# TI-610 Spring Framework

Antonio Carvalho - Treinamentos

# Spring Security + JWT

# O que é o JWT

- JSON Web Token é um sistema aberto que define um forma segura e auto contida para transmitir dados entre partes através de um objeto JSON. A informação pode ser verificada e assegurada devido a sua assinatura digital, por meio de uma chave secreta ou através de um par de certificados público/privado

# Componentes do JWT

- Cabeçalho, praticamente composto por duas partes:
  - O tipo do token normalmente “JWT”
  - Algoritmo usado
- Carga
  - As informações contidas na chave, estas informações são chamadas de Afirmações, em inglês Claims.
  - Há 3 tipos de Claims, Registradas, Públicas e Privadas
- Assinatura
  - É usada para verificar se a mensagem foi alterada no percurso.

# Implementação do JWT

- Adicionar a dependência no gradle.

```
implementation group:'io.jsonwebtoken', name:'jjwt', version:'0.9.1'
```

# Implementação do JWT

- Criar uma classe de utilidades para usar o JWT

```
@Service
public class JWTUtil {
// ...
    private static final String SECRET_KEY =
            "<Coloque aqui a chave secreta>";
    private static final long EXPIRATION_TIME =
                1000 * 60 * 60; // 1 hora
// ...
}
```

# Implementação do JWT

- Adicionar à classe JWTUtil as funções para **verificar o Token**

```
@Service
public class JwtUtil {
// ...
    private boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

   public Boolean validateToken(String token, UserDetails userDetails) {
        final String userName = extractUsername(token);
        return (userName.equals(userDetails.getUsername())
              && !isTokenExpired(token));
    }
// ...
}
```

# Implementação do JWT

- Adicionar à classe JWTUtil as funções para **extrair informações do Token**

```
@Service
public class JwtUtil {
// ...
    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    private <T> T extractClaim(String token,
                               Function<Claims, T> claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    private Claims extractAllClaims(String token) {
        return Jwts.parser()
                .setSigningKey(SECRET_KEY)
                .parseClaimsJws(token)
                .getBody();
    }
// ...
}
```

# Implementação do JWT

- Adicionar à classe JWTUtil as funções para **gerar o Token**

```
@Service
public class JwtUtil {
// ...
    public String generateToken(UserDetails userDetails) {
        Map<String, Object> claims = new HashMap<>();
        return createToken(claims, userDetails.getUsername());
    }

    private String createToken(Map<String, Object> claims, String subject) {
        return Jwts.builder()
                .setClaims(claims)
                .setSubject(subject)
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis()
                    + EXPIRATION_TIME))
                .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                .compact();
    }
// ...
}
```

# Implementação do JWT

- Criar uma classe para receber o usuário e senha no corpo do request do /login

```
public class UsuarioCredenciais {
    private String usuario;
    private String senha;

    public String getUsuario() {
        return usuario;
    }
    public void setUsuario(String usuario) {
        this.usuario = usuario;
    }

    public String getSenha() {
        return senha;
    }
    public void setSenha(String senha) {
        this.senha = senha;
    }
}
```

# Implementação do JWT

- Criar controller mapeado para /login de forma a receber o usuário, a senha e gerar o Token

```
@RestController
public class UsuarioController {
    @Autowired
    AuthenticationManager authenticationManager;

    @Autowired
    UsuarioService usuarioService;

    @Autowired
    JwtUtil jwtUtil;

    @RequestMapping(
            value = "/login",    method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON_VALUE)
    public String login(@RequestBody UsuarioCredenciais usuarioCredenciais) throws Exception {
        authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(usuarioCredenciais.getUsuario(),
                        usuarioCredenciais.getSenha())
        );
        final UserDetails userDetails = usuarioService
                .loadUserByUsername(usuarioCredenciais.getUsuario());
        final String token = jwtUtil.generateToken(userDetails);
        return "{\"token\":\"" + token + "\"}";
    }
}
```

# Implementação do JWT

- Definir um bean para produzir o AuthenticationManager

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    @Bean
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
}
```

# Implementação do JWT

- Definir um filtro para evitar verificar quais requisições possuem o token

```
@Component
public class JwtRequestFilter extends OncePerRequestFilter {
    @Autowired
    private JwtUtil jwtUtil;
    @Autowired
    private UsuarioService usuarioService;
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {
        final String authorizationHeader = request.getHeader("Authorization");
        String username = null;
        String token = null;
        if (authorizationHeader != null && authorizationHeader.startsWith("Bearer")) {
            token = authorizationHeader.substring(7);
            username = jwtUtil.extractUsername(token);
        }
        if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = this.usuarioService.loadUserByUsername(username);
            if (jwtUtil.validateToken(token, userDetails)) {
                UsernamePasswordAuthenticationToken usernamePasswordAuth =
                        new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
                usernamePasswordAuth.setDetails(
                        new WebAuthenticationDetailsSource().buildDetails(request) );
                SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuth);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

# Implementação do JWT

- Inserir o filtro na classe SecurityConfig no método configure

```
@Autowired JWTRequestFilter jwtRequestFilter;

@Override
protected void configure(HttpSecurity http) throws Exception
{
   http.authorizeRequests()
   .antMatchers("/livro").hasRole("USER")
   .and()
   .sessionManagement()
   .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
   ...
   .and().csrf().disable();
   ...
   http.addFilterBefore(jwtRequestFilter,
   UsernamePasswordAuthenticationFilter.class);
}
```