

# Personalizar accesos a URLs por ROLES

```
@Configuration
@EnableWebSecurity
public class DatabaseWebSecurity {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        // . . . .

        // Asignar permisos a URLs por ROLES
        .requestMatchers("/vacantes/**").hasAnyAuthority("SUPERVISOR", "ADMINISTRADOR")
        .requestMatchers("/categorias/**").hasAnyAuthority("SUPERVISOR", "ADMINISTRADOR")
        .requestMatchers("/usuarios/**").hasAnyAuthority("ADMINISTRADOR")

        // . . . .

    }
}
```

# Thymeleaf – sec:authorize="hasAnyAuthority('ANYROLE')"

En nuestras vistas podemos renderizar contenido HTML, dependiendo del ROL que tenga asignado un usuario. Para esto se usa la expresión:

```
sec:authorize="hasAnyAuthority('SUPERVISOR','ADMINISTRADOR')"
```

Nota: La expresión se aplica a un elemento HTML.

**Un ejemplo práctico en nuestra aplicación sería para renderizar las opciones del menú dependiendo del ROL del usuario:**

```
<li class="nav-item" sec:authorize="hasAnyAuthority('SUPERVISOR','ADMINISTRADOR')">  
    <a class="nav-link" th:href="@{/vacantes/indexPaginate}">Vacantes</a>  
</li>
```

# Thymeleaf – Expresiones comunes.

- Renderizar atributos almacenados en la sesión.

✓ **th:text="\${session.nombreAtributo}"**

Ejemplo:

```
<span th:if="${session.usuario != null}" class="text-light" th:text="'Bienvenido ' + ${session.usuario.nombre}"></span>&nbsp;
```

- Renderizar en la vista el nombre del usuario que inicio sesión.

✓ **sec:authentication="name"**

Ejemplo:

```
<span class="text-light">Bienvenido</span>&nbsp;<span class="text-light" sec:authentication="name"></span>
```

- Renderizar un elemento HTML para usuarios anónimos (usuarios que no han iniciado sesión).

✓ **sec:authorize="isAnonymous()"**

Ejemplo: Renderizar el botón Ingresar y Registrarse solo para usuarios anónimos.

```
<div sec:authorize="isAnonymous()">
  <a class="btn btn-primary" th:href="@{/index}">Ingresar</a>&nbsp;
  <a class="btn btn-primary" th:href="@{/signup}">Registrarse</a>
</div>
```

- Renderizar un elemento HTML para usuarios autenticados (usuarios que ya han iniciado sesión).

✓ **sec:authorize="isAuthenticated()"**

Ejemplo: Renderizar el nombre del usuario y el botón Salir solo para usuarios autenticados.

```
<div sec:authorize="isAuthenticated()">
  <span class="text-light">Bienvenido</span>&nbsp;<span class="text-light" sec:authentication="name"></span>&nbsp;
  <a class="btn btn-primary" th:href="@{/logout}">Salir</a>
</div>
```

# Encriptar contraseñas con Spring Security

- En nuestra aplicación hasta este momento tenemos guardadas las contraseñas de los usuarios en texto plano.

id	nombre	email	username	password	estatus	fechaRegistro
2	Luis Esparza Gomez	luis@itinajero.net	luis	{noop}luis123	1	2019-06-10
3	Marisol Salinas Rodarte	marisol@itinajero.net	marisol	{noop}mari123	1	2019-06-10

- ✓ En entornos de desarrollo se puede usar.

- ✓ **En producción esto es muy riesgoso.**

- P. ¿Cuál es la solución a este problema de seguridad?

- ✓ R. Guardar las contraseñas de los usuarios encriptadas.

id	nombre	email	username	password	estatus	fechaRegistro
2	Luis Esparza Gomez	luis@itinajero.net	luis	\$2y\$12\$yiRBBPu.vFoQL35r4ZMIpuD9DqbegjDr74jel4FJ47XN46ppMgrQu	1	2019-06-10
3	Marisol Salinas Rodarte	marisol@itinajero.net	marisol	\$2y\$12\$d2G8U1Tdrq3zxSppuqXh2O0tp6MCuXIC2pTDtjf2OWuP6G.BKscI	1	2019-06-10

- ✓ De esta forma, suponiendo que nuestra base de datos estuviera en poder de usuarios ajenos a nuestra organización, sería mucho más difícil **(casí imposible)** conocer las contraseñas de los usuarios.

# Encriptar contraseñas con el algoritmo bcrypt

Spring Security recomienda usar el algoritmo **bcrypt** para encriptar passwords.

<https://docs.spring.io/spring-security/site/docs/current/reference/html/ns-config.html#ns-password-encoder>

## ➤ Ventajas:

- ✓ Ejecuta un hashing one-way (solo se encripta, pero no se puede desencriptar).
- ✓ Fácil de configurar en nuestra aplicación web.

## ➤ Configuración: declarar un Spring Bean de tipo **PasswordEncoder**

- ✓ Como implementación podemos usar la clase **BCryptPasswordEncoder**

```
@Configuration
@EnableWebSecurity
public class DatabaseWebSecurity {

    // . . .

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

}
```

# ¿Cómo encriptar passwords en nuestra webapp?

## Controlador

```
@Controller
@RequestMapping("/usuarios")
public class UsuariosController {

    @Autowired
    private PasswordEncoder passwordEncoder;

    @GetMapping("/demo-bcrypt")
    public String pruebaBcrypt() {
        String password = "mari123";
        String encriptado = passwordEncoder.encode(password);
        System.out.println("Password encriptado: " + encriptado);
        return "usuarios/demo";
    }
}
```

# Estructura del formulario de login personalizado.

```
<form name="form" th:action="@{/login}" method="POST">
    Usuario:<br>
    <input type="text" name="username" ><br>
    Password:<br>
    <input type="password" name="password"><br>
    <input name="submit" type="submit" value="Ingresar" >
</form>
```

## Explicación:

- **th:action="@{/login}"**: URL del atributo action del formulario por defecto.
- **method="POST"**: Por seguridad, nunca poner GET.
- **name="username"**: El nombre del input tiene que ser **username** porque así lo espera Spring Security.
- **name="password"**: El nombre del input tiene que ser **password** porque así lo espera Spring Security.
- **type="password"**: Para que no sea visible el password al ingresarse.
- **type="submit"**: El botón tiene que ser tipo SUBMIT. No dejarlo por ejemplo type="button".

**NOTA:** Se puede agregar cualquier código HTML, CSS e IMAGENES para darle formato. En nuestro caso utilizaremos el diseño del archivo formLogin.html que viene en la carpeta de la plantilla del proyecto.

# Configurar formulario de login personalizado.

```
@Configuration
@EnableWebSecurity
public class DatabaseWebSecurity {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        // . . . .

        // El formulario de Login no requiere autenticacion
        http.formLogin(form -> form.loginPage("/login").permitAll());

        // . . . .

    }
}
```

Spring Security

```
@GetMapping("/login")
public String mostrarLogin() {
    return "formLogin";
}
```

HomeController

Archivo HTML (view) con el  
formulario HTML de login.



# Controlador para cerrar la sesión

El objeto request es necesario para que Spring obtenga la sesión actual para invalidarla.

Implementación de Spring Security encargada de destruir la sesión.

## Controller

```
@GetMapping("/logout")
public String logout(HttpServletRequest request){

    SecurityContextLogoutHandler logoutHandler =
    new SecurityContextLogoutHandler();

    logoutHandler.logout(request, null, null);

    return "redirect:/login";
}
```

Después de cerrar sesión redireccionamos al formulario de login.

### Method Detail

#### logout

```
public void logout(javax.servlet.http.HttpServletRequest request,
    javax.servlet.http.HttpServletResponse response,
    Authentication authentication)
```

Requires the request to be passed in.

#### Specified by:

logout in interface LogoutHandler

#### Parameters:

request - from which to obtain a HTTP session (cannot be null)

response - not used (can be null)

authentication - not used (can be null)

Documentación Oficial.

<https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/web/authentication/logout/SecurityContextLogoutHandler.html>

# Notificación de usuario/contraseña incorrectos (1).

- Por defecto cuando es proporcionado un usuario/contraseña incorrectos Spring Security realiza un REDIRECCIONAMIENTO a la URL del formulario de login y le pasa un parámetro llamado error. Ejemplo:  
`http://localhost:8080/login?error`
- Este parámetro puede ser usado en la vista del formulario de login para mostrar una notificación al usuario.

```
<form name='form' th:action="@{/login}" method='POST'>
```

```
<div th:if="${param.error}">
```

```
<span class="text-danger">Usuario / Password incorrectos!</span>
```

```
</div>
```

```
</form>
```