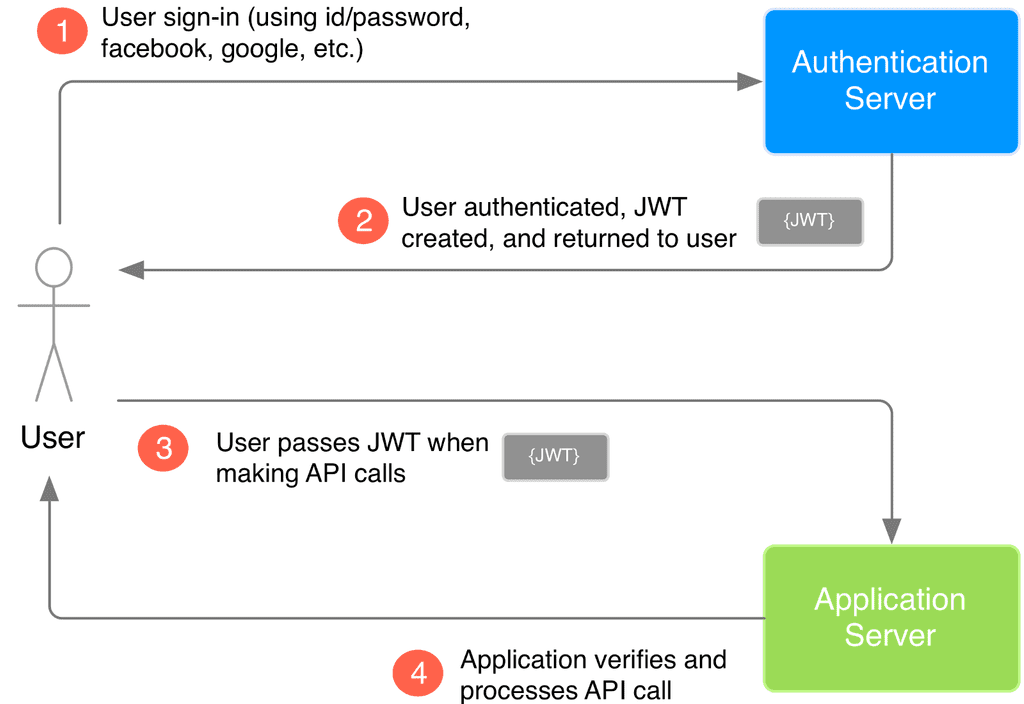
**Giới thiệu**

Xin chào các bạn, Trong hai bài trước tôi đã giới thiệu cách sử dụng **Spring Security** và kết nối với database để xác thực người dùng.

1. [Spring Security Cơ bản](https://loda.me/huong-dan-spring-security-co-ban-de-hieu-loda1556592373421)
2. [Spring Security + Hibernate](https://loda.me/huong-dan-spring-security-jpa-hibernate-loda1556631547828)

Trong bài hôm nay chúng ta sẽ tìm hiểu một phần cực kỳ quan trọng trong các hệ thống bảo mật ngày nay, đó là **JWT**.

**JWT (Json web Token)** là một chuỗi mã hóa được gửi kèm trong Header của client request có tác dụng giúp phía server xác thực request người dùng có hợp lệ hay không. Được sử dụng phổ biến trong các hệ thống API ngày nay.



**Cài đặt**

Chúng ta sử dụng Maven giống bài trước, tuy nhiên có update thêm thư viện io.jsonwebtoken.jwtt để giúp chúng ta mã hóa thông tin thành jwt

*pom.xml*

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.springframework</groupId>

<artifactId>spring-security-jwt</artifactId>

<version>0.1.0</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.5.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

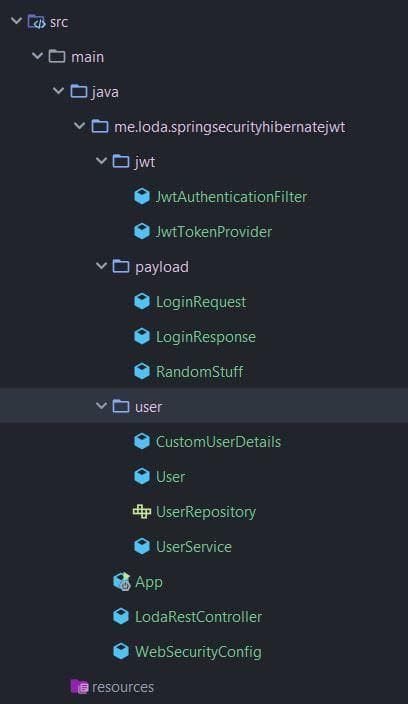
</plugin>

</plugins>

</build>

</project>

Cấu trúc thư mục code bao gồm:



**Implement**

Ban đầu, chúng ta sẽ tạo ra class User và UserDetails để giao tiếp với **Spring Security**. Phần này giống hệt với bài viết [Spring Security + Hibernate](https://loda.me/huong-dan-spring-security-jpa-hibernate-loda1556631547828)

Trong bài viết có sử dụng [Lombok](https://loda.me/general-huong-dan-su-dung-lombok-giup-code-java-nhanh-hon-69-loda1552789752787)

**Tạo User**

Tạo ra class User tham chiếu với database.

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Table;

import lombok.Data;

@Entity

@Table(name = "user")

@Data // lombok

public class User {

@Id

@GeneratedValue

private Long id;

@Column(nullable = false, unique = true)

private String username;

private String password;

}

Tạo UserRepository kế thừa JpaRepository để truy xuất thông tin từ database.

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

User findByUsername(String username);

}

**Tham chiếu User với UserDetails**

Mặc định **Spring Security** sử dụng một đối tượng UserDetails để chứa toàn bộ thông tin về người dùng. Vì vậy, chúng ta cần tạo ra một class mới giúp chuyển các thông tin của User thành UserDetails

*CustomUserDetails.java*

import org.springframework.security.core.GrantedAuthority;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.security.core.userdetails.UserDetails;

import lombok.AllArgsConstructor;

import lombok.Data;

@Data

@AllArgsConstructor

public class CustomUserDetails implements UserDetails {

User user;

@Override

public Collection<? extends GrantedAuthority> getAuthorities() {

// Mặc định mình sẽ để tất cả là ROLE\_USER. Để demo cho đơn giản.

return Collections.singleton(new SimpleGrantedAuthority("ROLE\_USER"));

}

@Override

public String getPassword() {

return user.getPassword();

}

@Override

public String getUsername() {

return user.getUsername();

}

@Override

public boolean isAccountNonExpired() {

return true;

}

@Override

public boolean isAccountNonLocked() {

return true;

}

@Override

public boolean isCredentialsNonExpired() {

return true;

}

@Override

public boolean isEnabled() {

return true;

}

}

Khi người dùng đăng nhập thì **Spring Security** sẽ cần lấy các thông tin UserDetails hiện có để kiểm tra. Vì vậy, bạn cần tạo ra một class kế thừa lớp UserDetailsService mà **Spring Security** cung cấp để làm nhiệm vụ này.

*UserService.java*

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.stereotype.Service;

@Service

public class UserService implements UserDetailsService {

@Autowired

private UserRepository userRepository;

@Override

public UserDetails loadUserByUsername(String username) {

// Kiểm tra xem user có tồn tại trong database không?

User user = userRepository.findByUsername(username);

if (user == null) {

throw new UsernameNotFoundException(username);

}

return new CustomUserDetails(user);

}

}

**JWT**

Sau khi có các thông tin về người dùng, chúng ta cần mã hóa thông tin người dùng thành chuỗi JWT. Tôi sẽ tạo ra một class JwtTokenProvider để làm nhiệm vụ này.

import org.springframework.stereotype.Component;

import io.jsonwebtoken.\*;

import lombok.extern.slf4j.Slf4j;

import me.loda.springsecurityhibernatejwt.user.CustomUserDetails;

@Component

@Slf4j

public class JwtTokenProvider {

// Đoạn JWT\_SECRET này là bí mật, chỉ có phía server biết

private final String JWT\_SECRET = "lodaaaaaa";

//Thời gian có hiệu lực của chuỗi jwt

private final long JWT\_EXPIRATION = 604800000L;

// Tạo ra jwt từ thông tin user

public String generateToken(CustomUserDetails userDetails) {

Date now = new Date();

Date expiryDate = new Date(now.getTime() + JWT\_EXPIRATION);

// Tạo chuỗi json web token từ id của user.

return Jwts.builder()

.setSubject(Long.toString(userDetails.getUser().getId()))

.setIssuedAt(now)

.setExpiration(expiryDate)

.signWith(SignatureAlgorithm.HS512, JWT\_SECRET)

.compact();

}

// Lấy thông tin user từ jwt

public Long getUserIdFromJWT(String token) {

Claims claims = Jwts.parser()

.setSigningKey(JWT\_SECRET)

.parseClaimsJws(token)

.getBody();

return Long.parseLong(claims.getSubject());

}

public boolean validateToken(String authToken) {

try {

Jwts.parser().setSigningKey(JWT\_SECRET).parseClaimsJws(authToken);

return true;

} catch (MalformedJwtException ex) {

log.error("Invalid JWT token");

} catch (ExpiredJwtException ex) {

log.error("Expired JWT token");

} catch (UnsupportedJwtException ex) {

log.error("Unsupported JWT token");

} catch (IllegalArgumentException ex) {

log.error("JWT claims string is empty.");

}

return false;

}

}

**Cấu hình và phân quyền**

Bây giờ, chúng ta bắt đầu cấu hình **Spring Security** bao gồm việc kích hoạt bằng @EnableWebSecurity.

Bước này giống với bài viết [Spring + Hibernate](https://loda.me/huong-dan-spring-security-jpa-hibernate-loda1556631547828) nên tôi sẽ không nói những phần lặp lại.

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.BeanIds;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import me.loda.springsecurityhibernatejwt.jwt.JwtAuthenticationFilter;

import me.loda.springsecurityhibernatejwt.user.UserService;

@EnableWebSecurity

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

UserService userService;

@Bean

public JwtAuthenticationFilter jwtAuthenticationFilter() {

return new JwtAuthenticationFilter();

}

@Bean(BeanIds.AUTHENTICATION\_MANAGER)

@Override

public AuthenticationManager authenticationManagerBean() throws Exception {

// Get AuthenticationManager bean

return super.authenticationManagerBean();

}

@Bean

public PasswordEncoder passwordEncoder() {

// Password encoder, để Spring Security sử dụng mã hóa mật khẩu người dùng

return new BCryptPasswordEncoder();

}

@Override

protected void configure(AuthenticationManagerBuilder auth)

throws Exception {

auth.userDetailsService(userService) // Cung cáp userservice cho spring security

.passwordEncoder(passwordEncoder()); // cung cấp password encoder

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.cors() // Ngăn chặn request từ một domain khác

.and()

.authorizeRequests()

.antMatchers("/api/login").permitAll() // Cho phép tất cả mọi người truy cập vào địa chỉ này

.anyRequest().authenticated(); // Tất cả các request khác đều cần phải xác thực mới được truy cập

// Thêm một lớp Filter kiểm tra jwt

http.addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);

}

}

Điểm khác biệt ở đây là sự xuất hiện của JwtAuthenticationFilter. Đây là một lớp Filter do tôi tự tạo ra.

JwtAuthenticationFilter Có nhiệm vụ kiểm tra request của người dùng trước khi nó tới đích. Nó sẽ lấy Header Authorization ra và kiểm tra xem chuỗi JWT người dùng gửi lên có hợp lệ không.

@Slf4j

public class JwtAuthenticationFilter extends OncePerRequestFilter {

@Autowired

private JwtTokenProvider tokenProvider;

@Autowired

private UserService customUserDetailsService;

@Override

protected void doFilterInternal(HttpServletRequest request,

HttpServletResponse response, FilterChain filterChain)

throws ServletException, IOException {

try {

// Lấy jwt từ request

String jwt = getJwtFromRequest(request);

if (StringUtils.hasText(jwt) && tokenProvider.validateToken(jwt)) {

// Lấy id user từ chuỗi jwt

Long userId = tokenProvider.getUserIdFromJWT(jwt);

// Lấy thông tin người dùng từ id

UserDetails userDetails = customUserDetailsService.loadUserById(userId);

if(userDetails != null) {

// Nếu người dùng hợp lệ, set thông tin cho Seturity Context

UsernamePasswordAuthenticationToken

authentication = new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());

authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authentication);

}

}

} catch (Exception ex) {

log.error("failed on set user authentication", ex);

}

filterChain.doFilter(request, response);

}

private String getJwtFromRequest(HttpServletRequest request) {

String bearerToken = request.getHeader("Authorization");

// Kiểm tra xem header Authorization có chứa thông tin jwt không

if (StringUtils.hasText(bearerToken) && bearerToken.startsWith("Bearer ")) {

return bearerToken.substring(7);

}

return null;

}

}

**Tạo Controller**

Vì phần này chúng ta làm việc với JWT, nên các request sẽ dưới dạng Rest API.

Tôi tạo ra 2 api:

1. /api/login: Cho phép request mà không cần xác thực.
2. /api/random: Là một api bất kỳ nào đó, phải xác thực mới lấy được thông tin.

@RestController

@RequestMapping("/api")

public class LodaRestController {

@Autowired

AuthenticationManager authenticationManager;

@Autowired

private JwtTokenProvider tokenProvider;

@PostMapping("/login")

public LoginResponse authenticateUser(@Valid @RequestBody LoginRequest loginRequest) {

// Xác thực từ username và password.

Authentication authentication = authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(

loginRequest.getUsername(),

loginRequest.getPassword()

)

);

// Nếu không xảy ra exception tức là thông tin hợp lệ

// Set thông tin authentication vào Security Context

SecurityContextHolder.getContext().setAuthentication(authentication);

// Trả về jwt cho người dùng.

String jwt = tokenProvider.generateToken((CustomUserDetails) authentication.getPrincipal());

return new LoginResponse(jwt);

}

// Api /api/random yêu cầu phải xác thực mới có thể request

@GetMapping("/random")

public RandomStuff randomStuff(){

return new RandomStuff("JWT Hợp lệ mới có thể thấy được message này");

}

}

**Tạo thông tin User trong database**

Trước hết bạn cần cấu hình cho hibernate kết tới tới h2 database trong file resources/appication.properties

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

# Enabling H2 Console

spring.h2.console.enabled=true

Để phục vụ demo, mỗi khi chạy chương trình, chúng ta cần insert một User vào database.

Có thể làm việc này bằng cách sử dụng CommandLineRunner. Một interface của Spring cung cấp, có tác dụng thực hiện một nhiệm vụ khi Spring khởi chạy lần đầu.

@SpringBootApplication

public class App implements CommandLineRunner {

public static void main(String[] args) {

SpringApplication.run(App.class, args);

}

@Autowired

UserRepository userRepository;

@Autowired

PasswordEncoder passwordEncoder;

@Override

public void run(String... args) throws Exception {

// Khi chương trình chạy

// Insert vào csdl một user.

User user = new User();

user.setUsername("loda");

user.setPassword(passwordEncoder.encode("loda"));

userRepository.save(user);

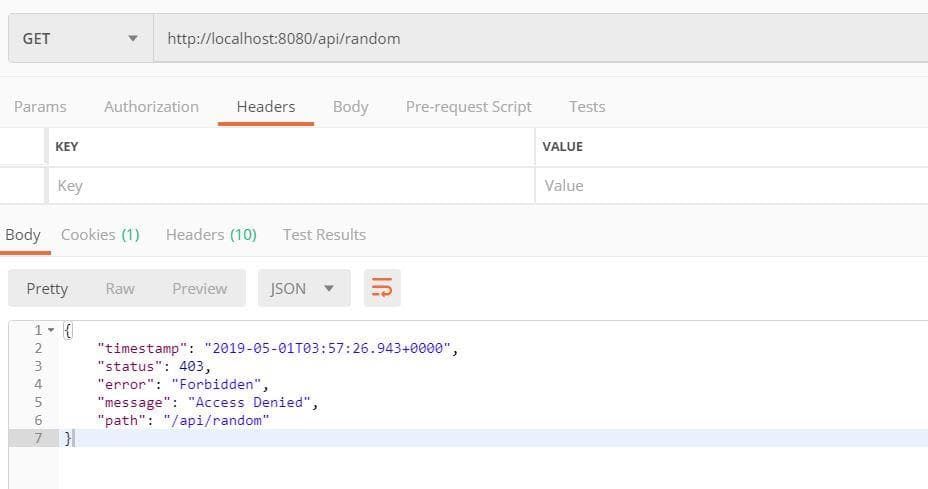
System.out.println(user);

}

}

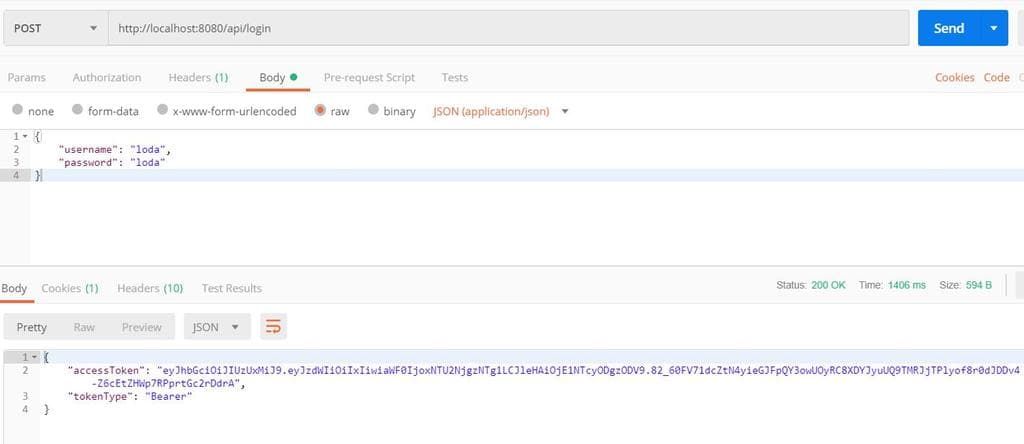
**Chạy thử**

Khi server on, chúng ta request thử tới địa chỉ <http://localhost:8080/api/random> mà không xác thực.

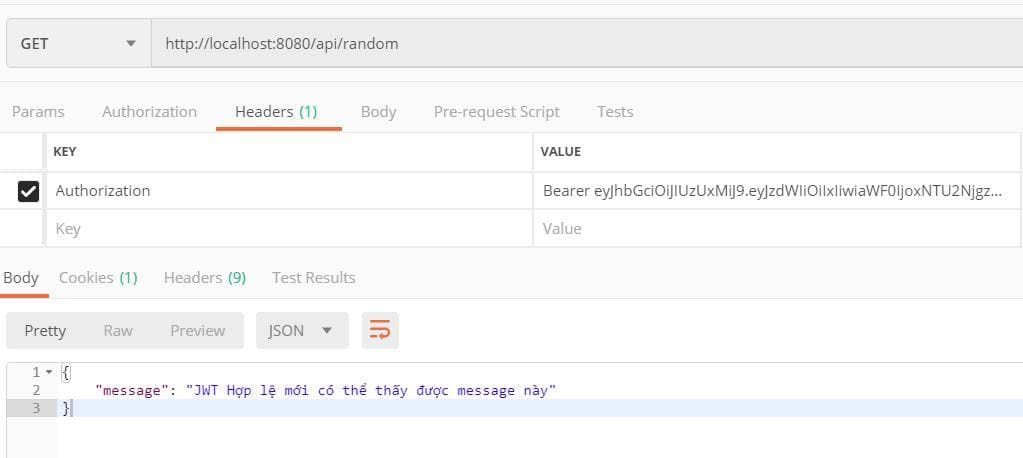


Kết quả trả về mã lỗi 403 kèm theo message Access Denied.

Để có thể request được, chúng ta đăng nhập vào hệ thống bằng api/login để lấy jwt.



Sau đó sử dụng thông tin jwt server trả về để thực hiện các request khác.



**Kết**

Trong bài này, chúng ta đã tìm hiểu cách sử dụng **Spring Security** và **JWT** để có thể xác thực người dùng trong các hệ thống Restful API. Chúng ta sẽ tìm hiểu các cách xác thực OAuth 2.0 ở các bài sau.