El futuro digital es de todos | MinTIC

UNIVERSIDAD EL BOSQUE | MisiónTIC 2022

Ciclo 3

Semana 5

Conceptos de Cloud Computing con Amazon Web Services (AWS) y seguridad en ambientes web.

Lectura 4 - Seguridad Java Web con Spring (Spring Security)

Convenio de asociación no. 777 de 2021 suscrito entre el Fondo Único de Tecnologías de la Información y las Comunicaciones y la Universidad el Bosque.

Seguridad Java Web con Spring (Spring Security)

Algunos de los conceptos de seguridad en una aplicación Web que se deben tener en cuenta al momento de desarrollar una, entre otras, están los siguientes puntos:

- Filtros de Autenticación: Evitar escribir una URL sin que se haya hecho login a la aplicación correctamente.
- Control de Páginas de Error: Controlar errores HTTP tipo 404 (recurso no encontrado), o 500 (Error de Servidor) mediante la configuración de páginas.
- Envío de peticiones seguras: Configuración de envío de parámetros para envío entre una página JSP y un Servlet, que deben estar ocultos.

Spring Security es un framework de Spring que permite gestionar completamente la seguridad las aplicaciones Java. Permite:

- Gestionar seguridad a varios niveles: No solamente control de acceso a la aplicación, sino acceso a recursos de bajo nivel, tales como clases, métodos, etc.
- Configuración de seguridad portable: Todo lo configurado se empaquetará en el archivo .JAR, o .WAR para la publicación en servidores, o nube.
- Soporta muchos modelos de autenticación (HTTP Basic, LDAP – directorio activo, OAuth, Http Digest entre otros).

Esta característica se instala en la aplicación como una dependencia más de Maven en nuestra aplicación de backend – recordemos que es la aplicación `ciclo3back`, la cual se realiza, por medio de la aplicación de los siguientes pasos:

Paso 1: Configurar la dependencia Maven `spring-boot-starter-security` en el archivo de configuración `pom.xml`.

Se agrega la descripción de la siguiente dependencia:

```
<!-- Dependencia Spring Security  -->
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Se puede colocar, después de la definición de la dependencia MySQL, la cual debería verse la siguiente forma:

ciclo3back/pom.xml

```
34        <dependency>
35            <groupId>org.springframework.boot</groupId>
36            <artifactId>spring-boot-devtools</artifactId>
37            <scope>runtime</scope>
38            <optional>true</optional>
39        </dependency>
40        <dependency>
41            <groupId>mysql</groupId>
42            <artifactId>mysql-connector-java</artifactId>
43            <scope>runtime</scope>
44        </dependency>
45        <!-- Dependencia Spring Security -->
46        <dependency>
47            <groupId>org.springframework.boot</groupId>
48            <artifactId>spring-boot-starter-security</artifactId>
49        </dependency>
```

Paso 2: Excluir la configuración por defecto de seguridad.

Por defecto, SpringBoot realiza una configuración básica por defecto de seguridad, la cual debe ser excluida mediante la agregación del atributo `exclude=` dentro de la anotación @SpringBootApplication. en la clase principal de la aplicación, de la siguiente forma:

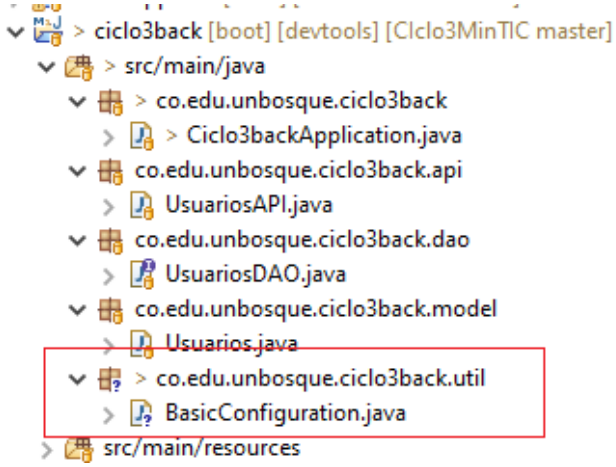
ciclo3back/pom.xml | Ciclo3backApplication.java

```
1  package co.edu.unbosque.ciclo3back;
2
3  import org.springframework.boot.SpringApplication;
6
7  @SpringBootApplication(exclude = {SecurityAutoConfiguration.class})
8  public class Ciclo3backApplication {
9
10     public static void main(String[] args) {
11         SpringApplication.run(Ciclo3backApplication.class, args);
12     }
13
14 }
```

En este punto marcará un error en la parte de `SecurityAutoConfiguration`, lo cual puede ser solucionado, agregando la respectiva librería, mediante las teclas <Crtl>+>shift>+<O>.

Paso 3: Crear una clase para la configuración de la aplicación de Spring Security.

Para efectos de tener una organización de nuestras clases y paquetes de la aplicación, haremos un nuevo package donde colocar la clase, que llamaremos `util` (`co.edu.unbosque.util`). En este package, posteriormente, crearemos la nueva clase, llamada BasicConfiguration. Nos debe quedar algo similar a esto:



Dentro de la clase `BasicConfigurarion.java`, vamos a agregar la configuración de seguridad, por medio de las anotaciones `@Configuration`, y `@EnableWebSecurity`. Agregamos las librerías por medio de las teclas <Crtl>+>shift>+<O>. Debemos hasta ahora tener esto:

```
1  package co.edu.unbosque.ciclo3back.util;
2
3  import org.springframework.context.annotation.Configuration;
4  import org.springframework.security.config.annotation.web.builders.HttpSecurity;
5  import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
6  import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
7
8  @Configuration
9  @EnableWebSecurity
10 public class BasicConfiguration extends WebSecurityConfigurerAdapter {
11
12
13 }
```

Luego, deberemos agregar un método que le dé instrucciones a la aplicación de qué tipo de autenticación y seguridad queremos para la aplicación para lo cual, vamos a escribir el siguiente método dentro de la clase, con nombre `configure` (es `protected` porque hereda de `WebSecurityConfigurerAdapter`) , el cual tendrá la siguiente estructura:

```
protected void configure(HttpSecurity http) throws Exception {
      http.authorizeRequests().anyRequest().authenticated().and().httpBasic().and
      ().csrf().disable();
}
```

Este método maneja un método de seguridad por HTTP, (`HttpSecurity`), el cual debe lanzar una excepción, si se da un error en seguridad, y en el parámetro `http` se autorizan los Request, de cualquier tipo (`anyRequest()`) , de forma autenticada (`authenticated()`), con autenticación de tipo `usuario: contraseña` (`httpBasic()`), y deshabilitando la opción csrf (*Cross-site request forgery*: falsificación de petición en sitios cruzados) es un ataque en que usuario final ejecuta acciones no autorizadas en una aplicación web en la que está autenticado, tales como hacer un POST o GET directamente utilizando la URL de la API, en donde solamente autorizará acceso a la aplicación de frontend. Finalmente, todo lo codificado, debería verse de la siguiente forma:

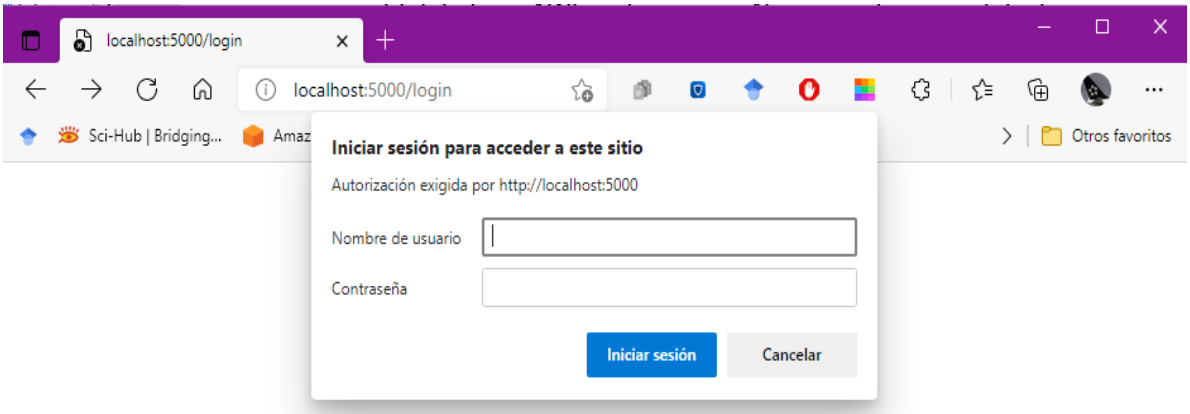
```
package co.edu.unbosque.ciclo3back.util;

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.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableWebSecurity
public class BasicConfiguration extends WebSecurityConfigurerAdapter {

        protected void configure(HttpSecurity http) throws Exception {
           http.authorizeRequests().anyRequest().authenticated().and().httpBasic().and().csrf().disable();
        }
}
```

Paso 4: Crear una entrada de configuración en el archivo `configuration.properties` de una clave de acceso.

Para crear una combinación de tipo `usuario:clave`, vamos a agregar en el archivo `configuration.properties`, las siguientes dos entradas:

```
spring.security.user.name=usuario
spring.security.user.password=tiendagenerica
```

Compilamos con Maven la aplicación, y veremos que, si escribimos en el browser http://localhost:5000/ (recuerden que cambiamos el número del puerto del servidor de 8080 a 5000 por compatibilidad con AWS en la semana 4),nos conduce por defecto una página de autenticación (/login), de la siguiente forma:



Con lo anterior, se establece que ya en este momento, no se puede acceder a las APIs libremente cuando, por ejemplo, ejecutábamos la opción http://localhost:5000/usuarios/listar en el navegador, y obteníamos los resultados, por lo que, debemos agregar la autenticación en nuestra aplicación de frontend, en el siguiente paso.

Paso 5: Modificación en el frontend para agregar la autenticación requerida por el backend.

Si en este momento con el cambio anterior, utilizáramos nuestra aplicación de frontend para listar usuarios (en otras palabras, consumir la API de /usuarios/listar), nos generaría una `Exception` de IO, por medio de un error HTTP de autenticación 401 de la siguiente forma:

```
INFO: Server startup in [1058] milliseconds
java.io.IOException: Server returned HTTP response code: 401 for URL: http://localhost:5000/usuarios/listar
        at java.base/sun.net.www.protocol.http.HttpURLConnection.getInputStream0(HttpURLConnection.java:1913)
        at java.base/sun.net.www.protocol.http.HttpURLConnection.getInputStream(HttpURLConnection.java:1509)
        at co.edu.unbosque.ciclo3demo.TestJSON.getJSON(TestJSON.java:32)
        at co.edu.unbosque.ciclo3demo.DemoServlet.listarUsuarios(DemoServlet.java:80)
        at co.edu.unbosque.ciclo3demo.DemoServlet.doGet(DemoServlet.java:40)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:626)
```

Por lo anterior, debemos modificar nuestra aplicación de frontend (`ciclo3demo`), para que, al momento de consumir cualquier API del backend, se realice la autenticación, mediante la agregación del usuario y contraseña establecidos en el backend.

En la clase `TestJSON.java`, debemos incluir las siguientes instrucciones para la autenticación, en el método `getJSON()` el cual nos permite recuperar la lista de usuarios, que, inicialmente lo tenemos codificado de esta forma:

```
public static ArrayList<Usuarios> getJSON() throws IOException, ParseException{
	url = new URL(sitio+"usuarios/listar");
	HttpURLConnection http = (HttpURLConnection)url.openConnection();
	http.setRequestMethod("GET");
	http.setRequestProperty("Accept", "application/json");
	InputStream respuesta = http.getInputStream();
	byte[] inp = respuesta.readAllBytes();
	String json = "";
	for (int i = 0; i<inp.length ; i++) {
		json += (char)inp[i];
	}
	ArrayList<Usuarios> lista = new ArrayList<Usuarios>();
	lista = parsingUsuarios(json);
	http.disconnect();
	return lista;
}
```

Vamos a agregar dos (2) instrucciones para la autenticación, por medio de la generación de una cadena de autenticación con una variable tipo `String` con el usuario/contraseña (1) , y luego agregamos una nueva propiedad en `RequestProperty()`, con el atributo "`Authorization`" (2).

```
public static ArrayList<Usuarios> getJSON() throws IOException, ParseException{
    url = new URL(sitio+"usuarios/listar");
    String authStr = Base64.getEncoder().encodeToString("usuario:tiendagenerica".getBytes());   (1)
    HttpURLConnection http = (HttpURLConnection)url.openConnection();
    http.setRequestMethod("GET");
    http.setRequestProperty("Accept", "application/json");
    http.setRequestProperty("Authorization", "Basic " + authStr);   (2)
    InputStream respuesta = http.getInputStream();
    byte[] inp = respuesta.readAllBytes();
    String json = "";
    for (int i = 0; i<inp.length ; i++) {
        json += (char)inp[i];
    }
    ArrayList<Usuarios> lista = new ArrayList<Usuarios>();
    lista = parsingUsuarios(json);
    http.disconnect();
    return lista;
}
```

Con lo anterior, agregamos una capa de seguridad entre frontend y backend para consumo de las API REST, y que sea la aplicación de frontend autorizada para consumirla. Estas dos líneas se deben agregar en todos los métodos que vayan a consumir las APIs, o se puede generar un método para tal fin.

Debemos hacer otra modificación en el código del método postJSON(), para agregar de forma similar el header de autenticación. Recordemos cómo teníamos el código de este método:

```
public static int postJSON(Usuarios usuario) throws IOException {
	url = new URL(sitio+"usuarios/guardar");

	HttpURLConnection http;
	http = (HttpURLConnection)url.openConnection();
	try {
		http.setRequestMethod("POST");
	} catch (ProtocolException e) {
		e.printStackTrace();
	}
	http.setDoOutput(true);
	http.setRequestProperty("Accept", "application/json");
	http.setRequestProperty("Content-Type", "application/json");
	String data = "{"
		+ "\"cedula_usuario\":\""+ usuario.getCedula_usuario()
		+"\",\"email_usuario\": \""+usuario.getEmail_usuario()
		+"\",\"nombre_usuario\": \""+usuario.getNombre_usuario()
		+"\",\"password\":\""+usuario.getPassword()
		+"\",\"usuario\":\""+usuario.getUsuario()
		+ "\"}";
	byte[] out = data.getBytes(StandardCharsets.UTF_8);
	OutputStream stream = http.getOutputStream();
	stream.write(out);
	int respuesta = http.getResponseCode();
	http.disconnect();
	return respuesta;
}
```

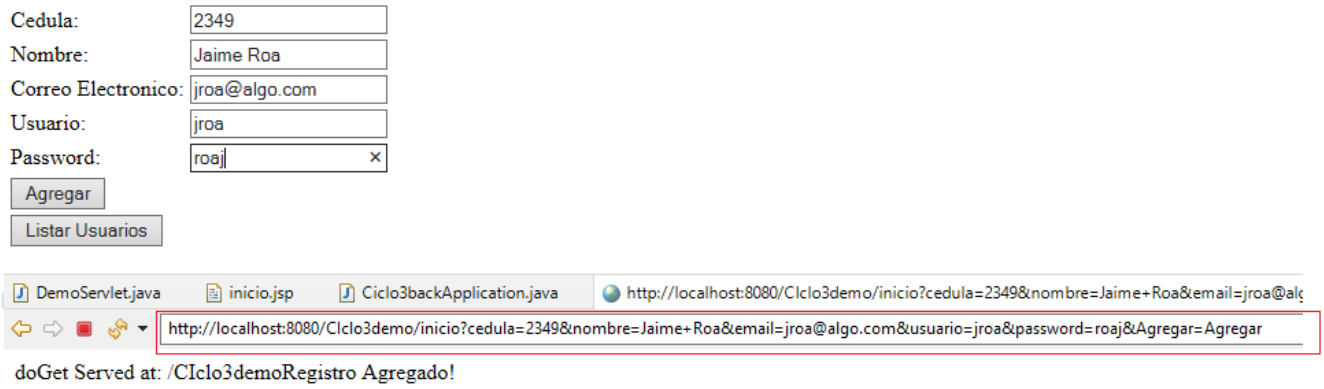
Introducimos los cambios denotados abajo, que, en la práctica son las mismas dos líneas de código que nos permite generar una codificación de la clase `usuario:tiendagenérica` (1), y posteriormente, agregar el encabezado correspondiente (2), quedando de la siguiente forma:

```
public static int postJSON(Usuarios usuario) throws IOException {
    url = new URL(sitio+"usuarios/guardar");

    HttpURLConnection http;
    http = (HttpURLConnection)url.openConnection();
(1) String authStr = Base64.getEncoder().encodeToString("usuario:tiendagenerica".getBytes());
    try {
        http.setRequestMethod("POST");
    } catch (ProtocolException e) {
        e.printStackTrace();
    }
    http.setDoOutput(true);
    http.setRequestProperty("Accept", "application/json");
(2) http.setRequestProperty("Authorization", "Basic " + authStr);
    http.setRequestProperty("Content-Type", "application/json");

    String data = "{"
            + "\"cedula_usuario\":\""+ usuario.getCedula_usuario()
            +"\",\"email_usuario\": \""+usuario.getEmail_usuario()
            +"\",\"nombre_usuario\": \""+usuario.getNombre_usuario()
            +"\",\"password\":\""+usuario.getPassword()
            +"\",\"usuario\":\""+usuario.getUsuario()
            + "\"}";
    byte[] out = data.getBytes(StandardCharsets.UTF_8);
    OutputStream stream = http.getOutputStream();
    stream.write(out);
    int respuesta = http.getResponseCode();
    http.disconnect();
    return respuesta;
}
```

Con los cambios introducidos, podemos tener acceso a las APIs del backend, por medio de método de seguridad de autenticación y autorización para nuestras aplicaciones, desarrolladas en el frontend.

Paso 6: Modificación de método de hacer el Request para ocultamiento de parámetros en la página web `inicio.jsp`.

Como recordamos de la semana 3, el método que utilizamos para realizar el Request de todos los parámetros desdela página web `inicio.JSP` al Servlet DemoServlet.java, era por medio de GET, el cual, mostraba todos los parámetros en forma de texto plano en el browser de la siguiente forma:



Para evitar esto, vamos a hacer un cambio simple en la página en la parte donde dice :

```
<form method="get" action="./inicio">
```

Cambiamos “get”, por “post”,

```
<%@ page language="java" contentType="text/html; charset=ISO-8859
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Tienda Generica</title>
</head>
<body>
    <p align="center"> CREACION DE USUARIOS</p>
    <form method="post" action="./inicio">
        <table>
            <tr>
                <td><label>Cedula:</label></td>
```

Volvemos a ejecutar el servidor Tomcat para la aplicación, y al ingresar un usuario, ya no se ven los parámetros, de la siguiente forma: