JSON Web Token(JWT) And Authetication Understand

Contents

[Understanding Spring Security Architecture 2](#_Toc70767320)

[How To Keep Authenticaion (Client vs server, API) 7](#_Toc70767321)

[Session Management 9](#_Toc70767322)

[JWT (thực hành với jjwt) 10](#_Toc70767323)

[Giới thiệu 10](#_Toc70767324)

[Khởi tạo Token khi login 11](#_Toc70767325)

[Xác thực Và Phân quyền JWT 13](#_Toc70767326)

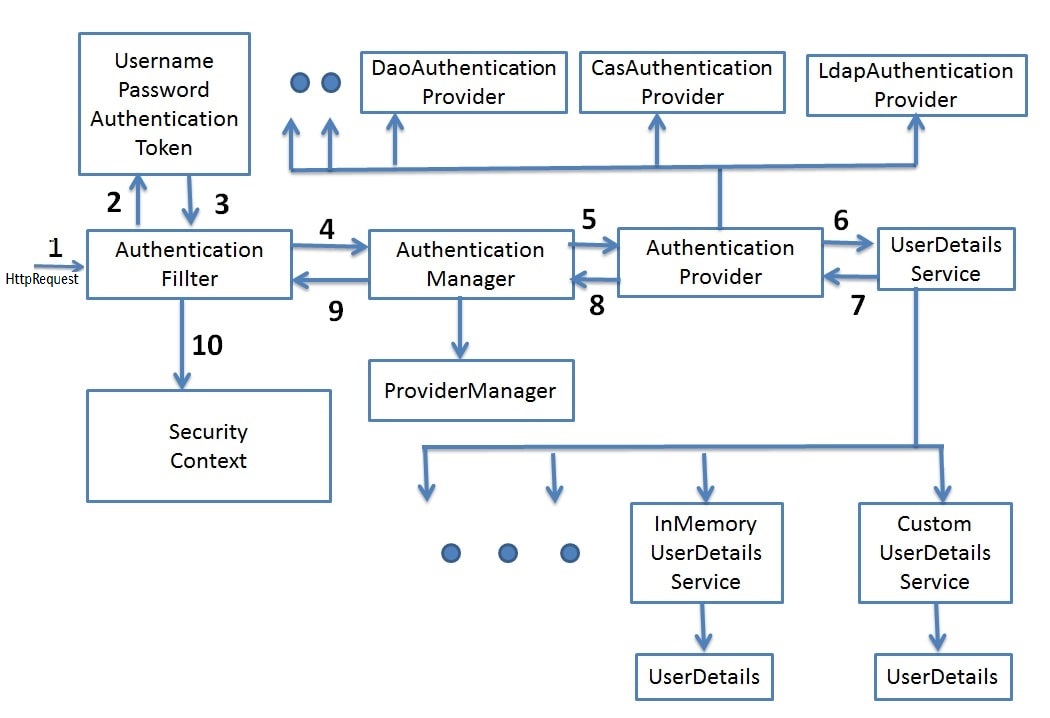
[Expired Time Token 15](#_Toc70767327)

[Test 16](#_Toc70767328)

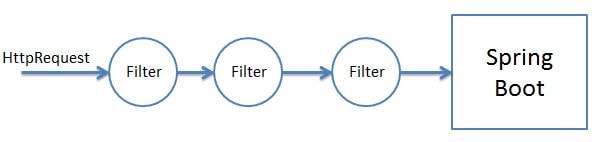
[JWT (thực hành với com.auth0) 19](#_Toc70767329)

[Reference 19](#_Toc70767330)

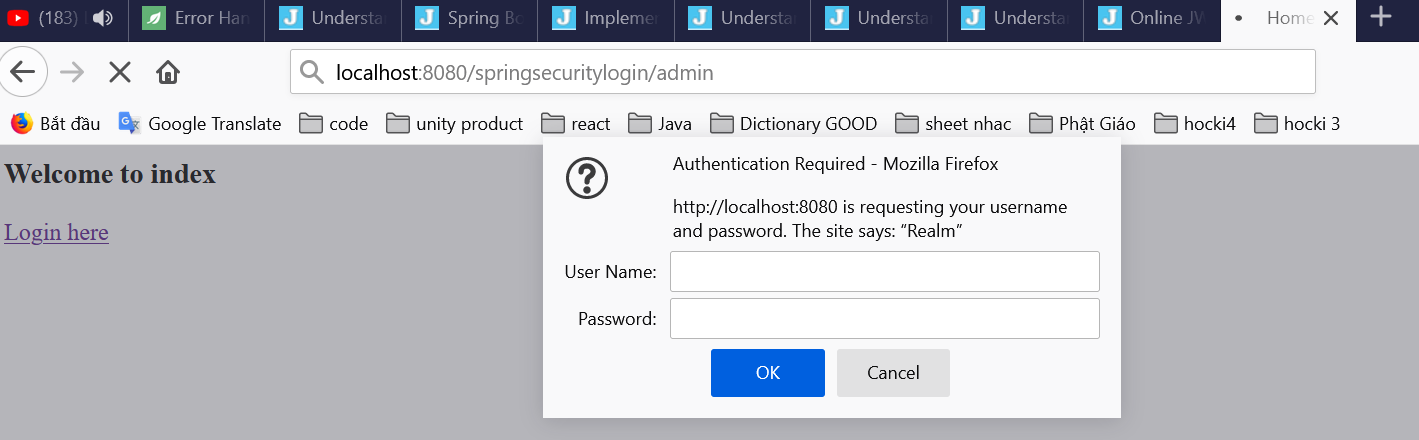
# Understanding Spring Security Architecture



**1 Filter:** Trước khi request tới được Dispatcher Servlet (Controller) nó được chặn bởi các chuỗi filter.

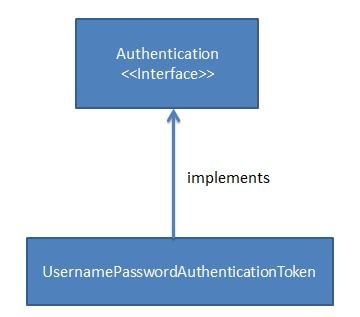


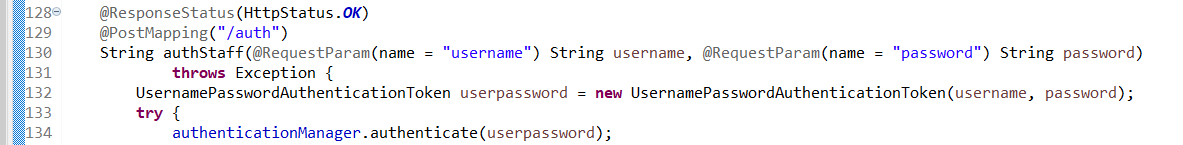
Các filter này có nhiệm vụ của spring security nên mỗi khi có bất kì request nào đi qua những filter này nó sẽ bắt để xác thực và phân quyền đựa trên kiểu request khác nhau của Authentication Filters là BasicAutheticationFilter (Login bằng http brower)



Hoặc UsernamePasswordAuthenticationFilter (này là kiểu custom nó để xác thực)

**2 Authentication Object Creattion**: Khi những request bị chặn lại để xác thực với AuthenticationFilter nó sẽ lấy cái username password từ HttpRequest và create Authentication Object đó chính là tạo ra UsernamePasswordAuthenticationToken để tiếp tục xác thực (có thể là object khác RememberMeAuthenticationToken ...).

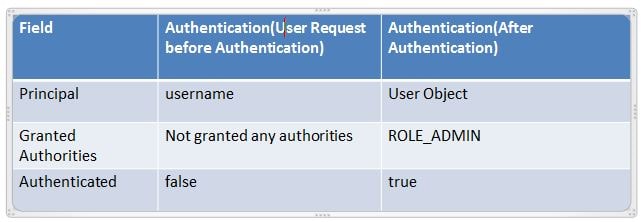




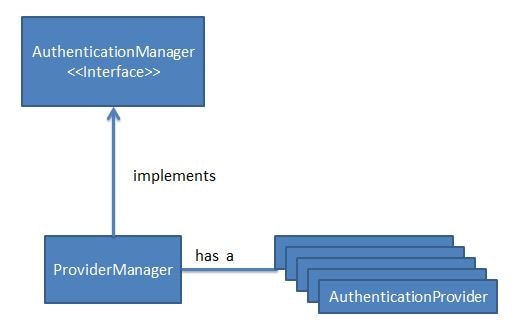
**3-4 Authentication Manager:** user object authentioncation vừa tạo (UsernamePasswordAuthenticationToken) sau đó gọi hàm authenticate() của Authentication Manager. Authentication Manager là 1 interface chỉ có duy nhất 1 hàm authenticate() được triển khai (implements) bởi các ProviderManager



Điểm quan trọng được chú ý ở đây là Authentication Manager đem 1 authentication object để đưa vào input và sau khi xác thực thành công cũng trả 1 object có kiểu authention luôn. Thông thường thì cái UserObject ở trường Principal của object authentication là instance của class UserDetails ắ.



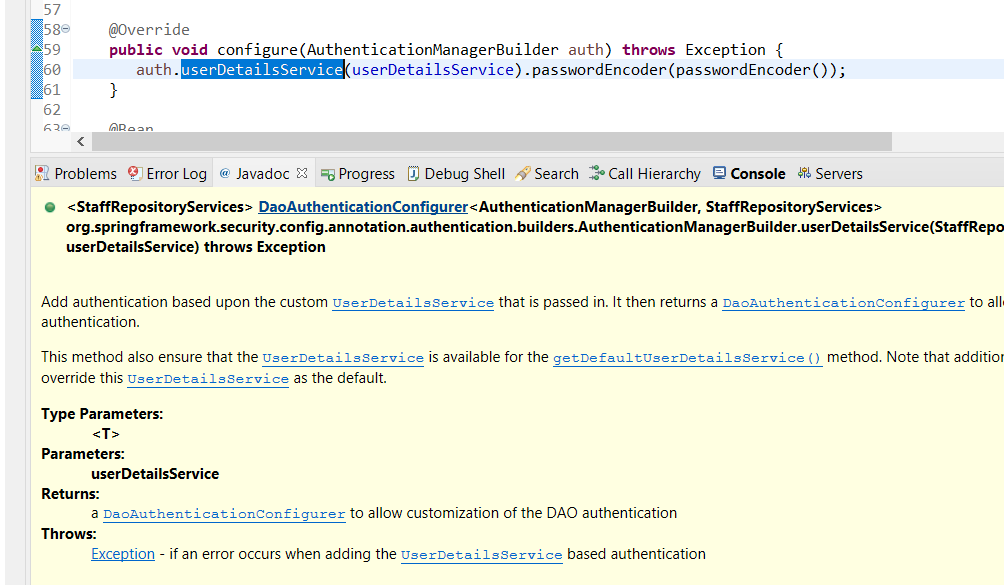
ProviderManager chứa một list AuthenticationProvider , từ hàm authenticate() của AuthenticationManager có thể gọi các hàm authenticate() được implement từ các AuthenticationProvider này . Khi xác thực thành công nó sẽ trả về đối tượng authentication cho AuthenticationManager .



**5 AuthenticationProvider** nó là 1 interface chỉ có 2 hàm là authenticate() và hàm support (chỉ định class object authenticate nào được hỗ trợ vd: chỉ có đối tượng UsernamePasswordAuthenticationToken được xác thực bằng DaoAuthenticationProvider thôi )



Nó có nhiều (triển khai) implements giống như CasAuthenticationProvider, DaoAuthenticationProvider. Dựa trên đối tượng AuthenicationProvider của nó mà đã implemenet AuthenticationManager từ trước mà được sử dụng. Mặc định nếu không chỉ định sử dụng mà ta chỉ sử dụng AuthenticationManagerBuilder gọi hàm userDetailsService Thì nó sẽ nhận DaoAuthenticationProvider thôi.



**6 Trong AuthenicationProvider** nó xác định các implement của nó qua đăng kí config từ class AbstractConfiguredSecurityBuilder

Ví dụ: Để đăng kí RememberMeAuthenticationProvider với AuthenticationProvider ta sẽ phải làm như sau thêm dòng http.rememberMe()

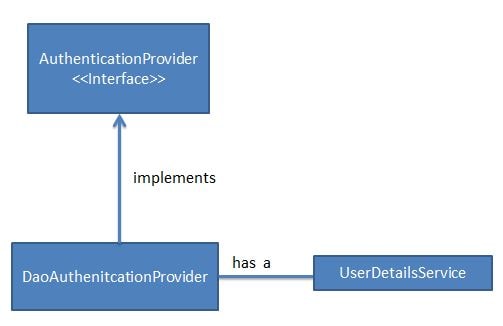
@Override protected void configure(HttpSecurity http) throws Exception {

http.rememberMe();

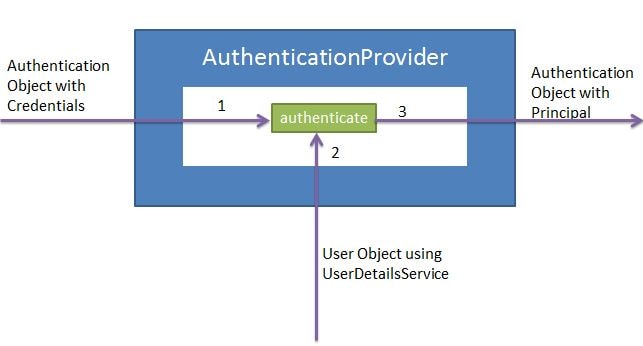
}

Trong các implement của AuthenicationProvider nó sẽ gọi tới UserDetailService để fetch data từ dababse hoặc memory do ta tự setting dựa theo username từ đối tượng authentication mà ta đã input

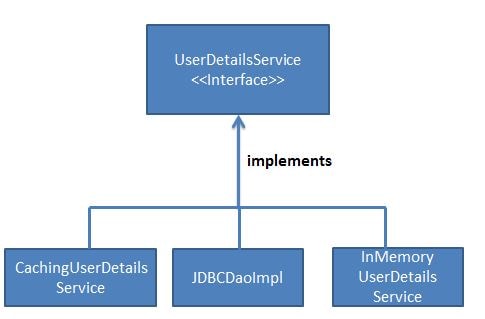
Ví dụ ảnh:



Rồi khi load được user từ Resouce(DB or Memory) UserdetailService sẽ trả về 1 object UserDetail rồi AuthenticationProvider sẽ check password hay check các thiết yếu do các Implement Authentication Provider quyết định , nhưng tất cả các implement AuthentionProvider sau khi check xác thực thành công sẽ trả về 1 đối Authentication ngược lại sẽ xuất 1 exception.



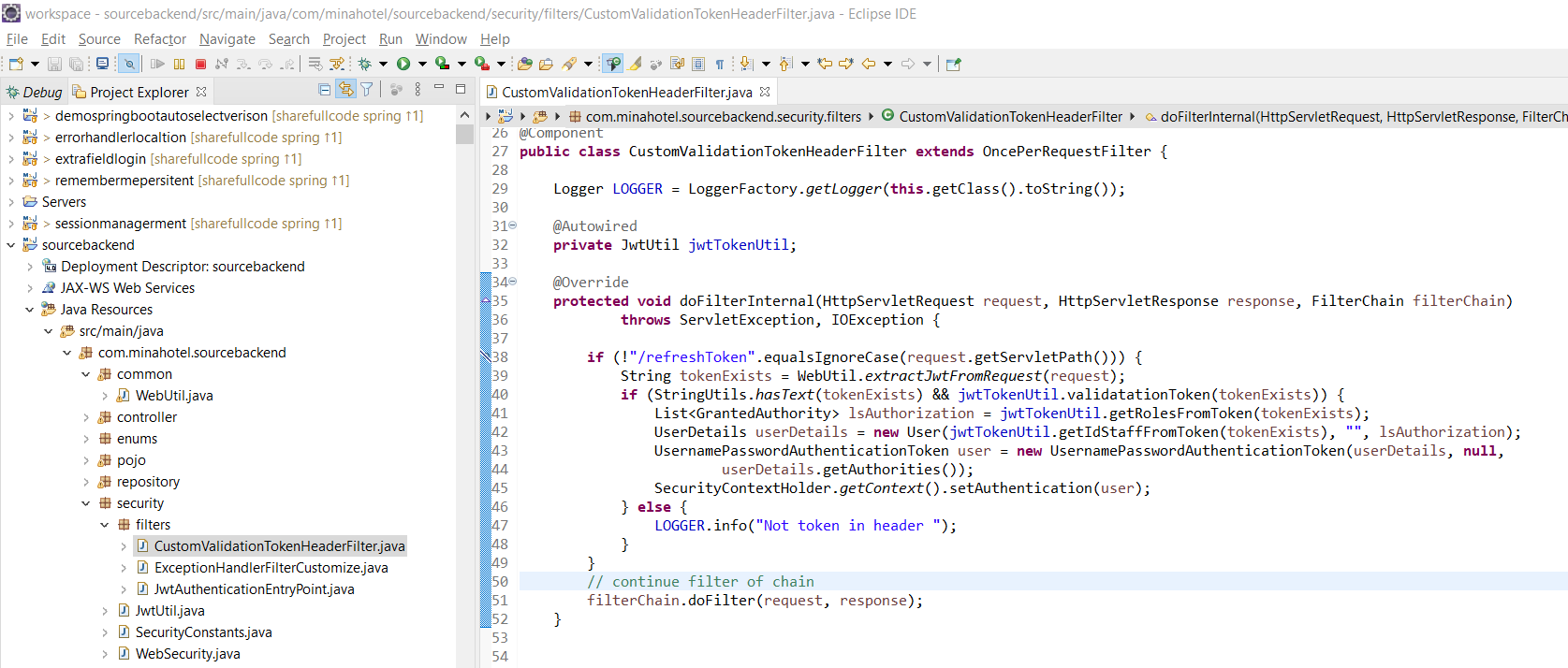
**7 UserdetailService** : là 1 interface chỉ duy nhất có 1 class loadUserByUsername . ở đó thì có nhiều implement của interface userdetailservice là CachingUserDetailsService, JDBCDaoImpl etc



Nó có nhiệm vụ thực thi hàm loadUserByUsername để fetch data để trả về Object (Userdetail) để so sánh với Object input mà đã truyền vào. Tùy từng các implement of AuthenicationProvider mà có cách xử lí khác nhưng chủ yếu cũng là check object input vs object vừa get lên từ Userdetailservice.

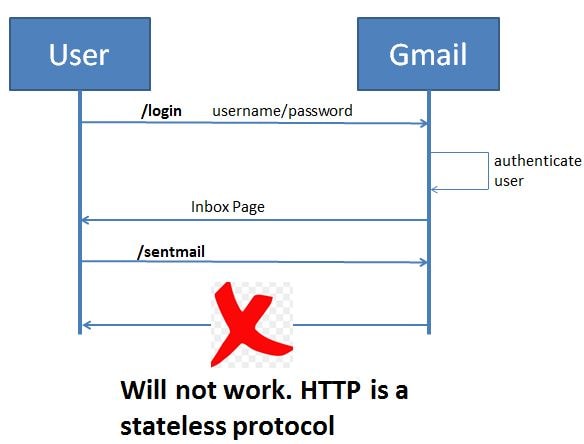
**10 SecurityContext :** là 1 class nếu trường hợp ta sử dụng authenticationProvider của hệ thống không customize thì ta không cần set object Authentication để cho hệ thống biết ta đã login thành công . nếu không thì ta cần set object đó . Ví dụ về validateToken thành công và add object vào ngược lại nếu không hệ thống sẽ check authenticaion faiule ở các filter sau :

SecurityContextHolder.*getContext*().setAuthentication(user);



# How To Keep Authenticaion (Client vs server, API)

Hình minh họa thể hiện xác thực login trong stateless protocal



Khi /login với username-password thì ta có thể truy cập và server trả về nội dung của inboxPage,

Sau đó client tiếp giử /sentmail đến server nhưng không được quyền truy cập do nó chưa được authenticaion nên server trả lỗi về client.

Lí do là do trạng thái stateless protocol, mỗi request đều xem như là 1 request mới không phân biệt được nó đã từng xác thực hay chưa.

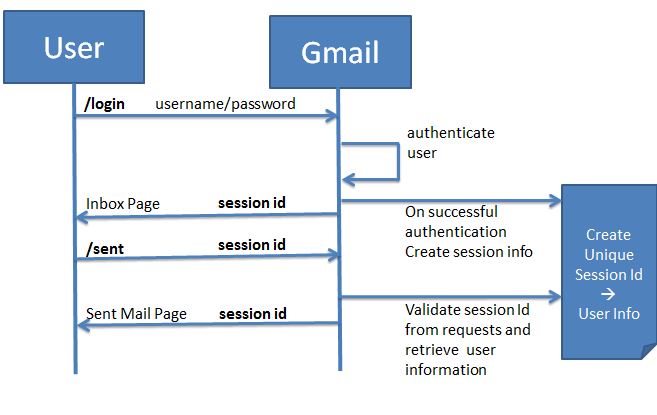
Để khác phục đều này ta có 2 hướng giả quyết

**Session Management**

**JWT**

# Session Management

Trong trường hơp này ta chỉnh policy session về ifRequire để giữ client và server có sự nhân biết với nhau bằng 1 session id . Vì thế tất cả các request con của request xác thực đều được chấp nhận và thực hiện bởi server do có session id trong mỗi header giửi lên và trả về .

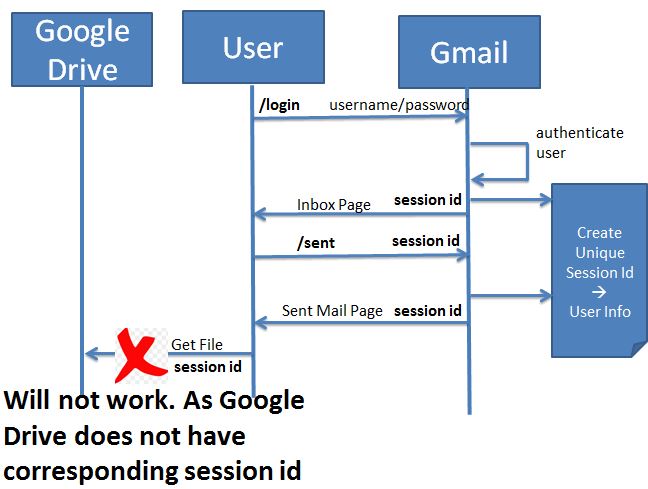


Hạn chế của Session management

Không có phù hợp với tầng kiến trúc microservice.

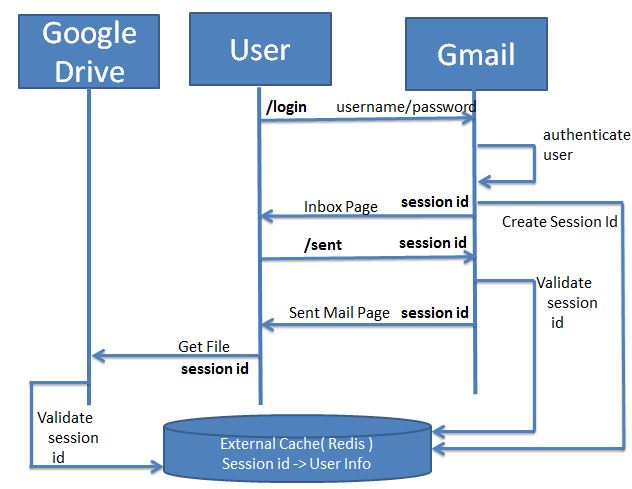
Mỗi lần nhận request server phải check các thông tin cần thiết tương ứng với thông tin từ DB

Có thể tấn công bởi crfs **Cross-site Request Forgery**.



# JWT (thực hành với jjwt)

## Giới thiệu

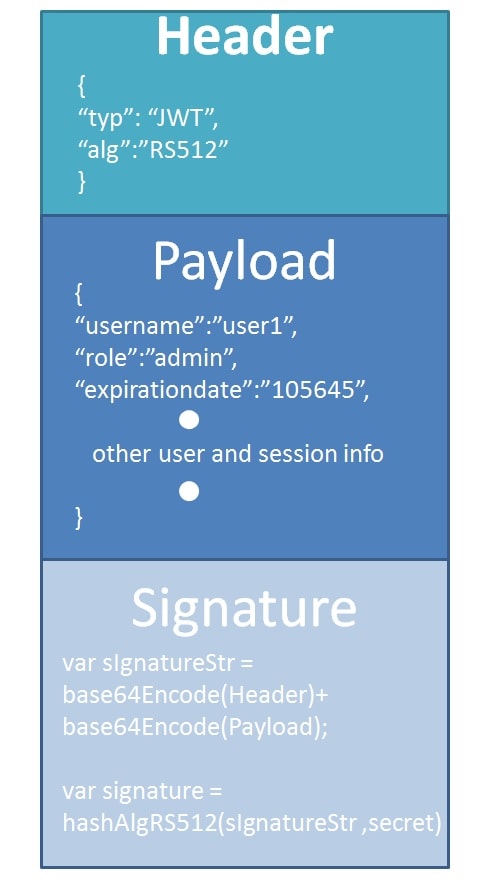
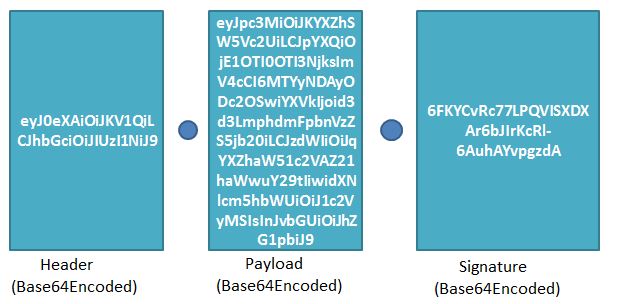


JWT tự đóng giói thông tin của nó, nên server không cần phải fetch data từ DB để check,

Dùng chữ kí bảo mật khóa private , mỗi khi bị thay đổi server sẽ nhận biết được ngay

Phù hợp với tần kiến trúc mcroservice . không cần phải chống crfs.

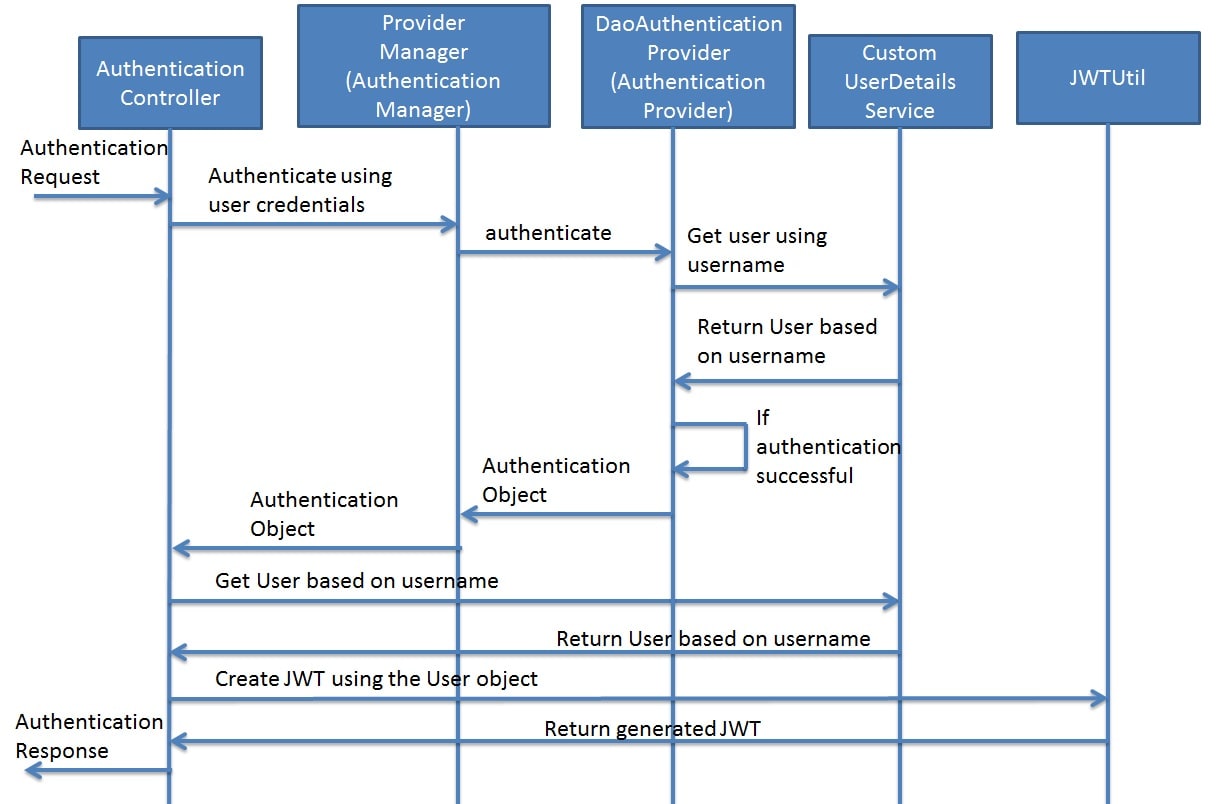
Cấu tạo gồm có 3 phần được mã hóa bởi base64

Đều quan trọng phải ghi nhớ về JWT là thông tin của PayLoad đều có thể nhìn thấy bởi bất kì ai . Nên chúng ta không đừng để password hay nội dung đặc biệt nào vào trong payload (claim). Chúng ta có thể encrypt mã hóa payload nếu thấy cần thiết cho nó thêm bảo mật. Tuy nhiên không ai có thể thay đổi kết cấu payload Nếu đều đó sảy ra server sẽ nhân ra ngay.

## Khởi tạo Token khi login

Login thành công trả về token cho client



Graphical user interface, text, application

Description automatically generated

Để tạo được 1 token ta cần phải cung cấp cho nó 2 tham số:

Secret key: Để có được chữ ký (signature) trong token ta cần phải cung cấp secret key để chỉ có server mới có thể kiểm tra và giải mã check xem token đó có đúng cấu trúc và đúng là token mà server đã cung cấp cho client hay không .

Expired Time: Token có 1 khoản thời gian sử dụng nó , được xác định bằng tham số này trong ví dụ nó được xác định bằng mili giây , 1000 ms = 1s

2 tham số này được lưu vào application.properties hoặc trong class code cũng ok

Graphical user interface, text, application, email

Description automatically generated

Để nhận giá trị từ file application.properties trong class ta sử dụng SpEL (Spring Expression Language #{expression} ${property.name}) cùng với annotation @Value làm như thế nó sẽ tự động được truyền giá trị từ @Value vào hàm khi mà ta gọi @Autowired của class

Text

Description automatically generated

Hàm tạo token cần có claim này nó là map đại diện các đối tượng có trong payload , setSubject cũng là trường trong payload thông thường nó là id của user đang chữa token, setIssuedAt giá trị timestamp của token lúc tạo ra là trường trong payload luôn , setExpiration cũng là trường trong payload thể hiện lúc mà token hết hạn không dc sử dụng hợp lệ nữa.

Graphical user interface, application

Description automatically generated

## Xác thực Và Phân quyền JWT

Sau khi tạo nhận được token từ server , client muốn truy cập vào server phải gắn token vào header mỗi request để hành động này hợp lệ , **để check mỗi request đó server cần phải tạo ra 1 filter kiểu (extends) OncePerRequestFilter** và add trước UsernamePasswordAuthenticationFilter.class.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Nếu validation token thành công sẽ tạo object UsernamePasswordAuthenticationToken set nó vào SecurityContextHolder.getContext().setAuthentication(user); để chứng minh rằng object request này đã xác thực thành công các filter sau ko cần phải check tiếp (FilterSecurityInterceptor) nếu không set nó sẽ check access denied

Text

Description automatically generated

## Expired Time Token

Trường hợp code nếu vừa có AuthenticationEntryPoint nếu thì sẽ hứng tất cả các lỗi về xác thực để customize lỗi trả về cho client . nên muốn hứng ExpiredExcetionToken riêng thì ta phải tạo ra 1 filter xử lí riêng các exception muốn hứng. Filter này cũng có kiểu là **OncePerRequestFilter và được add trước** CorsFilter.class Filter.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

## Test

Login

Graphical user interface, text, application, email

Description automatically generated

Request Get API

Graphical user interface, text, email

Description automatically generated

Request Expired Time

A picture containing text

Description automatically generated

Được hứng từ PointApp nên customize lỗi thật sự

Graphical user interface, text, application, email

Description automatically generated

Request Token invalid

Được xử lí bởi filer expired time

Graphical user interface, text, application, email

Description automatically generated

# JWT (thực hành với com.auth0)

JWTAuthenticationFilter extends UsernamePasswordAuthenticationFilter (Login trả về token) cùng với cái hàm setFilterProcessesUrl("/sourcebackend/auth"); chỉ định truy cập link này sẽ vào filter này sẽ tốt hơn cách làm ở trên là custom từ controller . này ko cần customize controller chỉ cần thêm cái link “/sourcebackend/auth” là ok tự nó hiểu cho filter login.

JWTAuthorizationFilter extends BasicAuthenticationFilter (xác thực mỗi request của client) thực hiện giống như cách làm ở trên verify token ở header nếu thành công set object authentication vào SecurityContextHolder.*getContext*().setAuthentication(authentication); để chỉ định xác thực thành công và tiếp tục đi tiếp

Text

Description automatically generated

# Reference

page: <https://www.javainuse.com/webseries/spring-security-jwt/>