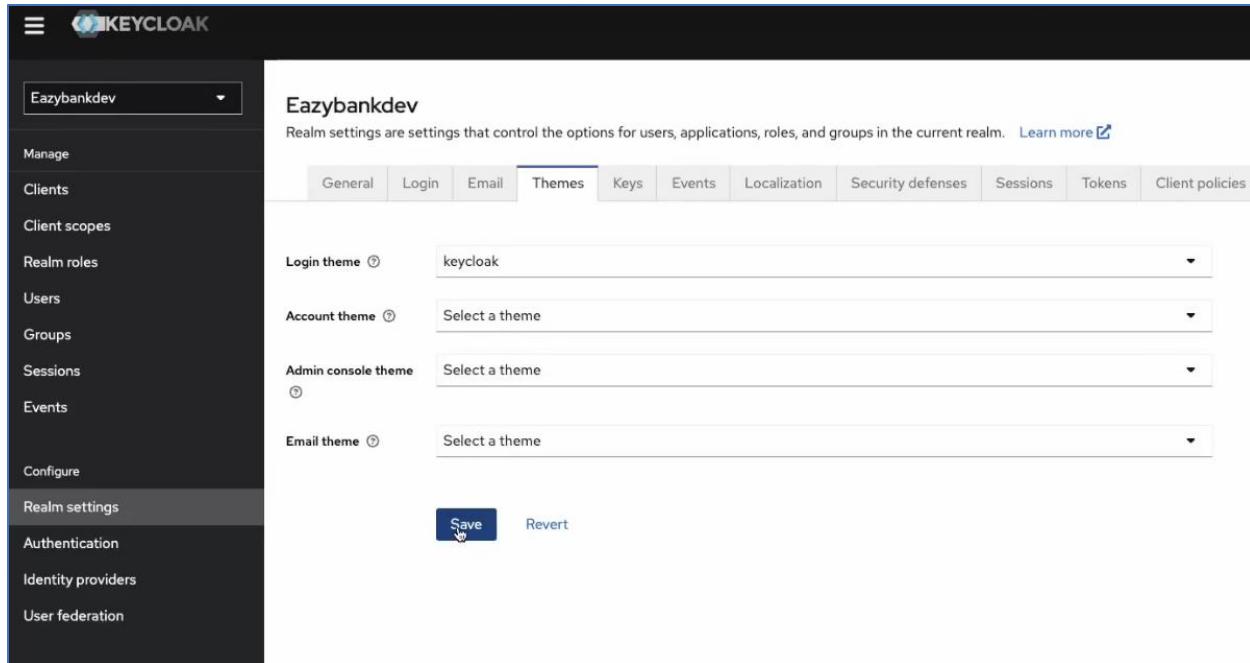
  ngOnInit(): void {
    this.user = JSON.parse(sessionStorage.getItem('userdetails')!);
    if(this.user){
      this.dashboardService.getAccountDetails(this.user.email).subscribe(
        responseData => {
          this.account = <any> responseData.body;
        });
    }
  }
}

```

Bên trong route, chúng tôi đang sử dụng lớp này, đó là AuthKeyCloackGuard. Bên trong lớp này, bất cứ khi nào xác thực thành công, chúng tôi sẽ điền giá trị email này từ thông tin **userProfile** mà chúng tôi đã nhận được từ KeyCloakAuth server.

#### 14. Important features of Keycloak

Mặc dù Keycloak là một sản phẩm mã nguồn mở và miễn phí sử dụng, nó cung cấp nhiều tính năng tích hợp sẵn mà chúng ta có thể sử dụng trong các ứng dụng web product. Thông tin đầu tiên tôi muốn chia sẻ ở đây là, khi bạn nhấp vào trang đăng nhập, bạn có thể thấy chúng ta đang nhận được trang đăng nhập được cung cấp bởi Keycloak theo mặc định. Nhưng trong các ứng dụng web product, chúng ta không muốn sử dụng loại trang đăng nhập này. Chúng ta muốn tùy chỉnh dựa trên yêu cầu của chúng ta. Chúng ta muốn cung cấp mã HTML riêng của mình, mã CSS riêng của mình để theo đúng thương hiệu của tổ chức của mình. Để đạt được điều đó, nếu chúng ta vào bảng điều khiển quản trị, bạn có thể thấy rằng dưới các thiết lập Realm, có một tab Themes.



Eazybankdev

Realm settings are settings that control the options for users, applications, roles, and groups in the current realm. [Learn more](#)

Manage

Clients

Client scopes

Realm roles

Users

Groups

Sessions

Events

Configure

Realm settings

Authentication

Identity providers

User federation

General Login Email Themes Keys Events Localization Security defenses Sessions Tokens Client policies

Login theme: keycloak

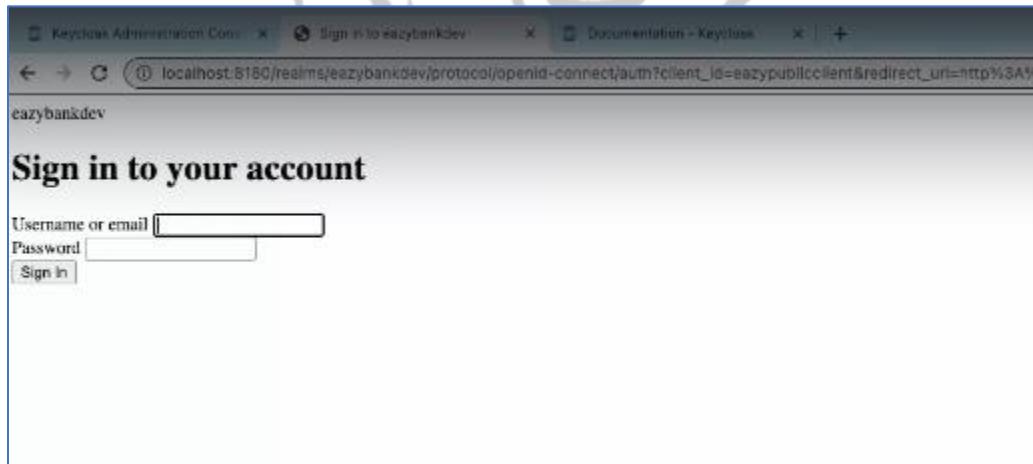
Account theme: Select a theme

Admin console theme: Select a theme

Email theme: Select a theme

Save Revert

Ở đây, bạn có thể xác định những chủ đề mà bạn muốn chọn cho login page, Account theme, administrator console, hoặc Email theme. Nhưng đối với các Login themes, hiện tại, chúng ta đang sử dụng Keycloak theo mặc định. Nếu tôi thử thay đổi điều này thành base và lưu thông tin này, sau đó tôi sẽ quay lại Angular application của mình, và tôi nhấp vào nút LOGIN này. Lần này, bạn có thể thấy tôi đang nhận được một trang HTML cơ bản mà không có bất kỳ CSS, hình ảnh hoặc nền tảng nào. Vì vậy, điều này rất đơn giản và nhảm chán.



Sign in to your account

Username or email

Password

Sign in

Để định nghĩa custom theme của chúng ta, bạn luôn có thể truy cập trang web tài liệu chính thức của Keycloak, đó là keycloak.org/documentation. Ở trang này, bạn có thể nhấp vào Server Developer documentation. Nếu bạn mở nó, trong trang này, nếu bạn nhấp vào Themes, sẽ có rất nhiều thông tin chi tiết do Keycloak cung cấp. Cách xác định custom theme của riêng bạn, cách cấu hình chủ đề đó để Keycloak administrator của tôi có thể thấy chủ đề bên trong danh sách dropdown mà anh ấy có ở đây. Vì vậy, vui lòng đọc tất cả tài liệu chính thức này nếu bạn muốn xác định custom Login theme của riêng mình.

Keycloak cũng cung cấp API REST cho tất cả các loại hành động hoặc chức năng mà bạn có thể thực hiện với sự trợ giúp của admin login. Vì vậy, nếu bạn có thể cuộn xuống bên trong trang tài liệu chính thức này, sẽ có một liên kết có tên là Administration REST API. Vì vậy, nếu bạn có thể truy cập Administration REST API, tại đây, có nhiều API REST được xác định, chẳng hạn như nếu bạn muốn lấy AccessToken, API REST bạn cần gọi là gì. Nếu bạn muốn tạo một user, nếu bạn muốn tạo một role mới. Nếu bạn đang ở trong một tình huống mà bạn muốn kích hoạt các API này từ backend hoặc từ UI application của mình, thì đó có thể là Angular hoặc có thể là React, bất kể nó có thể là gì. Vì chúng tôi có API REST nên không có ràng buộc hoặc giới hạn nào. Chúng tôi có thể sử dụng các API REST này từ bất kỳ loại ứng dụng web nào. Thậm chí chúng ta cũng có thể sử dụng chúng từ các ứng dụng di động.

Và thông tin tiếp theo mà tôi muốn chia sẻ ở đây là liệu bạn có thể chuyển đến Client scope hay không, đây là những scope hiện được xác định bên trong Keycloak auth server của tôi.

Client scopes				
Client scopes are a common set of protocol mappers and roles that are shared between multiple clients. <a href="#">Learn more</a>				
Name	Assigned type	Protocol	Display order	Description
acr	Default	OpenID Connect	–	OpenID Connect scope for add acr (authentication context class reference)
address	Optional	OpenID Connect	–	OpenID Connect built-in scope: address
email	Default	OpenID Connect	–	OpenID Connect built-in scope: email
micropattern-jwt	Optional	OpenID Connect	–	Microprofile - JWT built-in scope
offline_access	Optional	OpenID Connect	–	OpenID Connect built-in scope: offline_access
phone	Optional	OpenID Connect	–	OpenID Connect built-in scope: phone
profile	Default	OpenID Connect	–	OpenID Connect built-in scope: profile
role_list	Default	SAML	–	SAML role list
roles	Default	OpenID Connect	–	OpenID Connect scope for add user roles to the access token
web-origins	Default	OpenID Connect	–	OpenID Connect scope for add allowed web origins to the access token

Bạn có thể thấy một số scope gán cho giá trị mặc định, nghĩa là bất cứ khi nào tôi tạo client, tất cả các scope mặc định này sẽ được gán cho client mà tôi đang tạo bên trong auth server của mình. Vì vậy, tất cả những thứ này được hỗ trợ bởi giao thức OpenID. Và nếu bạn nhấp vào scope email này, bạn có thể xem loại trường nào chúng tôi sẽ nhận được bất cứ khi nào ai đó đang cố lấy scope này.

Name	Category	Type
email	Token mapper	User Property
email verified	Token mapper	User Property

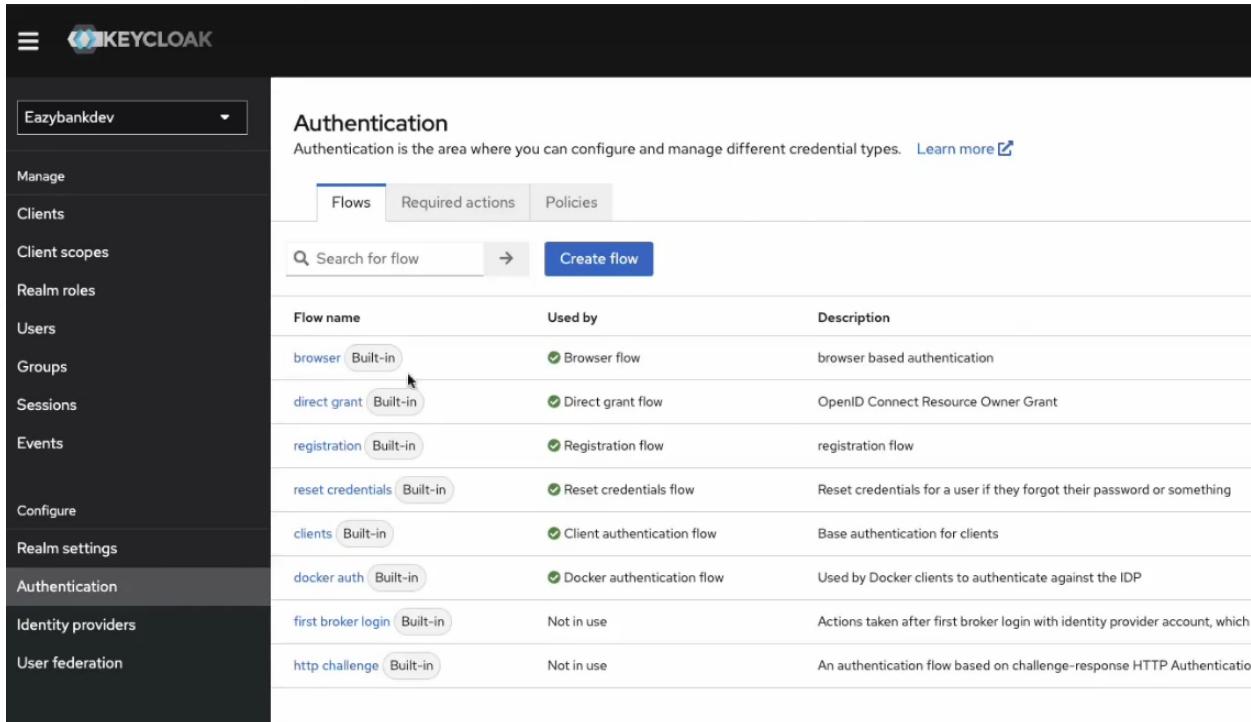
Giống như với phạm vi email, các client của tôi, họ sẽ nhận được email và các giá trị được xác minh qua email của user của tôi. Và trong các cài đặt này, chúng tôi có thể thay đổi loại này từ **Default** thành **Optional** hoặc **Non**.

Name *	email
Description	OpenID Connect built-in scope: email
Type	Default
Display on consent screen	None
Consent screen text	Optional
Include in token scope	<input checked="" type="checkbox"/> On
Display Order	

Và chúng ta cũng có thể tạo **Group**. Và với các **Group**, chúng tôi cũng có thể tạo một **Group** và chúng tôi có thể chỉ định Session người dùng hoặc nhóm người dùng cho một nhóm cụ thể. Và trong

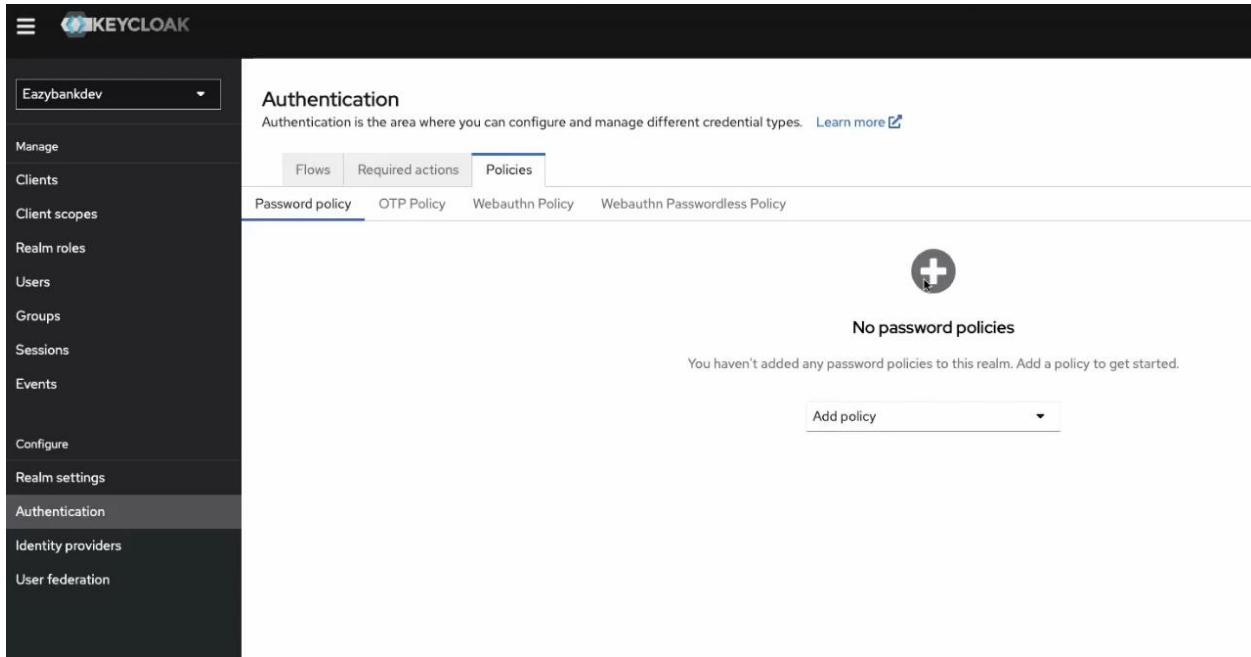
**Sessions**, chúng ta có thể xem các **Sessions** đang hoạt động là gì. Nếu cần, bạn có thể kill chúng, bạn có thể vô hiệu hóa chúng. Và với sự trợ giúp của **Events**, chúng tôi có thể định cấu hình bất kỳ **Events** người dùng hoặc **Events** quản trị viên nào mà chúng tôi muốn đăng nhập bên trong Keycloak auth server, chỉ với mục đích theo dõi. Vì vậy, tất cả đây là những chủ đề nâng cao mà quản trị viên Keycloak sẽ thực hiện.

Bây giờ, đến phần authentication ở đây.



Flow name	Used by	Description
browser	Built-in	Browser flow browser based authentication
direct grant	Built-in	Direct grant flow OpenID Connect Resource Owner Grant
registration	Built-in	Registration flow
reset credentials	Built-in	Reset credentials flow Reset credentials for a user if they forgot their password or something
clients	Built-in	Client authentication flow Base authentication for clients
docker auth	Built-in	Docker authentication flow Used by Docker clients to authenticate against the IDP
first broker login	Built-in	Not in use Actions taken after first broker login with identity provider account, which is not in use
http challenge	Built-in	Not in use An authentication flow based on challenge-response HTTP Authentication

Trong quá trình authentication, chúng tôi có thể xác định các chính sách như mật khẩu của tôi trông như thế nào, Chính sách OTP là gì? Giống như nếu cần, tôi có thể tạo chính sách mật khẩu nói rằng mật khẩu của tôi phải có độ dài tối thiểu là tám ký tự.



Eazybankdev

Manage

Clients

Client scopes

Realm roles

Users

Groups

Sessions

Events

Configure

Realm settings

Authentication

Identity providers

User federation

Authentication

Flows

Required actions

Policies

>Password policy

OTP Policy

Webauthn Policy

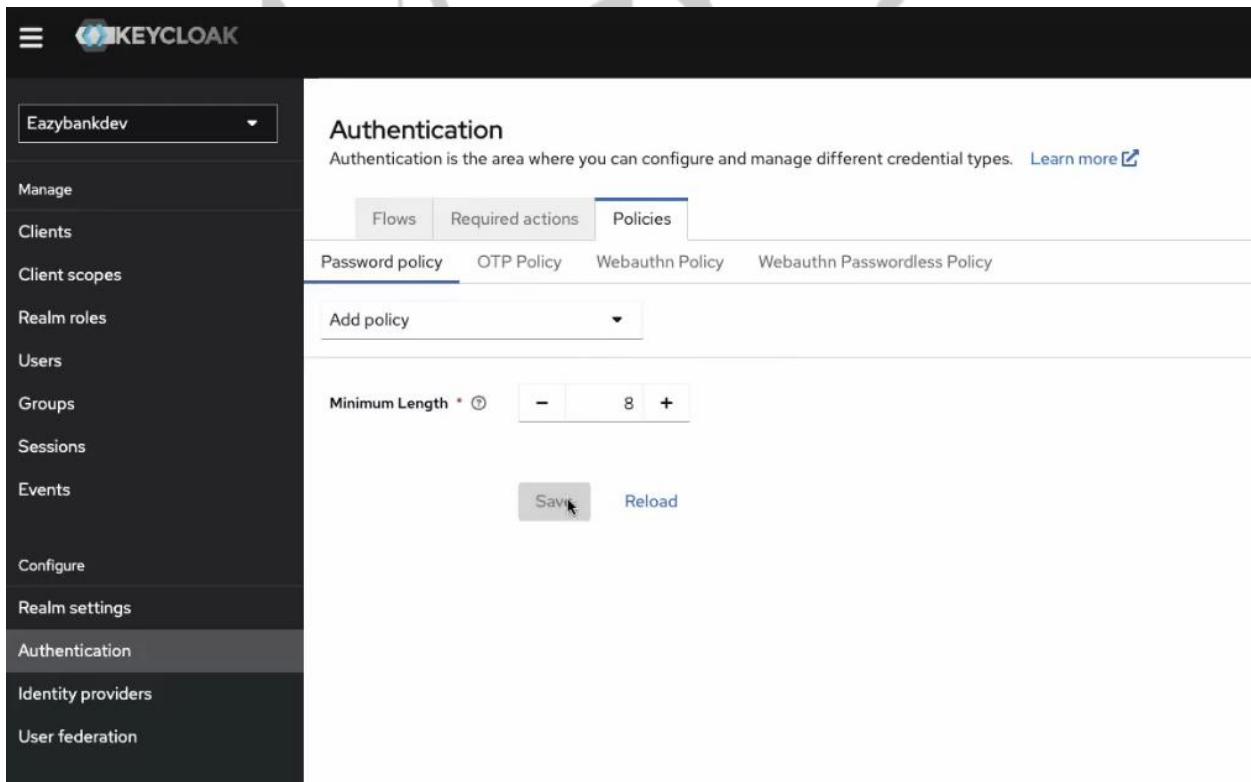
Webauthn Passwordless Policy

No password policies

You haven't added any password policies to this realm. Add a policy to get started.

Add policy

Nếu tôi lưu cái này và nếu tôi cố gắng tạo một user và user đang cung cấp một số mật khẩu ít hơn tám ký tự, thì Keycloak của tôi sẽ không chấp nhận.



Eazybankdev

Manage

Clients

Client scopes

Realm roles

Users

Groups

Sessions

Events

Configure

Realm settings

Authentication

Identity providers

User federation

Authentication

Flows

Required actions

Policies

Password policy

OTP Policy

Webauthn Policy

Webauthn Passwordless Policy

Add policy

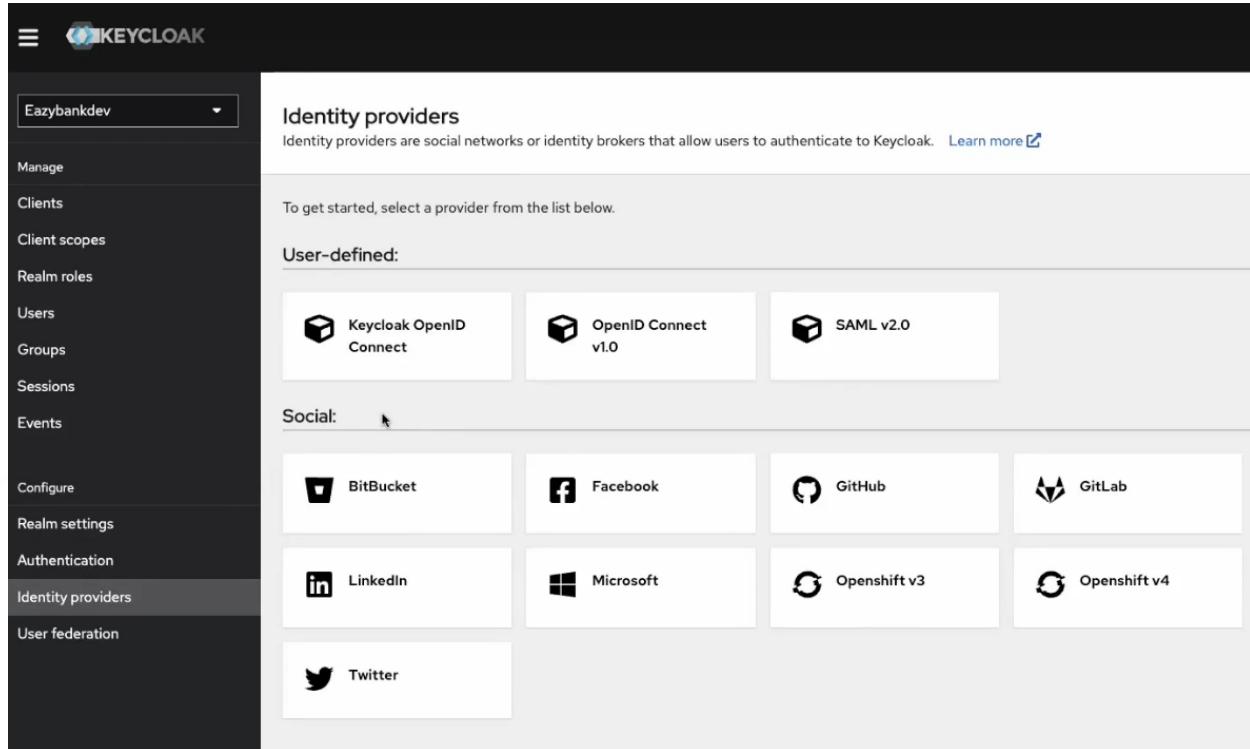
Minimum Length \* ②

- 8 +

Save Reload

Vì vậy, hãy để tôi thử tạo người dùng này. Và bên dưới Credentials, tôi đang cố đặt mật khẩu với độ dài năm ký tự, chẳng hạn như một, hai, ba, bốn, năm. Nhưng điều này sẽ không hiệu quả vì tôi có chính sách mật khẩu yêu cầu mật khẩu của tôi có ít nhất tám ký tự. Vì vậy, đó là mục đích của xác thực.

Đến với Identity, tại đây, nếu bạn muốn thêm một số thông tin social login vào ứng dụng của mình, bạn có thể tận dụng những thứ này.



Identity providers

Identity providers are social networks or identity brokers that allow users to authenticate to Keycloak. [Learn more](#)

To get started, select a provider from the list below.

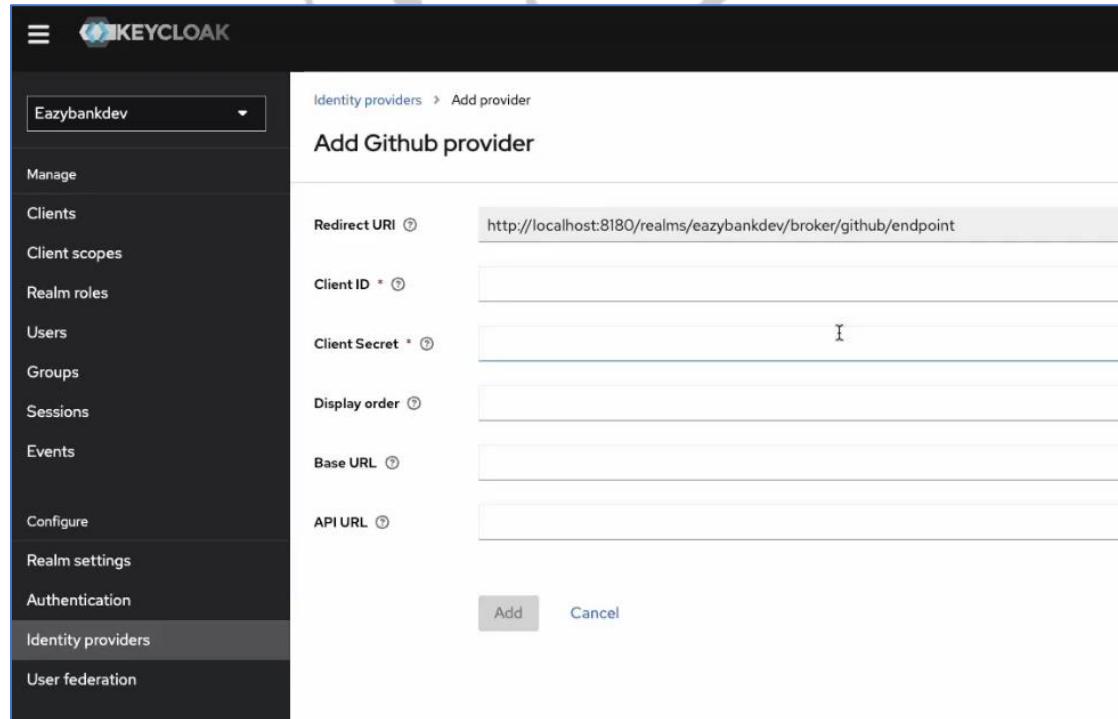
User-defined:

- Keycloak OpenID Connect
- OpenID Connect v1.0
- SAML v2.0

Social:

- BitBucket
- Facebook
- Github
- GitLab
- LinkedIn
- Microsoft
- Openshift v3
- Openshift v4
- Twitter

## 15. Social Login integration with the help of KeyCloak Server



Identity providers > Add provider

### Add Github provider

Redirect URI [?](#) http://localhost:8180/realms/eazybankdev/broker/github/endpoint

Client ID [?](#)

Client Secret [?](#)

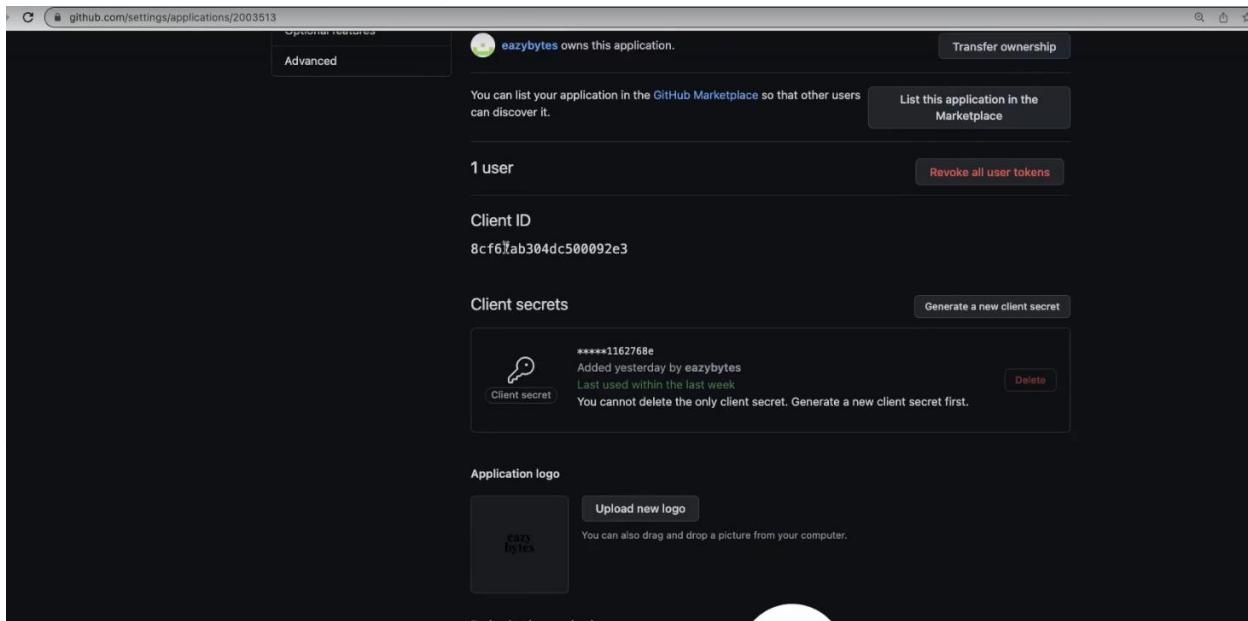
Display order [?](#)

Base URL [?](#)

API URL [?](#)

[Add](#) [Cancel](#)

Ở đây, Keycloak của tôi đang yêu cầu, "Vui lòng cung cấp client ID và client secret mà bạn đã nhận được từ GitHub." Nếu bạn có thể truy cập GitHub, trước đây chúng tôi đã tạo một ứng dụng OAuth. Vì vậy, nếu bạn có thể nhấp vào đây, bạn có thể thấy đây là client ID và client secret.



Client ID và client secret này, tôi phải cung cấp bên trong Keycloak của mình.

Bây giờ, trước khi cố gắng lưu các chi tiết này, tôi phải lấy redirect URI này từ Keycloak và đề cập đến điều tương tự bên trong back URL bên trong GitHub. Vì vậy, tôi chỉ thay thế back URL này bằng URL tôi có bên trong máy chủ xác thực Keycloak. Đồng thời, URL trang chủ hiện tại của Angular không phải là 8080, mà là 4.200. Vì vậy, tôi cũng đã cập nhật URL trang chủ đó. Vì vậy, trước tiên hãy để tôi lưu những chi tiết này vào GitHub của mình.

Badge background color

#ffffff



The hex value of the badge background color.

Application name \*

EazyBytes

Something users will recognize and trust.

Homepage URL \*

http://localhost:4200

The full URL to your application homepage.

Application description

EazyBytes Client Application

This is displayed to all users of your application.

Authorization callback URL \*

http://localhost:8180/realmseazybankdev/broker/github/endpoi

Your application's callback URL. Read our OAuth documentation for more information.

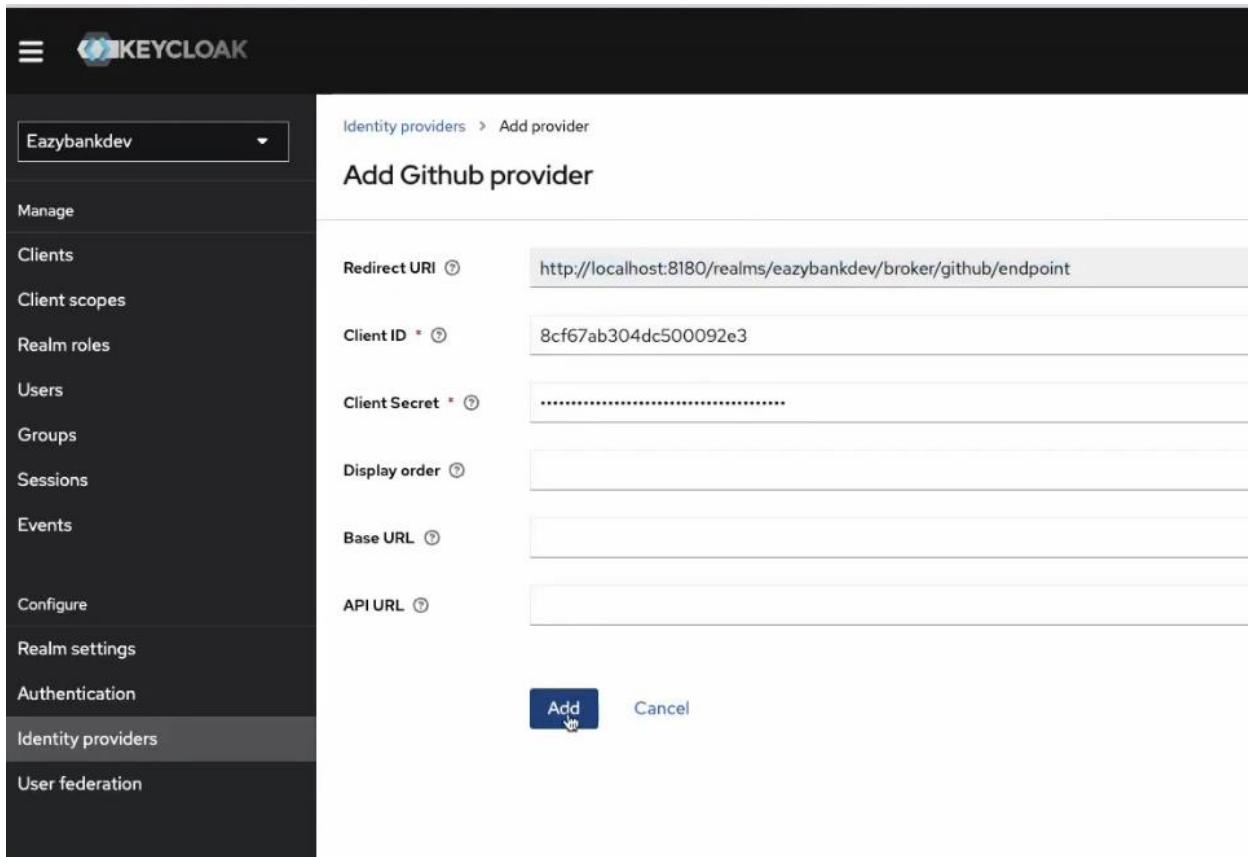
**Enable Device Flow**

Allow this OAuth App to authorize users via the Device Flow.  
Read the [Device Flow documentation](#) for more information.

---

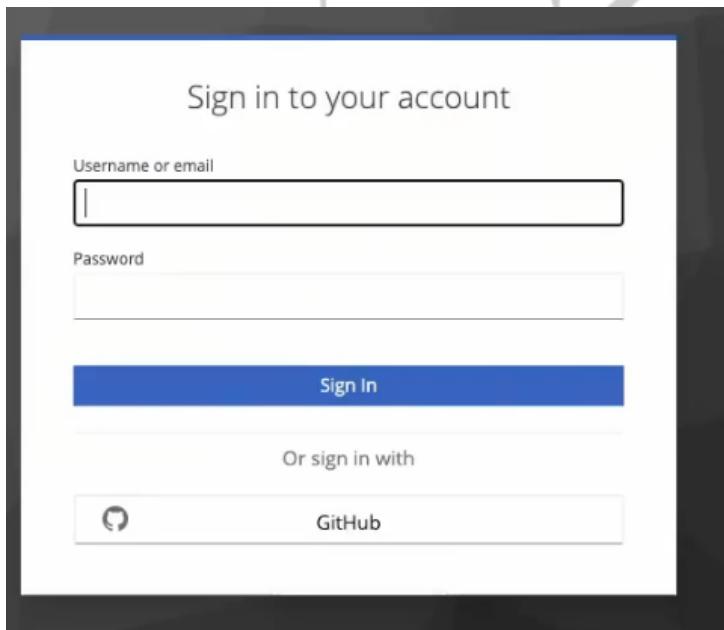
**Update application**

Sau đó, tôi có thể truy cập Keycloak của mình và nhấp vào nút “Add” này.

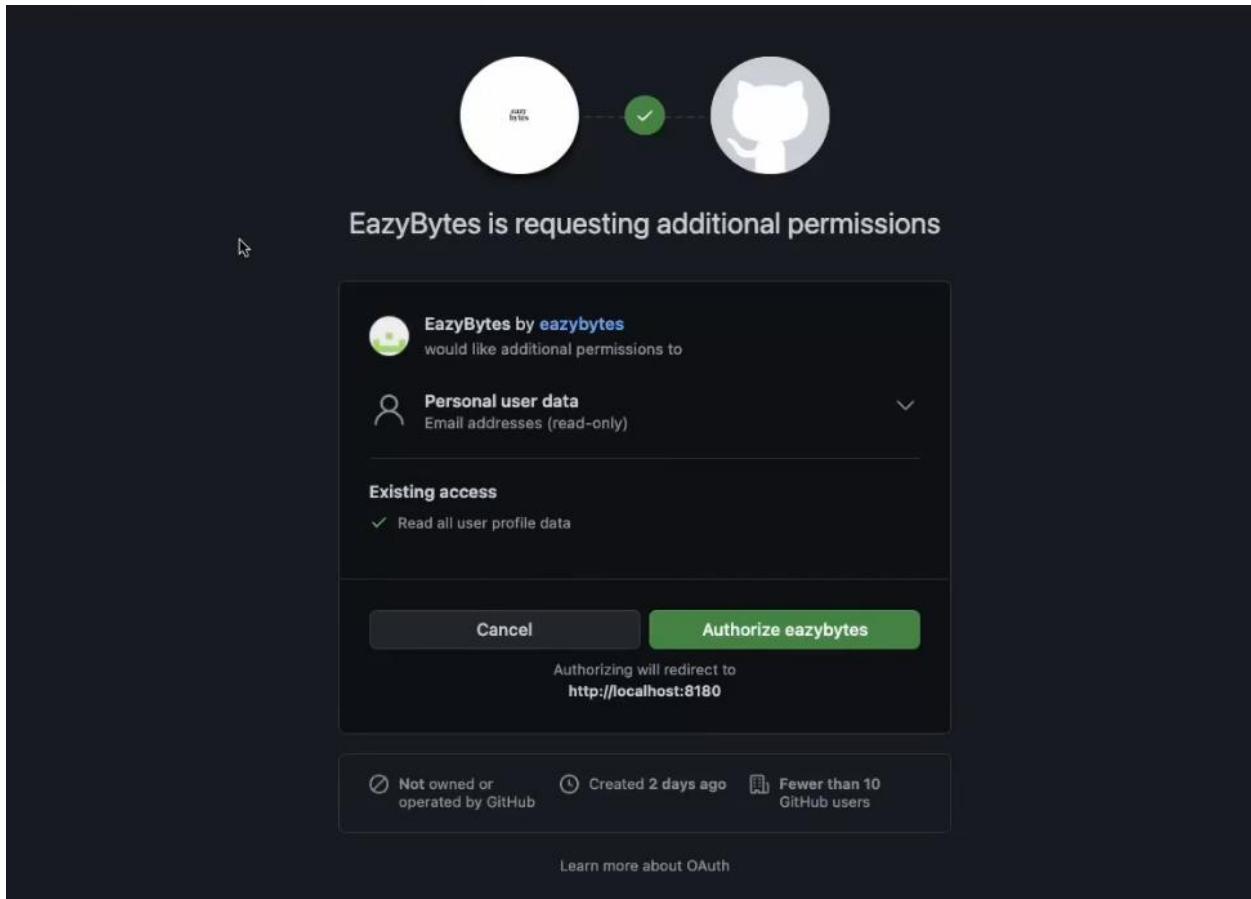


The screenshot shows the Keycloak administration interface. The left sidebar is titled 'Eazybankdev' and contains the following navigation items: Manage, Clients, Client scopes, Realm roles, Users, Groups, Sessions, Events, Configure, Realm settings, Authentication, Identity providers (which is currently selected and highlighted in grey), and User federation. The main content area is titled 'Add Github provider'. It contains several configuration fields: 'Redirect URI' with the value 'http://localhost:8180/realms/eazybankdev/broker/github/endpoint', 'Client ID' with the value '8cf67ab304dc500092e3', 'Client Secret' (redacted), 'Display order' (empty), 'Base URL' (empty), and 'API URL' (empty). At the bottom of the form are two buttons: 'Add' (highlighted with a blue background) and 'Cancel'.

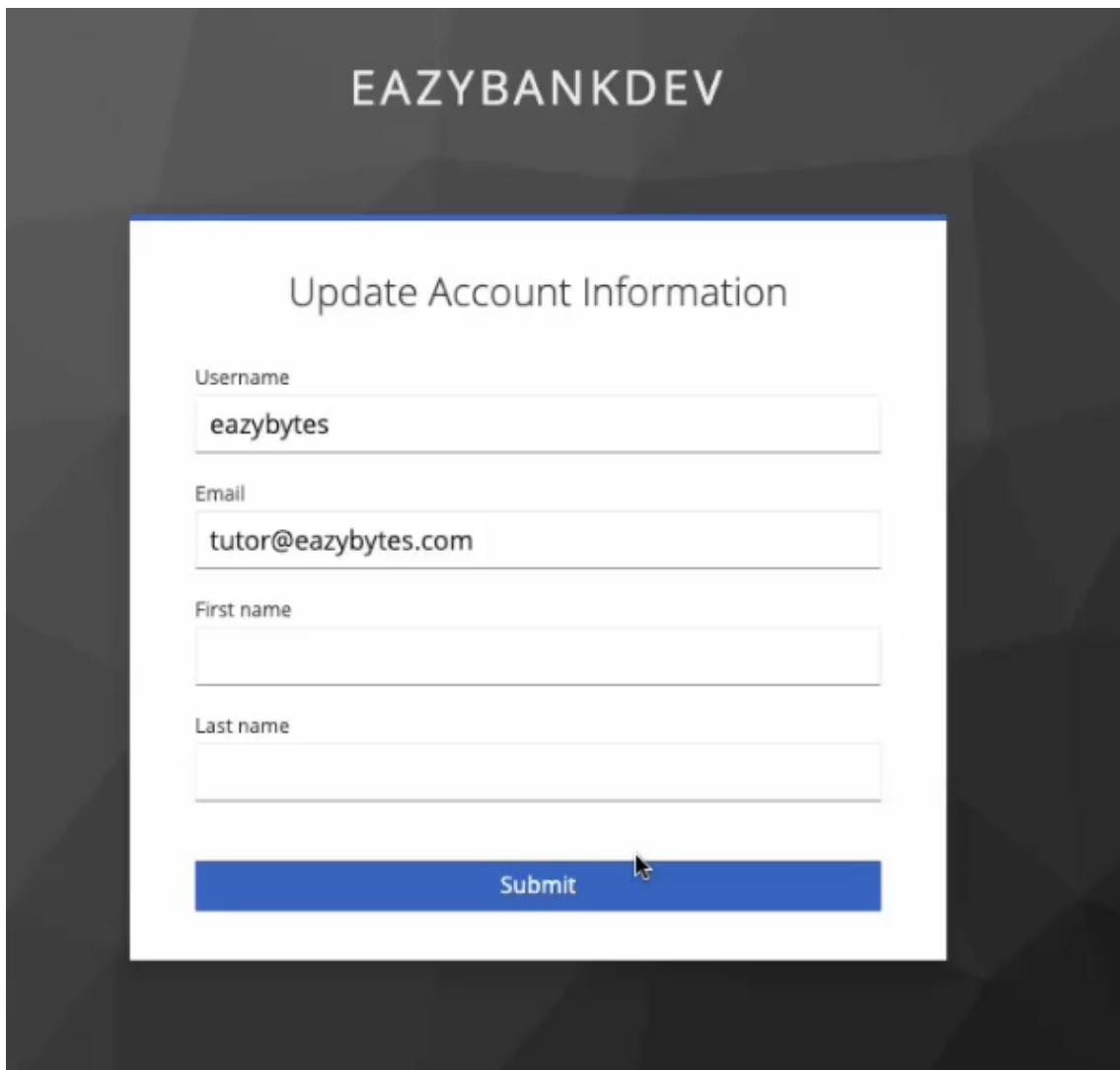
Cùng với đó, bây giờ trang đăng nhập Keycloak của tôi cũng sẽ có tùy chọn GitHub. Vì vậy, hãy để tôi đi đến Angular application của mình. Và đây, tôi đang bấm vào nút đăng nhập này. Lần này, bạn có thể thấy, cùng với tên người dùng và mật khẩu, tôi cũng có tùy chọn GitHub.



Tại đây, bạn có thể thấy, GitHub của tôi đang cố lấy sự đồng ý của tôi nói rằng "Eazy Bytes đang yêu cầu quyền bổ sung. Dữ liệu người dùng cá nhân của bạn sẽ được chia sẻ trở lại Auth Server.

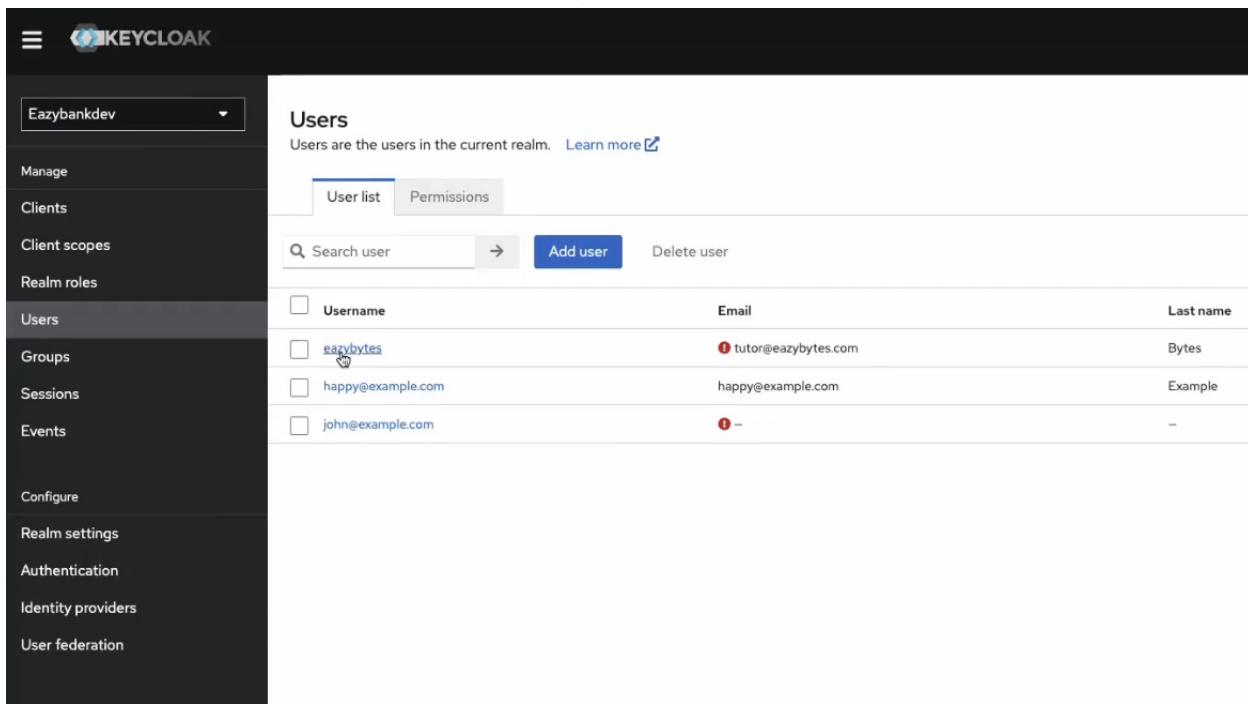


Authorization sẽ chuyển hướng nó đến localhost:8180." Vì vậy, tôi chỉ cần nhấp vào EazyByte Authorization này. Và ở đây, Keycloak auth servers của tôi, vì nó nhận được thông tin user của tôi từ GitHub lần đầu tiên, nên nó yêu cầu tôi cập nhật một số trường như tên và họ.



Nhưng nó nhận được tên người dùng và email từ GitHub. Vì vậy, tôi chỉ cần nhấp vào **“Submit”**. Điều này sẽ đưa tôi đến **dashboard**. Vì vậy, tôi có thể truy cập **dashboard**. Bây giờ, nếu tôi cố gắng nhấp vào tài khoản này, sẽ không có gì xảy ra vì GitHub của tôi chưa bao giờ cấp cho tôi các role mà Angular application của tôi đang mong đợi. Cùng với đó, tôi chỉ có thể truy cập các API không yêu cầu bất kỳ role nào.

Một thông tin quan trọng nữa mà tôi muốn chia sẻ ở đây là, nếu bạn có thể truy cập vào phần người dùng ngay bây giờ, bạn có thể thấy có một người dùng mới được tạo, Eazy Bytes với email [tutor@eazybytes.com](mailto:tutor@eazybytes.com). Vì vậy, người dùng này được tạo tự động sau khi tôi đăng nhập thành công, lần đầu tiên với sự trợ giúp của thông tin đăng nhập GitHub.



The screenshot shows the Keycloak admin interface. The left sidebar has a dropdown for realms, currently set to 'Eazybankdev'. The main content area is titled 'Users' and displays a list of users in the current realm. The table has columns for 'Username', 'Email', and 'Last name'. A user named 'eazybytes' is selected, as indicated by a cursor icon over the 'Username' cell. The table data is as follows:

Username	Email	Last name
<input type="checkbox"/> eazybytes	tutor@eazybytes.com	Bytes
<input type="checkbox"/> happy@example.com	happy@example.com	Example
<input type="checkbox"/> john@example.com	john@example.com	-

Keycloak auth server của tôi cũng sẽ lưu trữ các chi tiết hồ sơ mà nó nhận được từ GitHub và nó cũng sẽ lưu trữ access token để nếu người dùng cuối của tôi cố gắng đăng nhập lại vào ứng dụng, nó có thể tận dụng mã access token, thay vào đó buộc user của tôi phải đăng nhập lại vào GitHub nhiều lần.