

# JAVA WEB

## Unit 7: Spring Security



# Mục lục

## 1. Spring Security

- Giới thiệu
- Xây dựng project minh họa xác thực bằng User, Password ()
- Bổ sung xác thực, phân quyền cho ứng dụng (Chapter7SpringBoot\_Jpa\_Jsp\_Security)

## 2. JSON Web Token (JWT)

- Giới thiệu
- Xây dựng project minh họa

## 3. Giới thiệu Oauth2

# Spring Security

**Spring Security** là một framework được tích hợp trong hệ sinh thái của Spring, cung cấp một loạt các tính năng và dịch vụ để xác thực (authentication), phân quyền (authorization), bảo vệ các URL.

## **Các thành phần chính của Spring Security:**

- **Authentication (Xác thực):** Spring Security quản lý việc xác thực người dùng bằng cách xác định ai đang truy cập vào hệ thống. Nó hỗ trợ nhiều phương thức xác thực khác nhau như username/password, JWT , OAuth2, ...
- **Authorization (Phân quyền):** Sau khi xác thực, Spring Security quyết định người dùng có quyền truy cập vào các tài nguyên nào trong hệ thống. Quyền truy cập có thể được định nghĩa ở mức URL (web security), mức phương thức (method security), hoặc mức biểu thức (expression-based security).

Spring Security tích hợp chặt chẽ với các thành phần khác của Spring như Spring MVC, Spring Boot, và Spring Data, giúp việc triển khai bảo mật trở nên dễ dàng và hiệu quả.

## Cấu hình Spring Security

**@Configuration**: Khi ứng dụng Spring khởi động, nó sẽ tìm lớp này và gọi thực thi phương thức `configure()`.

**@EnableWebSecurity**: Khi ứng dụng Spring khởi động, web security sẽ được bật lên. Phương thức `configure()` sẽ thiết lập cấu hình security cho ứng dụng web.

**@EnableGlobalMethodSecurity**: Để bảo mật các phương thức trong ứng dụng web, chúng ta phải sử dụng annotation là **@PreAuthorize()**.

# Chapter7SpringBootAuth User Password

The screenshot displays an IDE with the following components:

- Project Explorer (Left):** Shows the project structure for 'Chapter7SpringBootAuth\_User\_Password [boot-war]'. The 'src/main/java/org.hanbo.boot.app/config' directory is selected, highlighting the 'WebAppSecurityConfig' class. Other visible directories include 'controllers' (with 'HelloController', 'LoginController', 'SecuredPageController') and 'security' (with 'UserAuthenticationService'). The 'target' directory contains 'boot-war.iml', 'pom.xml', and 'Readme.txt'.
- Code Editor (Right):** Displays the code for 'WebAppSecurityConfig.java'. The code is as follows:

```
1 package org.hanbo.boot.app.config;
2
3 import ...
4
5 @Configuration
6 @EnableWebSecurity
7 @EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true)
8 public class WebAppSecurityConfig extends WebSecurityConfigurerAdapter
9 {
10     @Autowired
11     private UserAuthenticationService authenticationProvider;
12
13     @Override
14     protected void configure(HttpSecurity http) throws Exception
15     {
16         http.authorizeRequests() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry
17             .antMatchers( ...antPatterns: "/public/**", "/assets/**").permitAll()
18             .anyRequest().authenticated()
19             .and() HttpSecurity
20             .formLogin() FormLoginConfigurer<HttpSecurity>
21                 .loginPage("/login")
22                 .permitAll()
23                 .usernameParameter("username")
24                 .passwordParameter("password")
25                 .defaultSuccessUrl( defaultSuccessUrl: "/secure/index", alwaysUse: true).failureUrl("/public/aut
26                 .successHandler(new SavedRequestAwareAuthenticationSuccessHandler())
```

## Cấu hình cách xử lý http requests

```
@Override
protected void configure(HttpSecurity http) throws Exception
{
    http.authorizeRequests()
        .antMatchers("/public/**", "/assets/**").permitAll()
        .anyRequest().authenticated() không yêu cầu xác thực
        .and()
    .formLogin()
        .loginPage("/login")
        .permitAll()
        .usernameParameter("username")
        .passwordParameter("password")
        .defaultSuccessUrl("/secure/index", true).failureUrl("/public/authFailed")
        .successHandler(new SavedRequestAwareAuthenticationSuccessHandler())
        .and()
        .exceptionHandling().accessDeniedPage("/public/accessDenied")
        .and()
    .logout()
        .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
        .logoutSuccessUrl("/public/logout").permitAll();
}
```

## Cấu hình cách xử lý http requests

```
http.authorizeRequests()  
    .antMatchers("/public/**", "/assets/**").permitAll()  
    .anyRequest().authenticated()
```

Các request có dạng: *<context root>/public/\*\** and *<context root>/assets/\*\**, sẽ không yêu cầu xác thực.



## Cấu hình cách xử lý http requests

```
.formLogin()  
  .loginPage("/login")  
  .permitAll()  
  .usernameParameter("username")  
  .passwordParameter("password")  
  .defaultSuccessUrl("/secure/index", true).failureUrl("/public/authFailed")  
  .successHandler(new SavedRequestAwareAuthenticationSuccessHandler())
```

Cấu hình trang đăng nhập là `<context root>/login`, trên form có 1 ô text là `username` và 1 ô text là `password`. User đăng nhập thành công thì sẽ được chuyển đến trang `<context root>/secure/index`. User cố gắng đăng nhập mà thất bại thì sẽ chuyển đến trang `<context root>/public/authFailed`.

Nếu người dùng muốn truy cập một trang bảo mật cụ thể, sau khi đăng nhập thành công, `successHandler` sử dụng đối tượng `SavedRequestAwareAuthenticationSuccessHandler` để thực hiện việc này.



## Cấu hình cách xử lý http requests

```
.exceptionHandling().accessDeniedPage("/public/accessDenied")
```

Đối với user đã login, truy cập đến trang không có quyền truy cập thì sẽ được chuyển đến trang này.

```
.logout()
```

```
.logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
```

```
.logoutSuccessUrl("/public/logout").permitAll();
```

*<context root>/logout* là trang để logout; Nếu logout thành công thì sẽ chuyển đến trang *<context root>/public/logout*.

# User Authentication Service

```

@Service public class UserAuthenticationService implements AuthenticationProvider {

    @Override public Authentication authenticate( Authentication auth) throws AuthenticationException {
        Authentication retVal = null; List<GrantedAuthority> grantedAuths = new ArrayList<GrantedAuthority>();
        if (auth != null) {
            String name = auth.getName(); String password = auth.getCredentials().toString();
            if (name.equals("admin") && password.equals("admin12345")) {
                grantedAuths.add(new SimpleGrantedAuthority("ROLE_ADMIN"));
                grantedAuths.add(new SimpleGrantedAuthority("ROLE_STAFF"));
                grantedAuths.add(new SimpleGrantedAuthority("ROLE_USER"));
                retVal = new UsernamePasswordAuthenticationToken( name, "", grantedAuths );
            }
            else if (name.equals("staff1") && password.equals("staff12345")) {
                grantedAuths.add(new SimpleGrantedAuthority("ROLE_STAFF"));
                grantedAuths.add(new SimpleGrantedAuthority("ROLE_USER"));
                retVal = new UsernamePasswordAuthenticationToken( name, "", grantedAuths );
            }
            else if (name.equals("user1") && password.equals("user12345")) {
                grantedAuths.add(new SimpleGrantedAuthority("ROLE_USER"));
                retVal = new UsernamePasswordAuthenticationToken( name, "", grantedAuths );
            }
        }
        else {
            retVal = new UsernamePasswordAuthenticationToken( null, null, grantedAuths ); }
        return retVal;
    }

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

```

# User Authentication Service

Giao diện **AuthProvider** có 2 phương thức:

- **Authentication authenticate(Authentication auth)**
- **boolean supports(Class<?> tokenType)**

Phương thức **supports()** trả về true nếu form login trả về đúng kiểu đối tượng `UsernamePasswordAuthenticationToken`. Sau khi **supports()** trả về true thì phương thức **authenticate()** sẽ xử lý request.

UVINA

# Secure the Web Pages

```
@Controller
public class SecuredPageController
{
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    @RequestMapping(value="/secure/adminPage", method = RequestMethod.GET)
    public ModelAndView adminPage()
    {
        ModelAndView retVal = new ModelAndView();
        retVal.setViewName("webAccess");
        retVal.addObject("pageInfo", "The AWESOME Admin Page");
        retVal.addObject("userInfo", "Awesome Admin User.");
        return retVal;
    }

    @PreAuthorize("hasRole('ROLE_STAFF')")
    @RequestMapping(value="/secure/staffPage", method = RequestMethod.GET)
    public ModelAndView staffPage()
    {
        ModelAndView retVal = new ModelAndView();
        retVal.setViewName("webAccess");
        retVal.addObject("pageInfo", "The SUPPORTING Staff Page");
        retVal.addObject("userInfo", "T.L.C Staff User.");
        return retVal;
    }

    @PreAuthorize("hasRole('ROLE_USER')")
    @RequestMapping(value="/secure/userPage", method = RequestMethod.GET)
    public ModelAndView userPage()
    {
        ModelAndView retVal = new ModelAndView();
        retVal.setViewName("webAccess");
        retVal.addObject("pageInfo", "The LAMMO User Page");
        retVal.addObject("userInfo", "an ordinary User.");
        return retVal;
    }
}
```

# Demo

13

← → ↻ ⓘ localhost:8080/login 🔑 🔍 ☆ 🔄 📄 🎵 H

User Name:

user1

Password:

.....

Login

```
if (name.equals("admin") && password.equals("admin12345")) {  
    grantedAuths.add(new SimpleGrantedAuthority("ROLE_ADMIN"));  
    grantedAuths.add(new SimpleGrantedAuthority("ROLE_STAFF"));  
    grantedAuths.add(new SimpleGrantedAuthority("ROLE_USER"));  
    retVal = new UsernamePasswordAuthenticationToken( name, "", grantedAuths );  
}  
else if (name.equals("staff1") && password.equals("staff12345")) {  
    grantedAuths.add(new SimpleGrantedAuthority("ROLE_STAFF"));  
    grantedAuths.add(new SimpleGrantedAuthority("ROLE_USER"));  
    retVal = new UsernamePasswordAuthenticationToken( name, "", grantedAuths );  
} else if (name.equals("user1") && password.equals("user12345")) {  
    grantedAuths.add(new SimpleGrantedAuthority("ROLE_USER"));  
    retVal = new UsernamePasswordAuthenticationToken( name, "", grantedAuths );  
}  
else {  
    retVal = new UsernamePasswordAuthenticationToken( null, null, grantedAuths );  
}  
return retVal;
```

# Chapter7SpringBoot Jpa Jsp Security

The screenshot displays an IDE window for a project named "Chapter7SpringBoot\_Jpa\_Jsp\_Security". The left sidebar shows the project structure, with the "mvc" package containing several classes, including "WebSecurityConfig", which is highlighted with a red box. The main editor area shows the code for "WebSecurityConfig.java".

```
1 package mta.mvc;
2
3 import ...
4
12
13 @Configuration
14 @EnableWebSecurity
15 public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
16     @Autowired
17     private DataSource dataSource;
18     @Autowired
19     @Override
20     public void configure(AuthenticationManagerBuilder authBuilder) throws Exception {
21         authBuilder.jdbcAuthentication()
22             .dataSource(dataSource)
23             .passwordEncoder(new BCryptPasswordEncoder())
24             .usersByUsernameQuery("select username, password, enabled from users where username=?")
25             .authoritiesByUsernameQuery("select username, role from users where username=?")
26     }
27
28     @Override
29     protected void configure(HttpSecurity http) throws Exception {
30         http.authorizeRequests( ) ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry
31
32         .antMatchers( ...antPatterns: "/edit/*", "/delete/*").hasRole("ADMIN")
33
34         .anyRequest().authenticated()
```

# Bổ sung xác thực, phân quyền cho Project

Thêm các Dependency vào pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<!-- https://mvnrepository.com/artifact/org.springframework.security/spring-security-taglibs -->
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-taglibs</artifactId>
  <version>5.4.5</version>
</dependency>
```

Tạo bảng và chèn 02 bản ghi vào bản users

```
CREATE TABLE `users` (
  `user_id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(45) NOT NULL,
  `password` varchar(64) NOT NULL,
  `role` varchar(45) NOT NULL,
  `enabled` tinyint(4) DEFAULT NULL,
  PRIMARY KEY (`user_id`)
);

INSERT INTO `users` (`username`,`password`,`role`,`enabled`)
VALUES ('test', '$2a$10$G3s28odHo8pzgbZxAzkkDuOPsWIE5FIW/GLWzuzcSS1o55839DJ4u', 'ROLE_USER', 1);

INSERT INTO `users` (`username`,`password`,`role`,`enabled`)
VALUES ('admin', '$2a$10$mOhFQi0323z.gAZzWvAINugiAseueSXW8PcxM70xlhrZZCCh6ks6a', 'ROLE_ADMIN', 1);
```

# Cách cấu hình Spring Security và cấu hình xác thực

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

    @Autowired
    private DataSource dataSource;

    @Autowired
    public void configAuthentication(AuthenticationManagerBuilder authBuilder) throws Exception {
        authBuilder.jdbcAuthentication()
            .dataSource(dataSource)
            .passwordEncoder(new BCryptPasswordEncoder())
            .usersByUsernameQuery("select username, password, enabled from users where username=?")
            .authoritiesByUsernameQuery("select username, role from users where username=?")
        ;
    }
}
```



# Cấu hình xác thực

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests( ) ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry

        .antMatchers( ...antPatterns: "/edit/*", "/delete/*").hasRole("ADMIN")

        .anyRequest().authenticated()

        .and() HttpSecurity
        .formLogin().permitAll() FormLoginConfigurer<HttpSecurity>

        .and() HttpSecurity
        .logout().permitAll() LogoutConfigurer<HttpSecurity>

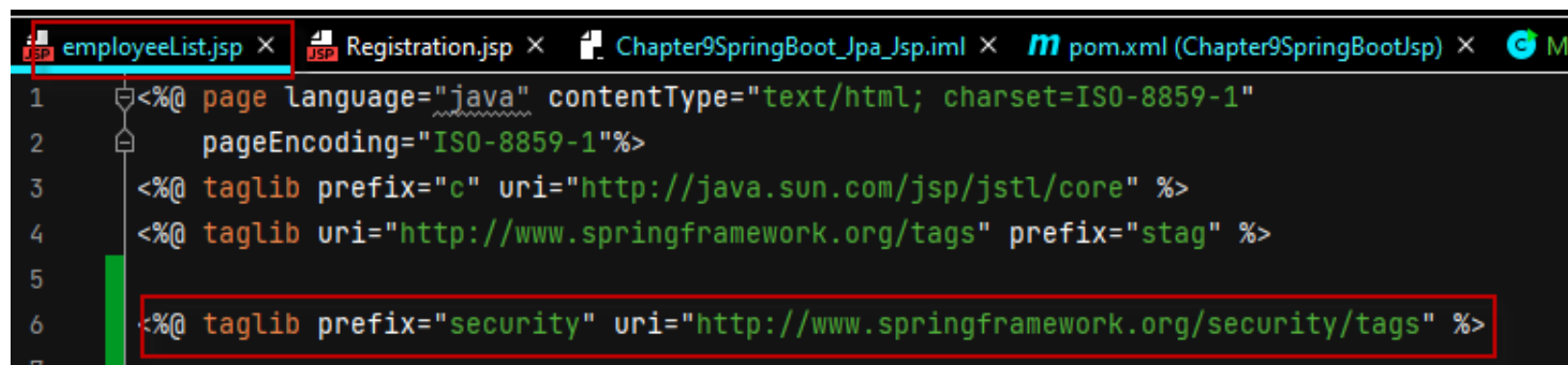
        .and() HttpSecurity
        .exceptionHandling().accessDeniedPage("/403") ExceptionHandlingConfigurer<HttpSecurity>
    ;
}
```

# Sử dụng Spring Security Taglibs trong JSP

Thêm dependency

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-taglibs</artifactId>
  <version>5.2.2.RELEASE</version>
</dependency>
```

Khai báo



```
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2   pageEncoding="ISO-8859-1"%>
3 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4 <%@ taglib uri="http://www.springframework.org/tags" prefix="stag" %>
5
6 <%@ taglib prefix="security" uri="http://www.springframework.org/security/tags" %>
7
```

Sử dụng

```
<security:authorize access="hasRole('ADMIN')">
  <td>
    <a href="/edit?id=${emp.id}" >Edit</a>
  </td>
  <td>
    <form action="/delete?id=${emp.id}" method="post">
      <input type="submit" value="Delete" style="..." />
    </form>
  </td>
</security:authorize>
```



## Please sign in

## Employee Registration

First Name :

Last Name :

User Name :

Email Id :

Emp. Id :

Blood Group :

Age :

Personal Email :

Mobile No :

[Employee List](#)

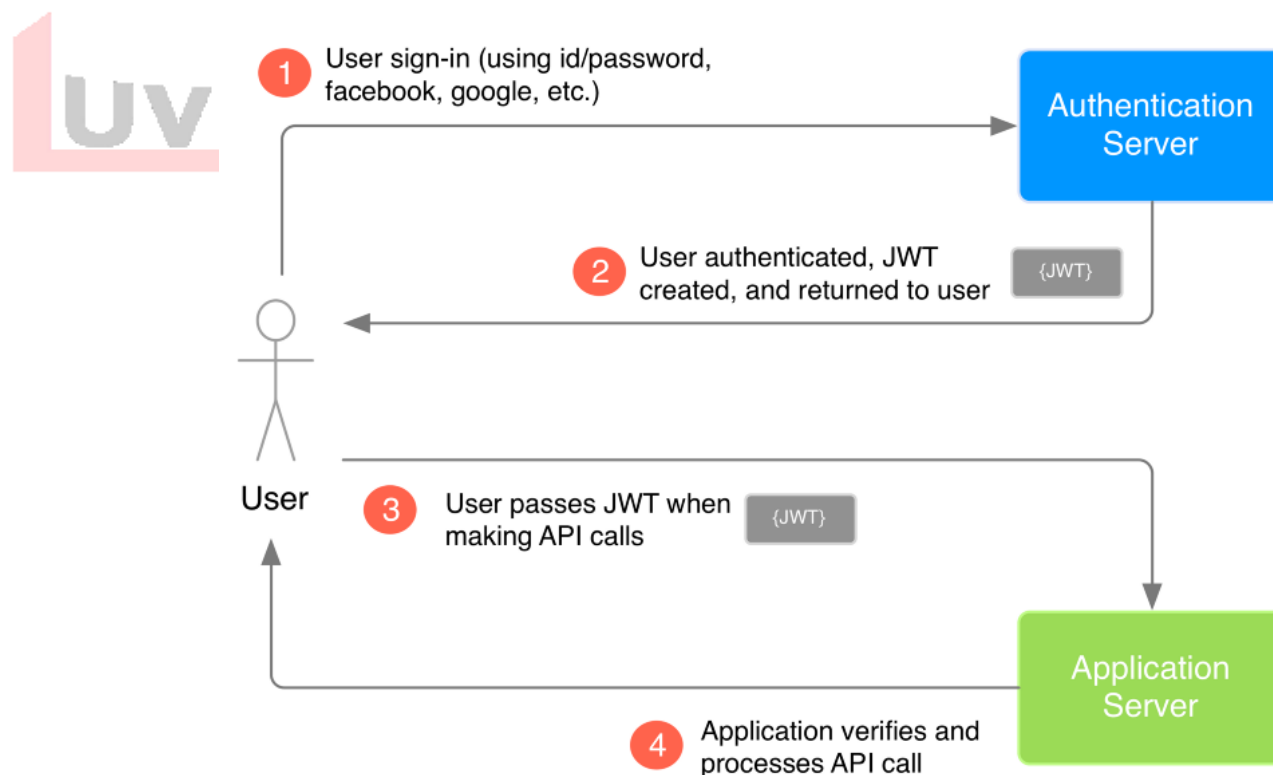
# JSON Web Token

## Giới thiệu:

Json Web Token (JWT) là một tiêu chuẩn mở (RFC 7519) nhằm xác minh thông tin an toàn giữa các bên Client-Server dưới dạng JSON object.

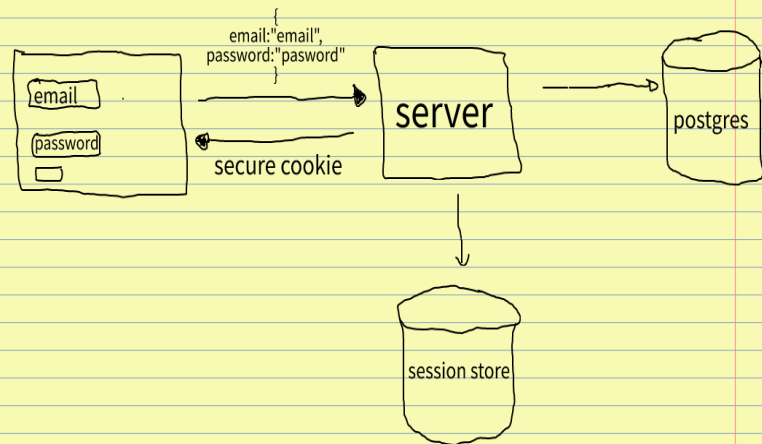
Thông tin này có thể được xác minh và tin cậy vì nó được ký điện tử - digitally signed. JWT có thể được ký bằng cách sử dụng một secret (với thuật toán HMAC) hoặc cặp public/private key dùng chuẩn RSA hoặc ECDSA.

## Luồng xác thực sử dụng JWT:

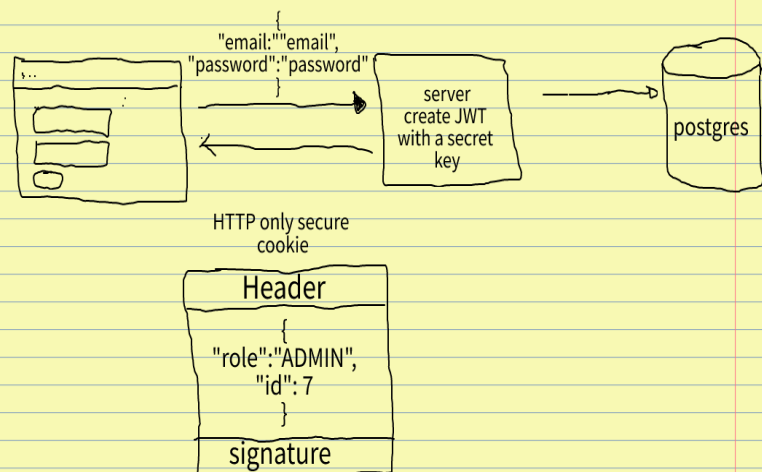


# So sánh Session và JWT

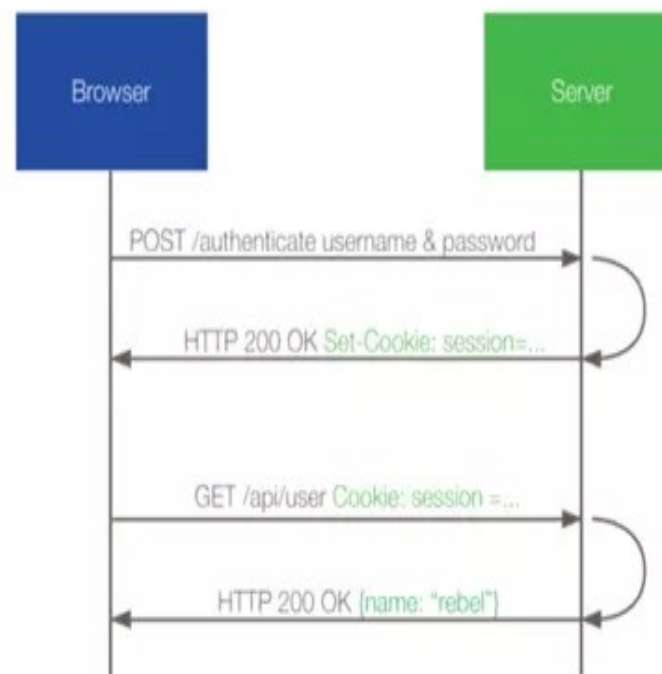
## Session



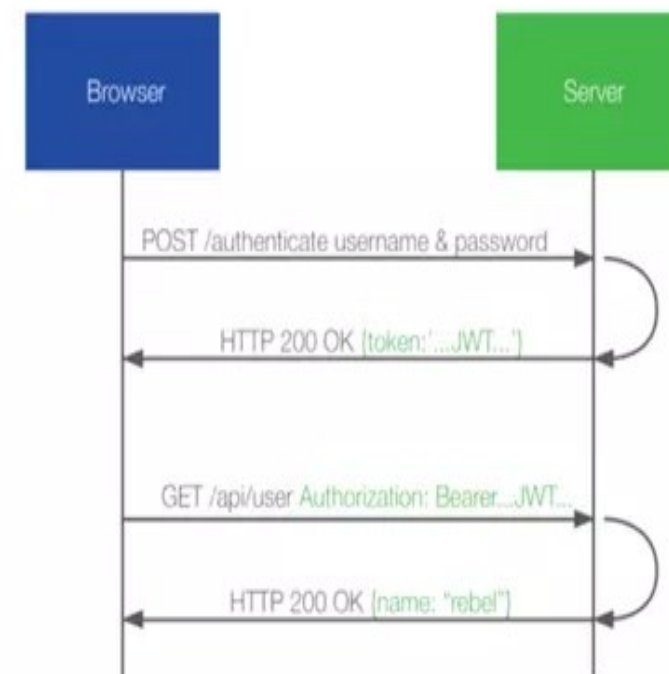
## JWT



## Traditional Cookie-based Authentication



## Modern Token-based Authentication



# Các thành phần của JWT (Header, Payload, Signature)

## Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

## Payload

Gồm tập các trường lưu lại thông tin như **danh tính** hay các quyền mà người dùng được cho phép.

## Signature

Phần cuối cùng và cũng là phần quan trọng nhất của một token, đó là phần **signature**.

HS256 ( base64UrlEncode(header) + "." + base64UrlEncode(payload), secret )

<https://jwt.io/#debugger-io>

The screenshot shows the JWT.io debugger interface. The browser address bar displays `jwt.io/#debugger-io`. The page has a dark header with the JWT logo and navigation links: Debugger, Libraries, Introduction, Ask. It also mentions 'Crafted by Auth0 by Okta'.

The interface is split into two main sections: 'Encoded' and 'Decoded'.

**Encoded:** Labeled 'PASTE A TOKEN HERE', it contains a long base64-encoded string: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.qHiFAsyHQy10FSojTHR543QiXVB6Y2p6LUd5x1V41Xc`.

**Decoded:** Labeled 'EDIT THE PAYLOAD AND SECRET', it shows the decoded components of the token:

- HEADER: ALGORITHM & TOKEN TYPE:**

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```
- PAYLOAD: DATA:**

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```
- VERIFY SIGNATURE:**

Shows the signature algorithm: `HMACSHA256(`

The input field contains: `base64UrlEncode(header) + "." + base64UrlEncode(payload),`

The secret field contains: `your-256-bit-sec123ret`

There is a checkbox for `secret` which is checked, and the output is `base64 encoded`.

# Tạo project Spring security JWT

The screenshot displays an IDE with the project structure on the left and the source code of `JwTokenFilter.java` on the right. The project is named `Chapter7SpringJwtAuthentication` and is a Spring Boot application. The package structure is as follows:

- `src/main/java/jwtDemo/jwt`
  - `JwTokenFilter`
  - `JwTokenUtil`
- `src/main/java/product`
  - `Product`
  - `ProductApi`
  - `ProductRepository`
- `src/main/java/user`
  - `api`
    - `User`
  - `UserRepository`
- `src/main/java/applicationSecurity`
  - `SpringJwtAuthExampleApplication`
- `src/main/resources`
  - `application.properties`
- `src/test/java/jwtDemo/user`
  - `UserRepositoryTests`

The `JwTokenFilter.java` file is highlighted, showing the following code:

```
19 @Component
20 public class JwTokenFilter extends OncePerRequestFilter {
21     /* Một filter (bộ lọc) chỉ được thực thi một lần trong suốt một yêu cầu HTTP (re
22     dù cho có xảy ra việc dispatch nhiều lần */
23     @Autowired
24     private JwTokenUtil jwtUtil;
25     @Override
26     protected void doFilterInternal(HttpServletRequest request,
27                                     HttpServletResponse response, FilterChain filterChain)
28         throws ServletException, IOException {
29
30         if (!hasAuthorizationBearer(request)) {
31             filterChain.doFilter(request, response);
32             return;
33         }
34
35         String token = getAccessToken(request);
36
37         if (!jwtUtil.validateAccessToken(token)) {
38             filterChain.doFilter(request, response);
39             return;
40         }
41
42         setAuthenticationContext(token, request);
43         filterChain.doFilter(request, response);
44     }
45 }
```

# Lớp JwtTokenUtil

```
public boolean validateAccessToken(String token) {  
    try { //io.jsonwebtoken.Jwts;  
        Jwts.parser().setSigningKey(SECRET_KEY).parseClaimsJws(token);  
        return true;  
    } catch (ExpiredJwtException ex) {  
        LOGGER.error("JWT expired", ex.getMessage());  
    } catch (IllegalArgumentException ex) {  
        LOGGER.error("Token is null, empty or only whitespace", ex.getMessage());  
    } catch (MalformedJwtException ex) {  
        LOGGER.error("JWT is invalid", ex);  
    } catch (UnsupportedJwtException ex) {  
        LOGGER.error("JWT is not supported", ex);  
    } catch (SignatureException ex) {  
        LOGGER.error("Signature validation failed");  
    }  
    return false;  
}
```

Dòng lệnh **Jwts.parser().setSigningKey(SECRET\_KEY).parseClaimsJws(token);** được sử dụng để phân tích và xác thực một JSON Web Token (JWT) bằng cách sử dụng một khóa bí mật (secret key).



## Lớp JwtTokenFilter extends OncePerRequestFilter

```
@Component
public class JwtTokenFilter extends OncePerRequestFilter {
    /* Một filter (bộ lọc) chỉ được thực thi một lần trong suốt một yêu cầu HTTP (re
    dù cho có xảy ra việc dispatch nhiều lần */
    @Autowired
    private JwtTokenUtil jwtUtil;
    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {
        if (!hasAuthorizationBearer(request)) {
            filterChain.doFilter(request, response);
            return;
        }

        String token = getAccessToken(request);

        if (!jwtUtil.validateAccessToken(token)) {
            filterChain.doFilter(request, response);
            return;
        }

        setAuthenticationContext(token, request);
        filterChain.doFilter(request, response);
    }
}
```

Nếu header of the request không chứa token thì tiếp tục thực hiện chuỗi các filter.  
Else, nếu token not verified thì cũng tiếp tục thực hiện chuỗi các filter.

Nếu token is verified, update the authentication context with the user details ID and email.  
In other words, it tells Spring that the user is authenticated, and tiếp tục chuỗi các filters.

# Lớp ApplicationSecurity - Cấu hình security

```
@Autowired private UserRepository userRepo;
@Autowired private JwtTokenFilter jwtTokenFilter;
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable();
    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

    http.authorizeRequests()
        .antMatchers(...antPatterns: "/auth/login").permitAll()
        .anyRequest().authenticated();

    http.exceptionHandling()
        .authenticationEntryPoint(
            (request, response, ex) -> {
                response.sendError(
                    HttpServletResponse.SC_UNAUTHORIZED,
                    ex.getMessage()
                );
            }
        );

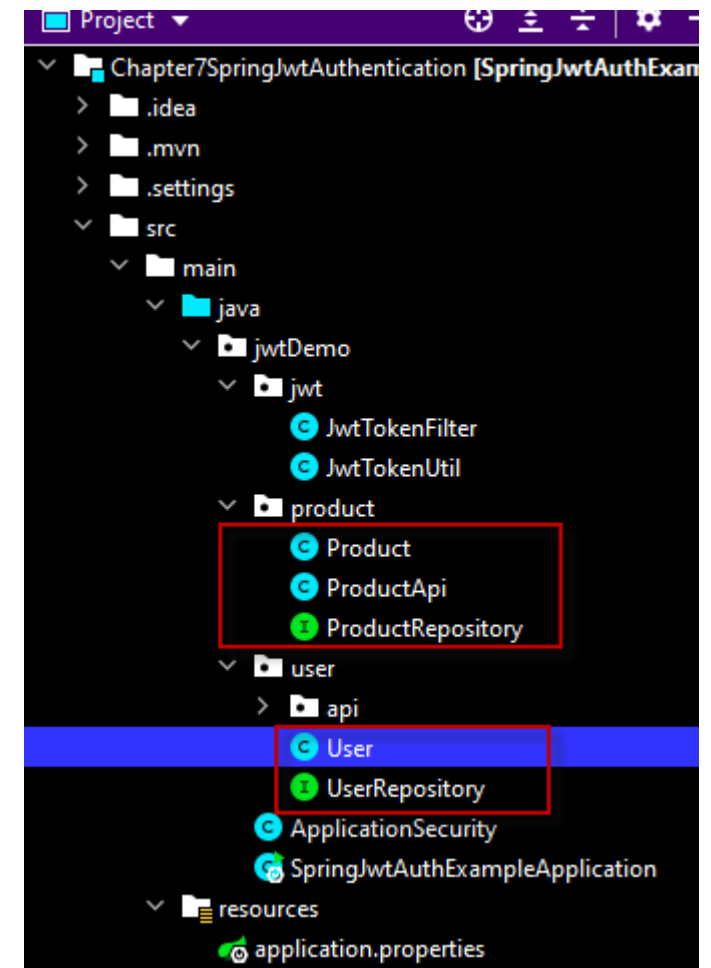
    http.addFilterBefore(jwtTokenFilter, UsernamePasswordAuthenticationFilter.class);
}
```

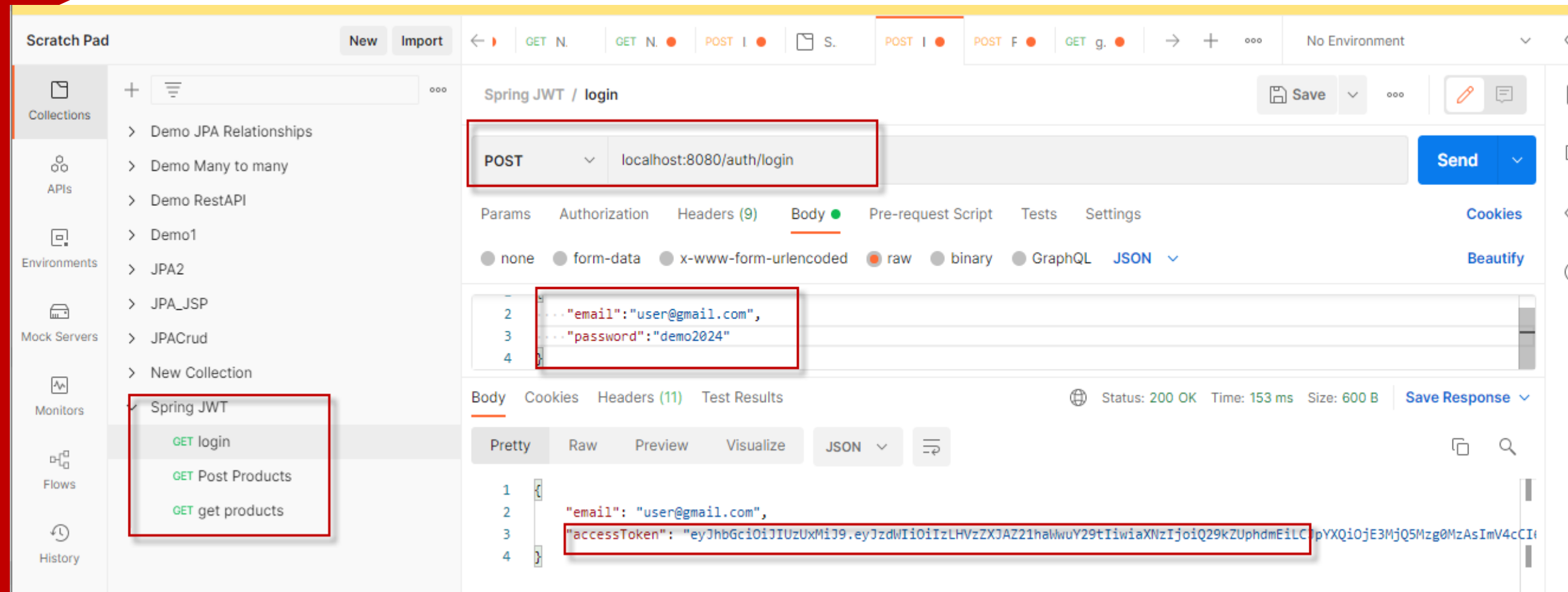
```
@Autowired private UserRepository userRepo;
@Autowired private JwtTokenFilter jwtTokenFilter;
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable();
    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

    http.authorizeRequests()
        .antMatchers(...antPatterns: "/auth/login").permitAll()
        .anyRequest().authenticated();

    http.exceptionHandling()
        .authenticationEntryPoint(
            (request, response, ex) -> {
                response.sendError(
                    HttpServletResponse.SC_UNAUTHORIZED,
                    ex.getMessage()
                );
            }
        );

    http.addFilterBefore(jwtTokenFilter, UsernamePasswordAuthenticationFilter.class);
}
```





# Test trên Postman

The screenshot displays the Postman application interface. On the left, a sidebar shows a collection named "Spring JWT" with several endpoints, including "GET Post Products" which is currently selected. The main workspace shows a POST request configuration for the URL "localhost:8080/products". The "Authorization" tab is active, and a dropdown menu is open, highlighting the "Bearer Token" option. A warning message states: "Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)". The token value "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIxLG5hb..." is entered in the text field. The "Body" tab is selected at the bottom, showing a JSON response in "Pretty" format: 

```
{  "id": 1,  "name": "Product 1",  "price": 100}
```

. The status bar at the bottom right indicates "Status: 201 Created", "Time: 90 ms", and "Size: 420 B".

## Giới thiệu OAuth2

**OAuth2** là một phiên bản tiếp theo của giao thức OAuth (Open Authorization), là một tiêu chuẩn được thiết kế để cho phép một trang web (ứng dụng) truy cập vào tài nguyên được lưu trữ bởi các ứng dụng web khác thay mặt cho người dùng.

### **OAuth2 xác định có bốn vai trò chính:**

- 1) **Resource Owner** (Chủ sở hữu tài nguyên): Người dùng có tài khoản và tài nguyên trên dịch vụ (ví dụ: tài liệu Google Drive, thông tin cá nhân trên Facebook). Người dùng này sẽ cấp quyền cho ứng dụng để truy cập vào tài nguyên của họ.
- 2) **Client** (Ứng dụng khách): Ứng dụng muốn truy cập vào tài nguyên của người dùng. Ứng dụng này phải được người dùng cấp phép trước khi có thể truy cập vào tài nguyên.
- 3) **Authorization Server** (Máy chủ ủy quyền): Máy chủ chịu trách nhiệm xác thực người dùng và cấp mã thông báo (token) để ứng dụng có thể truy cập tài nguyên. Nó xử lý việc cấp mã ủy quyền (authorization code) và mã thông báo (token).
- 4) **Resource Server** (Máy chủ tài nguyên): Máy chủ lưu trữ tài nguyên mà ứng dụng muốn truy cập (ví dụ: máy chủ Google Drive, máy chủ Facebook API). Nó xác thực mã thông báo và cung cấp tài nguyên nếu mã thông báo hợp lệ.

# Luồng xác thực trong OAuth2

## Abstract Protocol Flow

