

SPRING FRAMEWORK

Introducción

La clase responsable de configurar la seguridad en tu proyecto de Spring cumple una función esencial en la implementación de medidas de protección, ya que permite definir el comportamiento del sistema en cuanto al inicio y cierre de sesión, así como regular el acceso a distintas direcciones según el rol de cada usuario, lo que no solo refuerza la seguridad de la aplicación, sino que también garantiza una gestión eficaz y segura.

1. Configuración avanzada de seguridad en Spring

La clase de seguridad es la encargada de definir los métodos necesarios para gestionar el inicio y cierre de sesión, así como para establecer los permisos de acceso a diferentes rutas según el rol del usuario autenticado. Esta clase juega un papel crucial en la protección de la aplicación, permitiendo un control granular sobre qué usuarios pueden acceder a qué recursos y garantizando un

A continuación, se muestra un ejemplo de configuración que incluye autenticación:

```
package com.egg.biblioteca;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SeguridadWeb {

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

```

@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        .authorizeHttpRequests((authorize) -> authorize
            .requestMatchers("/css/", "/js/", "/img/").permitAll())
        .formLogin((form) -> form
            .loginPage("/login")
            .loginProcessingUrl("/logincheck")
            .usernameParameter("email")
            .passwordParameter("password")
            .defaultSuccessUrl("/inicio", true)
            .permitAll())
        .logout((logout) -> logout
            .logoutUrl("/logout")
            .logoutSuccessUrl("/login")
            .permitAll())
        .csrf(csrf -> csrf.disable());

    return http.build();
}
}

```

Desglose de la configuración

- **@Configuration:** Esta anotación marca la clase `SeguridadWeb` como una clase de configuración, indicando a Spring que contiene definiciones de beans administrados por el contenedor de Spring.
- **@EnableWebSecurity:** Habilita la configuración de seguridad web en la aplicación, permitiendo que Spring gestione el acceso a los recurrentes.
- **passwordEncoder():** Este método crea y devuelve un objeto `BCryptPasswordEncoder`, que se utiliza para cifrar las contraseñas de manera segura. BCrypt es un algoritmo de hash robusto que se emplea comúnmente para proteger las contraseñas.
- **@Bean:** La anotación `@Bean` se utiliza para definir el método `filterChain` como un bean administrado por Spring. Este método devuelve un objeto de tipo `SecurityFilterChain`, que define las reglas de seguridad para la aplicación.
- **SecurityFilterChain:** La interfaz `SecurityFilterChain` permite definir una serie de filtros que se aplicarán a las solicitudes HTTP para controlar el acceso a los recursos.
- **Método filterChain(HttpSecurity http):** Este método configura las reglas de seguridad utilizando el objeto `HttpSecurity`. Las configuraciones clave incluyen:
 - **authorizeHttpRequests():** Establece las reglas de autorización para las solicitudes HTTP. Por ejemplo, se configura que solo los usuarios con el rol

ADMIN puedan acceder a `/admin/`, mientras que los recursos estáticos como archivos CSS, JS e imágenes son accesibles para todos los usuarios.

- **formLogin()**: Configure el formulario de inicio de sesión. Los detalles importantes incluyen:
 - `loginPage("/login")`: La URL de la página de inicio de sesión.
 - `loginProcessingUrl("/logincheck")`: La URL a la que se enviarán los datos del formulario de inicio de sesión.
 - `usernameParameter("email")` y `passwordParameter("password")`: Los nombres de los campos en el formulario de inicio de sesión.
 - `defaultSuccessUrl("/inicio", true)`: La URL a la que se redirige al usuario después de un inicio de sesión exitosa.
- **logout()**: Configure la funcionalidad de cierre de sesión:
 - `logoutUrl("/logout")`: La URL para cerrar sesión.
 - `logoutSuccessUrl("/login")`: La URL a la que el usuario será redirigido después de cerrar sesión.
- **csrf().disable()**: Deshabilita la protección contra ataques CSRF (Cross-Site Request Forgery). En este caso, se desactiva porque no estamos manejando formularios de autenticación específicos en esta configuración.
- **http.build()**: Este método finaliza la configuración de `HttpSecurity` y devuelve el objeto `SecurityFilterChain` configurado.

2. Autorización de acceso a métodos según autenticación previa

En Spring Security, podemos autorizar el acceso a métodos específicos de manera más controlada, utilizando autenticación previa. Esto resulta útil cuando, dentro de un controlador, existen métodos de acceso público y otros restringidos.

La anotación `@PreAuthorize` permite aplicar autorización a nivel de método, utilizando expresiones SpEL (Spring Expression Language). Esta anotación se evalúa antes de ejecutar un método, lo que permite definir reglas de seguridad directamente en el método en lugar de configurarlas de forma global en la clase de configuración de seguridad.

La expresión dentro de las comillas en **@PreAuthorize** se evalúa para determinar si el usuario actual tiene los permisos necesarios para ejecutar el método anotado. Por ejemplo, la expresión `hasAnyRole('ROLE_USER', 'ROLE_ADMIN')` verifica si el usuario tiene al menos uno de los roles especificados, es decir, "ROLE_USER" o "ROLE_ADMIN".

Explicación de la expresión `hasAnyRole('ROLE_USER', 'ROLE_ADMIN')`:

- `hasAnyRole('ROLE_USER', 'ROLE_ADMIN')`: Esta expresión asegura que el método solo se puede ejecutar si el usuario tiene al menos uno de los roles indicados. Si el usuario posee el rol "ROLE_USER" o "ROLE_ADMIN", se le concede acceso y se ejecuta el método. Si no tiene ninguno de estos roles, la autorización se deniega y se lanzará una excepción de acceso denegado.

Es relevante recordar que, en Spring Security, los roles se definen convencionalmente con el prefijo "ROLE_". Sin embargo, al usar `hasAnyRole`, no es necesario incluir este prefijo en los nombres de los roles dentro de la expresión.

La anotación **@PreAuthorize** resulta especialmente útil cuando se desea aplicar lógica de autorización a métodos específicos, brindando un control más detallado sobre quién puede acceder a cada uno, en lugar de establecer reglas globales a nivel de configuración.

Ejemplo de implementación:

```
@PreAuthorize("hasAnyRole('USER', 'ADMIN')")
@GetMapping("/inicio")
public String inicio() {
    // Métodos
    return "inicio.html";
}
```

En este ejemplo, solo las personas autenticadas que tengan los roles "USER" o "ADMIN" podrán acceder al recurso `inicio.html`.

3. Conectando tu configuración de inicio/cierre de sesión con HTML

Una vez que hayas configurado correctamente los métodos de inicio y cierre de sesión en el backend, es fundamental que te asegures de que tus archivos HTML estén listos para manejar estas acciones de manera efectiva.

Formulario de inicio de sesión

En primer lugar, debes asegurarte de que el formulario de inicio de sesión en tus archivos HTML incluya el atributo **th:action**. Este atributo indica la URL a la que se enviará el formulario para su procesamiento, y debe coincidir exactamente con lo que hayas declarado en la clase **SeguridadWeb** para **loginProcessingUrl**. Es decir, el valor de **th:action** en tu formulario debe ser idéntico al nombre proporcionado en **loginProcessingUrl**. Esto garantizará que la solicitud de inicio de sesión se maneje correctamente por tu aplicación backend.

```
<form class="formulario" th:action="@{/logincheck}" method="POST">

  <div class="form-group my-3">
    <h3 class="card-title">Accede a la Biblioteca!</h3>
  </div>
  <div class="form-group my-3">
    <input type="email" class="form-control" name="email" placeholder="EMAIL">
  </div>
  <div class="form-group my-3">
    <input type="password" class="form-control" name="password" placeholder="CONTRASEÑA">
  </div>

  <button type="submit" class="btn btn-primary mt-3 botones">Ingresar!</button>
</form>
```

Declaración de URL en tu clase de seguridad

```
loginProcessingUrl("/login");
```

Este es el enlace entre el formulario HTML y la configuración de Spring Security en el backend.

```

.formLogin((form) -> form
    .loginPage(loginPage:"/login")
    .loginProcessingUrl(loginProcessingUrl:"/logincheck")
    .usernameParameter(usernameParameter:"email")
    .passwordParameter(passwordParameter:"password")
    .defaultSuccessUrl(defaultSuccessUrl:"/inicio", alwaysUse:true)
    .permitAll()
)

```

Cierre de sesión

En cuanto al cierre de sesión, es necesario que especifiques explícitamente el destino de la acción de cierre mediante el atributo **href** en el HTML. Este atributo debe apuntar al método correspondiente en el backend, que por defecto es **"/logout"**.

Al asignar la URL **"/logout"** al atributo **href** de un elemento HTML (como un botón, un enlace u otro tipo de elemento), estarás asegurando que, al hacer clic en este, se enviará una solicitud de cierre de sesión al backend. Esto permitirá que el usuario cierre su sesión de manera correcta y segura.

Elemento para cerrar sesión

```


<ul class="dropdown-menu" aria-labelledby="navbarDropdown">
  <li><a class="dropdown-item" href="/logout" >Cerrar Sesión</a></li>
</ul>

```

```

.logout((logout) -> logout
    .logoutUrl(logoutUrl:"/logout")
    .logoutSuccessUrl(logoutSuccessUrl:"/login")
    .permitAll()
)

```

 **IMPORTANTE:** Es importante que sepas que, para configurar las funcionalidades de inicio y cierre de sesión (login y logout), no es necesario crear métodos separados en tu código. Estas acciones están integradas dentro del flujo de Spring Security y se manejan automáticamente a través de la configuración que hayas proporcionado en tu clase de seguridad. Spring Security gestionará automáticamente estas solicitudes y ejecutará las acciones correspondientes según lo que hayas configurado.

