SECURITY API

CYBERSOFT.EDU.VN

Nội dung

- 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.

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

```java
@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);
    }
}
```

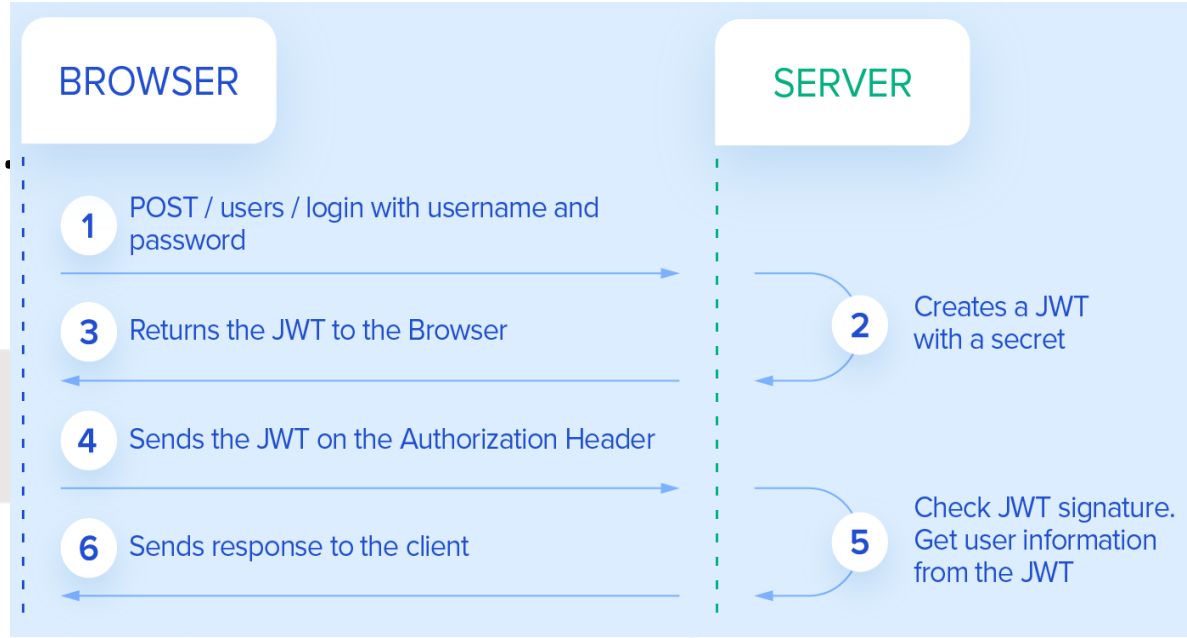
- **allowOrigins():** Chỉ định những domain được phép truy cập. Nếu cấu hình Origins là (*) thì nó sẽ cho mọi domain có thể truy cập tài nguyên.
- **allowedMethods()**: Chỉ định những phương thức nào được phép, nếu sử dụng (*) thì tất cả các Http Method sẽ được truy cập. Theo mặc định, các phương thức 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).

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 security

❑ Các bước thực hiện

- ✓ **Bước 1**: Tạo lớp **CustomUserDetails** (chứa thông tin user).
- ✓ **Bước 2**: Tạo lớp **UserDetailsServiceImpl** (lấy thông tin user + quyền của user), phương thức trả về một đối tượng CustomUserDetails.
- ✓ **Bước 3**: Tạo lớp **LoginController**: Lớp này cấu hình lấy thông tin đăng nhập từ request để đăng nhập và tạo token.
- ✓ **Bước 4**: Tạo lớp **JWTAuthorizationFilter**: Lớp này cấu hình lấy token từ header trong mỗi request để kiểm tra đăng nhập.
- ✓ **Bước 5**: Tạo lớp **WebSecurityConfig**.

CustomUserDetails

```java
public class CustomUserDetails extends User implements UserDetails{

    private static final long serialVersionUID = 1L;

    private String fullname;

    public CustomUserDetails(String username, String password,
            Collection<? extends GrantedAuthority> authorities) {
        super(username, password, authorities);
    }

    public String getFullname() {
        return fullname;
    }

    public void setFullname(String fullname) {
        this.fullname = fullname;
    }
}
```

UserDetailsServiceImpl

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

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {

        User user = userRepository.findByEmail(email);
        if(user == null) {
            throw new UsernameNotFoundException("Không tìm thấy thông tin!");
        }

        Collection<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
        authorities.add(new SimpleGrantedAuthority(user.getRole().getName()));

        CustomUserDetails userDetails = new CustomUserDetails(user.getEmail(), user.getPassword(), authorities);
        userDetails.setFullname(user.getFullname());

        return userDetails;
    }

}
```

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;
}
```

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

```java
@Configuration
@EnableWebSecurity
@ComponentScan("com.myclass")
@Order(2)
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
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {

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

Bean **AuthenticationManager** được sử dụng ở Controller để thực hiện chức năng đăng nhập.

ApiSecurityConfig

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

    http
    .csrf().disable()
    .antMatcher("/api/**")
    .authorizeRequests()
    .antMatchers("/api/login", "/api/register")
    .permitAll()
    .anyRequest()
    .authenticated();

    // Sử dụng JWTAuthorizationFilter để check token => Lấy thông tin người dùng
    http.addFilter(new JWTAuthorizationFilter(authenticationManager(), userDetailsService));
    // Cấu hình không sử dụng Session lưu thông tin client
    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
}
```

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.