Ejercicio Base Spring Security

**Paso 1: Dependencias**

Como primer paso vamos a crear un nuevo proyecto en **Initializr** incluyendo las dependencias de **Spring Web** y de **Spring Security**.

<https://start.spring.io/#!type=maven-project&language=java&platformVersion=3.4.3&packaging=jar&jvmVersion=21&groupId=ar.edu.centro8.ps&artifactId=proyectobaseseguridad&name=proyectobaseseguridad&description=Proyecto%20Base%20Seguridad&packageName=ar.edu.centro8.ps.proyectobaseseguridad&dependencies=devtools,lombok,web,security>

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**Paso 2: Creación de API de ejemplo**

Como segundo paso vamos a levantar el proyecto generado en Initializr en nuestro IDE.  A partir de eso, crearemos en primer lugar la estructura básica de cualquier proyecto donde utilizamos o desarrollamos apis, respetando la arquitectura multicapas. Agregaremos como extra, un nuevo paquete llamado security y un subpaquete llamado config para colocar allí las configuraciones de seguridad.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Luego, crearemos en nuestro Controller dos endpoints de “Hello World”, uno que vamos a segurizar (el usuario deberá autenticarse para poder acceder a él) y otro que no estará segurizado (a futuro, en principio ambos lo estarán).

Para que sea más sencillo identificar cada end-point, podemos diferenciarlos con nombres de métodos o mensajes distintos.

package ar.edu.centro8.ps.proyectobaseseguridad.controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class HelloWorldController {

    @GetMapping("/holaseg")

    public String secHelloWorld() {

        return "Hola Mundo con seguridad";

    }

    @GetMapping("/holanoseg")

    public String noSecHelloWorld() {

        return "Hola mundo sin seguridad";

    }

}

Por otro lado, nuestro archivo de configuración nos queda así:

package ar.edu.centro8.ps.proyectobaseseguridad.security.config;

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.web.SecurityFilterChain;

import static org.springframework.security.config.Customizer.withDefaults;

@Configuration

@EnableWebSecurity

public class SecurityConfig {

    @Bean

    public SecurityFilterChain filterChain (HttpSecurity httpSecurity) throws Exception {

        return httpSecurity

                .authorizeHttpRequests(authorize -> authorize

                    .requestMatchers("/holanoseg").permitAll()

                    .anyRequest().authenticated())

                .formLogin(withDefaults())

                .build();

    }

}

Veamos en detalle cada parte de este código:

1. **public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception**: Este método recibe un objeto HttpSecurity como argumento y devuelve un objeto SecurityFilterChain (una cadena de filtros de seguridad que establezcamos). La excepción Exception se usa para manejar posibles errores.
2. **httpSecurity.authorizeHttpRequests():** Comienza la configuración de la autorización para las peticiones HTTP.
3. **.requestMatchers(«/holanoseg»).permitAll():** Define una regla de autorización específica para la ruta /holanoseg, permitiendo el acceso a esta ruta sin requerir autenticación. Es decir, cualquier solicitud a esta ruta será permitida para todos los usuarios.
4. **.anyRequest().authenticated()**: Para cualquier otra solicitud que no coincida con la ruta especificada arriba, se requiere autenticación. Esto significa que el acceso a cualquier otra ruta necesitará que el usuario esté autenticado para acceder.
5. **.and():** Se utiliza para encadenar las configuraciones de seguridad.
6. **.formLogin().permitAll():** Configura el formulario de inicio de sesión y permite que esta página de inicio de sesión sea accesible para todos, incluso para aquellos que aún no han iniciado sesión.
7. **.build():** Finaliza la configuración y construye la cadena de filtros de seguridad.

**Paso 3: Ejecución**

Una vez listo nuestro Controller, procederemos a ejecutar nuestra aplicación. Si todo sale correctamente, el navegador nos redireccionará a una pantalla de Login donde tendremos que «autenticarnos», es decir, ingresar un usuario y contraseña para hacer uso de los end-points en cuestión.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Como actualmente no configuramos ningún usuario ni contraseña, debemos usar los datos por defecto que nos ofrece Spring Security. Por defecto, el usuario es «user» y la contraseña asociada a nuestra sesión la encontramos en la consola de ejecución del IDE que estemos utilizando:

Captura de pantalla de un celular con texto e imagen

El contenido generado por IA puede ser incorrecto.

Si colocamos correctamente los datos en la pantalla de login, vamos a ver que la autenticación se produce y podemos acceder correctamente a cualquiera de nuestros endpoints (dado que por el momento, ambos están securizados):

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**⚠️ IMPORTANTE:** Una vez autenticados, si intentamos acceder a cualquier end-point, vamos a poder hacerlo. ¿Por qué? Porque **Spring Security** detecta que nos hemos autenticado correctamente mediante usuario y contraseña, por lo que tendremos acceso a TODOS los recursos, a menos que configuremos lo contrario (por lo que tenemos acceso tanto al end-point seguro como al no seguro).

**Nota**: Una vez logueados se crea una «sesión» a nuestra aplicación. La misma la podemos cerrar accediendo a la ruta **/logout**, por ejemplo **https://localhost:8080/logout**si todo está ok, veremos un cartel de confirmación para cerrar nuestra sesión:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto. Interfaz de usuario gráfica, Aplicación, Sitio web

El contenido generado por IA puede ser incorrecto.

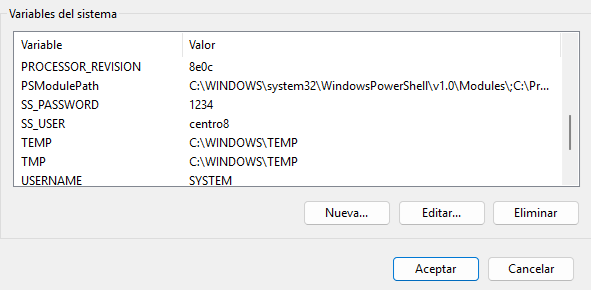
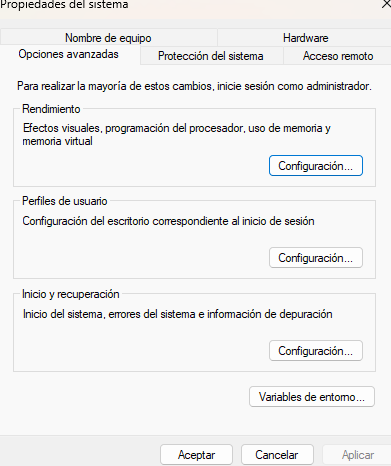
**Paso 4: Personalizar usuario y contraseña**

Para poder configurar un usuario y contraseña diferentes a los por defecto de Spring Security, debemos llevar a cabo una serie de configuraciones

*En primer termino debemos configurar las variables de entorno del usuario y contraseña:*

SS\_USER=centro8

SS\_PASSWORD=1234



Una vez configurado esto, debemos modificar el archivo de configuración de seguridad:

package ar.edu.centro8.ps.proyectobaseseguridad.security.config;

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.web.SecurityFilterChain;

import static org.springframework.security.config.Customizer.withDefaults;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

import java.util.Collections;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration

@EnableWebSecurity

public class SecurityConfig {

    @Value("${SS\_USER}")

    private String user;

    @Value("${SS\_PASSWORD}")

    private String password;

    @Bean

    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        return http

            .authorizeHttpRequests(auth -> auth

                .requestMatchers("/holanoseg").permitAll()

                .anyRequest().authenticated()

            )

            .formLogin(withDefaults())

            .logout(withDefaults())

            .userDetailsService(userDetailsService())  // Configura el usuario

            .build();

    }

    @Bean

    public UserDetailsService userDetailsService() {

        UserDetails user = User.withUsername(this.user)

            .password(passwordEncoder().encode(this.password))

            .roles("USER")

            .build();

        return new InMemoryUserDetailsManager(Collections.singleton(user));

    }

    @Bean

    public PasswordEncoder passwordEncoder() {

        return new BCryptPasswordEncoder();

    }

}

Este código configura la seguridad para una aplicación Spring Boot usando Spring Security. Veamos su estructura y funcionamiento:

@Configuration

@EnableWebSecurity

public class SecurityConfig {

@Configuration indica que esta clase contiene configuraciones de Spring

@EnableWebSecurity activa las características de seguridad de Spring

Diagrama

El contenido generado por IA puede ser incorrecto.

Diagrama

El contenido generado por IA puede ser incorrecto.

El diagrama muestra dos aspectos principales:

La configuración inicial donde SecurityConfig establece los servicios necesarios

El proceso de autenticación donde las solicitudes pasan por el formulario de login y son validadas.

Veamos ahora los componentes específicos del código:

@Value("${SS\_USER}")

private String user;

@Value("${SS\_PASSWORD}")

private String password;

Estas propiedades se cargan desde las variables de entorno o archivo de configuración

Contienen el nombre de usuario y contraseña predeterminados para el sistema.

La configuración principal del filtro de seguridad:

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

return http

.authorizeHttpRequests(auth -> auth

.requestMatchers("/holanoseg").permitAll()

.anyRequest().authenticated()

)

.formLogin(withDefaults())

.logout(withDefaults())

.userDetailsService(userDetailsService())

.build();

}

permitAll() permite acceso libre a la ruta "/holanoseg"

.anyRequest().authenticated() requiere autenticación para todas las demás rutas

formLogin() habilita el formulario de inicio de sesión predeterminado

logout() configura la funcionalidad de cierre de sesión

El servicio de usuarios y codificador de contraseñas:

@Bean

public UserDetailsService userDetailsService() {

UserDetails user = User.withUsername(this.user)

.password(passwordEncoder().encode(this.password))

.roles("USER")

.build();

return new InMemoryUserDetailsManager(Collections.singleton(user));

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

Crea un usuario con rol "USER" usando credenciales configuradas

Utiliza BCrypt para encriptar la contraseña

Almacena el usuario en memoria (sin base de datos).

Este código proporciona una configuración básica pero segura de Spring Security, donde cualquier ruta que no sea "/holanoseg" requerirá autenticación para acceder.

**Paso 5: Pruebas en navegador**

Ejecutaremos nuestra aplicación e intentaremos acceder al endpoint no securizado **https://localhost:8080/holanoseg**y si todo sale bien, deberíamos de poder ingresar sin necesidad de autorización:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Mientras que si intentamos acceder al end-point securizado, deberemos loguearnos de forma obligatoria:

Interfaz de usuario gráfica, Aplicación, Sitio web

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.