SPRING BOOT + SECURITY + BOOTSTRAP

# SEGURIDAD CON SPRING BOOT + THYMELEAF Y MYSQL. CAPÍTULO 3. CLASES DEL PRODUCTO

En esta entrada vamos a crear las clases que gestionan los productos, las mismas de los tutoriales de JWT: el modelo, el repositorio JPA, el servicio y el controlador.

**Modelo:**

En el paquete modelo creamos la clase Producto.

|  |  |
| --- | --- |
|  | package com.inezpre5.thymeleaf\_security.model; |
|  |  |
|  | import javax.persistence.\*; |
|  | import javax.validation.constraints.Min; |
|  | import javax.validation.constraints.NotBlank; |
|  | import javax.validation.constraints.NotNull; |
|  |  |
|  | @Entity |
|  | public class Producto { |
|  | @Id |
|  | @GeneratedValue(strategy = GenerationType.IDENTITY) |
|  | private long id; |
|  | @NotBlank |
|  | @Column(unique = true) |
|  | private String nombre; |
|  | @NotNull |
|  | @Min(value = 1) |
|  | private double precio; |
|  |  |
|  | public Producto() { |
|  | } |
|  |  |
|  | public Producto(@NotBlank String nombre, @NotNull @Min(value = 1) double precio) { |
|  | this.nombre = nombre; |
|  | this.precio = precio; |
|  | } |
|  |  |
|  | public long getId() { |
|  | return id; |
|  | } |
|  |  |
|  | public void setId(long id) { |
|  | this.id = id; |
|  | } |
|  |  |
|  | public String getNombre() { |
|  | return nombre; |
|  | } |
|  |  |
|  | public void setNombre(String nombre) { |
|  | this.nombre = nombre; |
|  | } |
|  |  |
|  | public double getPrecio() { |
|  | return precio; |
|  | } |
|  |  |
|  | public void setPrecio(double precio) { |
|  | this.precio = precio; |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/cavanosa/90ae5998d11f1a02f623c9ff2bc48211/raw/a8fd62420987840163f22ccbed8c36da6e67d329/producto.java)[**producto.java**](https://gist.github.com/cavanosa/90ae5998d11f1a02f623c9ff2bc48211#file-producto-java) hosted with  by [**GitHub**](https://github.com/)

**Repositorio:**

En el paquete repositorio creamos la interfaz ProductoRepository:

|  |  |
| --- | --- |
|  | package com.inezpre5.thymeleaf\_security.repository; |
|  |  |
|  | import com.inezpre5.thymeleaf\_security.model.Producto; |
|  | import org.springframework.data.jpa.repository.JpaRepository; |
|  | import org.springframework.stereotype.Repository; |
|  |  |
|  | import java.util.Optional; |
|  |  |
|  | @Repository |
|  | public interface ProductoRepository extends JpaRepository<Producto, Long> { |
|  | boolean existsByNombre(String nombre); |
|  | Optional<Producto> findByNombre(String nombre); |
|  | } |

[**view raw**](https://gist.github.com/cavanosa/10c07c8d329e72b2bceec156aca9ea03/raw/f0465f04d7cfbaeaf1798c4d0536e4ea7c7d8831/productorepository.java)[**productorepository.java**](https://gist.github.com/cavanosa/10c07c8d329e72b2bceec156aca9ea03#file-productorepository-java) hosted with  by [**GitHub**](https://github.com/)

**Servicio:**

En el paquete servicio creamos la clase ProductoService:

|  |  |
| --- | --- |
|  | package com.inezpre5.thymeleaf\_security.service; |
|  |  |
|  | import com.inezpre5.thymeleaf\_security.model.Producto; |
|  | import com.inezpre5.thymeleaf\_security.repository.ProductoRepository; |
|  | import org.springframework.beans.factory.annotation.Autowired; |
|  | import org.springframework.stereotype.Service; |
|  |  |
|  | import java.util.List; |
|  | import java.util.Optional; |
|  |  |
|  | @Service |
|  | public class ProductoService { |
|  |  |
|  | @Autowired |
|  | ProductoRepository productoRepository; |
|  |  |
|  | public List<Producto> getList(){ |
|  | return productoRepository.findAll(); |
|  | } |
|  |  |
|  | public Optional<Producto> getProductoId(long id){ |
|  | return productoRepository.findById(id); |
|  | } |
|  |  |
|  | public Optional<Producto> getProductoNombre(String nombre){ |
|  | return productoRepository.findByNombre(nombre); |
|  | } |
|  |  |
|  | public void save(Producto producto){ |
|  | productoRepository.save(producto); |
|  | } |
|  |  |
|  | public void delete(long id){ |
|  | productoRepository.deleteById(id); |
|  | } |
|  |  |
|  | public boolean existsProductoId(long id){ |
|  | return productoRepository.existsById(id); |
|  | } |
|  |  |
|  | public boolean existsProductoNombre(String nombre){ |
|  | return productoRepository.existsByNombre(nombre); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/cavanosa/4a592881f3a6a3acc0426d24316a27c6/raw/9ac48f7f722f9a5b392fafff644e0c860bdbc02c/productoservice.java)[**productoservice.java**](https://gist.github.com/cavanosa/4a592881f3a6a3acc0426d24316a27c6#file-productoservice-java) hosted with  by [**GitHub**](https://github.com/)

**Controlador:**

En el paquete controlador creamos la clase ProductoController: (¡pero qué repetitivo me está quedando el artículo!!)

|  |  |
| --- | --- |
|  | package com.inezpre5.thymeleaf\_security.controller; |
|  |  |
|  | import com.inezpre5.thymeleaf\_security.model.Producto; |
|  | import com.inezpre5.thymeleaf\_security.service.ProductoService; |
|  | import org.apache.commons.lang3.StringUtils; |
|  | import org.springframework.beans.factory.annotation.Autowired; |
|  | import org.springframework.stereotype.Controller; |
|  | import org.springframework.web.bind.annotation.\*; |
|  | import org.springframework.web.servlet.ModelAndView; |
|  |  |
|  | import java.util.List; |
|  |  |
|  | @Controller |
|  | @RequestMapping("/producto") |
|  | public class ProductoController { |
|  |  |
|  | @Autowired |
|  | ProductoService productoService; |
|  |  |
|  | @GetMapping |
|  | public ModelAndView lista(){ |
|  | ModelAndView mv = new ModelAndView("producto/lista"); |
|  | List<Producto> lista = productoService.getList(); |
|  | mv.addObject("productos", lista); |
|  | return mv; |
|  | } |
|  |  |
|  | @GetMapping("/detalle/{id}") |
|  | public ModelAndView detalle(@PathVariable("id") long id){ |
|  | if(!productoService.existsProductoId(id)) |
|  | return new ModelAndView("redirect:/producto"); |
|  | ModelAndView mv = new ModelAndView("/producto/detalle"); |
|  | Producto producto = productoService.getProductoId(id).get(); |
|  | mv.addObject("producto", producto); |
|  | return mv; |
|  | } |
|  |  |
|  | @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"); |
|  | } |
|  |  |
|  | @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; |
|  | } |
|  |  |
|  | @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"); |
|  | } |
|  |  |
|  | @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/077a509270ca9b38bff984fad039c5f9/raw/23f342a34e64f5f72e0c2211e4fd9b70d3a6aaf8/productocontroller.java)[**productocontroller.java**](https://gist.github.com/cavanosa/077a509270ca9b38bff984fad039c5f9#file-productocontroller-java) hosted with  by [**GitHub**](https://github.com/)

Bueno, el controlador es el más complicadillo de todos. En los artículos de JWT teníamos un @RestController, pero en el caso que nos ocupa tenemos un @Controller “a secas”. ¿Qué significa esto? Pues que los métodos no nos van a devolver un objeto json, nos van a dirigir a una vista -las plantillas de thymeleaf– a las cuales se les podrán pasar parámetros. Por esto utilizamos la clase ModelAnView.

Por ejemplo, en el primer método (el de obtener la lista de productos) creamos un objeto ModelAndView pasándole como parámetro al constructor el nombre de la plantilla -producto/lista en nuestro caso-; a continuación obtenemos la lista de productos con el servicio y se la asignamos al ModelAndView con el método addObject; finalmente el método devuelve el ModelAndView.

En el método “detalle” le pasamos el id del producto como parámetro y, en caso de no existir ningún producto con ese id, nos redirigirá a la lista.

El método nuevo devuelve simplemente una cadena pues la vista es un formulario vacío que no recibe ningún parámetro.

Una vez que en el formulario se rellenan los campos y se pulsa el botón submit, se llama al método crear (que es tipo Post). En dicho método primero se comprueba que los valores sean correctos:

* el nombre no es una cadena vacía ni una cadena de longitud 0 ni uno o más espacios en blanco (esto lo comprueba el método estático StringUtils.isBlank())
* el precio es mayor que 0
* el nombre no existe, es decir, no hay ningún producto con ese nombre porque es un atributo único.

Una vez que se ha comprobado todo lo anterior, se crea un nuevo producto, se llama al servicio para que lo guarde en la base de datos y se redirige a la vista de la lista de productos: Observad que aquí el parámetro del ModelAndView es “redirect:/producto”. Lo que redirect hace es llamar al @RequestMaping “producto”, que en esta clase es el que invoca al método lista().

El método editar es como el método nuevo, nos dirige a la vista de un formulario, pero esta vista sí va a recibir un objeto de la clase producto, el cual lo vamos a obtener a partir de su id. Los campos del formulario estarán rellenos con los valores del producto a modificar. Es por esto por lo que aquí si utilizamos un ModelAndView que recibe el nombre de la vista y el producto a editar.

El método actualizar se ejecutará al pulsar el botón submit del formulario de edición. Es muy similar al método crear, solo que aquí en vez de crear uno nuevo se actualiza uno existente. Con la línea:

if (productoService.existsProductoNombre(nombre) && productoService.getProductoNombre(nombre).get().getId( )!= id)

comprobamos si hay un producto con ese nombre y su id es diferente a la del producto a editar. En caso afirmativo, nos redirigirá la formulario pues no se puede repetir el nombre de un producto.

Y por último, el método eliminar comprueba que exista un producto con el id y lo elimina.

Bien, hasta aquí el artículo. En el siguiente post crearemos las clases Usuario, y Rol así como UsuarioDetailsService y UsuarioPrincipal, que son clases propias de Spring Security.