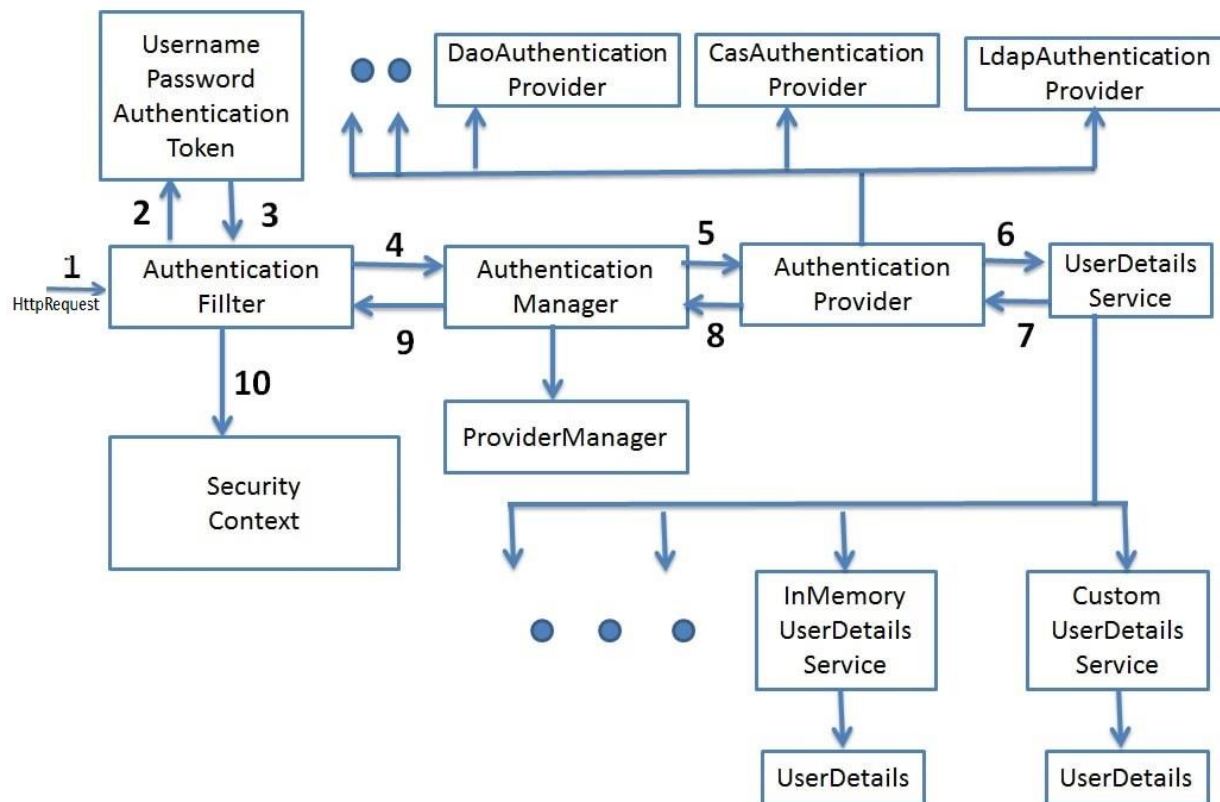


# JSON Web Token(JWT) And Authentication Understand

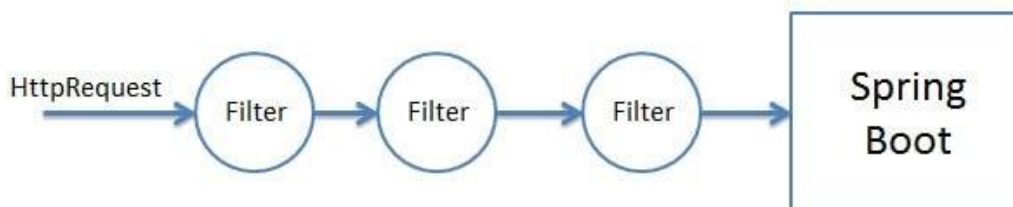
## Contents

Understanding Spring Security Architecture .....	2
How To Keep Authenticaion (Client vs server, API) .....	7
Session Management.....	9
JWT (thực hành với jjwt).....	10
Giới thiệu .....	10
Khởi tạo Token khi login .....	11
Xác thực Và Phân quyền JWT .....	13
Expired Time Token.....	15
Test.....	16
JWT (thực hành với com.auth0) .....	19
Reference .....	19

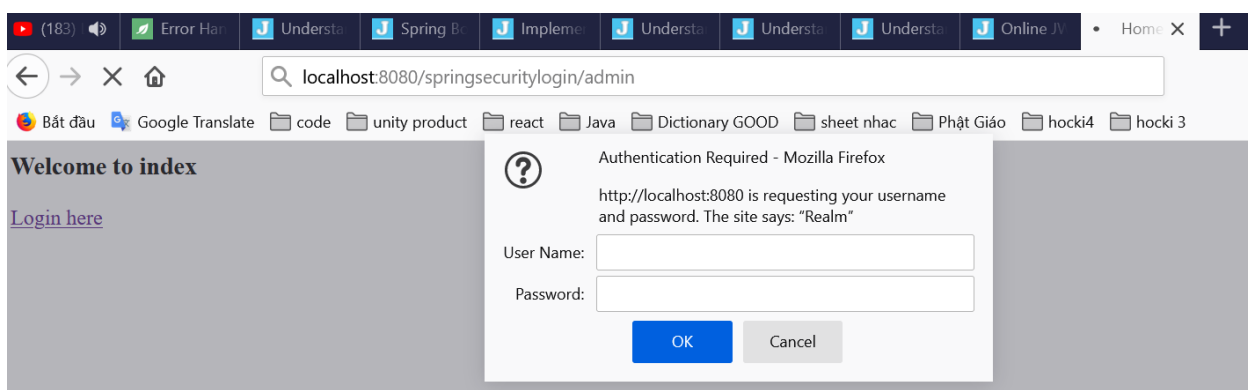
# Understanding Spring Security Architecture



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

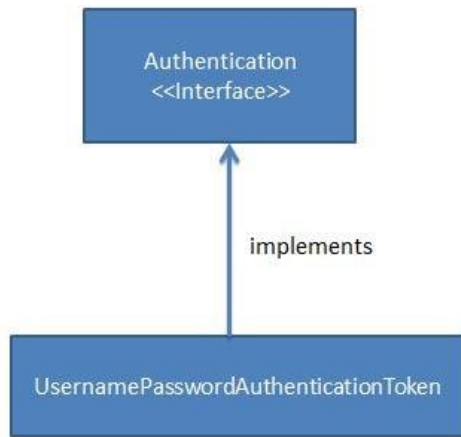


Các filter này có nhiệm vụ của spring security nên mỗi khi có bất kì request nào đi qua những filter này nó sẽ bắt để xác thực và phân quyền dựa trên kiểu request khác nhau của Authentication Filters là BasicAutheticationFilter (Login bằng http brower)



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

**2 Authentication Object Creation:** 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 ...).



```
128 @ResponseStatus(HttpStatus.OK)
129 @PostMapping("/auth")
130 String authStaff(@RequestParam(name = "username") String username, @RequestParam(name = "password") String password)
131     throws Exception {
132     UsernamePasswordAuthenticationToken userpassword = new UsernamePasswordAuthenticationToken(username, password);
133     try {
134         authenticationManager.authenticate(userpassword);
```

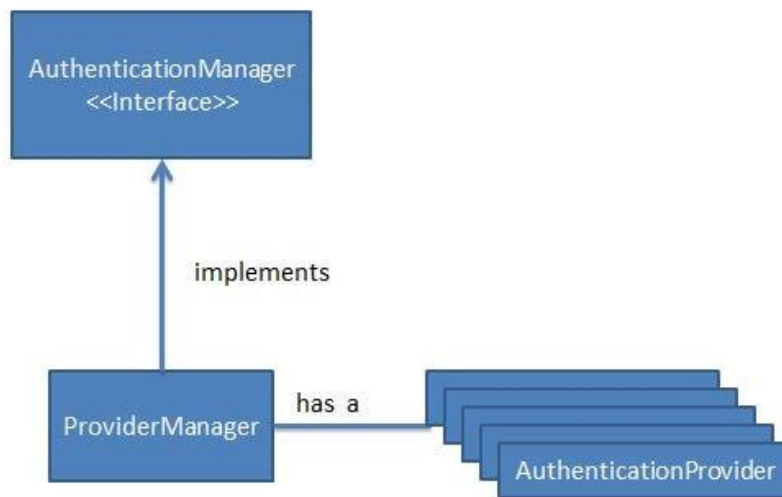
**3-4 Authentication Manager:** user object authentication vừa tạo (UsernamePasswordAuthenticationToken) sau đó gọi hàm authenticate() của Authentication Manager. Authentication Manager là 1 interface chỉ có duy nhất 1 hàm authenticate() được triển khai (implements) bởi các ProviderManager



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 authentication luôn. Thông thường thì cái UserObject ở trường Principal của object authentication là instance của class UserDetails ă.

Field	Authentication(User Request before Authentication)	Authentication(After Authentication)
Principal	username	User Object
Granted Authorities	Not granted any authorities	ROLE_ADMIN
Authenticated	false	true

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 AuthenticationProvider của nó mà đã implement 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.

```

57
58 @Override
59 public void configure(AuthenticationManagerBuilder auth) throws Exception {
60     auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
61 }
62
63 @Bean

```

Problems Error Log Javadoc Progress Debug Shell Search Call Hierarchy Console Servers

• <StaffRepositoryServices> **DaoAuthenticationConfigurer** <AuthenticationManagerBuilder, StaffRepositoryServices>  
 org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder.userDetailsService(StaffRepository userDetailsService) throws Exception

Add authentication based upon the custom [UserDetailsService](#) that is passed in. It then returns a [DaoAuthenticationConfigurer](#) to all authentication.

This method also ensure that the [UserDetailsService](#) is available for the [getDefaultUserDetailsService\(\)](#) method. Note that addition override this [UserDetailsService](#) as the default.

**Type Parameters:**  
 <T>

**Parameters:**  
 userDetailsService

**Returns:**  
 a [DaoAuthenticationConfigurer](#) to allow customization of the DAO authentication

**Throws:**  
[Exception](#) - if an error occurs when adding the [UserDetailsService](#) based authentication

**6 Trong AuthenticationProvider** 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()`

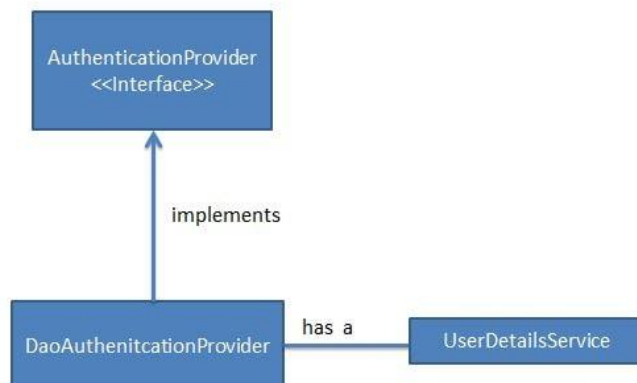
```

@Override protected void configure(HttpSecurity http) throws Exception {
    http.rememberMe();
}

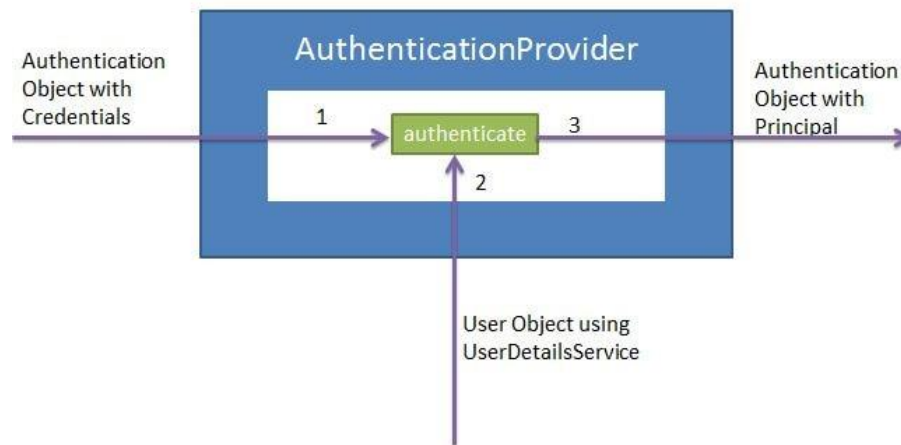
```

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

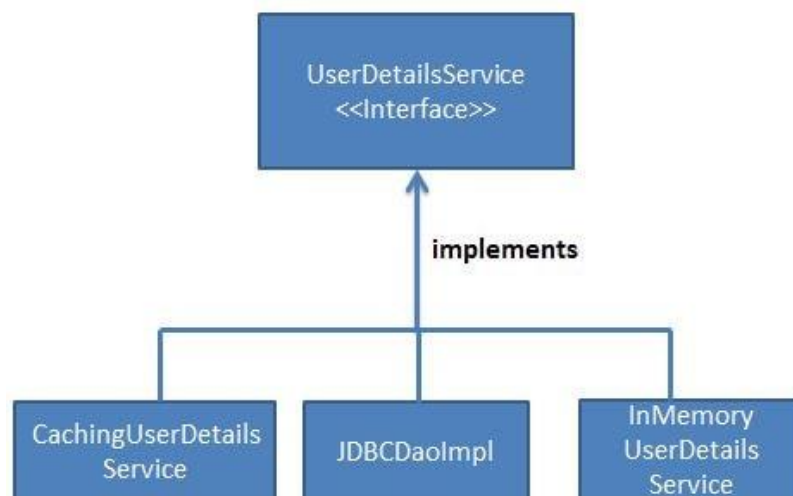
Ví dụ ảnh:



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



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

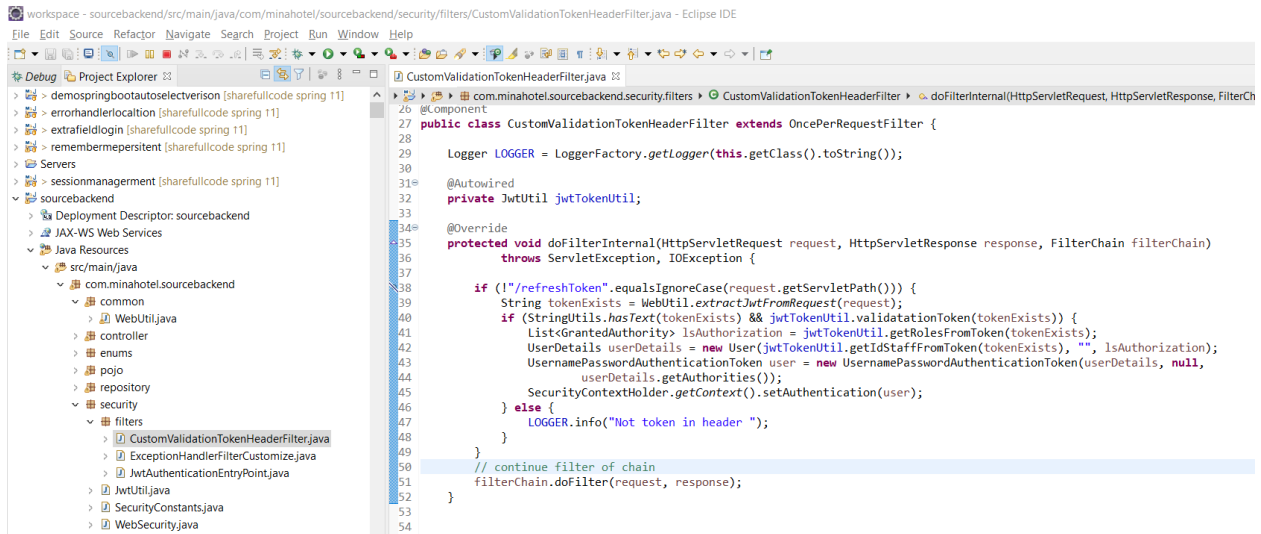


Nó có nhiệm vụ thực thi hàm loadUserByUsername để fetch data để trả về Object (Userdetail) để so sánh với Object input mà đã truyền vào. Tùy từng các implement of AuthenticationProvider mà có cách xử lý khác nhưng chủ yếu cũng là check object input vs object vừa get lên từ UserDetailsService.

**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

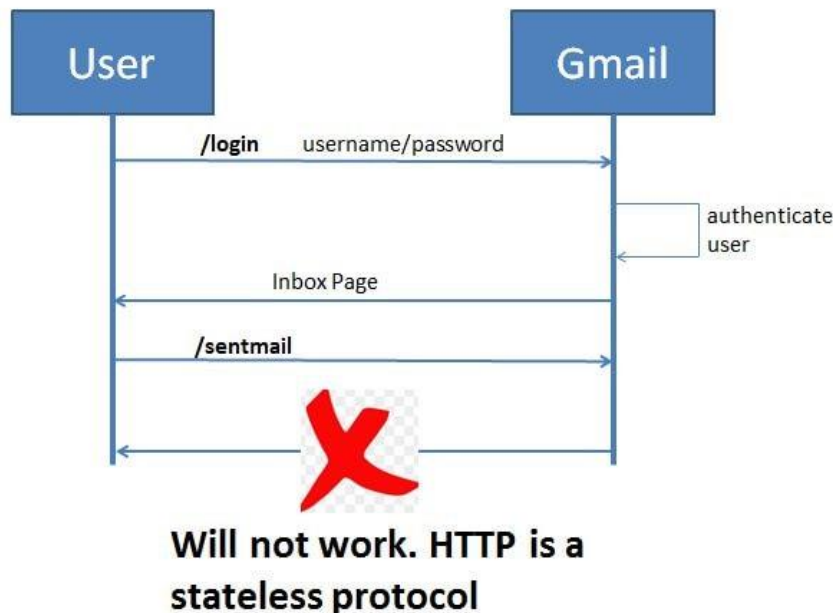
validateToken thành công và add object vào ngược lại nếu không hệ thống sẽ check authentication failure ở các filter sau :

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



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

Hình minh họa thể hiện xác thực login trong stateless protocol



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

Sau đó client tiếp gửi /sentmail đến server nhưng không được quyền truy cập do nó chưa được authentication 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 khắc phục điều này ta có 2 hướng giải quyết

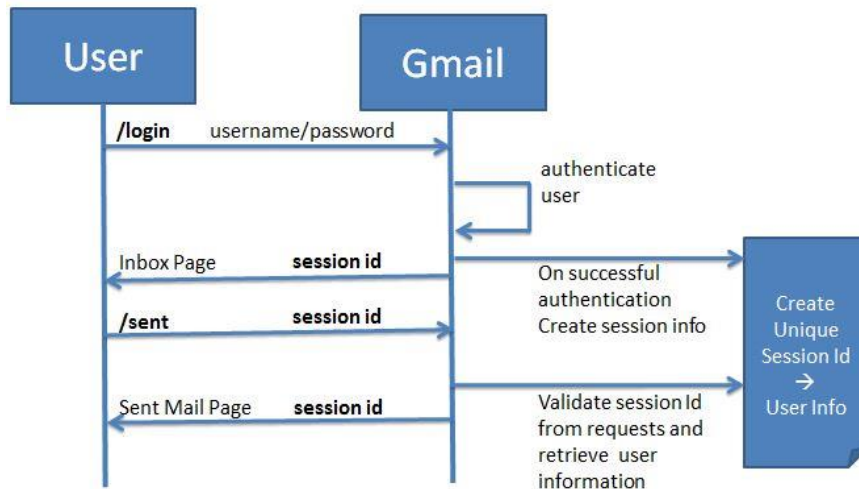
## **Session Management**

### **JWT**



# Session Management

Trong trường hợp này ta chỉnh policy session về ifRequire để giữ client và server có sự nhận 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 gửi lên và trả về .

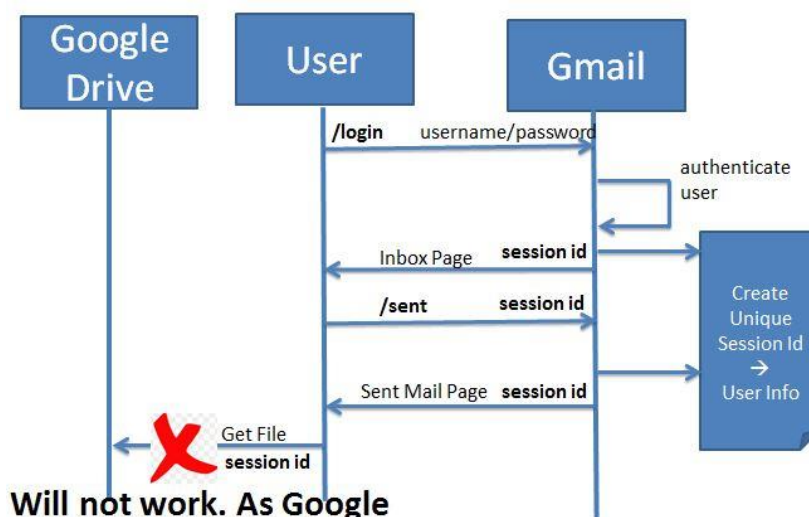


Hạn chế của Session management

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

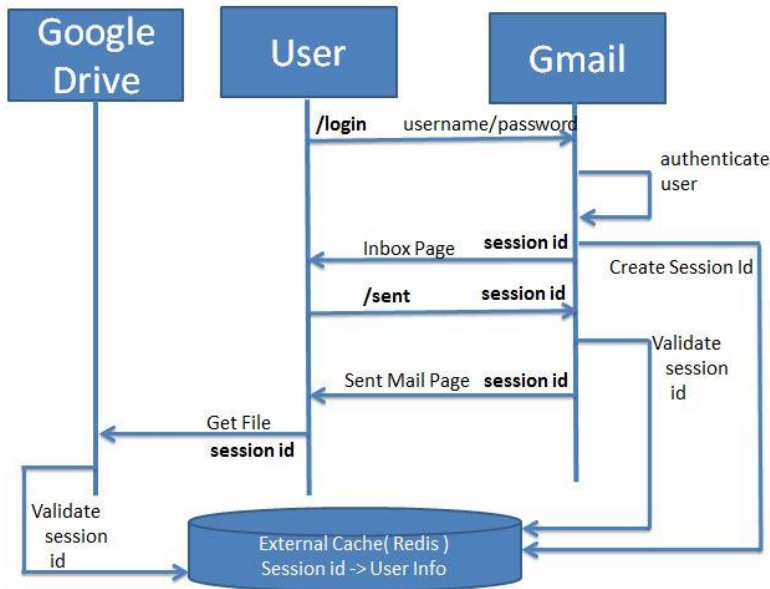
Có thể tấn công bởi crfs **Cross-site Request Forgery**.



Will not work. As Google Drive does not have corresponding session id

# JWT (thực hành với jjwt)

## Giới thiệu

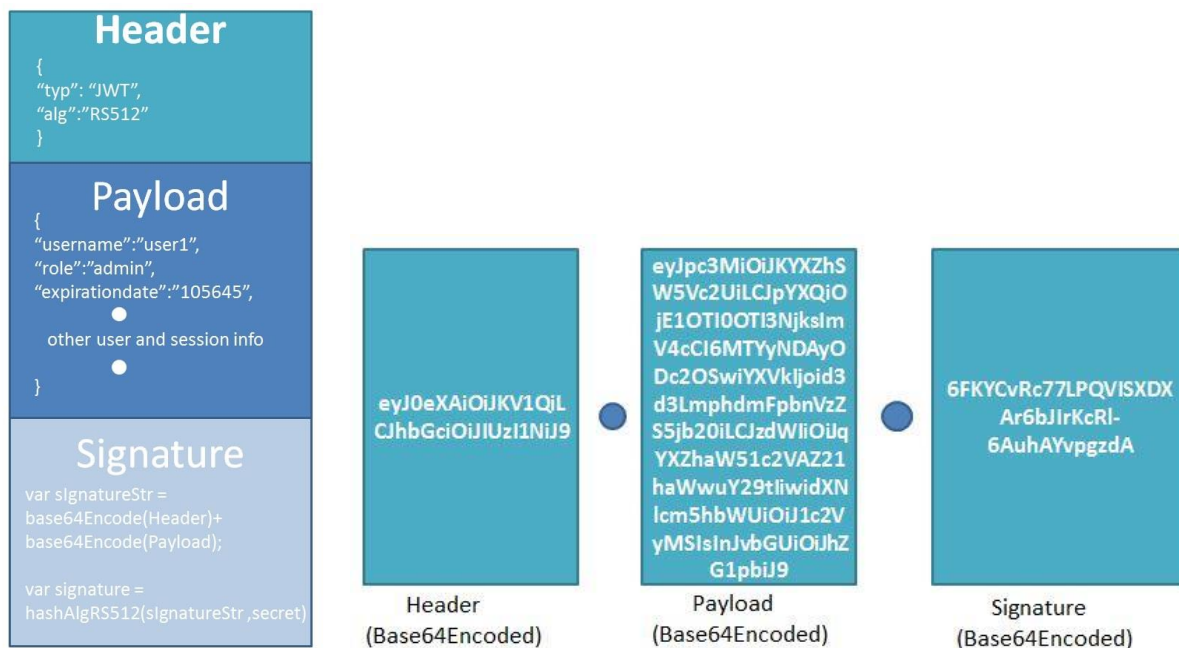


JWT tự đóng gói thông tin của nó, nên server không cần phải fetch data từ DB để check,

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.

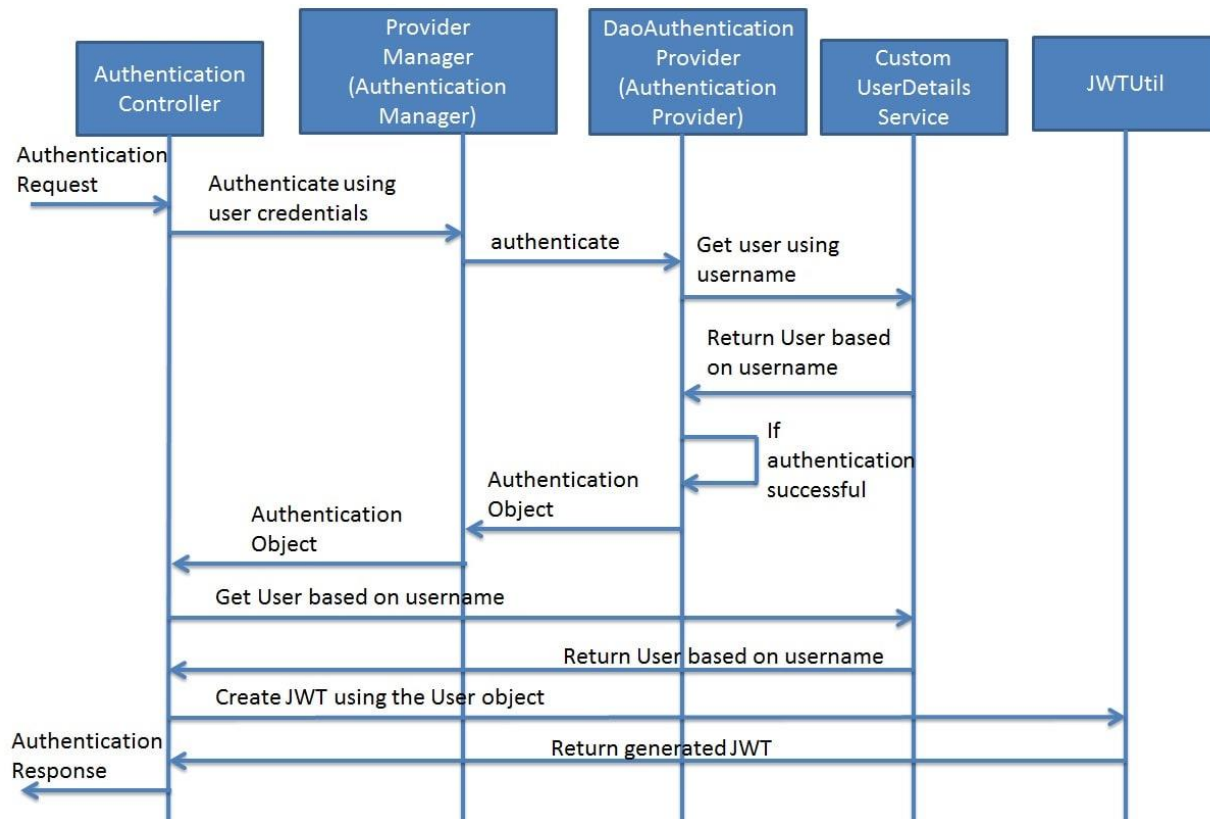
Cấu tạo gồm có 3 phần được mã hóa bởi base64



Điều 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 được 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 nhận ra ngay.

## Khởi tạo Token khi login

Login thành công trả về token cho client



workspace - sourcebackend/src/main/java/com/minahotel/sourcebackend/controller/StaffController.java - Eclipse IDE

```

125 //
126 //
127 // }
128
129 @ResponseStatus(HttpStatus.OK)
130 @PostMapping("/auth")
131 String authStaff(@RequestParam(name = "username") String username, @RequestParam(name = "password") String password)
132     throws Exception {
133     UsernamePasswordAuthenticationToken userpassword = new UsernamePasswordAuthenticationToken(username, password);
134     try {
135         authenticationManager.authenticate(userpassword);
136     } catch (DisabledException ex) {
137         LOGGER.info(ex.getMessage());
138         throw new Exception("USER_DISABLED", ex);
139     } catch (BadCredentialsException ex) {
140         LOGGER.info(ex.getMessage());
141         throw new Exception("INVALID_CREDENTIALS", ex);
142     } catch (AuthenticationException e) {
143         LOGGER.info(e.getMessage());
144         throw e;
145     }
146     final UserDetails userDetails = staffRepositoryServices.loadUserByUsername(username);
147     final String token = jwtUtil.generateToken(userDetails);
148     return token;
149 }
150
151
152

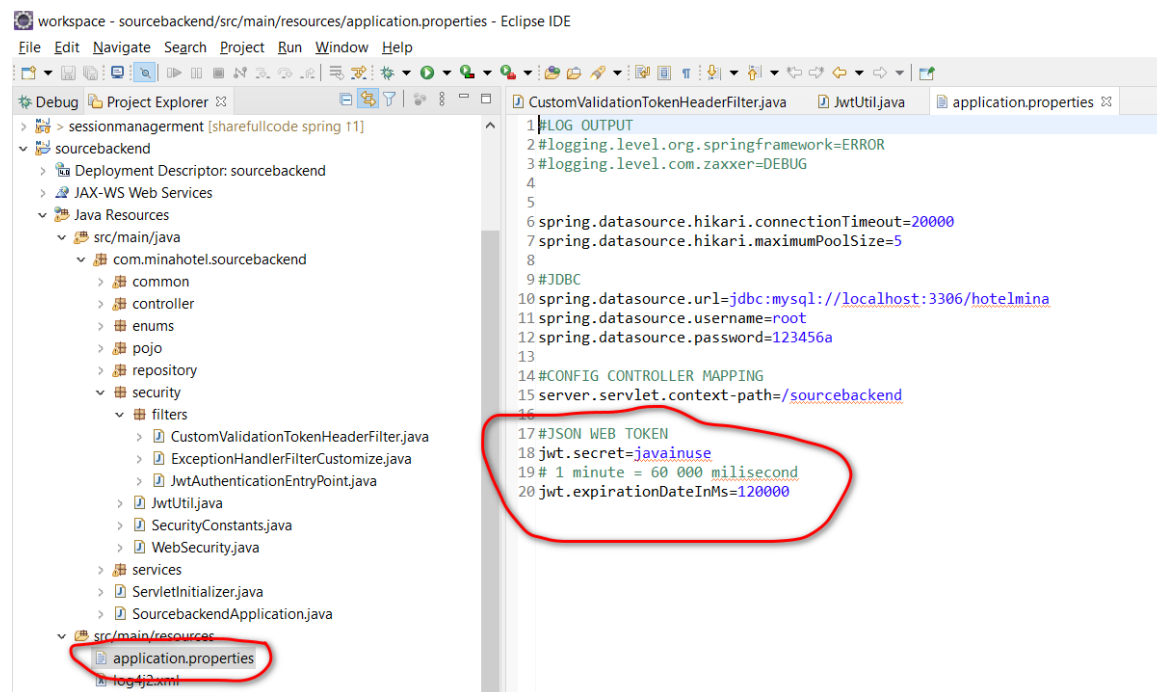
```

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

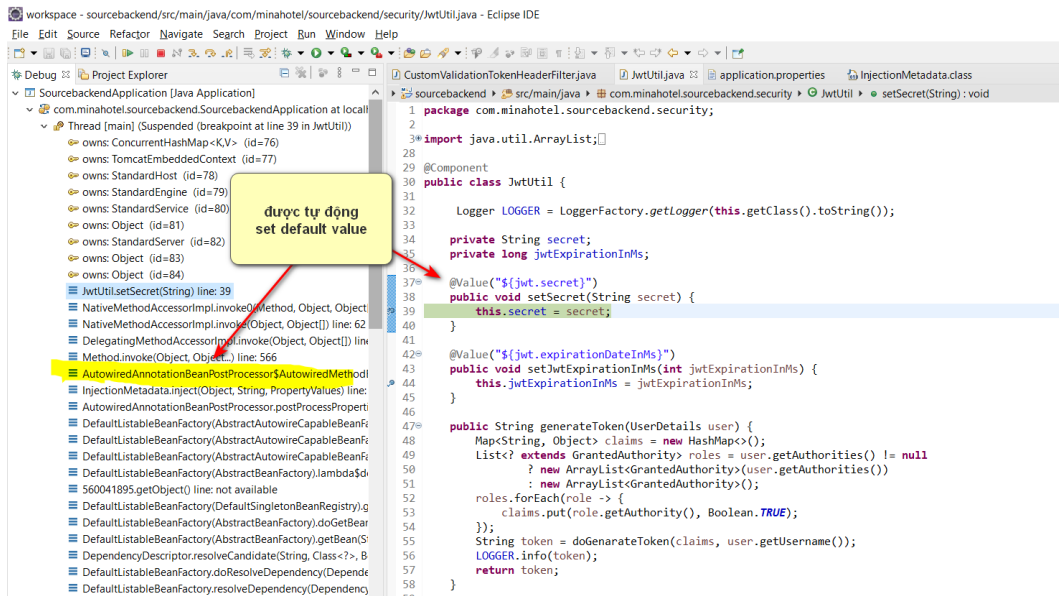
Secret key: Để có được chữ ký (signature) trong token ta cần phải cung cấp secret key để chỉ có server mới có thể kiểm tra và giải mã check xem token đó có đúng cấu trúc và đúng là token mà server đã cung cấp cho client hay không .

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

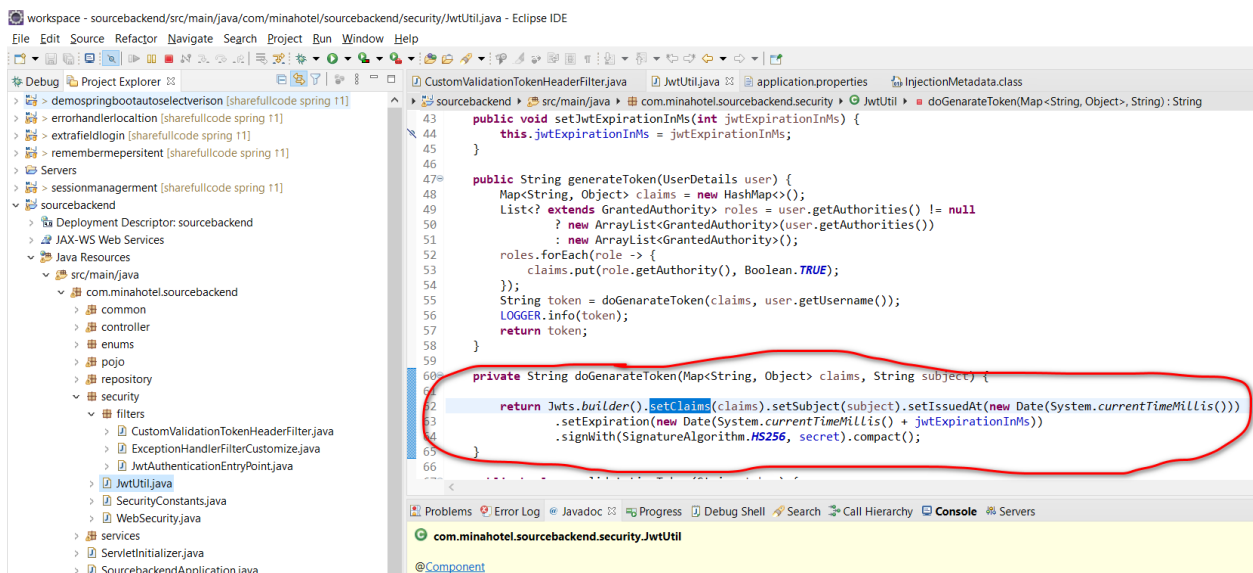
2 tham số này được lưu vào application.properties hoặc trong class code cũng ok



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



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 chứa 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.



## 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.

```

CustomValidationTokenHeaderFilter.java WebSecurity.java
doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) : void
25
26 @Component
27 public class CustomValidationTokenHeaderFilter extends OncePerRequestFilter {
28
29     Logger LOGGER = LoggerFactory.getLogger(this.getClass().toString());
30
31     @Autowired
32     private JwtUtil jwtTokenUtil;
33
34     @Override
35     protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
36         throws ServletException, IOException {
37
38         if (!"/refreshToken".equalsIgnoreCase(request.getServletPath())) {
39             String tokenExists = WebUtil.extractJwtFromRequest(request);
40             if (StringUtils.hasText(tokenExists) && jwtTokenUtil.validateToken(tokenExists)) {
41                 List<GrantedAuthority> lsAuthorization = jwtTokenUtil.getRolesFromToken(tokenExists);
42                 UserDetails userDetails = new User(jwtTokenUtil.getIdStaffFromToken(tokenExists), "", lsAuthorization);
43                 UsernamePasswordAuthenticationToken user = new UsernamePasswordAuthenticationToken(userDetails, null,
44                     userDetails.getAuthorities());
45                 SecurityContextHolder.getContext().setAuthentication(user);
46             } else {
47                 LOGGER.info("Not token in header ");
48             }
49         }
50         // continue filter of chain
51         filterChain.doFilter(request, response);
52     }
53
54 }
55

```

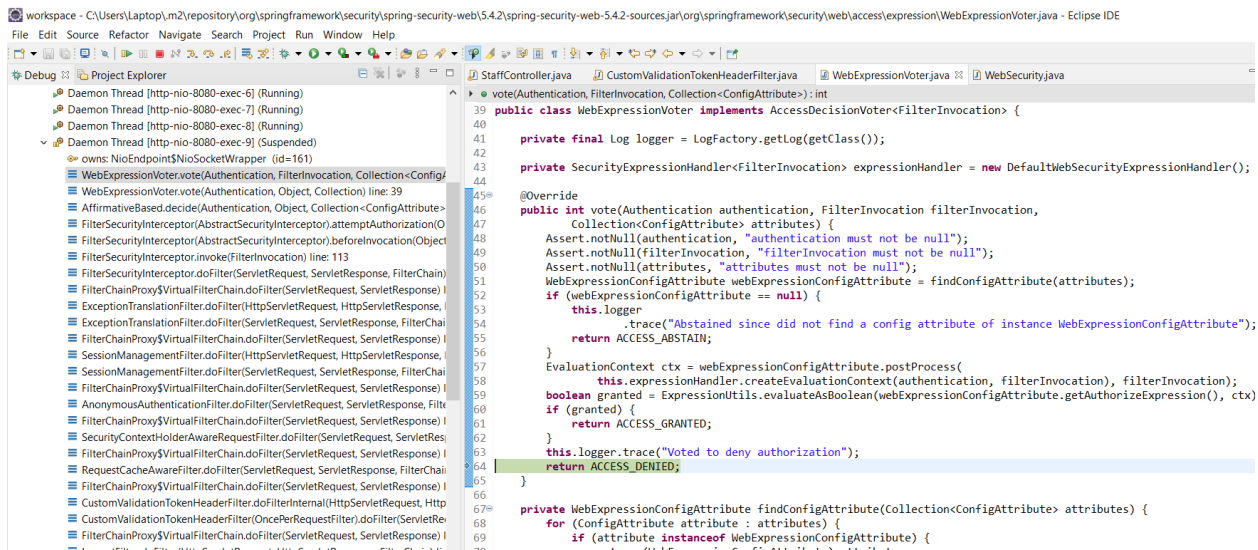
```

CustomValidationTokenHeaderFilter.java WebSecurity.java
sourcebackend src/main/java com.minahotel.sourcebackend.security WebSecurity configure(HttpSecurity) : void
32 JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;
33
34 @Autowired
35 CustomValidationTokenHeaderFilter customValidationTokenHeaderFilter;
36
37 @Autowired
38 ExceptionHandlerFilterCustomize exceptionHandlerFilterCustomize;
39
40 @Override
41 protected void configure(HttpSecurity http) throws Exception {
42
43     http.cors().and().csrf().disable();
44     http.antMatcher("/**").authorizeRequests()
45         .antMatchers(SecurityConstants.SIGN_UP_URL
46             , SecurityConstants.RESET_PASS_URL
47             , SecurityConstants.RESET_TOKEN_URL).permitAll()
48         .anyRequest().authenticated()
49         .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
50         .and()
51         .exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint); // handler error
52     http.addFilterBefore(customValidationTokenHeaderFilter, UsernamePasswordAuthenticationFilter.class); // validation token header
53     http.addFilterBefore(exceptionHandlerFilterCustomize, CorsFilter.class); // handler error runtime as expired jwt
54
55 }

```

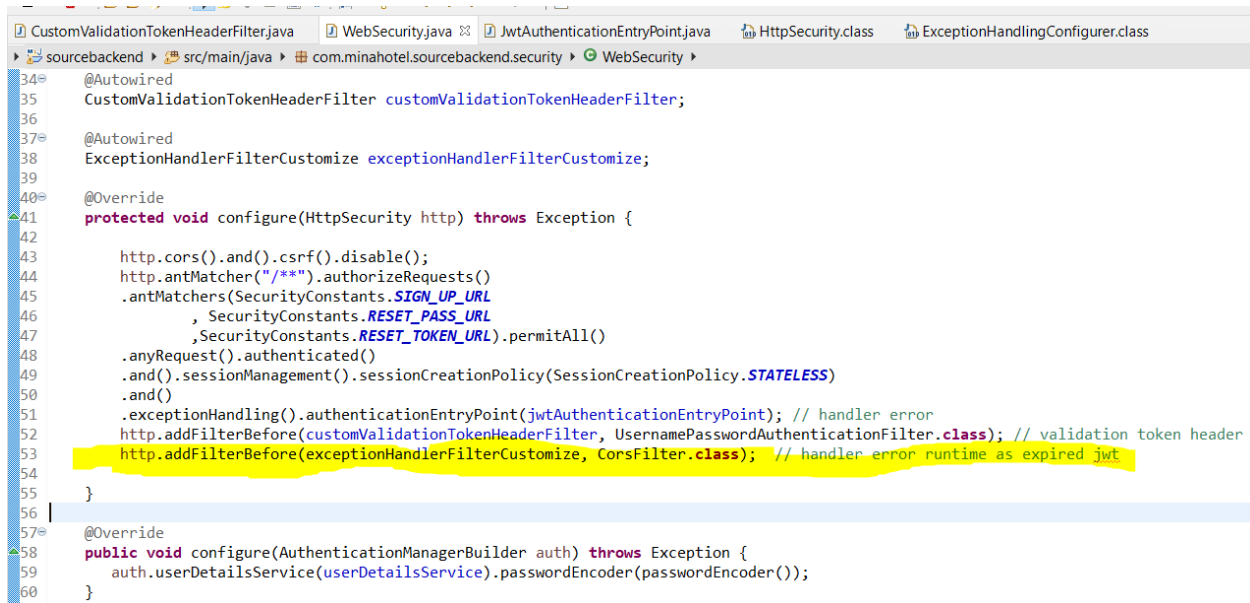
Nếu validation token thành công sẽ tạo object UsernamePasswordAuthenticationToken set nó vào SecurityContextHolder.getContext().setAuthentication(user); để chứng minh rằng object request này đã xác thực thành công các filter sau ko cần phải check tiếp (FilterSecurityInterceptor) nếu không set nó sẽ check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





## 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.



```

ExceptionHandlerFilterCustomize.java
sourcebackend > src/main/java > com.minahotel.sourcebackend.security.filters > ExceptionHandlerFilterCustomize
1 package com.minahotel.sourcebackend.security.filters;
2
3 import java.io.IOException;
20
21 @Component
22 public class ExceptionHandlerFilterCustomize extends OncePerRequestFilter {
23
24     @Override
25     protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
26         throws ServletException, IOException {
27         try {
28             filterChain.doFilter(request, response);
29         } catch (ExpiredJwtException ex) {
30             Map<String, Object> map= Collections.singletonMap("error", ex.getMessage() );
31             response.setStatus(HttpStatus.UNAUTHORIZED.value());
32             response.getWriter().write(convertObjectToJson(map));
33         }
34     }
35
36     private String convertObjectToJson(Object object) throws JsonProcessingException {
37         if (object == null) {
38             return null;
39         }
40         ObjectMapper mapper = new ObjectMapper();
41         return mapper.writeValueAsString(object);
42     }
43 }

```

## Test

### Login

[http://localhost:8080/sourcebackend/auth?username=staff\\_02&password=123](http://localhost:8080/sourcebackend/auth?username=staff_02&password=123)
Save

POST
http://localhost:8080/sourcebackend/auth?username=staff\_02&password='...
Send

Params
Auth
Headers (7)
Body
Pre-req.
Tests
Settings
Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	username	staff_02			
<input checked="" type="checkbox"/>	password	123			

Body

200 OK
917 ms
591 B
Save Response

Pretty
Raw
Preview
Visualize
Text

```

1 eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJzdGFmZ18wMiIsInRpZXB0YW4iOnRydWUsImV4cCI6MTYxOTg0MzM4NiwiYWFWOiJjoxNjE5ODQzMjY2fQ.W1NQMoMZKP7zowA1GXUye7FsSBhUL5ApGVs6wqjvdds

```



## Request Get API

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

- URL:** `http://localhost:8080/sourcebackend/staff`
- Method:** GET
- Headers:** 7 headers are listed, including a Bearer Token: `eyJhbGciOiJIUzI1NiJ9.eyJzdWwiOiJz...`
- Body:** The response is a JSON object with the following structure:

```
1 {
2   "idstaff": "staff_01",
3   "username": "nguyen thi vip",
4   "pass": "$2a$2a$10$aK7ynWil8RK8Rlu9QXJIqeGjq2s2IJYpQZ78YNZvuxbVSxiw8.iYe",
5   "role": "admin",
6 }
```
- Status:** 200 OK, 433 ms, 2.93 KB

## Request Expired Time

The screenshot shows a Java IDE console with the following log entries:

```
SourcebackendApplication [Java Application] D:\Document\openjdk-11+28_windows-x64_bin\jdk-11\bin\javaw.exe (May 1, 2021, 11:26:55 AM)
2021-05-01T11:29:05.514+0700 DEBUG RequestResponseBodyMethodProcessor.traceDebug(91) - Writing [[com.minahotel.sourcebackend.pojo.Staff@26729326, com.minahotel.
2021-05-01T11:29:05.518+0700 DEBUG OpenEntityManagerInViewInterceptor.afterCompletion(111) - Closing JPA EntityManager in OpenEntityManagerInViewInterceptor
2021-05-01T11:29:05.518+0700 DEBUG DispatcherServlet.logResult(1131) - Completed 200 OK
2021-05-01T11:29:05.519+0700 DEBUG SecurityContextPersistenceFilter.doFilter(118) - Cleared SecurityContextHolder to complete request
2021-05-01T11:29:13.214+0700 DEBUG FilterChainProxy.doFilterInternal(208) - Securing GET /staff
2021-05-01T11:29:13.214+0700 DEBUG SecurityContextPersistenceFilter.doFilter(102) - Set SecurityContextHolder to empty SecurityContext
2021-05-01T11:29:13.216+0700 DEBUG SecurityContextPersistenceFilter.doFilter(118) - Cleared SecurityContextHolder to complete request
2021-05-01T11:29:13.217+0700 ERROR [dispatcherServlet].log(175) - Servlet.service() for servlet [dispatcherServlet] in context with path [/sourcebackend] threw
org.springframework.security.authentication.BadCredentialsException: INVALID_CREDENTIALS
    at com.minahotel.sourcebackend.security.JwtUtil.validateToken(JwtUtil.java:74) ~[classes/:?]
    at com.minahotel.sourcebackend.security.filters.CustomValidationTokenHeaderFilter.doFilterInternal(CustomValidationTokenHeaderFilter.java:40) ~[classes/
```

Được hứng từ PointApp nên customize lỗi thật sự

http://localhost:8080/sourcebackend/staff

Save

GET http://localhost:8080/sourcebackend/staff... Send

Params Auth Headers (7) Body Pre-req. Tests Settings Cookies

Type  
Bearer Token  
Token  
eyyJhbGciOiJIUzI1NiJ9.eyJzdWliOi...

The authorization header

Body 401 Unauthorized 45 ms 504 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "error": "Full authentication is required to access this resource"  
3 }
```

Request Token invalid

Được xử lí bởi filer expired time

http://localhost:8080/sourcebackend/staff

Save

GET http://localhost:8080/sourcebackend/staff... Send

Params Auth Headers (7) Body Pre-req. Tests Settings Cookies

Type  
Bearer Token  
Token  
eyJhbGciOiJIUzI1NiJ9.eyJzdWliOiJz...

The authorization header

Body 401 Unauthorized 14 ms 434 B Save Response

Pretty Raw Preview Visualize JSON

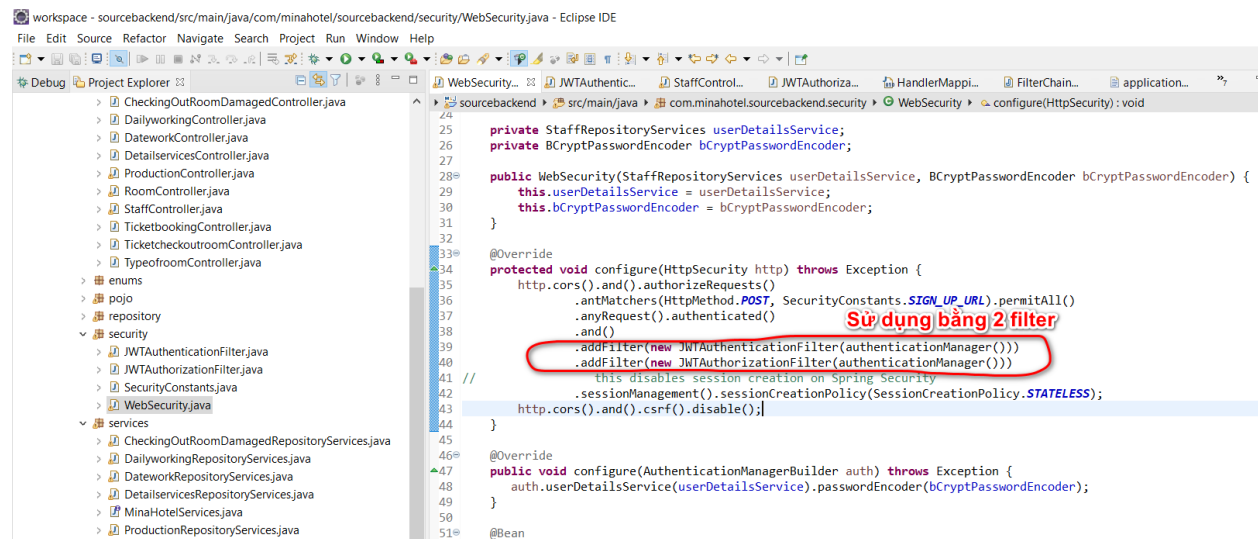
```
1 {  
2   "error": "Token has Expired"  
3 }
```

## 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



```
workspace - sourcebackend/src/main/java/com/minahotel/sourcebackend/security/WebSecurity.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
WebSecurity... src/main/java com.minahotel.sourcebackend.security WebSecurity configure(HttpSecurity): void
> CheckingOutRoomDamagedController.java
> DailyworkingController.java
> DateworkController.java
> DetailservicesController.java
> ProductionController.java
> RoomController.java
> StaffController.java
> TicketbookingController.java
> TicketcheckoutroomController.java
> TypeofroomController.java
> enums
> pojo
> repository
> security
  > JWTAuthenticationFilter.java
  > JWTAuthorizationFilter.java
  > SecurityConstants.java
  > WebSecurity.java
> services
  > CheckingOutRoomDamagedRepositoryServices.java
  > DailyworkingRepositoryServices.java
  > DateworkRepositoryServices.java
  > DetailservicesRepositoryServices.java
  > MinaHotelServices.java
  > ProductionRepositoryServices.java
44
45
46 private StaffRepositoryServices userDetailsService;
47 private BCryptPasswordEncoder bCryptPasswordEncoder;
48
49 public WebSecurity(StaffRepositoryServices userDetailsService, BCryptPasswordEncoder bCryptPasswordEncoder) {
50     this.userDetailsService = userDetailsService;
51     this.bCryptPasswordEncoder = bCryptPasswordEncoder;
52 }
53
54 @Override
55 protected void configure(HttpSecurity http) throws Exception {
56     http.cors().and().authorizeRequests()
57         .antMatchers(HttpMethod.POST, SecurityConstants.SIGN_UP_URL).permitAll()
58         .anyRequest().authenticated()
59         .and()
60         .addFilter(new JWTAuthenticationFilter(authenticationManager()))
61         .addFilter(new JWTAuthorizationFilter(authenticationManager()))
62     // this disables session creation on Spring Security
63     .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
64     http.cors().and().csrf().disable();
65 }
66
67 @Override
68 public void configure(AuthenticationManagerBuilder auth) throws Exception {
69     auth.userDetailsService(userDetailsService).passwordEncoder(bCryptPasswordEncoder);
70 }
71
72 @Bean
```

## Reference

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