

# HACIENDO UNA APLICACIÓN WEB SEGURA



# SEGURIDAD CON SPRING = SPRING SECURITY

- ▶ Ofrece servicios de seguridad a aplicaciones Java EE.
- ▶ Integración muy sencilla con proyectos Spring MVC a través de Spring Boot.
- ▶ Dos procesos:
  - ▶ **Autenticación:** ¿quién eres?
  - ▶ **Autorización:** ¿para qué tienes permiso?

# SPRING SECURITY EN NUESTRO POM.XML

```
<dependency>  
<groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-security</artifactId>  
</dependency>
```



“

*Learning by doing*: vamos a aprender a utilizar Spring Security utilizándolo en nuestro proyecto.

# PASO 1: CONFIGURACIÓN DE LA SEGURIDAD

- ▶ Paquete *seguridad*
- ▶ Extiende a *WebSecurityConfigurerAdapter*
- ▶ Anotada con *@EnableWebSecurity*

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

}
```

## PASO 2: CONFIGURAR EL MÉTODO DE AUTENTICACIÓN

- ▶ Autenticación: *¿tú quién eres?*
- ▶ En este primer ejemplo lo hacemos en memoria.
- ▶ Tampoco codificamos las contraseñas.
- ▶ Un solo usuario *admin/admin/admin*.

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth
        .inMemoryAuthentication()
        .passwordEncoder(NoOpPasswordEncoder.getInstance())
        .withUser("admin")
        .password("admin")
        .roles("ADMIN");
}
```

# HASTA AHORA...

Solo con este código ya tenemos

- ▶ Requisito de autenticación en todas las URLs
- ▶ Generación de un formulario de *login*
- ▶ Permitir el acceso a un usuario con Username *admin* y Password *admin*
- ▶ Permitir hacer logout
- ▶ Prevención de ataques CSRF, entre otras (*Session Fixation*, *X-XSS-Protection*, *Clickjacking*,...)

## HASTA AHORA...

- ▶ Integración con métodos del api Servlet (HttpServletRequest)
  - ▶ *getRemoteUser()*
  - ▶ *getUserPrincipal()*
  - ▶ *isUserInRole(...)*
  - ▶ *login(...)*
  - ▶ *logout()*



## ADEMÁS...

- ▶ La clase de configuración ha registrado un filtro de Servlet especial, llamado *springSecurityFilterChain*.
- ▶ Es una cadena de filtros responsable de toda la seguridad (proteger las URLs, validar usuario y contraseña, redirigir al formulario de login, ...)

## PASO 3: AUTORIZACIÓN

- ▶ **Autorización:** *¿para qué tienes permiso?*
- ▶ Lo configuramos mediante un objeto de tipo *HttpSecurity*.
- ▶ Uso de method chain.

```
@Override  
protected void configure(HttpSecurity http) throws Exception {  
  
}
```

## PASO 4: LOGIN

- ▶ Modificamos lo necesario para
  - ▶ Cambiar el formulario de login y hacer uno propio
  - ▶ Cambiar la ruta a */login*
  - ▶ Permitir que cualquiera acceda al formulario

## PASO 4: LOGIN

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .anyRequest().authenticated()
            .and()
        .formLogin()
            .loginPage("/login")
            .permitAll();
}
```

## PASO 4: LOGIN

- ▶ Para customizar el formulario, necesitaríamos un controlador que nos lleve a la plantilla de *login*.
- ▶ Si un controlador tan solo sirve para llevarnos a la plantilla, lo podemos simplificar.

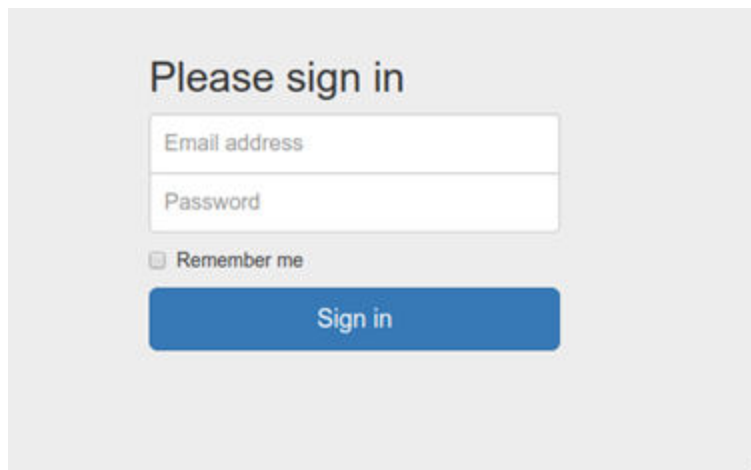
## PASO 4: LOGIN

- ▶ Creamos una clase de configuración, que implementa *WebMvcConfigurer*.
- ▶ Podemos mapear una ruta a una plantilla.

```
@Override  
public void addViewControllers(ViewControllerRegistry registry) {  
    registry.addViewController("/login");  
}
```

## PASO 4: LOGIN (PLANTILLA)

- ▶ Creamos una nueva plantilla (*login.html*)
- ▶ Nos podemos basar en otro ejemplo de bootstrap.



Please sign in

Email address

Password

☐ Remember me

Sign in

## PASO 4: LOGIN (PLANTILLA)

Para que funcione

- ▶ Apuntamos a las webjars
- ▶ Modificamos el formulario de login
  - ▶ **th:action="@{/login}"**
  - ▶ Los campos deben tener **name="username"** y **name="password"**.



## PASO 5: AUTORIZAR PETICIONES

- ▶ Nuestra seguridad es tan restrictiva que no permite *servir* ni los estilos `css`.
- ▶ Añadimos rutas para las que permitiremos el acceso siempre.
- ▶ Lo hacemos mediante el método `antMatchers(...)`, que acepta un *varargs* de `String` con rutas (podemos usar *glob*).

## PASO 6: LOGOUT

- ▶ Permitimos que un usuario autenticado realice el logout para
  - ▶ Invalidar la sesión
  - ▶ Limpiar el contexto de seguridad (*SecurityContextHolder*)
  - ▶ Redirigir al login (*/login?logout*)
- ▶ Añadimos los elementos necesarios a la plantilla.

## PASO 6: LOGOUT

¡**ATENCIÓN!** Desde las últimas versiones de Spring, la petición logout es POST (no GET).

- ▶ **Motivo:** uso de CSRF
- ▶ Solución:
  - ▶ Deshabilitar CSRF (no recomendado)
  - ▶ Realizar una petición POST (fácil)

## PASO 6: LOGOUT

- ▶ De esta forma, no tenemos que deshabilitar CSRF, y aprovechamos los estilos de bootstrap sin modificaciones.

```
<ul class="nav navbar-nav navbar-right">
  <li class="dropdown">
    <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" a
    <ul class="dropdown-menu">
      <li>
        <a href="javascript:document.getElementById('logoutForm').submit()">Salir</a>
      </li>
    </ul>
  </li>
</ul>
</div><!--/.nav-collapse -->
</div>
</nav>
<form th:action="@{/logout}" method="POST" id="logoutForm">
</form>
```

## PASO 7: MOSTRAR EL NOMBRE DEL USUARIO LOGUEADO

- ▶ Usual que podamos ver los datos del usuario logueado.
- ▶ Lo podemos sacar del objeto SecurityContextHolder y pasarlo a través del modelo.
- ▶ **O podemos tomarlo directamente en la plantilla.**

## PASO 7: MOSTRAR EL NOMBRE DEL USUARIO LOGUEADO

- ▶ [Thymeleaf](http://www.thymeleaf.org) ofrece una librería de extras con Spring Security (v. 3, 4 y 5).
- ▶ Añadimos un nuevo espacio de nombres:

<html

xmlns:th="http://www.thymeleaf.org"

xmlns:sec="http://www.thymeleaf.org/extras/spring-security">

- ▶ Podemos visualizar el nombre:

<div sec:authentication="name"></div>

<div th:text="\${#authentication.name}"></div>

# ALGO PARA PRACTICAR

Un ejercicio para hacer por tu cuenta

## AÑADIR INFORMACIÓN DE ERROR

- ▶ Al intentar un login fallido, la url a la que nos remite es /login?error.
- ▶ Intenta añadir algún elemento de interfaz de usuario (qué tal un *alert* de bootstrap) en el que se muestre un mensaje de error.
- ▶ Para obtener un *request param* llamado error thymeleaf podemos usar la expresión `${param.error}`



## MODIFICAR LA AUTORIZACIÓN

- ▶ En el ejemplo, tanto el listado de empleados como el alta de uno nuevo está autenticado.
- ▶ ¿Serías capaz de que el listado de empleados fuera de acceso público, pero no el alta de un nuevo empleado o la edición de uno existente?