



## Segurança orientada a componentes - Spring Security

Nas versões mais atualizadas do Spring, a classe de configuração **WebSecurityConfigurerAdapter** foi depreciada. Essa depreciação aconteceu para que as configurações de segurança fossem **orientadas a Componentes**.

Com **componentes**, queremos dizer **Beans** (nesse caso, específicos do Spring Security), a segurando do **HttpSecurity** pode ser substituída pelo componente **SecurityFilterChain**:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .httpBasic()
        .and()
        .authorizeHttpRequests()
        .antMatchers(HttpMethod.GET, "/parking-spot/**").permitAll()
        .antMatchers(HttpMethod.POST, "/parking-spot").hasRole(RoleName.ROLE_USER.name())
        .antMatchers(HttpMethod.DELETE, "/parking-spot").hasRole(RoleName.ROLE_ADMIN.name())
        .anyRequest().authenticated()
        .and()
        .csrf().disable();
}
```

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .antMatchers(HttpMethod.POST, "/api/users/register").permitAll()
        .antMatchers(HttpMethod.POST, "/api/users/login").permitAll()
        .anyRequest().authenticated()
        .and().cors().configure(HttpSecurity)
        .and().csrf().disable()
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and().addFilterBefore(new AuthenticationTokenFilter(tokenService, usuarioRepository),
            UsernamePasswordAuthenticationFilter.class);

    return http.build();
}
```

Não são

comparativos diretos.

A **autenticação injetada** agora **não se faz mais necessária até determinado ponto**. Isso porque o Spring já tem no contexto tanto o PasswordEncoder quanto a classe de UserDetailsService, a partir daí ele consegue identificar todo o contexto para autenticação do usuário.

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsServiceImpl userDetailsService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
    }
}
```



```
@EnableWebSecurity
@Configuration
public class SecurityConfiguration {

    @Autowired
    private AuthenticationLoginService autenticaoService;

    @Autowired
    private UserRepository usuarioRepository;

    @Autowired
    private TokenService tokenService;

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers(HttpMethod.POST, "/api/users/register").permitAll()
            .antMatchers(HttpMethod.POST, "/api/users/login").permitAll()
            .anyRequest().authenticated()
            .and().cors().configure(HttpSecurity)
            .and().csrf().disable()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and().addFilterBefore(new AuthenticationTokenFilter(tokenService, usuarioRepository),
                UsernamePasswordAuthenticationFilter.class);

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



E para ignorar arquivos estáticos poderíamos usar um **componente de WebSecurityCustomizer** como alternativa do **WebSecurity**:

```
@Override
public void configure(WebSecurity web) throws Exception {
}
```

```
@Bean
public WebSecurityCustomizer webSecurityCustomizer() {
    return (web) -> web.ignoring().antMatchers("/**.html", "/v2/api-docs", "/webjars/**",
        "/configuration/**", "/swagger-resources/**");
}
```