TI-610 Spring Framework

Antonio Carvalho - Treinamentos

Spring Security

O que é o Spring Security

- O Spring Security consiste em um framework do universo Spring voltado a prover funcionalidades para autenticação e autorização para aplicações corporativas.

Como funciona o Spring Security

- Para utilizar o Spring Security é preciso baixar os seguintes módulos:

  SpringSecurity-Core

  SpringSecurity-Config

  SpringSecurity-Web

  Os arquivos destes módulos podem ser baixados do Maven Repository

  https://mvnrepository.com/artifact/org.springframework.security

Configuração do Spring Security

- Primeiro será preciso criar uma classe para informar ao Spring que haverá um filtro para todas as solicitações. Esta classe deve herdar de **AbstractSecurityWebApplicationInitializer**

```
public class SecurityWebApplicationInitializer
extends AbstractSecurityWebApplicationInitializer {
}
```

Configuração do Spring Security

- Um arquivo de configuração do Spring Security deverá ser criado herdando de `WebSecurityConfigurerAdapter`

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends
WebSecurityConfigurerAdapter {
}
```

Configuração do Spring Security

- Será preciso definir o Algoritmo para Criptografia da senha, por meio de uma função que retorne uma interface **`PasswordEncoder`**

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

- Os tipos de algoritmo são:
  - `BCryptPasswordEncoder` `bcrypt`
  - `NoOpPasswordEncoder` `<sem criptografia>`
  - `Pbkdf2PasswordEncoder` `pbkdf2`
  - `SCryptPasswordEncoder` `scrypt`
  - `StandardPasswordEncoder` `sha256`

Configuração do Spring Security

- Nesta classe de configuração do Spring Security serão criados os métodos **`configure()`** um para definir os usuários, senhas e roles

```
@Override
public void configure(AuthenticationManagerBuilder auth) {
    try {
        auth.inMemoryAuthentication()
        .withUser("user")
        .password(passwordEncoder().encode("user"))
        .roles("USER");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Configuração do Spring Security

- **Ou por meio de uma classe que implemente a interface `UserDetailService`**

```
@Autowired
UserDetailService userDetailService;

@Override
public void configure(AuthenticationManagerBuilder auth) {
    try {
        auth.userDetailsService(userDetailsService)
        .passwordEncoder(passwordEncoder());
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Configuração do Spring Security

- **Criar a entidade User**

```
@Entity
public class User {
    private long id;
    private String username;
    private String password;
    private String role;

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    ...
```

Configuração do Spring Security

- **Criar o Repositorio para acessar o User no banco**

```
@Repository
public interface UserRepository extends
CrudRepository<User, Long>{

   @Query("SELECT u FROM User u WHERE u.username =
:user")
   User findUserByUserName(@Param("user")String
user);

}
```

Configuração do Spring Security

- **Criar o Wrapper para envolver o objeto User**

```
public class UserDetailsImpl implements UserDetails {
	private User user;

	public UserDetailsImpl(User u) {
		this.user = u;
	}

	@Override
	public Collection<? extends GrantedAuthority> getAuthorities() {
		GrantedAuthority ga = new SimpleGrantedAuthority(user.getRole());
		return Arrays.asList(ga);
	}

	@Override
	public String getPassword() {
		return user.getPassword();
	}

	@Override
	public String getUsername() {
		return user.getUsername();
	}

	@Override
	public boolean isAccountNonExpired() {
		return true;
	}

	@Override
	public boolean isAccountNonLocked() {
		return true;
	}

	@Override
	public boolean isCredentialsNonExpired() {
		return true;
	}

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

Configuração do Spring Security

- **Criar o UserDetailsServiceImplementation**

```
@Service
public class UserDetailsServiceImpl implements
UserDetailsService {
    @Autowired
    UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String user)
          throws UsernameNotFoundException {
       User u = userRepository.findUserByUserName(user);
       if (u == null) {
          throw new UsernameNotFoundException("Usuário não
encontrado");
       }
       UserDetails ud = new UserDetailsImpl( u );
       return ud;
    }
}
```

Configuração do Spring Security

- **Ou por meio de uma classe que implemente a interface `UserDetailsService`**

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService {
@Override
    public UserDetails loadUserByUsername(String userName) throws UsernameNotFoundException {
        AppUser appUser = this.appUserDAO.findUserAccount(userName);
        if (appUser == null) {
            System.out.printf("Usuário não encontrado %s\n", userName);
            throw new UsernameNotFoundException(
                  String.format("Usuário %s não foi encontrado", userName));
        }
        // [ROLE_USER, ROLE_ADMIN,..]
        List<String> roleNames = this.appRoleDAO.getRoleNames(appUser.getUserId());
        List<GrantedAuthority> grantList = new ArrayList<GrantedAuthority>();
        if (roleNames != null) {
            for (String role : roleNames) {
                // ROLE_USER, ROLE_ADMIN,..
                GrantedAuthority authority = new SimpleGrantedAuthority(role);
                grantList.add(authority);
            }
        }
        UserDetails userDetails = (UserDetails) new User(appUser.getUserName(), //
                appUser.getEncrytedPassword(), grantList);
        return userDetails;
    }
}
```

Configuração do Spring Security

- Resta definir o modo de autenticação que pode ser **`httpBasic()`** ou **`formLogin()`**

Configuração do Spring Security

- E depois será sobrescrito o método **`configure()`** :

```
@Override
protected void configure(HttpSecurity http) throws Exception
{
    http.authorizeRequests()
    .antMatchers("/livro").hasRole("USER")
    .antMatchers("/admin").hasRole("MANAGER")
    .antMatchers("/*").authenticated().and()
    .formLogin()
    .usernameParameter("username")
    .passwordParameter("password")
    .loginProcessingUrl("/loginController")
    .loginPage("/login")
    .failureUrl("/loginFail").defaultSuccessUrl("/principal")
    .permitAll()
    .and()
    .logout().logoutUrl("/logout").logoutSuccessUrl("/login")
    .and().csrf().disable();
}
```

Como funciona o Spring Security

- O objeto do tipo `ExpressionInterceptUrlRegistry` obtido por meio do método `authorizeRequests()` possui as seguintes propriedades
  - `.antMatchers()` – Permite a seleção de algumas URLs por meio de Regex
  - `.anyRequest()` – Refere-se a todas as URLs

- Estes métodos retornam objetos do tipo `AuthorizedUrl` que possuem os seguintes métodos
  - `.anonymous()` – Indica que qualquer usuário sem autenticação pode acessar a URL
  - `.authenticated()` – Indica que apenas os usuários autenticados podem acessar esta URL
  - `.denyAll()` - Esta URL será proibida para todos
  - `.hasAuthority()` - Apenas os usuários contendo a autoridade informada poderá acessar esta URL
  - `.hasRole()` - Apenas os usuários contendo a role informada poderá acessar esta URL
  - `.permitAll()` – Todos os usuários estarão autorizados a acessar aquela URL

Como funciona o Spring Security

- O objeto do tipo `FormLoginConfigurer<HttpSecurity>` obtido por meio do método `formLogin()` possui as seguintes propriedades

  `.loginPage()` – O nome da página ou controller customizado para fazer Login

  `.loginProcessingUrl()` – A URL para onde serão submetidos o **username** e **password**

  `.defaultSuccessUrl()` – A URL de destino para quando o login ocorrer com **sucesso**

  `.failureUrl()` – A URL de destino para quando o login resultar em **erro**

  `.usernameParameter()` - Nome do parâmetro que conterá o username

  `.passwordParameter()` - Nome do parâmetro que conterá a senha

Configuração do Spring Security

- Precisa fazer uma classe de Controller:

```
@Controller
public class LoginController {
    @GetMapping("/login")
    public String login() {
        return "login";
    }
    @GetMapping("/loginFail")
    public ModelAndView loginFail() {
        ModelAndView mv = new ModelAndView("login");
        mv.addObject("ERRO", "Usuario ou senha inválidos");
        return mv;
    }
    @PostMapping("/loginController")
    public String loginController() {
        return "livro";
    }
    @GetMapping("/logout")
    public String logout() {
        return "login";
    }
}
```

Configuração do Spring Security

- Precisa fazer a pagina login.html:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymleaf.org">
<head>
      <meta charset="ISO-8859-1">
      <title>Sistema de Livros</title>
      <link rel="stylesheet" th:href="@{'/resources/css/bootstrap.min.css'}"/>
      <script src="https://code.jquery.com/jquery-3.5.1.min.js" integrity="sha256-
9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0=" crossorigin="anonymous"></script>
</head>
<body>
      <h1>Acesso ao Sistema</h1>
      <div class="container">
            <form th:action="@{/loginController}" method="POST">
                  <div th:if="${ERRO}"  class="alert alert-danger" role="alert">
                        <h4 class="alert-heading">Erro</h4>
                        <p>Erro de autenticação</p>
                        <hr>
                        <span class="mb-0" th:text="${ERRO}"></span>
                  </div>
                  <div class="form-group">
                        <label>User Name</label>
                        <input type="text" class="form-control" name="username"/>
                  </div>
                  <div class="form-group">
                        <label>Password</label>
                        <input type="password" class="form-control" name="password"/>
                  </div>
                  <div class="form-group">
                        <button class="btn btn-primary"
                              name="cmd" value="login">Logar</button>
                        <button class="btn btn-primary"
                              name="cmd" value="registrar">Registrar</button>
                  </div>
            </form>
      </div>
</body>
</html>
```

Configuração do Spring Security

- A configuração do WebInitializer precisará ser modificada

```
public class WebInitializer
    extends AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class<?>[]{
            Config.class, SecurityConfig.class};
    }
    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] {};
    }
    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
```

Bibliografia

COSMINA, I.; HARROP, R.; SCHAEFER, C.; HO, C.; Pro Spring 5 - an in-depth guide to the Spring Framework and its tools, 5ª edição, Apress, 2017