

# Aplicar seguridad a las aplicaciones web



# Aplicando seguridad a nuestras aplicaciones web

En este curso aprenderás a aplicar seguridad a tus aplicaciones web desarrolladas con Spring Boot y Spring MVC.

- ✓ Configuración desarrollada solo con clases Java (no más configuración XML).
- ✓ Los usuarios y roles los recuperaremos de una base de datos relacional MySQL.
- ✓ Las contraseñas de los usuarios las guardaremos encriptadas (bcrypt).
- ✓ Desarrollaremos nuestro propio formulario de login con Bootstrap.
- ✓ Aplicaremos seguridad basada en los roles que tengan asignados los usuarios.

## Nota:

En estas secciones dedicadas a la seguridad, continuaremos con el proyecto terminado hasta la sección anterior: “Spring Boot - Integrar Spring MVC y Spring Data JPA”.

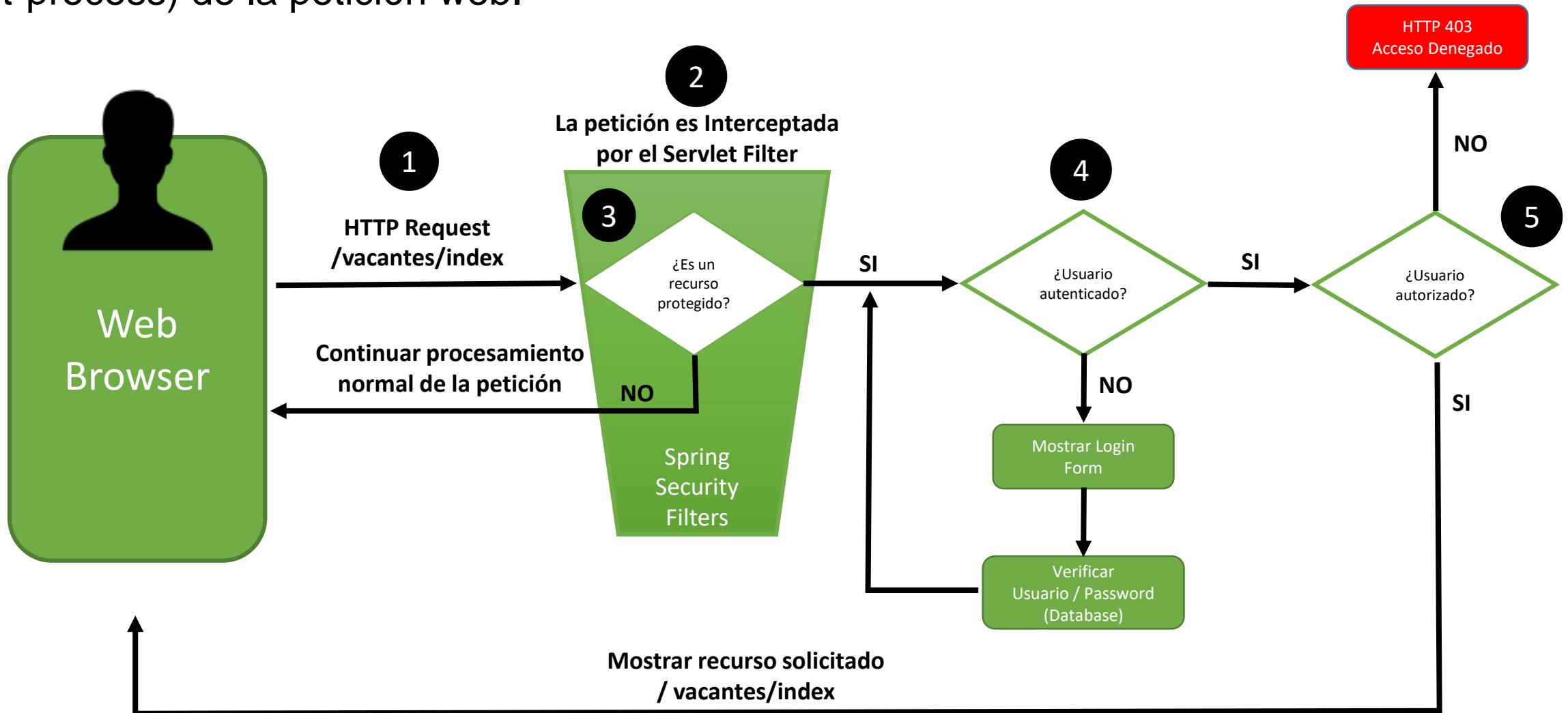
- ✓ Continuaremos con el proyecto descargado de la lección anterior:
  - Proyecto terminado con toda la funcionalidad, **PERO SIN SEGURIDAD.**

# ¿Qué es Spring Security?

- Es un framework de seguridad (módulo de Spring) que permite aplicar seguridad a tus aplicaciones desarrolladas con Spring.
  - ✓ En este curso integraremos **Spring Security** con **Spring Boot** para aplicar **Seguridad en Aplicaciones Web**.
- Spring Security aplica 2 tipos de seguridad:
  - ✓ **Autenticación**: ¿Es un usuario válido para acceder a la aplicación?
  - ✓ **Autorización**: ¿El usuario tiene permisos (ROL) para acceder al recurso solicitado?
- La seguridad es aplicada a nivel de **Petición Web (HTTP Request)** y a nivel de **Invocación de Métodos**.
- Spring Security esta basado en Spring Framework. Internamente utiliza:
  - ✓ Inyección de Dependencias (DI)
  - ✓ Programación orientada a aspectos (AOP).
- En aplicaciones web Spring Security utiliza **Servlet Filters** para aplicar seguridad a las peticiones web y restringir el acceso a nivel de URL.

# Spring Security – Servlet Filter

- Spring Security utiliza varios Servlet Filters para filtrar las peticiones web.
- Los Servlet Filters son componentes (Interceptors) ejecutados antes (pre-process) y después (post-process) de la petición web.



# Spring Boot – Soporte para Spring Security (pom.xml)

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



Spring Initializr  
Bootstrap your application

## Spring Security

Highly customizable authentication and access-control  
framework for Spring applications.



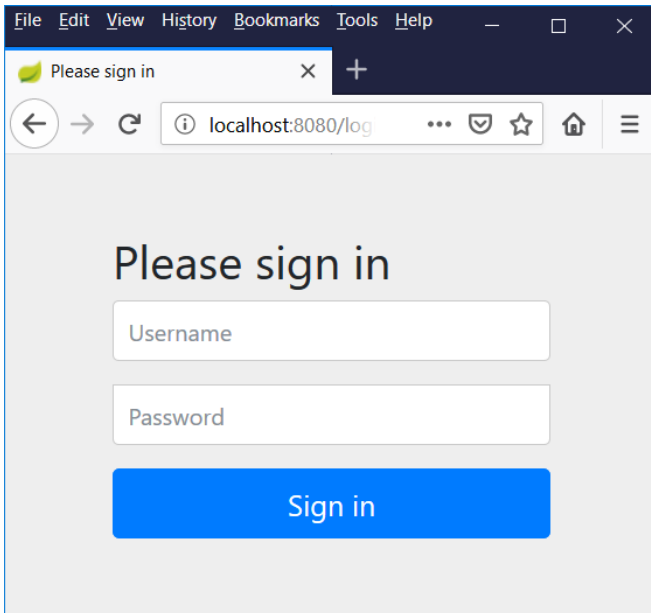
## OPCIONAL: Tags de Spring Security para Thymeleaf

```
<!-- Requerido para trabajar Thymeleaf Spring Security Tags -->
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>
```

# Configuración de Spring Security (1)

- Por defecto, con solo agregar la dependencia de Spring Security se aplicará la siguiente configuración:
  - ❖ Será requerida autenticación para todas las URLs.
  - ❖ Spring generará un formulario HTML de login de forma automática (COMPLETAMENTE FUNCIONAL).
  - ❖ Se agregará la configuración necesaria para prevenir ataques de tipo CSRF (Cross-site request forgery).
  - ❖ Se creará 1 usuario (username: **user**) en memoria con acceso todas las URLs. La contraseña es generada automáticamente y es desplegada en el log de la aplicación (console).

Formulario de login generado



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/log'. The page has a title 'Please sign in' and contains two input fields: 'Username' and 'Password'. Below these fields is a blue button labeled 'Sign in'.



The screenshot shows a Java IDE console window with the following logs:

```
empleos - EmpleosApplication [Spring Boot App] C:\java\jdk-11.0.2\bin\javaw.exe (10 may 2019 19:37:13)
2019-05-10 19:37:15.842 INFO 6528 --- [ restartedMain] org.apache.catalina.core.StandardEngine :
2019-05-10 19:37:15.929 INFO 6528 --- [ restartedMain] org.apache.catalina.core.StandardEngine :
2019-05-10 19:37:15.929 INFO 6528 --- [ restartedMain] org.apache.catalina.core.StandardEngine :
2019-05-10 19:37:16.142 INFO 6528 --- [ restartedMain] org.apache.catalina.core.StandardEngine :
2019-05-10 19:37:16.412 INFO 6528 --- [ restartedMain] org.apache.catalina.core.StandardEngine :

Using generated security password: bc81261c-8c48-4721-9df7-6d40a2d1d93b

2019-05-10 19:37:16.480 INFO 6528 --- [ restartedMain] org.apache.catalina.core.StandardEngine :
2019-05-10 19:37:16.538 INFO 6528 --- [ restartedMain] org.apache.catalina.core.StandardEngine :
2019-05-10 19:37:16.590 INFO 6528 --- [ restartedMain] org.apache.catalina.core.StandardEngine :
2019-05-10 19:37:16.595 INFO 6528 --- [ restartedMain] org.apache.catalina.core.StandardEngine :
```

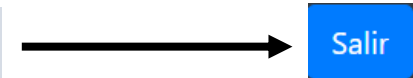
An arrow points from the text 'La contraseña es generada automáticamente y es desplegada en el log de la aplicación (console)' to the generated password in the console log.

# Spring Security – Cerrar Sesión

- La configuración de Spring Security por defecto también incluye la funcionalidad para cerrar la sesión en una aplicación web.
- Para incluir esta funcionalidad, en una vista, solo hay que agregar un link HTML a la URL `/logout`.

→ Utilizando thymeleaf el link quedaría así:

```
<a class="btn btn-primary" th:href="@{/logout}">Salir</a>
```



→ Al hacer clic en el botón aparecerá un cuadro de diálogo para confirmar:

Are you sure you  
want to log out?

Log Out



Please sign in

You have been signed out

Username

Password

Sign in

# Recuperar usuarios y roles de una Base de datos (MySQL)

➤ Hasta ahora en nuestra aplicación hemos tenido almacenado el usuario en memoria.

## ✓ Ventajas

- Configuración rápida
  - Usuario por defecto.
  - Declaramos el usuario y contraseña en el archivo `application.properties`.
- Adecuada para aplicaciones sencillas (solo existirá un usuario para toda la aplicación)

## ✓ Desventajas

- No es posible que el usuario pueda cambiar su contraseña.

➤ Spring Security permite recuperar los usuarios y contraseñas en una base de datos relacional como MySQL (Avanzado).

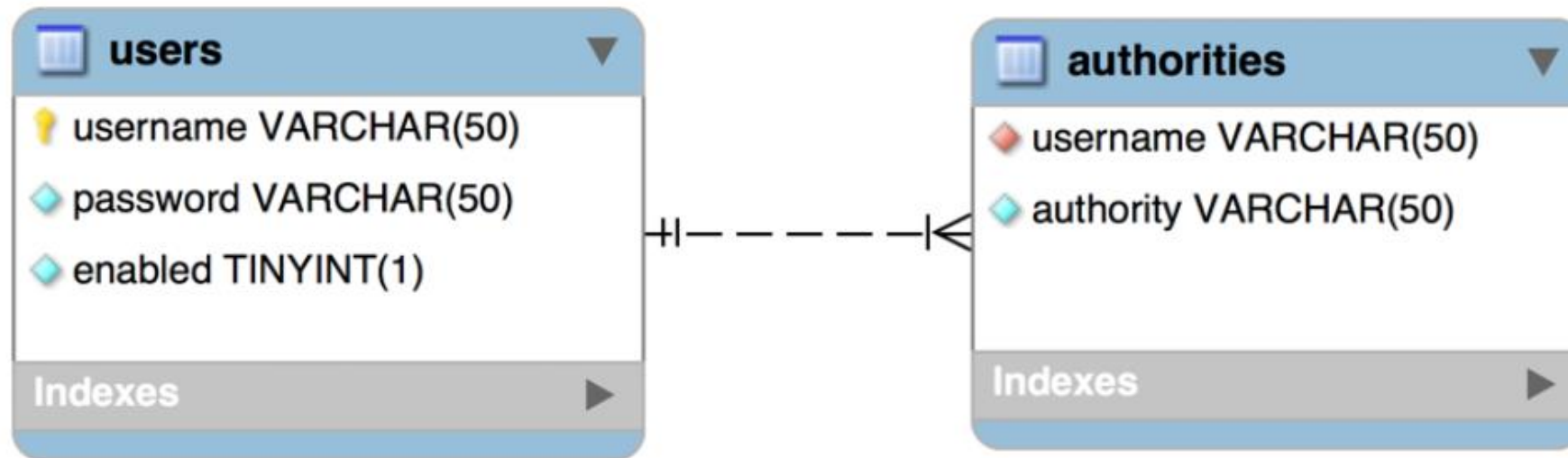
## ✓ Ventajas

- Se pueden agregar N usuarios en tiempo de ejecución (no es necesario reiniciar Apache Tomcat).
- Configuración adecuada para aplicaciones con muchos usuarios.
- Se puede crear un formulario (CRUD) para agregar usuarios.
- Solo se requieren dos tablas (usuarios y roles).
  - Se puede usar la base de datos por defecto.
- Podemos crear nuestra propia estructura de la base de datos.
  - Ideal para usar nuestras tablas de usuarios de acuerdo al contexto de nuestro proyecto.
  - El desarrollador es responsable de escribir las sentencias SQL para estas tablas.



# Recuperar usuarios y roles de una Base de datos (MySQL)

Esquema por defecto de la base de datos para Spring Security.



# Spring Security – Leer Usuario y Roles desde BD

➤ Configuración de Spring Security (Datasource) – Base de datos por defecto.

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

    @Autowired
    private DataSource dataSource;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.jdbcAuthentication().dataSource(dataSource);
    }
}
```

# Spring Security – Leer Usuario y Roles desde BD

➤ Configuración de Spring Security (Datasource) – Base de datos personalizada.

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

    @Autowired
    private DataSource dataSource;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {

        auth.jdbcAuthentication().dataSource(dataSource)
            .usersByUsernameQuery("select username, password, estatus from Usuarios where username=?")
            .authoritiesByUsernameQuery("select u.username, p.perfil from UsuarioPerfil up " +
                "inner join Usuarios u on u.id = up.idUsuario " +
                "inner join Perfiles p on p.id = up.idPerfil " +
                "where u.username = ?");

    }
}
```

# Spring Security – Personalizar accesos a URLs

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

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

        http.authorizeRequests()
            // Los recursos estáticos no requieren autenticación
            .antMatchers(
                "/bootstrap/**",
                "/images/**",
                "/tinymce/**",
                "/logos/**").permitAll()

            // Las vistas públicas no requieren autenticación
            .antMatchers("/",
                "/signup",
                "/search",
                "/vacantes/view/**").permitAll()

            // Todas las demás URLs de la Aplicación requieren autenticación
            .anyRequest().authenticated()

            // El formulario de Login no requiere autenticacion
            .and().formLogin().permitAll();
    }
}
```