SPRING BOOT + SECURITY + BOOTSTRAP

# SEGURIDAD CON SPRING BOOT + THYMELEAF Y MYSQL. CAPÍTULO 5. SEGURIDAD Y PRIVILEGIOS

En este nuevo post vamos a crear la clase que gestiona la seguridad del sitio y marcaremos algunos métodos del controlador de productos como métodos de acceso restringido a usuarios que tienen privilegios de administrador.

Lo primero será crear una nueva clase en el paquete config llamada CustomAccessDeniedHandler, que será la encargada de redirigir a los usuarios que no tengan privilegios de administrador a la vista correspondiente.

|  |  |
| --- | --- |
|  | package com.inezpre5.thymeleaf\_security.config; |
|  |  |
|  | import org.springframework.security.access.AccessDeniedException; |
|  | import org.springframework.security.web.access.AccessDeniedHandler; |
|  |  |
|  | import javax.servlet.ServletException; |
|  | import javax.servlet.http.HttpServletRequest; |
|  | import javax.servlet.http.HttpServletResponse; |
|  | import java.io.IOException; |
|  |  |
|  | public class CustomAccessDeniedHandler implements AccessDeniedHandler { |
|  |  |
|  | @Override |
|  | public void handle(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, AccessDeniedException e) throws IOException, ServletException { |
|  | httpServletResponse.sendRedirect(httpServletRequest.getContextPath() + "/forbidden"); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/cavanosa/8fdbed963b9510781dad38565e627918/raw/58ec2848cdddc9a352fba4220d344c93b0120a4c/customaccessdeniedhandler.java)[**customaccessdeniedhandler.java**](https://gist.github.com/cavanosa/8fdbed963b9510781dad38565e627918#file-customaccessdeniedhandler-java) hosted with  by [**GitHub**](https://github.com/)

Una clase simple donde las haya, lo único que hace es llevar a los usuarios a la página”forbidden” en aquellos casos en los que pretendan acceder a una vista para la que no poseenprivilegios.

Y ya el paquete security, crearemos la clase que gestiona la seguridad del sitio, le llamaremos SecurityConfig. Mostramos el código antes de explicar los métodos:

|  |  |
| --- | --- |
|  | package com.inezpre5.thymeleaf\_security.security; |
|  |  |
|  | import com.inezpre5.thymeleaf\_security.config.CustomAccessDeniedHandler; |
|  | import com.inezpre5.thymeleaf\_security.security.service.UsuarioDetailsServiceImpl; |
|  | import org.springframework.beans.factory.annotation.Autowired; |
|  | import org.springframework.context.annotation.Bean; |
|  | import org.springframework.context.annotation.Configuration; |
|  | import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder; |
|  | import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity; |
|  | 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; |
|  | import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder; |
|  | import org.springframework.security.crypto.password.PasswordEncoder; |
|  | import org.springframework.security.web.access.AccessDeniedHandler; |
|  | import org.springframework.security.web.util.matcher.AntPathRequestMatcher; |
|  |  |
|  | @Configuration |
|  | @EnableWebSecurity |
|  | @EnableGlobalMethodSecurity(prePostEnabled = true) |
|  | public class SecurityConfig extends WebSecurityConfigurerAdapter { |
|  |  |
|  | @Autowired |
|  | UsuarioDetailsServiceImpl usuarioDetailsService; |
|  |  |
|  | @Bean |
|  | PasswordEncoder passwordEncoder(){ |
|  | return new BCryptPasswordEncoder(); |
|  | } |
|  |  |
|  | @Bean |
|  | AccessDeniedHandler accessDeniedHandler(){ |
|  | return new CustomAccessDeniedHandler(); |
|  | } |
|  |  |
|  | @Override |
|  | protected void configure(AuthenticationManagerBuilder auth) throws Exception { |
|  | auth.userDetailsService(usuarioDetailsService).passwordEncoder(passwordEncoder()); |
|  | } |
|  |  |
|  | @Override |
|  | protected void configure(HttpSecurity http) throws Exception { |
|  | http.authorizeRequests() |
|  | .antMatchers("/bootstrap/\*\*", "/fontawesome/\*\*", "/img/\*\*").permitAll() |
|  | .antMatchers("/", "/index", "/usuario/registro", "/fragments", "/error", "/forbidden").permitAll() |
|  | .antMatchers("/usuario/registrar").permitAll() |
|  | .anyRequest().authenticated() |
|  | .and() |
|  | .formLogin() |
|  | .loginProcessingUrl("/signin") |
|  | .loginPage("/login").permitAll() |
|  | .defaultSuccessUrl("/index") |
|  | .usernameParameter("textUsername") |
|  | .passwordParameter("textPassword") |
|  | .and() |
|  | .exceptionHandling().accessDeniedHandler(accessDeniedHandler()) |
|  | .and() |
|  | .logout().logoutRequestMatcher(new AntPathRequestMatcher("/logout")).logoutSuccessUrl("/login?logout").permitAll() |
|  | .deleteCookies("JSESSIONID") |
|  | .and() |
|  | .rememberMe().tokenValiditySeconds(2592000).key("secret").rememberMeParameter("checkRememberMe"); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/cavanosa/9e5cca7be34fe6c3913e788ad0c2fca7/raw/338b65d51cbb84c661923d94a1f2a52424c2ead0/securityconfig.java)[**securityconfig.java**](https://gist.github.com/cavanosa/9e5cca7be34fe6c3913e788ad0c2fca7#file-securityconfig-java) hosted with  by [**GitHub**](https://github.com/)

Lo primero de todo son las anotaciones. Con @EnableWebSecurity indicamos que es la clase con la que se configura la seguridad (debe ir acompañada de la anotación @Configuration)  
@EnableGlobalMethodSecurity(prePostEnabled = true) la utilizamos para anotar algunos métodos con @PreAuthorize(hasRole=ADMIN), así sólo podrán ser invocados por el adiministrador.

Vamos a inyectar un objeto UsuarioDetailsServiceImpl para la autenticación.

Tendremos dos @Bean, un passwordEncoder para cifrar las contraseñas y un AccessDeniedHandler, que devolverá un objeto de la clase CustomAccessDeniedHandler para redirigir a la página de no autorizado (forbidden).

En esta clase, que hereda de WebSecurityConfigurerAdapter vamos a sobreeescribir dos métodos configure:

El primero, el que lleva como parámetro un AuthenticationManagerBuilder es el que se encarga de autenticar a los usuarios con nuestro UsuarioDetailsServiceImpl y de cifrar su contraseña con PasswordEncoder.

Por último, tenemos el método configure con el parámetro HttpSecurity que es en el que definimos las reglas de seguridad del sitio:

* con antMatchers().permitAll se indica qué vistas y qué documentos/carpetas se muestran a los usuarios sin autenticar, en nuestro caso las imágenes, los estilos css de Bootstrap y Font Awesome, así como las vistas de login, registro e index; para el resto de las vistas será necesario haber iniciado sesión (anyRequest().authenticated()).
* la página de login será la vista login (loginPage(“/login”) y la encargada de procesar la autenticación será signin (loginProcessingUrl(“/signin”). Una vez iniciada la sesión se nos redirige a index (defaultSuccessUrl(“/index)
* los parámetros de nombre de usuario y contraseña son, respectivamente textUsername y textPassword. Hay que poner estos nombres a los inputs correspondientes en el formulario del login.
* el acceso denegado lo controla el objeto accessDeniedhandler. (exceptionHandling().accessDenienhandler())
* al hacer logout se nos redirige de nuevo a login, pero lleva como parámetro al logout (login?logout) para que muestre el mensaje de despedida.
* la sesión durará 30 días (2592000 segundos) en caso de que marquemos la casilla rememberMe.

Lo siguiente que vamos a a hacer es anotar varios métodos  de la clase ProductoController con @Preauthorize(serán los métodos nuevo, crear, editar, actualizar y eliminar. Quedarán como sigue:

|  |  |
| --- | --- |
|  | @PreAuthorize("hasRole('ADMIN')") |
|  | @GetMapping("/nuevo") |
|  | public String nuevo(){ |
|  | return "producto/nuevo"; |
|  | } |
|  |  |
|  | @PostMapping("/crear") |
|  | public ModelAndView crear(String nombre, double precio){ |
|  | if(StringUtils.isBlank(nombre)){ |
|  | ModelAndView mv = new ModelAndView(); |
|  | mv.addObject("error", "El nombre no puede estar vacío"); |
|  | mv.setViewName("producto/nuevo"); |
|  | return mv; |
|  | } |
|  | if(precio < 1){ |
|  | ModelAndView mv = new ModelAndView(); |
|  | mv.addObject("error", "El precio debe ser mayor que cero"); |
|  | mv.setViewName("producto/nuevo"); |
|  | return mv; |
|  | } |
|  | if(productoService.existsProductoNombre(nombre)){ |
|  | ModelAndView mv = new ModelAndView(); |
|  | mv.addObject("error", "Ese nombre ya existe"); |
|  | mv.setViewName("producto/nuevo"); |
|  | return mv; |
|  | } |
|  | Producto producto = new Producto(nombre, precio); |
|  | productoService.save(producto); |
|  | return new ModelAndView("redirect:/producto"); |
|  | } |
|  |  |
|  | @PreAuthorize("hasRole('ADMIN')") |
|  | @GetMapping("/editar/{id}") |
|  | public ModelAndView editar (@PathVariable("id") long id){ |
|  | if(!productoService.existsProductoId(id)){ |
|  | return new ModelAndView("redirect:/producto"); |
|  | } |
|  | ModelAndView mv = new ModelAndView(); |
|  | Producto producto = productoService.getProductoId(id).get(); |
|  | mv.addObject("producto", producto); |
|  | mv.setViewName("producto/editar"); |
|  | return mv; |
|  | } |
|  |  |
|  | @PreAuthorize("hasRole('ADMIN')") |
|  | @PostMapping("/actualizar") |
|  | public ModelAndView actualizar(@RequestParam("nombre") String nombre, @RequestParam("precio") double precio, @RequestParam("id")long id){ |
|  | if(!productoService.existsProductoId(id)){ |
|  | ModelAndView mv = new ModelAndView(); |
|  | mv.setViewName("producto/lista"); |
|  | return mv; |
|  | } |
|  | Producto producto = productoService.getProductoId(id).get(); |
|  | if(productoService.existsProductoNombre(nombre) && productoService.getProductoNombre(nombre).get().getId()!=id){ |
|  | ModelAndView mv = new ModelAndView(); |
|  | mv.addObject("error", "Ese nombre ya existe"); |
|  | mv.addObject("producto", producto); |
|  | mv.setViewName("producto/editar"); |
|  | return mv; |
|  | } |
|  | if(StringUtils.isBlank(nombre)){ |
|  | ModelAndView mv = new ModelAndView(); |
|  | mv.addObject("error", "El nombre no puede estar vacío"); |
|  | mv.addObject("producto", producto); |
|  | mv.setViewName("producto/editar"); |
|  | return mv; |
|  | } |
|  | if(precio < 1){ |
|  | ModelAndView mv = new ModelAndView(); |
|  | mv.addObject("error", "El precio debe ser mayor que 0"); |
|  | mv.addObject("producto", producto); |
|  | mv.setViewName("producto/editar"); |
|  | return mv; |
|  | } |
|  | producto.setNombre(nombre); |
|  | producto.setPrecio(precio); |
|  | productoService.save(producto); |
|  | return new ModelAndView("redirect:/producto"); |
|  | } |
|  |  |
|  | @PreAuthorize("hasRole('ADMIN')") |
|  | @GetMapping("/eliminar/{id}") |
|  | public ModelAndView eliminar(@PathVariable("id") long id){ |
|  | if(productoService.existsProductoId(id)) { |
|  | productoService.delete(id); |
|  | return new ModelAndView("redirect:/producto"); |
|  | } |
|  | return null; |
|  | } |

[**view raw**](https://gist.github.com/cavanosa/3290c19c23a6c7eb0cc609a51a389c7e/raw/ea833d264afbcb3bae8769f30f48d8445231032b/productocontroller.java)[**productocontroller.java**](https://gist.github.com/cavanosa/3290c19c23a6c7eb0cc609a51a389c7e#file-productocontroller-java) hosted with  by [**GitHub**](https://github.com/)

**Creando al usuario admin y los roles:**

Y para finalizar el capítulo implementaremos una última clase con la que se insertarán en las base de datos los roles de USER y ADMIN, así como un usuario con privilegios de administrador. Le llamamos AdminCreate y la creamos en el paquete config:

|  |  |
| --- | --- |
|  | package com.inezpre5.thymeleaf\_security.config; |
|  |  |
|  | import com.inezpre5.thymeleaf\_security.enums.RolNombre; |
|  | import com.inezpre5.thymeleaf\_security.model.Rol; |
|  | import com.inezpre5.thymeleaf\_security.model.Usuario; |
|  | import com.inezpre5.thymeleaf\_security.service.RolService; |
|  | import com.inezpre5.thymeleaf\_security.service.UsuarioService; |
|  | import org.springframework.beans.factory.annotation.Autowired; |
|  | import org.springframework.boot.CommandLineRunner; |
|  | import org.springframework.security.crypto.password.PasswordEncoder; |
|  | import org.springframework.stereotype.Service; |
|  |  |
|  | import java.util.HashSet; |
|  | import java.util.Set; |
|  |  |
|  | @Service |
|  | public class AdminCreate implements CommandLineRunner { |
|  |  |
|  | @Autowired |
|  | UsuarioService usuarioService; |
|  |  |
|  | @Autowired |
|  | RolService rolService; |
|  |  |
|  | @Autowired |
|  | PasswordEncoder passwordEncoder; |
|  |  |
|  | @Override |
|  | public void run(String... args) throws Exception { |
|  | Rol rolAdmin = new Rol(); |
|  | rolAdmin.setRolNombre(RolNombre.ROLE\_ADMIN); |
|  | Rol rolUser = new Rol(); |
|  | rolUser.setRolNombre(RolNombre.ROLE\_USER); |
|  | Usuario adminUser = new Usuario("admin", passwordEncoder.encode("admin")); |
|  | Set<Rol> roles = new HashSet<>(); |
|  | roles.add(rolAdmin); |
|  | roles.add(rolUser); |
|  | adminUser.setRoles(roles); |
|  |  |
|  | //insert |
|  | rolService.save(rolAdmin); |
|  | rolService.save(rolUser); |
|  | usuarioService.save(adminUser); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/cavanosa/3fa4211589427e0a55f98705a1d639a3/raw/4e7468d7aa242082743944cb2dbb0f91353e11bc/admincreate.java)[**admincreate.java**](https://gist.github.com/cavanosa/3fa4211589427e0a55f98705a1d639a3#file-admincreate-java) hosted with  by [**GitHub**](https://github.com/)

Esta clase es un servicio que ejecuta la línea de comandos. Inyectamos los servicios de Rol y de Usuario y creamos los roles de usuario (ROLE\_USER) y de administrador (ADMIN\_USER). A continuación se crea un usuario de nombre admin y de contraseña admin, esta última se cifra con passwordEncoder. Luego sle asignamos los roles de usuario y administrador con setRoles(); y por último se guarda todo en la base de datos.

**OJO, MUY IMPORTANTE:** esta clase sólo se puede ejecutar una vez ya que a la segunda comenzará a lanzar excepciones (los roles tienen nombre único y los usuarios también). Por ello, una vez que ejecutemos deberemos eliminar esta clase o bien comentar todo el código del método run.

Y damos por concluido el post, en el siguiente crearemos las vistas.