Curso Presencial Programação Fullstack

Spring Security 🔐

Prof. MSc. Kelson | Senior Software Engineer

PROFKELSON.DEV

BORA CODAR?

PROFKELSON.DEV
BORA CODAR?

Introdução à Segurança em Aplicações Web

- Por que segurança é importante?
- Evita que usuários acessem dados de outros usuários
- Garante que apenas usuários autenticados possam acessar funcionalidades sensíveis
- Protege sua API contra acessos maliciosos

PROFKELSON.DEV
BORA CODAR?

Introdução à Segurança em Aplicações Web

🔐 Autenticação vs Autorização

✅ **Autenticação**
Processo de verificar quem é o usuário.

Exemplo: Login com username e senha.

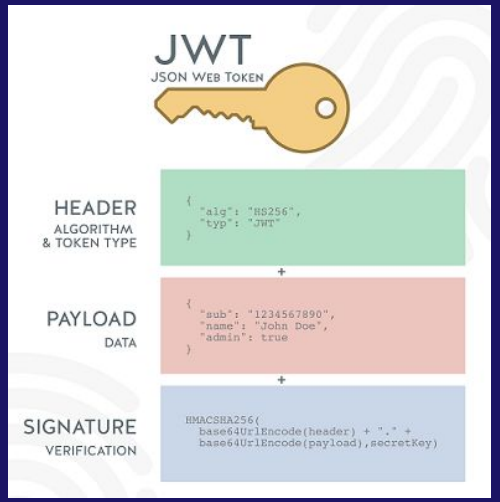
🔓 **Autorização**
Processo de verificar o que o usuário pode fazer.

Exemplo: Apenas usuários ADMIN podem excluir um produto.

🔐 O que é JWT (JSON Web Token)?

- Header: tipo do token (JWT), algoritmo (HS256)
- Payload: dados (username, role, validade, etc.)
- Signature: verificação se o token é autêntico

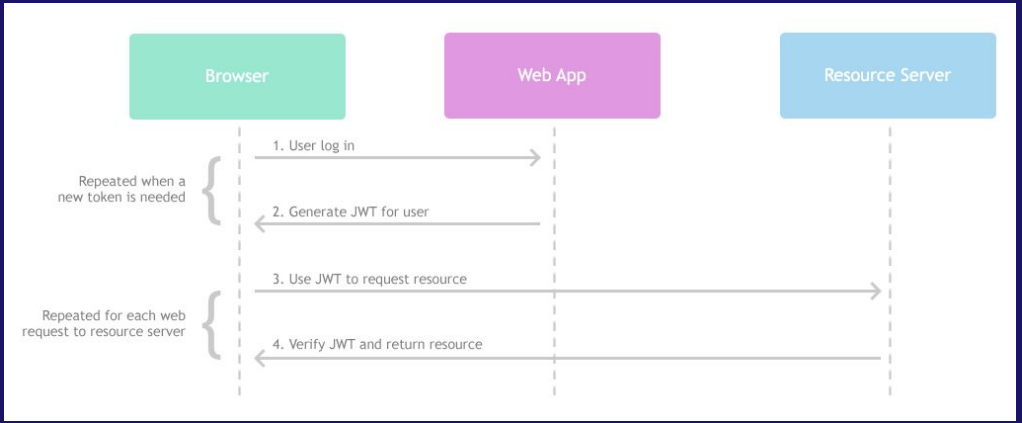
```
Header.Payload.Signature
```




```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 .
eyJzdWIiOiJhZG1pbiIsInJvbGUiOiJBRE1JTiIsImlhdCI6MTY5MzUxMjAwMCwiZXhwIjoxNjkzNTQ4MDAwfQ .
```

🔁 Como funciona o fluxo com JWT?

- Usuário faz login com username e senha
- O backend gera um token JWT
- O frontend guarda esse token (ex: localStorage)
- Em toda requisição futura, o token é enviado no header:
- O backend verifica o token e libera (ou bloqueia) o acesso




```
Authorization: Bearer <token>
```

O que o Spring Security faz?

É um módulo do Spring que cuida da autenticação e autorização de forma automática e segura.

- Valida usuários
- Bloqueia endpoints não autorizados
- Protege contra ataques (CSRF, brute force etc.)

🧩 Componentes principais no nosso projeto

- 1. Usuario

Classe do usuário com:

username, senha, email

Role (perfil: ADMIN ou USER)

```java
@Entity 6 usages
@Table(name = "usuarios")
@Getter @Setter
@NoArgsConstructor @AllArgsConstructor
@Builder
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true)
    private String username;

    private String email;

    private String senha;

    @Enumerated(EnumType.STRING)
    private Role role;
}
```

🧩 Componentes principais no nosso projeto

- 2. UsuarioService

Implementa UserDetailsService:

Busca o usuário no banco

Retorna um UserDetails (objeto que o Spring Security entende)

```java
@Service  no usages
@RequiredArgsConstructor
public class UsuarioService implements UserDetailsService {

    private final UsuarioRepository usuarioRepository;

    @Override  1 usage
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        return usuarioRepository.findByUsername(username) Optional<Usuario>
                .map( Usuario usuario -> User.builder()
                        .username(usuario.getUsername())
                        .password(usuario.getSenha())
                        .roles(usuario.getRole().name())
                        .build()) Optional<UserDetails>
                .orElseThrow(() -> new UsernameNotFoundException("Usuário não encontrado"));
    }
}
```

🧩 Componentes principais no nosso projeto

- 3. AuthService

register(): cria um novo usuário

login(): autentica e gera JWT

getUsuarioLogado(): retorna dados do usuário atual

```
@Service 2 usages
@RequiredArgsConstructor
public class AuthService {

    private final UsuarioRepository usuarioRepository;
    private final PasswordEncoder passwordEncoder;
    private final JwtService jwtService;
    private final AuthenticationManager authenticationManager;

    public AuthResponseDTO register(RegisterRequestDTO request) { 1 usage
        Usuario usuario = Usuario.builder()
                .username(request.username())
                .email(request.email())
                .senha(passwordEncoder.encode(request.senha()))
                .role(request.role())
                .build();
        usuarioRepository.save(usuario);

        // Cria UserDetails com a role
        UserDetails userDetails = User.builder()
                .username(usuario.getUsername())
```

🧩 Componentes principais no nosso projeto

- 4. JwtService

Gera tokens JWT

Extrai o username de um token

Valida se o token é válido e não expirado

```java
@Service 3 usages
public class JwtService {

    private final String SECRET_KEY = "sua-chave-super-secreta-de-256bits-para-teste"; 1 usage

    public String extractUsername(String token) { 2 usages
        return extractClaim(token, Claims::getSubject);
    }

    public <T> T extractClaim(String token, Function<Claims, T> resolver) { 2 usages
        final Claims claims = extractAllClaims(token);
        return resolver.apply(claims);
    }

    public String generateToken(UserDetails userDetails) { 2 usages
        return Jwts.builder()
                .setSubject(userDetails.getUsername())
                .claim( s: "roles", userDetails.getAuthorities()) // roles incluídos
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60 * 10)) // 10
                .signWith(getSignInKey(), SignatureAlgorithm.HS256)
                .compact();
```

🧩 Componentes principais no nosso projeto

- 6. SecurityConfig

Configura o SecurityFilterChain

Libera os endpoints públicos (/auth/**)

Define autenticação por token (sem sessão)

Adiciona o filtro JWT

```
19  @Configuration  no usages
20  @RequiredArgsConstructor
21  @EnableMethodSecurity // 👈 isso aqui ativa o uso de @PreAuthorize/@Secured
22  public class SecurityConfig {
23
24      private final JwtAuthenticationFilter jwtAuthFilter;
25      private final UsuarioService usuarioService;
26
27      @Bean  no usages
28  @   public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
29          return http
30                  .csrf(AbstractHttpConfigurer::disable)
31                  .authorizeHttpRequests( AuthorizationManagerRequestMat... auth -> auth
32                          .requestMatchers( ...patterns: "/auth/**").permitAll()
33                          .requestMatchers( ...patterns: "/v3/api-docs/**", "/swagger-ui/**", "/swagger-ui.
34                          .anyRequest().authenticated()
35                  )
36                  .sessionManagement( SessionManagementConfigurer<HttpSecurity> sess -> sess.sessionCreatio
37                  .authenticationProvider(authenticationProvider())
38                  .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
39                  .build();
40      }
```

🧩 Componentes principais no nosso projeto

- 6. SecurityConfig

Configura o SecurityFilterChain

Libera os endpoints públicos (/auth/**)

Define autenticação por token (sem sessão)

Adiciona o filtro JWT

```
19  @Configuration  no usages
20  @RequiredArgsConstructor
21  @EnableMethodSecurity // 👈 isso aqui ativa o uso de @PreAuthorize/@Secured
22  public class SecurityConfig {
23
24      private final JwtAuthenticationFilter jwtAuthFilter;
25      private final UsuarioService usuarioService;
26
27      @Bean  no usages
28  @   public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
29          return http
30                  .csrf(AbstractHttpConfigurer::disable)
31                  .authorizeHttpRequests( AuthorizationManagerRequestMat... auth -> auth
32                          .requestMatchers( ...patterns: "/auth/**").permitAll()
33                          .requestMatchers( ...patterns: "/v3/api-docs/**", "/swagger-ui/**", "/swagger-ui.
34                          .anyRequest().authenticated()
35                  )
36                  .sessionManagement( SessionManagementConfigurer<HttpSecurity> sess -> sess.sessionCreatio
37                  .authenticationProvider(authenticationProvider())
38                  .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
39                  .build();
40      }
```