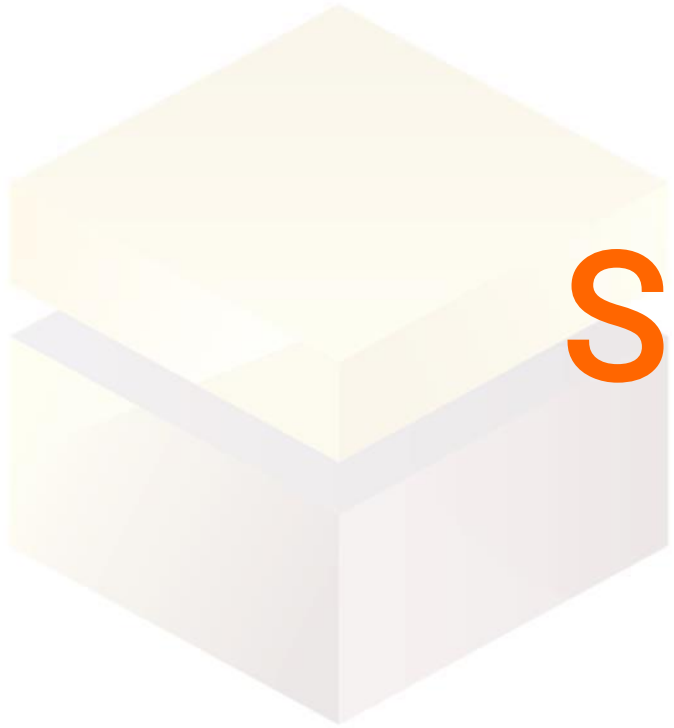


SECURITY API

CYBERSOFT.EDU.VN

Nội dung

- Tổng quan về CORS.
- Cấu hình CORS.
- Spring Sercurity.
- Json Web Token?
- Cấu trúc Json Web Token.
- Tạo token.
- Kiểm tra token.
- Các bước cấu hình Security Restful Api.

CORS

- **CORS**(Cross-origin resource sharing) là một cơ chế **cho phép các domain bên ngoài có thể truy cập vào tài nguyên trang web đó** (fonts, javascript, css,…).
- **CORS** được sinh ra là vì **same-origin policy**, một chính sách liên quan đến bảo mật được cài đặt vào toàn bộ các trình duyệt hiện nay. Chính sách này **ngăn chặn việc truy cập tài nguyên của các domain khác một cách vô tội vạ**.
- **CORS** sử dụng các **HTTP header** để "thông báo" cho trình duyệt rằng, một ứng dụng web chạy ở origin này có thể truy cập được các tài nguyên ở origin khác (domain khác).

Cấu hình Cors

```
@Configuration
public class AppConfig extends WebMvcConfigurerAdapter{

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**")
        .allowedOrigins("*")
        .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
        .allowCredentials(false)
        .maxAge(4800);
    }
}
```

CORS

- **allowOrigins():** Chỉ định những domain được phép truy cập. Nếu cấu hình Origins là (*) thì nó sẽ cho mọi domain có thể truy cập tài nguyên.
- **allowedMethods()**: Chỉ định những phương thức nào được phép, nếu sử dụng (*) thì tất cả các Http Method sẽ được truy cập. Theo mặc định, các phương thức GET, POST và HEAD sẽ được cho phép.
- **allowCredentials()**: Chỉ định cookie có được sử dụng hay không.
- **maxAge()**: Chỉ định thời gian request được lưu trong bộ nhớ đệm, nếu không cài đặt thì giá trị mặc định là 1800 (30 phút).

Spring Security

- Sercurity là gì?
- Sercurity Context.
- Đối tượng UserDetails.
- Lớp UserDetailsService.
- Các bước cấu hình Security.
- Đăng nhập trang quản trị, phân quyền.
- Đăng nhập trang người dùng, phân quyền.
- Viết chức năng đăng ký thành viên.

Spring Security là gì?

- **Spring security** là một framework (công cụ) **cung cấp và xử lý các vấn đề về xác thực** (authentication) **và phân quyền** (authorization) cho các ứng dụng web.
- **Spring security** sẽ **tự động tạo form đăng nhập**, **sau khi đăng nhập một đối tượng user sẽ được lưu trong session**, đối tượng user này sẽ gồm các thông tin như username, password, các quyền…
- **Spring Security** chống được các kỹ thuật hacking tinh vi:
  - ✓ **Session fixation:** Tấn công chiếm quyền điều khiển session của người dùng.
  - ✓ **Clickjacking:** Click chuột tự động (ví dụ click vào nút Like Facebook mà không xin phép người dùng).
  - ✓ **CSRF (Cross-site request forgery):** Tạo truy vấn (request) giả mạo truyền từ trang này sang trang khác.

Spring Security

❑ **Spring Security** cung cấp 2 cơ chế cơ bản:

✓ **Authentication**(Xác thực): Là tiến trình **xác thực** (kiểm tra) **danh tính của một người dùng hoặc một hệ thống** khác đang truy cập vào hệ thống bảo mật hiện tại.

  ✓ **Authentication** tương tác với người dùng thông qua form và xác thực dựa trên tên người dùng mà mật khẩu (password-based authentication).

✓ **Authorization** (Phân quyền): Là tiến trình **quyết định xem người dùng hoặc hệ thống sau khi xác thực có được quyền thực hiện một hành động nào đó trong ứng dụng** của bạn hay không.

❖ **Các hình thức phân quyền thường gặp:**

  ✓ **Role-based authorization**: Phân quyền dựa trên vai trò của người dùng.

  ✓ **Object-based authorization**: Phân quyền theo đối tượng.

SecurityContext

- **SecurityContext:** là interface cốt lõi của Spring Security, **lưu trữ tất cả các chi tiết liên quan đến bảo mật trong ứng dụng**.
- **SecurityContextHolder:** Lớp này **lưu trữ security context hiện tại của ứng dụng**, bao gồm chi tiết của **principal** đang tương tác với ứng dụng.
  - ✓ **Principal** có thể hiểu là **một người, một thiết bị hoặc một hệ thống nào đó** có thể thực hiện một hành động trong ứng dụng của bạn.

❖ Đoạn code dưới đây giúp lấy username của principal đã được xác thực:

```
Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();

if (principal instanceof UserDetails) {
    String username = ((UserDetails) principal).getUsername();
} else {
    String username = principal.toString();
}
```

UserDetails

- **UserDetails** là một interface cốt lõi của Spring Security. Nó **đại diện cho một principal** nhưng theo một cách mở rộng và cụ thể hơn.

✓ **getAuthorities():** trả về danh sách các quyền của người dùng.

✓ **getPassword():** trả về password đã dùng trong qúa trình xác thực.

✓ **getUsername():** trả về username đã dùng trong qúa trình xác thực.

✓ **isAccountNonExpired():** trả về true nếu tài khoản của người dùng chưa hết hạn.

✓ **isAccountNonLocked():** trả về true nếu người dùng chưa bị khóa.

✓ **isCredentialsNonExpired():** trả về true nếu chứng thực (mật khẩu) của người dùng chưa hết hạn.

✓ **isEnabled():** trả về true nếu người dùng đã được kích hoạt.

UserDetailsService

❑ UserDetailsService

❖ Là một interface có duy nhất một phương thức:

```
UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;
```

✓ Phương thức **loadUserByUsername()** sẽ trả về một implementation của **UserDetails**. Implementation ở đây có thể là:

- ✓ org.springframework.security.core.userdetails.User
- ✓ CustomUserDetails implements UserDetails.

❑ GrantedAuthority

✓ Là một **quyền được cấp cho principal**. Các quyền đều có tiền tố là **ROLE_**

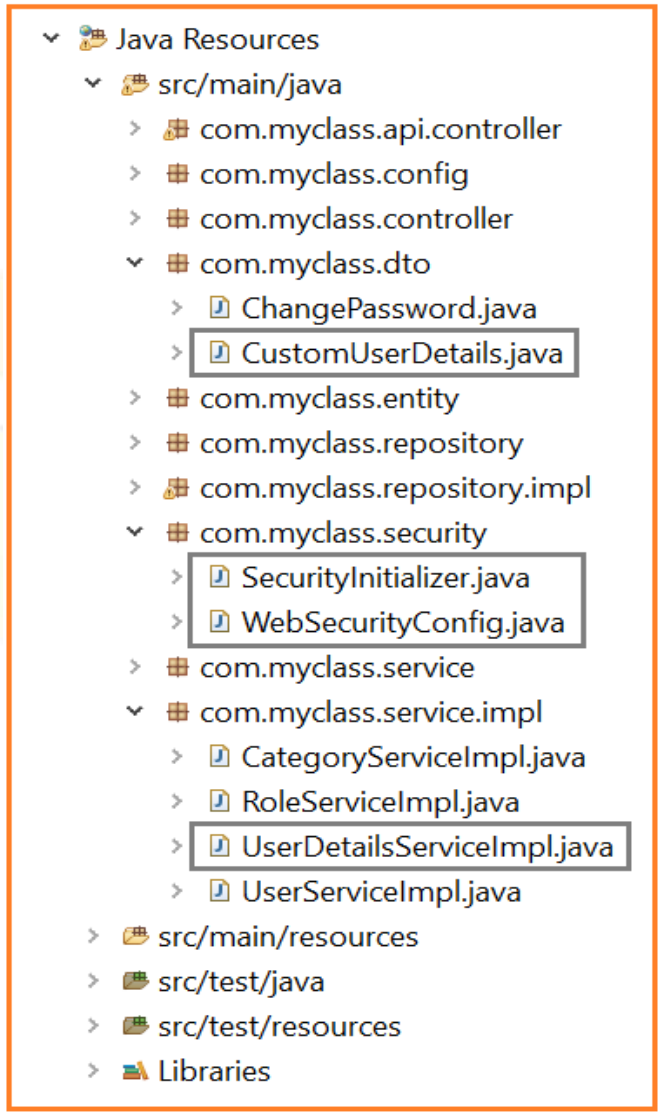
- ✓ Ví dụ: ROLE_ADMIN, ROLE_MEMBER,...

Các bước cấu hình AdminSecurity

- ✓ **Bước 1**: Tải thư viện **Spring Sercurity**.
- ✓ **Bước 2**: Định nghĩa hàm **findByEmail** trong tầng Repository.
- ✓ **Bước 3**: Tạo đối tượng **CustomUserDetails** implement từ **UserDetails** để thêm một số thuộc tính cho lớp UserDetails.
- ✓ **Bước 4**: Định nghĩa lớp **UserDetailsServiceImpl** imlement từ interface **UserDeatailsService** để load thông tin và quyền của người dùng.
- ✓ **Bước 5**: Tạo lớp **ApiSecurityConfig** kế thừa từ lớp **WebSecurityConfigurerAdapter** để cấu hình Security.
- ✓ **Bước 6**: Tạo lớp **SecurityInitializer** kế thừ từ lớp **AbstractSecurityWebApplicationInitializer**.
- ✓ **Bước 7**: Khai báo lớp **ApiSecurityConfig** vào DispatcherServlet.

Cấu trúc thư mục

Cấu trúc thư mục



Thư viện sử dụng

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>5.1.5.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>5.1.5.RELEASE</version>
</dependency>
```

UserRepository

Trả về một đối tượng User nếu như tìm thấy khớp email.

UserRepository

```
public User findByEmail(String email) {
    String hql = "FROM users WHERE email = :email";
    try {
        Session session = sessionFactory.getCurrentSession();
        Query<User> query = session.createQuery(hql, User.class);
        query.setParameter("email", email);
        return query.getSingleResult();
    }
    catch (HibernateException e) {
        e.printStackTrace();
    }
    return null;
}
```

CustomUserDetails

Mặc định Spring chỉ cung cấp cho UserDetails thuộc tính username, password và một danh sách chứa các quyền của người dùng để lưu thông tin vào Session.

```
public class CustomUserDetails extends User implements UserDetails{
    private static final long serialVersionUID = 1L;

    // Mặc định lớp User chỉ có username, password, role
    public CustomUserDetails(String username, String password,
                Collection<? extends GrantedAuthority> authorities) {
        super(username, password, authorities);
    }

    // Nếu muốn thêm thông tin để lưu vào session bạn có thể thêm ở đây
    // Ví dụ bạn có thể thêm fullname, avatar,...
}
```

Lớp CustomUserDetail mở rộng từ lớp User của Spring Sercurity cho phép thêm các thuộc tính vào nếu muốn.

UserDetailsService

```java
@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        // Lấy ra user có email giống với email người dùng gửi lên từ form đăng nhập
        User user = userRepository.findByEmail(email);
        if(user == null) throw new  UsernameNotFoundException("Không tìm thấy tài khoản!");

        // Tạo danh sách chứa tên quyền cả người dùng
        List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
        String roleName = user.getRole().getName(); // Lấy ra tên quyền
        authorities.add(new SimpleGrantedAuthority(roleName)); // Lưu vào danh sách

        // Trả về đối tượng chứa thông tin email, password và quyền
        return new CustomUserDetails(user.getEmail(), user.getPassword(), authorities);
    }

}
```

Lớp Service dùng để lấy ra thông tin tài khoản và quyền từ database sau đó gán vào cho đối tượng CustomUserDeails.

WebSecurityConfig

```java
@Configuration
@EnableWebSecurity
@ComponentScan("com.myclass")
public class ApiSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsService userDetailsService;

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

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        // TODO Auto-generated method stub
        return super.authenticationManagerBean();
    }

    // Khai báo service lấy thông tin user từ db và khai báo phương thức mã hóa password
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth
        .userDetailsService(userDetailsService)
        .passwordEncoder(passwordEncoder());
    }

    protected void configure(HttpSecurity http) throws Exception {
}
```

Bean PasswordEncoder dùng để giải mã mật khẩu (Sử dụng thư viện JBcrypt)

Bean sử dụng cho việc gọi hàm kiểm tra đăng nhập

Khai báo Service lấy thông tin user từ database và phương thức giải mã mật khẩu.

Phương thức cấu hình đăng nhập, phân quyền.

Cấu hình

SecurityInitializer

```
public class SecurityInitializer extends AbstractSecurityWebApplicationInitializer{
    // Lớp này chỉ cần kế thừa từ AbstractSecurityWebApplicationInitializer là được, không cần code.
}
```

DispatcherServlet

```
public class WebInitializer extends AbstractAnnotationConfigDispatcherServletInitializer{

    @Override
    protected Class<?>[] getRootConfigClasses() {
        // TODO Auto-generated method stub
        return new Class[] {
                HibernateConfig.class,
                SwaggerConfig.class,
                WebSecurityConfig.class
            };
    }

    protected Class<?>[] getServletConfigClasses() {

    protected String[] getServletMappings() {

    protected Filter[] getServletFilters() {
}
```

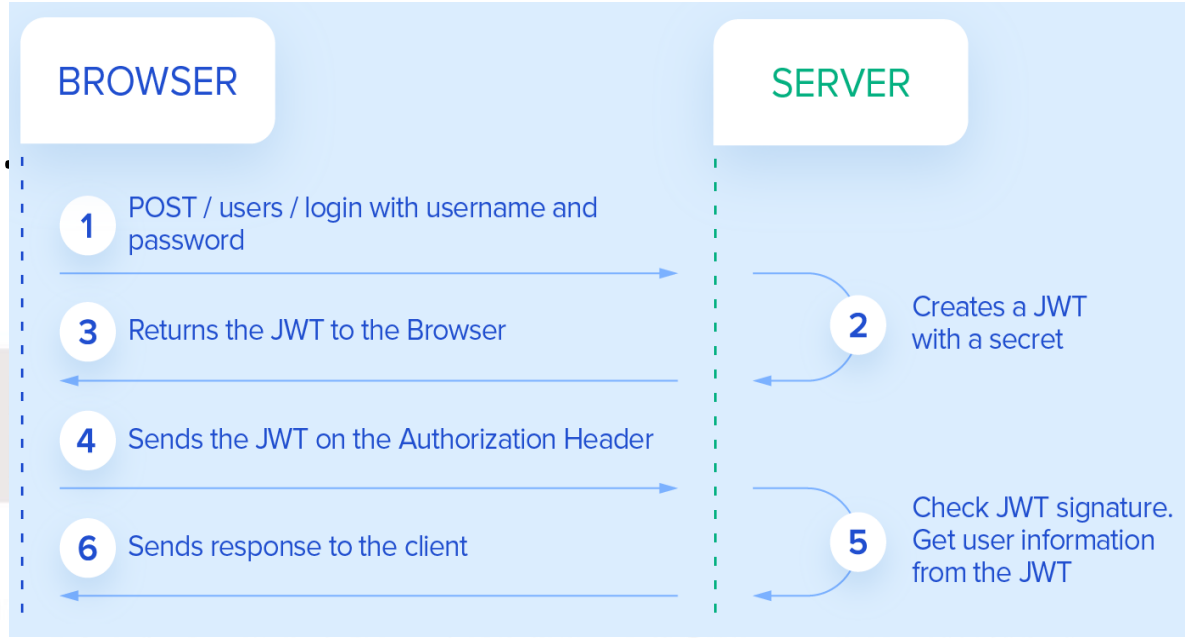
Khai báo lớp cấu hình Security.

Json Web Token

- **Json Web Token** (JWT) là một chuẩn để truyền tải thông tin một cách an toàn giữa các bên bằng một đối tượng Json.
- **Json Web Token** có kích thước nhỏ gọn do đó nó có thể được gửi qua Url, tham số POST hoặc bên trong tiêu đề HTTP.
- **Json Web Token** thường được dùng để xác thực người dùng (authentication), chuỗi JWT sẽ được gửi kèm trong phần header của request và server sẽ thông qua token đó để xác thực request.

Cách hoạt động

- ✓ **Bước 1**: Client gửi thông tin **username**, **password** để đăng nhập.
- ✓ **Bước 2 + 3**: Server xác nhận thông tin username, password để **tạo ra một chuỗi token** với thông tin cần thiết và gửi về cho client.



- ✓ **Bước 4**: Client thực hiện **gửi API kèm theo token vào trong header** của request.
- ✓ **Bước 5**: Server nhận được request sẽ lấy token trong header của request để **kiểm tra thông tin xác thực người dùng** và **trả dữ liệu về cho client**.

Cấu trúc JWT

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
lbWFpbCI6ImN5YmVyc29mdEBnbWFpbC5jb20iLCJ
uYW1lIjoiQ3liZXJzb2Z0In0.2AD4yKbuC_u-
TZ8MJJhIm2a4TMeM116LE4vHmObWFZk
```

Chuỗi token sau khi mã hóa

Chuỗi token có dạng: **header.payload.signature**

Signature được tạo ra bằng cách mã hóa **header** và **payload** bằng thuật toán **base64UrlEncode** sau đó mã hóa 2 chuỗi trên kèm theo **Secret** bằng thuật toán **HS256**.

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

Thuật toán mã hóa

Loại định dạng token

PAYLOAD: DATA

```
{
  "email": "cybersoft@gmail.com",
  "name": "Cybersoft"
}
```

Payload chứa thông tin muốn đặt trong token như emai, fullname, avatar

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  cyber
) secret base64 encoded
```

Khóa bí mật (secret)

Cấu trúc JWT

- **Header** bao gồm hai phần chính:
  - ✓Loại token (mặc định là **JWT**).
  - ✓Thuật toán đã dùng để mã hóa (**HMAC SHA256** - **HS256** hoặc **RSA**).
- **Payload**: Chứa **claims** (dữ liệu mà chúng ta muốn truyền đi như username, email, fullname,...), chứa các thông tin như **subject** (chủ đề), **issuer** (tổ chức phát hành token), **expired time** (ngày hết hạn).
- **Signature**: Là một chuỗi được mã hóa bởi **header**, **payload** cùng với một chuỗi bí mật (**secret**) theo nguyên tắc sau:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Thông tin trong Payload

❑ Một số thông tin thường đính kèm trong payload:

✓ **iss** (issuer): tổ chức phát hành token

✓ **sub** (subject): chủ đề của token

✓ **aud** (audience): đối tượng sử dụng token

✓ **exp** (expired time): thời điểm token sẽ hết hạn

✓ **nbf** (not before time): token sẽ chưa hợp lệ trước thời điểm này

✓ **iat** (issued at): thời điểm token được phát hành, tính theo UNIX time

✓ **jti**: JWT ID

Generate - Verify token

❑ Thư viện sử dụng

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.0</version>
</dependency>
```

❑ Tạo token

```
String jwt = Jwts.builder()
    .setClaims(claims)
    .setSubject(email)
    .setIssuedAt(new Date())
    .setExpiration(new Date(System.currentTimeMillis()+ JWT_EXPIRATION_TIME))
    .signWith(SignatureAlgorithm.HS256, JWT_SECRET)
    .compact();
```

Generate - Verify token

❑ Verify token

```
String result = Jwts.parser()
        .setSigningKey(SECRET)
        .parseClaimsJws(tokenHeader)
        .getBody()
        .getSubject();
```

- **SECRET**: Chuỗi ký tự bí mật sử dụng để tạo token.
- **tokenHeader**: Chuỗi token lấy từ header.

Cấu hình đăng nhập

❑ Các bước thực hiện

- ✓ **Bước 1**: Tạo lớp **LoginController**.
- ✓ **Bước 2**: Viết phương thức login, phương thức này có một tham số là đối tượng chứa thông tin email và mật khẩu sử dụng để đăng nhập.
- ✓ **Bước 3**: Thực hiện kiểm tra đăng nhập, sử dụng thư viện JJWT để tạo token trả về cho client.
- ✓ **Bước 4**: Cấu hình **WebSecurityConfig**.

LoginController

```java
@RestController
@RequestMapping("api")
public class ApiLoginController {

    @Autowired
    private AuthenticationManager authenticationManager;

    public ResponseEntity<String> login(@RequestBody UserLogin userLogin) {

    private String generateToken (Authentication authentication) {
}
```

LoginController

```java
@PostMapping("login")
public ResponseEntity<String> login(@RequestBody UserLogin userLogin) {
    Authentication authentication =  null;

    try {
        authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(userLogin.getEmail(), userLogin.getPassword()));

        SecurityContextHolder.getContext().setAuthentication(authentication);
        // Gọi phương thức tạo chuỗi token
        String token = generateToken(authentication);
        return new ResponseEntity<String>(token, HttpStatus.OK);
    }
    catch (AuthenticationException e) {
        e.printStackTrace();
    }

    return new ResponseEntity<String>("Sai tên đăng nhập hoặc mật khẩu", HttpStatus.BAD_REQUEST);
}
```

LoginController

```
private String generateToken (Authentication authentication) {
    // Đoạn JWT_SECRET này là bí mật, chỉ có phía server biết
    final String JWT_SECRET = "chuoi_bi_mat";
    //Thời gian có hiệu lực của chuỗi jwt (10 ngày)
    final long JWT_EXPIRATION = 864000000L;

    Date now = new Date();
    Date expiryDate = new Date(now.getTime() + JWT_EXPIRATION);

    UserDetails userDetails = (UserDetails) authentication.getPrincipal();
    // Tạo chuỗi json web token từ id của user.
    String token = Jwts.builder()
                .setSubject(userDetails.getUsername())
                .setIssuedAt(now)
                .setExpiration(expiryDate)
                .signWith(SignatureAlgorithm.HS512, JWT_SECRET)
                .compact();
    return token;
}
```

ApiSecurityConfig

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

    http.cors();

    // Cấu hình phân quyền
    http
    .csrf().disable()
    // Gặp link bắt đầu bằng /api/admin thì chạy các hàm bên dưới kiểm tra request
    .antMatcher("/api/admin/**")
    .authorizeRequests()
    // Gặp link /api/admin/login sẽ bỏ qua không kiểm tra
    .antMatchers("/api/admin/login")
    .permitAll()
    // Gặp link này chỉ cho quyền ADMIN và MANAGER truy cập
    .antMatchers("/api/admin/**")
    .hasAnyRole("ADMIN", "MANAGER")
    .anyRequest() // Những link còn lại phải yêu cầu đăng nhập trước
    .authenticated();

    // Không sử dụng session lưu thông tin đăng nhập
    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
}
```

Cấu hình phân quyền

❑ Các bước thực hiện

- ✓ **Bước 1**: Tạo filter **JWTAuthorizationFilter**: Lớp này cấu hình lấy token từ header trong mỗi request để kiểm tra đăng nhập.
- ✓ **Bước 2**: Triển khai phương thức doFilterInternal để lấy token từ header được gửi lên từ client.
- ✓ **Bước 3**: Giải mã token, lấy thông tin (email), sau đó xác nhận đăng nhập để phân quyền người dùng.
- ✓ **Bước 4**: Cấu hình **WebSecurityConfig**.

JWTAuthorizationFilter

```
public class JWTAuthorizationFilter extends BasicAuthenticationFilter {

    private UserDetailsService _userDetailsService;
    public JWTAuthorizationFilter(AuthenticationManager authenticationManager,
            UserDetailsService userDetailsService) {
        super(authenticationManager);
        _userDetailsService = userDetailsService;
    }

    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
}
```

JWTAuthorizationFilter

```java
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {

    final String JWT_SECRET = "chuoi_bi_mat";
    // Lấy chuỗi token từ header của request
    String tokenBearer = request.getHeader("Authorization");
    // Kiểm tra xem token đã đc đính kèm vào request chưa
    // và có đúng định dạng hay không ( token phải bắt đầu bằng Bearer )
    if(tokenBearer != null && tokenBearer.startsWith("Bearer ")) {
        // Thay thế "Bearer " bằng "" để lấy chuỗi token chính xác
        String token = tokenBearer.replace("Bearer ", "");
        // Giải mã token lấy email
        String email = Jwts.parser()
                .setSigningKey(JWT_SECRET)
                .parseClaimsJws(token)
                .getBody()
                .getSubject();
        // Lấy thông tin user từ database
        UserDetails userDetails = _userDetailsService.loadUserByUsername(email);
        // Nếu người dùng hợp lệ, set thông tin cho Seturity Context
        UsernamePasswordAuthenticationToken authenticationToken =
                new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());

        SecurityContextHolder.getContext().setAuthentication(authenticationToken);
    }
    chain.doFilter(request, response);
}
```

ApiSecurityConfig

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

    http.cors();

    // Cấu hình phân quyền
    http
    .csrf().disable()
    .antMatcher("/api/admin/**")
    .authorizeRequests()
    .antMatchers("/api/admin/login")
    .permitAll()
    .antMatchers("/api/admin/**")
    .hasAnyRole("ADMIN", "MANAGER")
    .anyRequest()
    .authenticated();

    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

    http.addFilter(new JWTAuthorizationFilter(authenticationManager(), userDetailsService));
}
```

Tổng kết

- ✓ Tổng quan về CORS.
- ✓ Cấu hình CORS.
- ✓ Json Web Token?
- ✓ Cấu trúc Json Web Token.
- ✓ Tạo token.
- ✓ Kiểm tra token.
- ✓ Các bước cấu hình Security Restful Api.