Programación Frontend y Backend

BLOQUE SPRING

Spring Security

GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
EOI Escuela de organización industrial
Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro
SISTEMA NACIONAL DE GARANTÍA JUVENIL

01 Spring Security

GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
EOI Escuela de organización industrial

Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro

SISTEMA NACIONAL DE GARANTÍA JUVENIL

Autenticación y Autorización

- La *autenticación* es el mecanismo que a través de él certificamos que un usuario es quien dice ser
- La *autorización* es el mecanismo por el cual acreditamos que un usuario tiene permisos para acceder al recurso solicitado

GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
EOI Escuela de organización industrial
Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro
SISTEMA NACIONAL DE GARANTÍA JUVENIL

Dependencias

Necesitamos inyectar la siguiente dependencia:

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

Primeros Pasos (1)

Además de incluir la dependencia en el pom.xml, para habilitar Spring Security necesitamos decir al proyecto que inicie la seguridad, para ello debemos:

- Incluir *@EnableWebSecurity* en cualquier clase anotada con @Configuration
- Crear un clase que herede de *WebSecurityConfigurerAdapter*, donde configuraremos la autenticación y la autorización

Primeros Pasos (2)

La clase padre *WebSecurityConfigurerAdapter* nos permite describir cómo será la seguridad, sobreescribiendo los siguientes métodos:

- Para la autenticación: *configure(AuthenticationManagerBuilder auth)*
- Para la autorización: *configure(HttpSecurity auth)*

Primeros Pasos (3)

Para que el usuario se autentique, necesitamos:

- Añadir un mapeo a la ruta */login*
- Añadir un nuevo template, *login.html*, con el formulario para identificar al usuario

Acceso a la Seguridad (Thymeleaf)

Necesitamos añadir la dependencia:

```
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity4</artifactId>
    <version>3.0.2.RELEASE</version>
</dependency>
```

Necesitamos añadir el Bean: `SpringSecurityDialect`

Necesitamos añadir el taglib:

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

Acceso a la Seguridad (Thymeleaf)

Expressions Utility Objects:

- *#authentication* representa el objeto de autenticación.

```
<div th:text="${#authentication.name}">
   El valor de la propiedad “name” del objeto authetication debe aparecer aquí.
</div>
```

- *#authorization:* contiene métodos para comprobar la autorización mediante expresiones

```
<div th:if="${#authorization.expression('hasRole(''ROLE_ADMIN'')')}">
    Esto sólo se mostrará si el usuario autenticado tiene el role ROLE_ADMIN.
</div>
```

Acceso a la Seguridad (Thymeleaf)

Thymeleaf-Security taglib attributes:

- *sec:authentication="prop"* imprime la propiedad indicada del objeto authentication

```
<div sec:authentication="name">
   El valor de la propiedad “name” del objeto authetication debe aparecer aquí.
</div>
```

- *sec:authorize="expr"* renderiza el contenido si el usuario está autorizado mediante la expresión especificada

```
<div sec:authorize="hasRole('ROLE_ADMIN')">
    Esto sólo se mostrará si el usuario autenticado tiene el role ROLE_ADMIN.
</div>
```

Acceso a la Seguridad (Thymeleaf)

Thymeleaf-Security taglib attributes:

- *sec:authorize-url="url"* renderiza el contenido si el usuario está autorizado a ver la URL especificada.

```
<div sec:authorize-url="/admin">
    Esto sólo se mostrará si el usuario está autorizado a ver “/admin”.
</div>
```

Securizar Métodos y Clases

Spring Security nos provee de dos anotaciones para securizar a nivel de clase o método, en un controlador:

- *@PreAuthorize(expr)* comprueba la expresión relacionada con la autorización al entrar al método.
- *@PostAuthorize(expr)* comprueba la expresión relacionada con la autorización al salir del método.

Es necesario anotar la clase de configuración con:

@EnableGlobalMethodSecurity(prePostEnabled = true)

Securizar Métodos y Clases

Spring Security nos provee de dos anotaciones para securizar a nivel de clase o método, en un controlador:

- *@PreAuthorize(expr)* comprueba la expresión relacionada con la autorización al entrar al método.
- *@PostAuthorize(expr)* comprueba la expresión relacionada con la autorización al salir del método.

Es necesario anotar la clase de configuración con:

@EnableGlobalMethodSecurity(prePostEnabled = true)

Gestión de No Autorización

Spring Security brinda la posibilidad de gestionar los errores de autorización. Para ello, necesitamos:

- Un mapeo para el error 403
- Un gestor de errores 403
- Adaptar el *Autenticador* para restringir los accesos

Autenticación desde BD

Para autenticar a un usuario desde la base de datos, necesitamos crear la infraestructura necesario de JPA:

- Entidad “User” y su repositorio
- Entidad “Role” y su repositorio

Además debemos implementar un *UserDetailsService* el cual será utilizado por el “autenticador”

Y un encriptador de claves del tipo: *BCryptPasswordEncoder*

Adicionalmente, vamos a necesitar tener los usuarios en BD. Para poder probar implementaremos un cargador al inicio de la app.

Ejemplo *Spring Security*

```
git clone https://github.com/SpringFrameworkWorkshop/billing-app.git
git checkout security
```

GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
EOI Escuela de organización industrial

Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro

SISTEMA NACIONAL DE GARANTÍA JUVENIL

Ref

Spring Security

- https://docs.spring.io/spring-batch/4.0.x/reference/html/index.html
- https://spring.io/guides/gs/scheduling-tasks/