# seguridad, AUTENTIFICACIÓN Identificar quién está usando la aplicación y verificar que realmente es quien dice ser.

creado 20220719T1002 guardado 20220719T1602 impreso 20220719T1602

## Cómo funciona el proceso

En una determinada sesión web (nueva sesión = cerrar el navegador y volverlo a abrir) cualquier intento de acceder a un recurso que no esté expresamente etiquetado como .permitAll() en la configuración de seguridad de la aplicación, desata automáticamente el que se le presente al usuario la página de login.

Cuando el usuario rellena los datos de login (‘username’ y ‘password’) y clica sobre el botón de ‘submit’. Spring se encarga automáticamente de toda la validación. Tan solo hay un punto desde donde controlamos el proceso: MyUserDetailsService

Hasta que no esté correctamente identificado y validado, se le seguirá presentando la página de login cada vez que pida cualquier recurso no .permitAll().

Una vez identificado y validado, esa sesión ya tiene un “current User” (un ‘Principal’) asignado. Y, a partir de ahí, será ese el usuario que el sistema utilizará cada vez que deba hacer una comprobación de seguridad (ver el documento “seguridad, AUTORIZACIÓN”).

## Cómo se indica a quien vamos a dejar pasar y a quien no

Como se ha comentado, solo hemos de preparar una clase que implemente el interfaz ‘UserDetailsService’ . (ver ejemplos un poco más adelante)

E indicarle al sistema de seguridad automático de Spring cual es esa clase:

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.core.Ordered;

**import** org.springframework.core.annotation.Order;

**import** org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

**import** org.springframework.security.config.annotation.authentication.configuration.GlobalAuthenticationConfigurerAdapter;

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

@Configuration

**public** **class** SecurityConfiguration {

../..

@Order(Ordered.***HIGHEST\_PRECEDENCE***)

@Configuration

**protected** **static** **class** AuthenticationSecurity **extends** GlobalAuthenticationConfigurerAdapter {

@Autowired

**private** Credenciales credenciales;

@Override

**public** **void** init(AuthenticationManagerBuilder auth) **throws** Exception {

auth.userDetailsService(**new** MyUserDetailsService()).passwordEncoder(**new** BCryptPasswordEncoder());

}

}

}

### ejemplo de MyUserDetailsService, poniendo los usuarios a mano, tal cual:

**import** org.springframework.stereotype.Service;

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

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

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

**import** org.springframework.security.core.userdetails.UsernameNotFoundException;

**import** java.util.Optional;

@Service

**public** **class** MyUserDetailsService **implements** UserDetailsService {

**private** CredencialesService credenciales;

**public** MyUserDetailsService(Credenciales credenciales) {

**this**.credenciales = credenciales;

}

@Override

**public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {

// Se trata de determinar si alguien que clama ser "username" puede o no entrar a la aplicación

// ==> si puede: se han de devolver todos sus datos (UserDetails), para que el autentificador (AuthenticationManager) pueda verificarlos

// ==> si no puede: se ha de devolver una excepcion (UsernameNotFoundException), y el autentificador rechazará la entrada.

**switch** (username) {

**case** "Benganito":

String contraseña01 = **new** BCryptPasswordEncoder().encode("zaq");

ArrayList<GrantedAuthority> roles01 = **new** ArrayList<>();

roles01.add(**new** SimpleGrantedAuthority("ROLE\_ADMINISTRADOR"));

roles01.add(**new** SimpleGrantedAuthority("ROLE\_REPARTIDOR"));

**return** **new** User(username, contraseña01, roles01);

**case** "Fulanito":

String contraseña02 = **new** BCryptPasswordEncoder().encode("mko");

ArrayList<GrantedAuthority> roles02 = **new** ArrayList<>();

roles02.add(**new** SimpleGrantedAuthority("ROLE\_REPARTIDOR"));

**return** **new** User(username, contraseña02, roles02);

**case** "Zutanito":

String contraseña03 = **new** BCryptPasswordEncoder().encode("vfr");

ArrayList<GrantedAuthority> roles03 = **new** ArrayList<>();

roles03.add(**new** SimpleGrantedAuthority("ROLE\_CLIENTE"));

**return** **new** User(username, contraseña03, roles03);

**default**:

**throw** **new** UsernameNotFoundException("[" + username + "] no está en la lista de autorizados");

}

}

}

nota: Se han puesto tres usuarios, pero se podrian haber puesto más, o menos. Esta manera de definir los usuarios válidos no es factible “en producción”, pero si que puede resultar útil para hacer pruebas “en desarrollo”.

### ejemplo de MyUserDetailsService, poniendo los usuarios en una tabla de la base de datos:

**import** org.springframework.stereotype.Service;

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

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

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

**import** org.springframework.security.core.userdetails.UsernameNotFoundException;

**import** java.util.Optional;

@Service

**public** **class** MyUserDetailsService **implements** UserDetailsService {

@Autowired

**private** CredencialesService credenciales;

@Override

**public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {

Optional<Credencial> credencial = credenciales.get(username);

**if** (credencial.isPresent()) {

Credencial credencialEncontrada = credencial.get();

**return** **new** User(credencialEncontrada.getNombre(), credencialEncontrada.getContraseña(), credencialEncontrada.getRoles());

} **else** {

**throw** **new** UsernameNotFoundException("[" + username + "] no está en la lista de autorizados");

}

}

### ejemplo de MyUserDetailsService, poniendo los usuarios repartidos en más de una tabla de la base de datos:

**import** org.springframework.stereotype.Service;

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

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

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

**import** org.springframework.security.core.userdetails.UsernameNotFoundException;

**import** java.util.Optional;

@Service

**public** **class** MyUserDetailsService **implements** UserDetailsService {

@Autowired

**private** RepartidoresService repartidores;

@Autowired

**private** ClientesService clientes;

@Override

**public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {

Optional<Credencial> repartidor = repartidores.get(username);

**if** (repartidor.isPresent()) {

Repartidor repartidorEncontrado = repartidor.get();

ArrayList<GrantedAuthority> roles = **new** ArrayList<>();

roles.add(**new** SimpleGrantedAuthority("ROLE\_REPARTIDOR"));

**return** **new** User(repartidorEncontrado.getUsername(), repartidorEncontrado.getPassword(), roles);

} **else** {

Optional<Credencial> cliente = clientes.get(username);

**if** (cliente.isPresent()) {

Cliente clienteEncontrado = cliente.get();

ArrayList<GrantedAuthority> roles = **new** ArrayList<>();

roles.add(**new** SimpleGrantedAuthority("ROLE\_CLIENTE"));

**return** **new** User(clienteEncontrado.getUsername(), clienteEncontrado.getPassword(), roles);

} **else** {

**throw** **new** UsernameNotFoundException("[" + username + "] no está en la lista de autorizados");

}

}

}

}

### referencias:

[UserDetailsService (spring-security-docs 5.7.2 API)](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/userdetails/UserDetailsService.html)

[UserDetails (spring-security-docs 5.7.2 API)](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/userdetails/UserDetails.html)

[User (spring-security-docs 5.7.2 API)](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/userdetails/User.html)

nota: En lugar de new User(---), también podemos usar un builder para crear instancias de User:

[User.UserBuilder (spring-security-docs 5.7.2 API)](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/userdetails/User.UserBuilder.html).

### truco:

Para insertar lineas en una tabla cualquiera de la base de datos, (y para muchas más cosas que se nos ocurran), podemos usar esta manera de ejecutar código arbitrario al iniciar la aplicación. Utilizando un @Bean “CommandLineRunner” en la clase principal de la aplicación:

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.boot.CommandLineRunner;

**import** org.springframework.context.annotation.Bean;

**import** es.susosise.pruebas\_springboot.seguridad.Credenciales;

**import** es.susosise.pruebas\_springboot.seguridad.Credencial;

**import** es.susosise.pruebas\_springboot.seguridad.Credencial.Rol;

@SpringBootApplication

**public** **class** App {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(App.**class**, args);

}

@Bean

**public** CommandLineRunner insertarUnaCredencialDePruebasEnLaBaseDeDatos(Credenciales credenciales) {

**return** args -> {

**if**(credenciales.get("Benganito").isEmpty()) {

Credencial credencial = **new** Credencial();

credencial.setNombre("Benganito ");

credencial.setContraseña("zaq");

credencial.setEstaActiva(**true**);

credencial.asignarleUnRol(Rol.***ROLE\_REPARTIDOR***);

credencial.asignarleUnRol(Rol.***ROLE\_ADMINISTRADOR***);

credenciales.guardar(credencial);

}

**if**(credenciales.get("Fulanito").isEmpty()) {

Credencial credencial = **new** Credencial();

credencial.setNombre("Fulanito");

credencial.setContraseña("mko");

credencial.setEstaActiva(**true**);

credencial.asignarleUnRol(Rol.***ROLE\_REPARTIDOR***);

credenciales.guardar(credencial);

}

};

}

}

notas complementarias:

**import** java.util.ArrayList;

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

**import** org.springframework.security.core.GrantedAuthority;

**import** org.springframework.security.core.authority.SimpleGrantedAuthority;

**public** **void** setContraseña(String contraseña) {

**this**.contraseña = **new** BCryptPasswordEncoder().encode(contraseña);

}

**private** ArrayList<GrantedAuthority> roles;

**public** **enum** Rol {

***ROLE\_CURRELA***,

***ROLE\_ADMINISTRADOR***

}

**public** Credencial() {

roles = **new** ArrayList<>();

}

**public** **void** asignarleUnRol(Rol rol) {

**if**(!roles.contains(**new** SimpleGrantedAuthority(rol.toString()))) {

roles.add(**new** SimpleGrantedAuthority(rol.toString()));

}

}

**public** **void** retirarleUnRol(Rol rol) {

roles.remove(**new** SimpleGrantedAuthority(rol.toString()));

}

## Cómo se indica el aspecto que queremos tenga la pantalla de login

Hemos de proporcionar una plantilla .htm con un <form> que recoja dos campos:

* username
* password

(exactamente con esos nombres)

<!DOCTYPE html>

<html lang=*"es"* xmlns:th=*"http://www.thymeleaf.org"*>

<head>

<meta charset=*"UTF-8"*>

<title>login - Pruebas\_SpringBoot</title>

<link rel=*"stylesheet"* type=*"text/css"* th:href=*"@{/css/susosise.css}"*>

</head>

<body>

<h1>Esta es la pagina para identificar al usuario.</h1>

<form action=*"#"* th:action=*"@{/login}"* th:object=*"${usuario}"* method=*"post"*>

<div class=*"campoDeFormulario"*>

<label>Usuario:</label>

<input id=*"username"* type=*"text"* name=*"username"* required autocomplete=*"off"* />

</div>

<div class=*"campoDeFormulario"*>

<label>Contraseña:</label>

<input id=*"password"* type=*"password"* name=*"password"* required autocomplete=*"off"* />

</div>

<p th:if=*"${error != null}"* class=*"error"*>Credenciales no válidas, vuelva a intentarlo...</p>

<button type=*"submit"* class=*"botonDeFormulario"*>ENTRAR</button>

</form>

</body>

</html>

Y hemos de proporcionar el correspondiente punto de entrada (/login)(obligatoriamente se ha de llamar así) en un @Controller

**import** org.springframework.stereotype.Controller;

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

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.ui.Model;

@Controller

**public** **class** ControladorVistasApp {

@GetMapping("/login")

**public** String mostrarLaPaginaParaIdentificarUsuario(Model model, String error, String logout) {

model.addAttribute("error", error);

**return** "login";

}

nota: En ninguna parte de nuestro código hemos de preocuparnos de cuándo se llamará a ese punto de entrada.. (ver el siguiente apartado).

## Cómo funciona el proceso

En una determinada sesión web (nueva sesión = cerrar el navegador y volverlo a abrir). Cualquier intento de acceder a un recurso que no esté expresamente etiquetado como .permitAll(), desata automáticamente el que se le presente al usuario la página de login.

Cuando el usuario rellena los datos (‘username’ y ‘password’) y clica sobre el botón de ‘submit’. Spring se encarga automáticamente de toda la validación. Tan solo hay un punto desde donde controlamos el proceso: MyUserDetailsService

Este punto se encarga de:

* Decir si dejar pasar a ‘username’ (devuelve un UserDetails) o si no dejarle pasar (lanza una UsernameNotFoundException).
* Proporcionar la contraseña contra la que validará el ‘password’ tecleado, (del proceso de validación se encarga Spring).
* Proporciona los roles que se le asignarán al usuario en caso de que sea validado.

Hasta que la sesión no tenga un usuario correctamente identificado y validado, se le seguirá presentando automáticamente la página de login cada vez que pida cualquier recurso restringido.

Una vez identificado y validado. Es decir, una vez esa sesión ya tiene asignado un “current User” (un ‘Principal’). Ese “current User” será utilizado cada vez que se deba hacer una comprobación de seguridad.

Si nosotros queremos de hacer alguna comprobación desde nuestro propio código, también podemos acceder a la información de ese “current User”.

**private** **void** ejemplosDeComoAccederAlCurrentUser() {

Authentication auth = SecurityContextHolder.*getContext*().getAuthentication();

**if** (auth != **null**) {

String nombreDelUsuarioLogeado = auth.getName();

// para usarlo donde deseemos, por ejemplo:

**if** (nombreDelUsuarioLogeado.equals("Zutanito")) {

//hacer algo que le corresponde a Zutanito...

} **else** {

//hacer algo que le corresponde a cualquier otro usuario...

}

ArrayList<GrantedAuthority> rolesDelUsuarioLogeado = (ArrayList<GrantedAuthority>) auth.getAuthorities();

// para usarlo donde deseemos, por ejemplo:

**if** (rolesDelUsuarioLogeado.contains(**new** SimpleGrantedAuthority("ROLE\_CLIENTE"))) {

//hacer algo que le corresponde a un cliente

} **else** {

//hacer algo que le corresponde a cualquier otro usuario que no sea cliente...

}

}

}

[Authentication (Spring Security 3.0.0.RELEASE API)](https://docs.spring.io/spring-security/site/apidocs/org/springframework/security/core/Authentication.html)