

# Spring Security

Prof. Luiz Real

# Spring Security

- Spring Security é um framework que fornece autenticação, autorização e proteção contra ataques comuns.
- É a ferramenta padrão para proteger aplicativos baseados em Spring.



# Adicionando a dependência

- implementation 'org.springframework.boot:spring-boot-starter-security'

*build.gradle*

```
dependencies {  
    implementation "org.springframework.boot:spring-boot-starter-security"  
}
```

- Testando...
- Response code: 401 – Unauthorized
- <http://localhost:8080> (login)

# Configuração padrão

- Por padrão, o Spring Security gera uma camada de autenticação e gera uma senha aleatória de desenvolvimento cada vez que a aplicação sobe (verificar o log).
- Exemplo:
  - Using generated security password: 95bf4dc3-3800-495a-b6e6-bde674fdebda

# Criando entidade Usuario

@Entity

```
public class Usuario implements UserDetails {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.UUID)
```

```
    private String id;
```

```
    private String login;
```

```
    private String senha;
```

```
    private String role;
```

```
}
```

# Gerando os métodos do implements

@Override

```
public Collection<? extends GrantedAuthority> getAuthorities() {  
    return List.of();  
}
```

@Override

```
public String getPassword() {  
    return senha;  
}
```

@Override

```
public String getUsername() {  
    return login;  
}
```

@Override

```
public boolean isAccountNonExpired() {  
    return true;  
}
```

@Override

```
public boolean isAccountNonLocked() {  
    return true;  
}
```

@Override

```
public boolean isCredentialsNonExpired() {  
    return true;  
}
```

@Override

```
public boolean isEnabled() {  
    return true;  
}
```

# Criando enum UserRole

```
public enum UserRole {  
    ADMIN("admin"),  
    USER("user");  
  
    private String role;  
  
    UserRole(String role) { this.role = role; }  
  
    public String getRole() { return role; }  
}
```

# Mapeando roles na entidade

```
private UserRole role;
```

```
@Override
```

```
public Collection<? extends GrantedAuthority> getAuthorities() {  
    if (UserRole.ADMIN.equals(this.role)) {  
        return List.of(new SimpleGrantedAuthority("ROLE_ADMIN")  
            , new SimpleGrantedAuthority("ROLE_USER"));  
    } else {  
        return List.of(new SimpleGrantedAuthority("ROLE_USER"));  
    }  
}
```

A precedência é importante...  
O usuário ADMIN recebe as  
permissões dele e dos usuários  
abaixo dele.



# Criando repository UsuarioRepository

@Repository

public interface UsuarioRepository

extends JpaRepository<Usuario, String> {

UserDetails findByLogin(String login);

}

# Criando service AuthService

@Service

```
public class AuthService implements UserDetailsService {
```

    @Autowired

```
    UsuarioRepository usuarioRepository;
```

    @Override

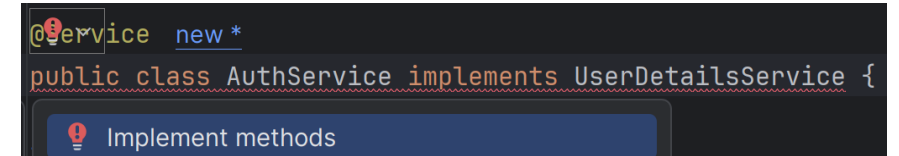
```
    public UserDetails loadUserByUsername(String username)
```

```
        throws UsernameNotFoundException {
```

```
        return usuarioRepository.findByLogin(username);
```

```
    }
```

```
}
```



# Sobrepondo configurações do Spring Security

- Criando a classe security.SecurityConfigurations
- Anotações que sobrepõem as configs do Spring

@Configuration

@EnableWebSecurity

```
public class SecurityConfigurations {  
    // ...  
}
```

# SecurityConfigurations

@Bean

```
public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws  
Exception {  
    return httpSecurity  
        // desativa uma config padrão de proteção  
        .csrf(csrf -> csrf.disable())  
        .sessionManagement(session -> session  
            // não armazena sessão de usuário  
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS))  
        .build();  
}  
// Testando... API voltou a responder
```

# Definindo requisições HTTP autorizadas

```
// define autorização de requisições HTTP
.authorizeHttpRequests(authorize -> authorize
    // restringindo o POST de /livros apenas a ADMIN
    .requestMatchers(HttpMethod.POST, "/livros").hasRole("ADMIN")
    // e liberando todos os outros métodos para qualquer usuário logado
    .anyRequest().authenticated()
)
.build();
// Testando... 403 - Forbidden
```

# Criando AuthController e método login

```
@RestController
@RequestMapping("/auth")
public class AuthController {
    @Autowired
    private AuthenticationManager authenticationManager; // erro de autowire

    @PostMapping("/login")
    public ResponseEntity login(@RequestBody @Valid AuthDTO authDTO) {
        // Gera um token do usuário e senha
        var usuarioSenha = new UsernamePasswordAuthenticationToken(authDTO.login(), authDTO.senha());
        // Autentica esse token
        var auth = this.authenticationManager.authenticate(usuarioSenha);
        return ResponseEntity.ok().build();
    }
}
```

# Criando AuthDTO

```
public record AuthDTO(String login, String senha) {}
```



# Adicionando recursos SecurityConfiguration

@Bean

```
public AuthenticationManager  
authenticationManager(AuthenticationConfiguration  
authenticationConfiguration) throws Exception {  
    return authenticationConfiguration.getAuthenticationManager();  
} // remove o erro de autowire no controller
```

@Bean

```
public PasswordEncoder passwordEncoder() {  
    return new BCryptPasswordEncoder();  
} // para trabalhar com hash da senha
```



# Criando método register

```
@PostMapping("/register")
public ResponseEntity register(@RequestBody @Valid RegisterDTO registerDTO) {
    if (usuarioRepository.findByLogin(registerDTO.login()) != null) {
        return ResponseEntity.badRequest().build();
    }
    String encryptedPassword = new
BCryptPasswordEncoder().encode(registerDTO.senha());
    Usuario novoUsuario = new Usuario(registerDTO.login(), encryptedPassword,
registerDTO.role());
    usuarioRepository.save(novoUsuario);
    return ResponseEntity.ok().build();
}
```

# Criando RegisterDTO

```
public record RegisterDTO(String login, String senha, UserRole role) {}
```

# Criando construtores da entidade

```
public Usuario() {}
```

```
public Usuario(String login, String senha, UserRole role) {  
    this.login = login;  
    this.senha = senha;  
    this.role = role;  
}
```

# Libera o login para todos os usuários

- No SecurityConfigurations:

```
.authorizeHttpRequests(authorize -> authorize
    // liberando o POST de /auth/login para todos
    .requestMatchers(HttpMethod.POST, "/auth/login").permitAll()
    // liberando o POST de /auth/register para todos
    .requestMatchers(HttpMethod.POST,
"/auth/register").permitAll()
```

- Testando... Registro e Login ok!

# JWT – JSON Web Tokens

- O JWT é um método aberto e padrão da indústria para representar reivindicações de forma segura entre duas partes
- <https://jwt.io/libraries?language=Java>
- Gradle
  - `implementation 'com.auth0:java-jwt:4.4.0'`

# Criando TokenService

@Service

```
public class TokenService {  
  
    @Value("${api.security.token.secret}")  
    private String secret;  
}
```

- E no application.properties, adicionamos a referência para nossa variável de ambiente JWT\_SECRET:

```
api.security.token.secret=${JWT_SECRET:my-secret-key}
```

# Criando TokenService

```
public String generateToken(Usuario usuario) {  
    try {  
        Algorithm algorithm = Algorithm.HMAC256(secret);  
        return JWT.create()  
            .withIssuer("auth-api")  
            .withSubject(usuario.getLogin())  
            .withExpiresAt(genExpirationDate())  
            .sign(algorithm);  
    } catch (JWTCreationException exception) {  
        throw new RuntimeException("Erro na geração de token", exception);  
    }  
}
```

# Criando TokenService

```
public String validateToken(String token) {  
    try {  
        Algorithm algorithm = Algorithm.HMAC256(secret);  
        return JWT.require(algorithm)  
            .withIssuer("auth-api")  
            .build()  
            .verify(token)  
            .getSubject();  
    } catch (JWTVerificationException exception) {  
        return "";  
    }  
}
```

# Criando TokenService

```
private Instant genExpirationDate() {  
    return LocalDateTime  
        .now()  
        .plusHours(2)  
        .toInstant(ZoneOffset.of("-03:00"));  
}
```



# SecurityConfigurations

- No final dos filtros:

```
.authorizeHttpRequests(  
    // ...  
)  
.addFilterBefore(securityFilter,  
    UsernamePasswordAuthenticationFilter.class)  
.build();
```

# Criando SecurityFilter

@Component

```
public class SecurityFilter extends OncePerRequestFilter {}
```

- Devemos implementar o método doFilterInternal

@Override

```
protected void doFilterInternal(HttpServletRequest request,  
    HttpServletResponse response, FilterChain filterChain) throws  
    ServletException, IOException {  
    var token = this.recoverToken(request);  
}
```

# Criando SecurityFilter

- Implementamos o método recoverToken

```
private String recoverToken(HttpServletRequest request) {  
    var authHeader = request.getHeader("Authorization");  
    if (authHeader == null) {  
        return null;  
    }  
    return authHeader.replace("Bearer ", "");  
}
```

# Criando SecurityFilter

- Injeção de dependências

@Autowired

private TokenService tokenService;

@Autowired

private UsuarioRepository usuarioRepository

# Criando SecurityFilter

- Continuando implementação do método doFilterInternal

```
if (token != null) {  
    var login = tokenService.validateToken(token);  
    UserDetails usuario = usuarioRepository.findByLogin(login);  
    var authentication = new UsernamePasswordAuthenticationToken(usuario, null,  
        usuario.getAuthorities());  
    SecurityContextHolder.getContext().setAuthentication(authentication);  
}  
filterChain.doFilter(request, response);
```

# SecurityConfigurations

- Injeção de dependência

@Autowired

```
private SecurityFilter securityFilter;
```

# AuthController

@Autowired

private TokenService tokenService;

// ...

var auth = authenticationManager.authenticate(usuarioSenha);

var token = tokenService.generateToken((Usuario) auth.getPrincipal());

return ResponseEntity.ok(new LoginResponseDTO(token));

# Criando LoginResponseDTO

```
public record LoginResponseDTO(String token) {}
```

- Testando...
- Criar um usuário ADMIN e um USER
- Fazer login e pegar o token
- Testar o uso dos endpoints GET e POST com os dois usuários
- Para passar o token, procurar o tipo Bearer e passar o prefixo Bearer
  - Postman: Header "Authorization", Param "Bearer `token`"



# OAuth2

- OAuth 2.0, que significa "Autorização Aberta", é um padrão projetado para permitir que um site ou aplicativo acesse recursos hospedados por outros aplicativos da web em nome de um usuário.
- <https://auth0.com/pt/intro-to-iam/what-is-oauth-2>

# OAuth2

- Gradle

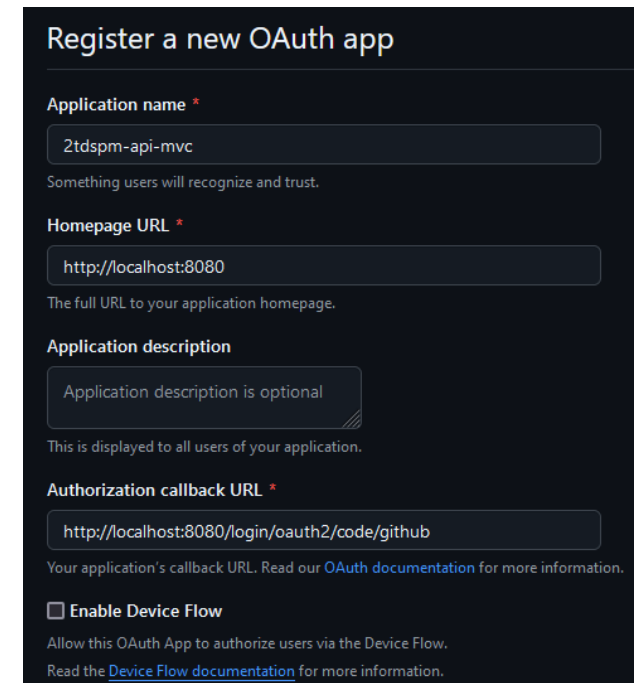
implementation 'org.springframework.boot:spring-boot-starter-oauth2-client'

implementation 'org.springframework.boot:spring-boot-starter-security'

implementation 'org.springframework.boot:spring-boot-starter-web'

# OAuth2

- Criando token de serviços para a aplicação
  - Github: <https://github.com/settings/applications/new>
  - Callback URL: <http://localhost:8080/login/oauth2/code/github>
  - Copiar Client ID e Client Secret para a aplicação



The screenshot shows the GitHub 'Register a new OAuth app' form. It includes fields for 'Application name', 'Homepage URL', 'Application description', and 'Authorization callback URL'. The 'Application name' field contains '2tdspm-api-mvc'. The 'Homepage URL' field contains 'http://localhost:8080'. The 'Application description' field is empty. The 'Authorization callback URL' field contains 'http://localhost:8080/login/oauth2/code/github'. There is also a checkbox for 'Enable Device Flow'.

**Register a new OAuth app**

**Application name \***

2tdspm-api-mvc

Something users will recognize and trust.

**Homepage URL \***

http://localhost:8080

The full URL to your application homepage.

**Application description**

Application description is optional

This is displayed to all users of your application.

**Authorization callback URL \***

http://localhost:8080/login/oauth2/code/github

Your application's callback URL. Read our [OAuth documentation](#) for more information.

☐ **Enable Device Flow**

Allow this OAuth App to authorize users via the Device Flow.  
Read the [Device Flow documentation](#) for more information.

# OAuth2

- Criando token de serviços para a aplicação
  - Google: <https://console.cloud.google.com/apis/credentials>
  - Configure consent screen (configurar)
  - Credentials, Create Credentials, OAuth Client ID
  - Application type: Web application
  - Authorized redirect URIs:
    - <http://localhost:8080/login/oauth2/code/google>
  - Copiar Client ID e Client Secret para a aplicação

The screenshot shows the 'Create OAuth client ID' page in the Google Cloud Console. At the top, the 'Name' field is filled with 'oauth\_exemplo'. Below it, a note states: 'The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.' A light blue information box contains a note: 'The domains of the URIs you add below will be automatically added to your OAuth consent screen as authorized domains.' The 'Authorized JavaScript origins' section has a sub-header with a help icon and the text 'For use with requests from a browser'. Below this is a '+ ADD URI' button. The 'Authorized redirect URIs' section also has a sub-header with a help icon and the text 'For use with requests from a web server'. Below this, the 'URIs 1 \*' field contains the URL 'http://localhost:8080/login/oauth2/code/google', with another '+ ADD URI' button below it. At the bottom, a note says 'Note: It may take 5 minutes to a few hours for settings to take effect'. Finally, there are 'SAVE' and 'CANCEL' buttons.

Name \*

oauth\_exemplo

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

**i** The domains of the URIs you add below will be automatically added to your [OAuth consent screen](#) as [authorized domains](#).

**Authorized JavaScript origins** ⓘ

For use with requests from a browser

+ ADD URI

**Authorized redirect URIs** ⓘ

For use with requests from a web server

URIs 1 \*

<http://localhost:8080/login/oauth2/code/google>

+ ADD URI

Note: It may take 5 minutes to a few hours for settings to take effect

SAVE CANCEL

# OAuth2

- <http://localhost:8080/secured>

Please sign in

Username

Password

Sign in

Login with OAuth 2.0

GitHub

Google