
FIAP

Domain Driven Design
using Java

AGENDA

1. Implementando segurança nas APIs

Protegendo Aplicações com Spring Security no Spring Boot

Prof. Evando Borges

O que é Spring Security?

- O que é Bean Validation?
  - É um **framework de segurança** do Spring usado para proteger aplicativos Java.
  - Oferece autenticação, autorização e defesa contra ataques.

Autenticação e Autorização

- Autenticação
  - Verifica a identidade do usuário (login e senha). Métodos suportados:
    - Banco de dados (JDBC, JPA).
    - LDAP (Active Directory);
    - OAuth2 (Google, Facebook);
    - JWT (JSON Web Token).
- Autorização
  - Define permissões para acesso (ADMIN, USER, GUEST).
  - Usa **roles** para permitir ou negar recursos.

Configuração do Spring Security

- Dependência Maven no pom.xml:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.11.5</version>
</dependency>
```

Como Spring Security Funciona?

- O Spring Security, por padrão, bloqueia todas as requisições e exige autenticação.
- Exemplo de usuário em memória:

```
@Bean
public UserDetailsService userDetailsService() {
    UserDetails user = User.withDefaultPasswordEncoder()
                           .username("admin")
                           .password("senha123")
                           .roles("ADMIN")
                           .build();
    return new InMemoryUserDetailsManager(user);
}
```

Configuração Personalizada

- Liberando Acesso a Algumas Rotas:

```
@Configuration  new *
@EnableWebSecurity
public class SecurityConfig {

    @Bean  new *
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .csrf( CsrfConfigurer<HttpSecurity> csrf -> csrf.disable())
                .authorizeHttpRequests( AuthorizationManagerRequestMat... auth -> auth
                        .requestMatchers(HttpMethod.POST, "/v1/auth/login").permitAll() // Login público
                        .anyRequest().authenticated() // Todas as demais requisições exigem autenticação
                )
                .sessionManagement( SessionManagementConfigurer<HttpSecurity> sess -> sess.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .addFilterBefore(new JwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }
}
```

Protegendo APIs REST com JWT

```java
public class JwtUtil {  3 usages  new *

    private static final SecretKey SECRET_KEY = Keys.hmacShaKeyFor("MinhaChaveSecretaSegura1234567890123456".getBytes(StandardCharsets.UTF_8));

    public static String gerarToken(String username, String role) {  1 usage  new *
        return Jwts.builder()
                .subject(username) // Mudança para `.subject()`
                .claim( s: "role", role)
                .expiration(new Date(System.currentTimeMillis() + 86400000)) // 24h
                .signWith(SECRET_KEY, Jwts.SIG.HS256) // Configuração correta
                .compact();
    }

    public static Claims validarToken(String token) {  1 usage  new *
        JwtParser parser = Jwts.parser().verifyWith(SECRET_KEY).build(); // Configuração correta
        return parser.parseSignedClaims(token).getPayload(); // Usa parseSignedClaims()
    }
}
```

Protegendo APIs REST com JWT

```java
public class JwtAuthenticationFilter extends GenericFilter {  1 usage  new *

    @Override  no usages  new *
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        String token = req.getHeader( s: "Authorization");

        if (token != null && token.startsWith("Bearer ")) {
            try {
                Claims claims = JwtUtil.validarToken(token.substring( beginIndex: 7));

                String role = claims.get( s: "role", String.class);
                Authentication auth = new UsernamePasswordAuthenticationToken(claims.getSubject(),  credentials: null, List.of(new SimpleGrantedAuthority( role: "ROLE_" + rol

                SecurityContextHolder.getContext().setAuthentication(auth);
            } catch (Exception e) {
                ((HttpServletResponse) response).sendError(HttpServletResponse.SC_FORBIDDEN,  s: "Token inválido ou expirado");
                return;
            }
        }

        chain.doFilter(request, response);
    }
}
```

Criando entidade Usuário

Prof. Evando Borges

Definição da Classe Entidade

```java
@Entity  new *
@Table(name = "USUARIO")
@Getter
@Setter
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID_USUARIO")
    private Integer id;

    @NotBlank(message = "O campo username é obrigatório.")
    @Size(max = 50, message = "O campo username deve ter no máximo 100 caracteres.")
    @Column(name = "USERNAME", nullable = false, length = 50)
    private String username;

    @NotBlank(message = "O campo password é obrigatório.")
    @Size(max = 50, message = "O campo password deve ter no máximo 100 caracteres.")
    @Column(name = "PASSWORD", nullable = false, length = 50)
    private String password;

    @NotBlank(message = "O campo role é obrigatório.")
    @Size(max = 50, message = "O campo role deve ter no máximo 100 caracteres.")
    @Column(name = "ROLE", nullable = false, length = 50)
    private String role; // ADMIN ou USER
}
```

Criando repositório

Prof. Evando Borges

Definição da Interface Repository

```java
import br.com.fiap.estoque.entities.Usuario;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;

public interface UsuarioRepository extends JpaRepository<Usuario, Integer> {  3 usages  new *
    Optional<Usuario> findByUsername(String username);  1 usage  new *
}
```

Criando o Serviço

Definição da Classe Service

```java
@Service  3 usages  new *
public class UsuarioService {
    private UsuarioRepository usuarioRepository;  3 usages

    public UsuarioService(UsuarioRepository usuarioRepository) {  new *
        this.usuarioRepository = usuarioRepository;
    }

    public Usuario buscarPorUsername(String username) {  1 usage  new *
        return usuarioRepository.findByUsername(username).orElseThrow(() ->
                new UsernameNotFoundException(username));
    }

    public Usuario salvar(Usuario usuario) {  no usages  new *
        return usuarioRepository.save(usuario);
    }
}
```

Criando o Controller

Definição da Classe Controller -> Autenticação

```java
@RestController  new *
@RequestMapping("/v1/auth")
public class AuthController {

    private final UsuarioService usuarioService;  2 usages

    public AuthController(UsuarioService usuarioService) {  new *
        this.usuarioService = usuarioService;
    }

    @PostMapping("/login")  new *
    public String login(@RequestParam String username, @RequestParam String password) {

        Usuario usuario = usuarioService.buscarPorUsername(username);
        if (usuario != null && usuario.getPassword().equals(password)) {
            return JwtUtil.gerarToken(usuario.getUsername(), usuario.getRole());
        } else {
            return "Credenciais inválidas!";
        }
    }
}
```

Definição da Classe Controller -> Usuário

```java
@RestController  new *
@RequestMapping("/v1/usuarios")
public class UsuarioController {

    private final UsuarioService usuarioService;  2 usages

    public UsuarioController(UsuarioService usuarioService) {  new *
        this.usuarioService = usuarioService;
    }

    @PostMapping  new *
    @ResponseStatus(HttpStatus.CREATED)
    @PreAuthorize("hasRole('ADMIN')")
    public void criarUsuario(@RequestBody Usuario usuario) {
        usuarioService.salvar(usuario);
    }
}
```

Modificando a Classe Controller -> Produto

```java
@RestController  new *
@RequestMapping("/v1/usuarios")
public class UsuarioController {

    private final UsuarioService usuarioService;  2 usages

    public UsuarioController(UsuarioService usuarioService) {  new *
        this.usuarioService = usuarioService;
    }

    @PostMapping  new *
    @ResponseStatus(HttpStatus.CREATED)
    @PreAuthorize("hasRole('ADMIN')")
    public void criarUsuario(@RequestBody Usuario usuario) {
        usuarioService.salvar(usuario);
    }
}
```

Referências

- **Documentação Oficial:** https://spring.io/projects/spring-data-jpa.
- **Documentação Oficial:** https://docs.spring.io/spring-security/reference/index.html
- Livro: *Spring in Action* - Craig Walls.


FIAP