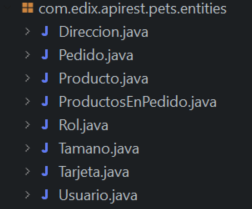
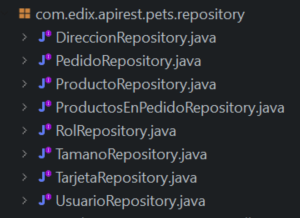
Lo primero que se ha hecho es definir las clases que corresponden a las tablas de la base de datos. Están en el paquete com.edix.apirest.pets.entities. Las clases son objetos que representan datos que se pueden guardar en la base de datos. Una clase representa una tabla guardada en una base de datos, y cada objeto de una clase representa una fila en la tabla. Cada clase tiene atributos que se relacionan con las columnas de cada tabla, un constructor vacío, los métodos para obtener (getters) y modificar (setters) los atributos, el hashcode y el equals.



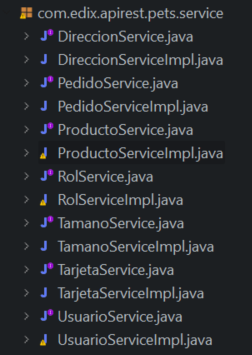
Luego se crean los ‘repository’, estos estan ubicados en el paquete com.edix.apirest.pets.repository, y desempeñan un papel fundamental al proporcionar interfaces que permiten la conexión y la interacción entre el sistema y la base de datos. Estas interfaces, que extienden las clases proporcionadas por JPA (Java Persistence API), actúan como contratos abstractos que definen los métodos y operaciones necesarios para gestionar la información almacenada en la base de datos.

Las interfaces en el 'repository' ofrecen una capa de abstracción que oculta los detalles de la implementación subyacente y permite a las clases de servicios y controladores trabajar con entidades y objetos de dominio en lugar de preocuparse por las consultas y manipulaciones específicas de la base de datos.

Cada interfaz en 'repository' define métodos que encapsulan operaciones comunes, como buscar, borrar, actualizar o crear registros en la base de datos. Estos métodos se basan en convenciones de nombres y anotaciones proporcionadas por JPA, que se encargan de generar automáticamente las consultas SQL correspondientes para interactuar con la base de datos.



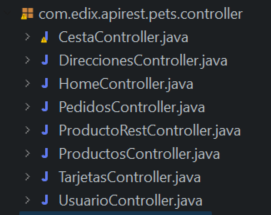
Seguidamente se crean los ‘service’, que están en el paquete com.edix.apirest.pets.service, los cuales se utilizan para albergar las interfaces que definen los métodos que encapsulan la lógica del negocio. Estas interfaces actúan como contratos que especifican las operaciones que se pueden realizar en el sistema. Dentro de estas interfaces, se definen métodos que realizan tareas como la manipulación de datos, la ejecución de cálculos o la interacción con otros componentes del sistema. Estas interfaces proporcionan una capa de abstracción entre la capa de presentación y la capa de acceso a datos, permitiendo una separación clara de responsabilidades y facilitando la modularidad y el mantenimiento del código. En este mismo paquete se encuentran los 'serviceImpl' que albergan las clases que implementan la lógica del negocio definida en las interfaces de los 'service'. Estas clases son responsables de aplicar y ejecutar los métodos declarados en las interfaces, llevando a cabo las operaciones específicas que se requieren para satisfacer los requisitos funcionales del sistema.



A continuación se crean los ‘controllers’, ubicado en el paquete com.edix.apirest.pets.controller, estos juegan un rol crucial al actuar como la capa de conexión entre el backend y las solicitudes realizadas desde fuera de la aplicación, ya sea a través de una interfaz de usuario o servicios externos.

En los controllers se encuentran las clases que se encargan de recibir y gestionar las solicitudes HTTP entrantes, así como de enviar las respuestas correspondientes. Estas clases actúan como controladores, ya que se encargan de orquestar la lógica necesaria para procesar las solicitudes y coordinar las acciones pertinentes en el backend.

Dentro de los métodos de los controladores, se realiza la validación de los datos ingresados, se invocan los servicios correspondientes para ejecutar la lógica de negocio y se generan las respuestas adecuadas, que pueden incluir datos recuperados de la base de datos, mensajes de error o información procesada.



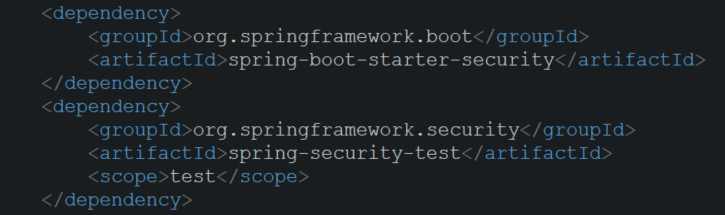
Para la parte de seguridad agregamos el paquete Spring Security que es una biblioteca incluida en el conjunto de herramientas de Spring que engloba una variedad de subproyectos o módulos que ofrecen funcionalidades adicionales para aplicaciones que lo consideren necesario. En particular, Spring Security se enfoca en unificar todas las características relacionadas con el control de acceso de usuarios en proyectos de Spring, que es para lo que lo usamos en este proyecto

El control de acceso permite restringir las acciones que un conjunto específico de usuarios o roles puede llevar a cabo en una aplicación. En este sentido, Spring Security se encarga de supervisar las llamadas a la lógica empresarial o limitar el acceso a ciertas URL en peticiones HTTP.

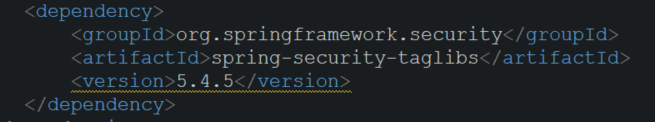
Para agregar la capa de seguridad realizamos una configuración sobre la aplicación indicando a Spring Security cómo debe comportarse la capa de seguridad.

El primer paso consiste en agregar las dependencias del paquete "spring-boot-starter-security". Esta inclusión traerá automáticamente las demás bibliotecas JAR necesarias para que Spring pueda implementar los mecanismos de seguridad requeridos. Simplemente con añadir esta dependencia Spring Boot protegerá por defecto todo el acceso a la aplicación, evitando que usuarios no identificados puedan invocar cualquier controlador.

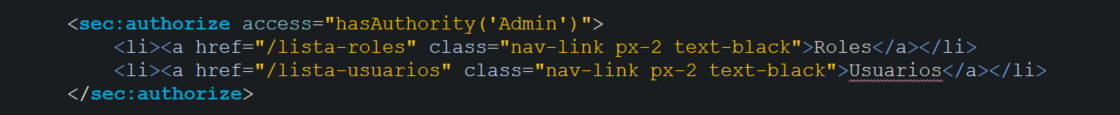
En el archivo pom.xml se pueden ver las dependencias que añade.



Adicinalmente también añadimos la dependencia de las ‘taglibs’ que nos proveen acceso a la información de seguridad del los usuarios y posibilitan el poder filtrar el contenido que se muestra al usuario en base a sus privilegios o tipo de usuario del que se trate.

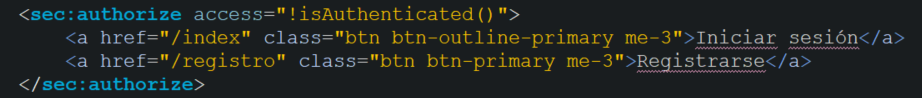


La etiqueta ‘authorize’ nos sirve para filtrar el contenido que se mostrará en base a los privilegios del usuario, por ejemplo, el contenido que solo será mostrado a aquellos usuarios con privilegios de administrador, esta etiqueta la utilizaremos de la siguiente manera:



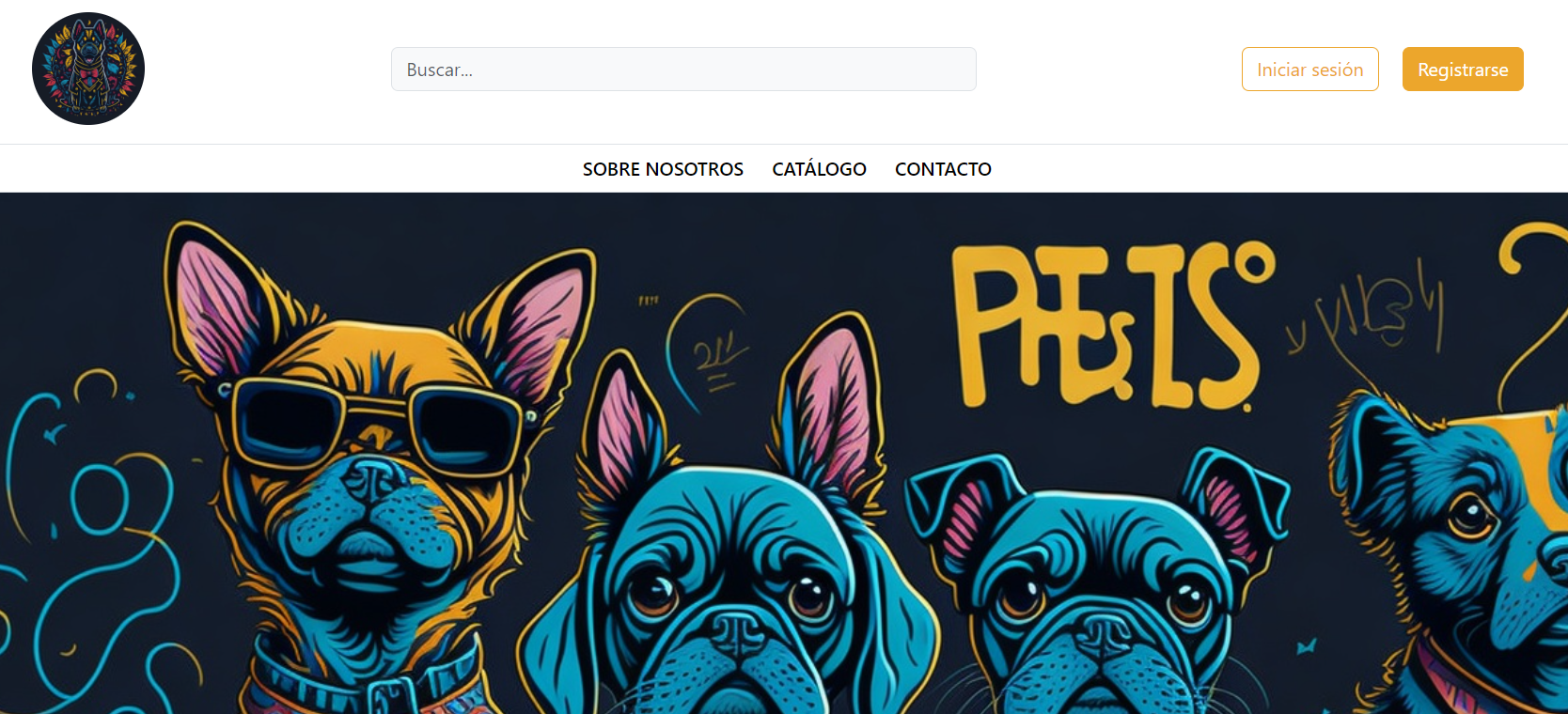
Aqui podemos ver como los elementos de ‘Roles’ y ‘Usuarios’ de la lista sólo los verán aquellos que tengan privilegios de ‘Admin’.

Otro ejemplo sería para los usuarios que no están registrados, como se puede ver a continuación:

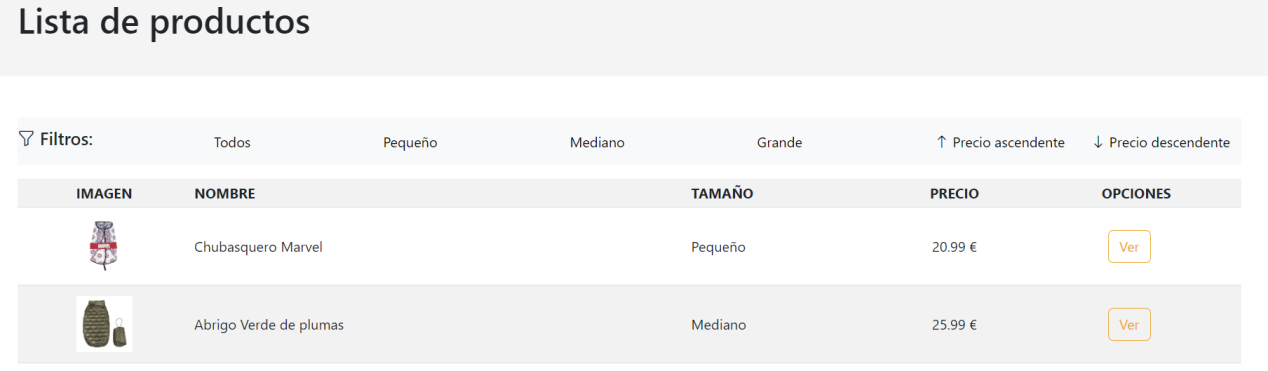


Con esto los usuarios que entran sin estar registrados o haber iniciado sesión, se les muestran las opciones de ‘iniciar sesión’ y ‘Registrarse’

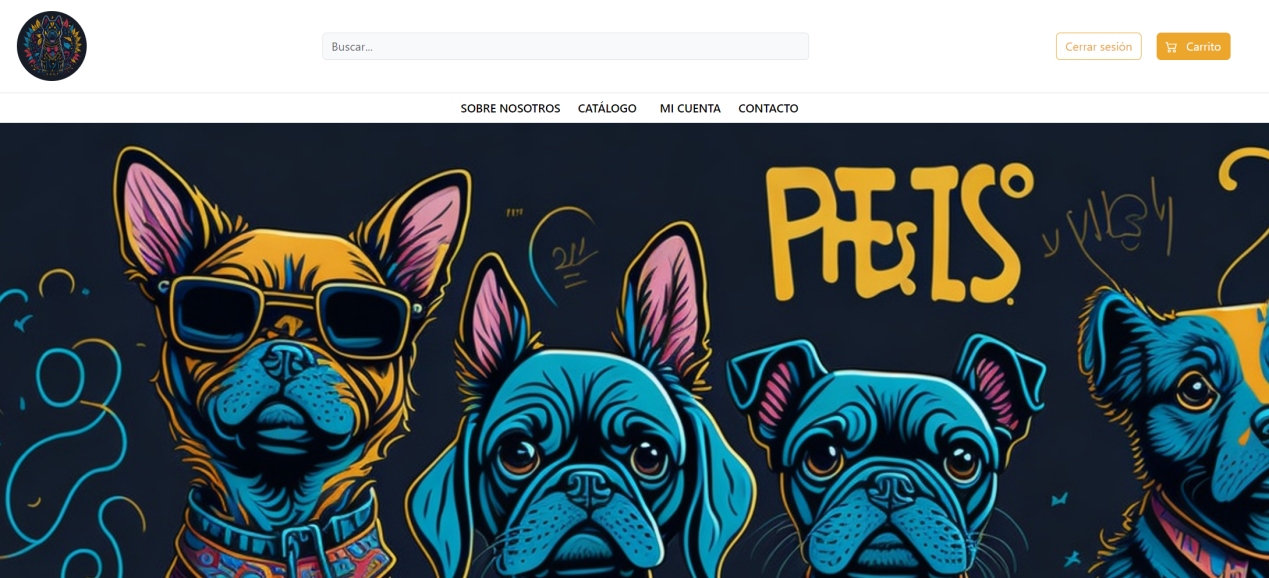
En nuestro caso tenemos dos roles, el usuario y el administrador, además también esta la persona que accede sin estar registrada o logueada, para esta última la aplicación solo mostrará en el menú la siguiente información.



El usuario tiene la opción de iniciar sesión y registrarse, en la opción de catálogo se puede ver el catálogo de productos y el detalle de cada uno.

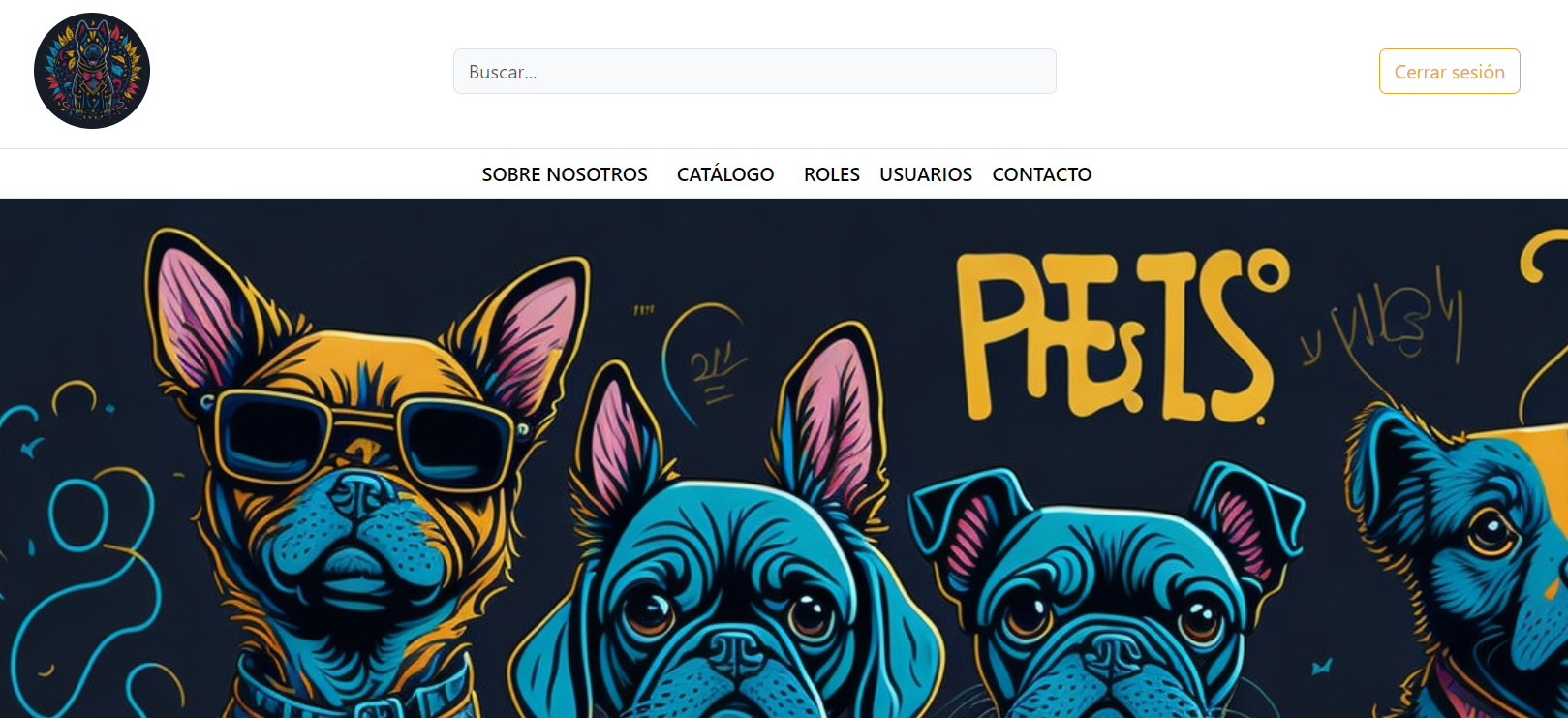


Para el caso de los usuarios registrados muestra la siguiente información.



Como se puede observar añade al menú la opción de ‘mi cuenta’ dónde están todos los datos personales del usuario y es donde podrá modificarlos de así quererlo y la opción de cerra sesión.

Luego esta el caso del administrador que nos muestra lo siguiente.



Como se puede apreciar, añade la opción de roles y de usuarios, ya que el administrador va a tener acceso a toda la información de la aplicación y a realizar cambios si es necesario.

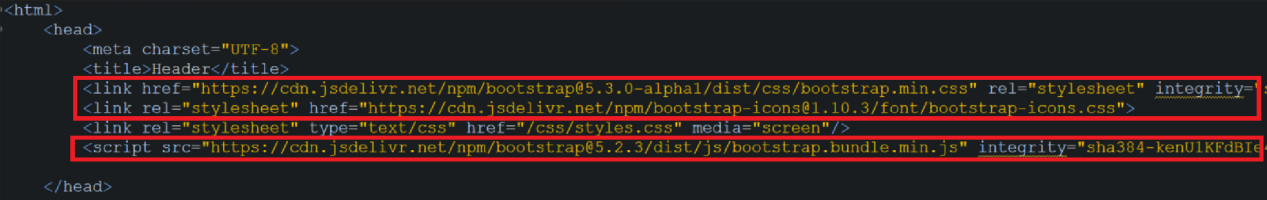
En el caso del catálogo para el administrador, incluye las opciones de agregar un nuevo producto y editar o eliminar uno existente.



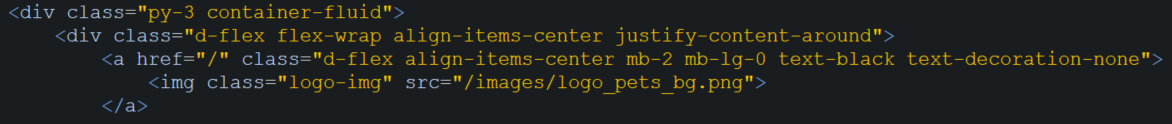
**Front-end**

Para el desarrollo del front-end usamos Bootstrap que es un framework de desarrollo web que combina HTML, CSS y JavaScript para facilitar la creación de interfaces, se basa en un sistema de rejillas (grid system) que organiza los elementos de la página en filas y columnas. Esto permite un diseño flexible y adaptable a diferentes dispositivos y tamaños de pantalla. Además, Bootstrap proporciona una amplia gama de componentes preestilizados, como botones, formularios, navegación, tarjetas, entre otros, que se pueden usar de manera sencilla, para usar estos componentes simplemente consiste en agregar las clases correspondientes que nos indica la documentación en las etiquetas HTML.

Para implementar Bootstrap en el proyecto, primeramente agregamos en el ‘head’ del HTML el <script> y los <link> que nos proporciona la documentación. Se pueden apreciar a continuación.



Luego es simplemente ir agregando las clases que se necesiten en cada caso, a continuación se puede ver un ejemplo de como se usan las clases.

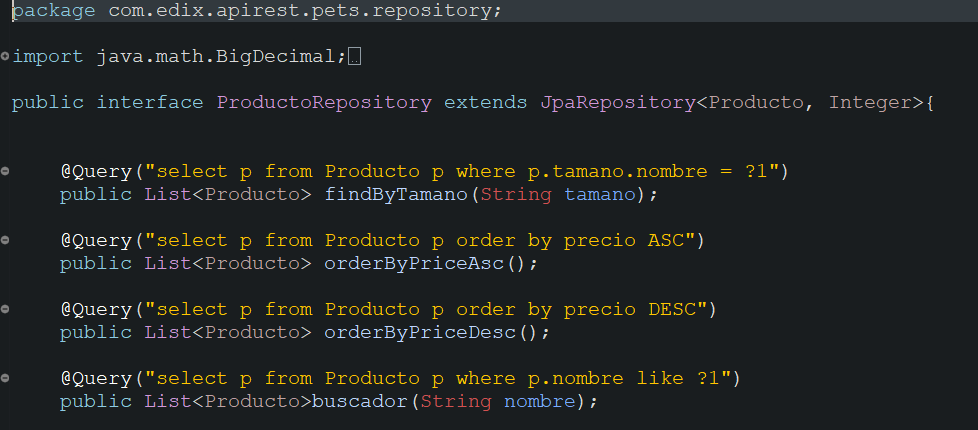


También hemos usado CCS para dar algunos estilos adicionales, para ello hemos hecho el link de la hoja de estilos que esta ubicada en la sigueiente ruta src/main/resources/static/css/style.css

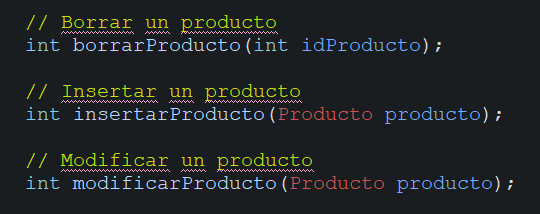
* **Creación de funcionalidades para la entidad Producto**

Una vez creada la entidad Producto es necesario crear el repositorio que tiene las consultas asociadas a los métodos que vamos a declarar para obtener los resultados deseados. En nuestro caso vamos a necesitar hacer consultas que nos devuelvan una lista de productos la cual este filtrada por tamaño, una lista de productos que este ordenada por precio ascendente y descendente y una lista de productos que contenga la palabra clave que el usuario solicite.

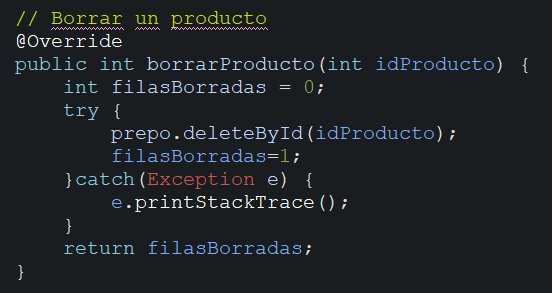
Para realizar estas consultas hacemos uso de la anotación @Query que permite escribir instrucciones de SQL y exponerlas como métodos, a continuación se muesran los métodos creados en el repositrio de Producto.

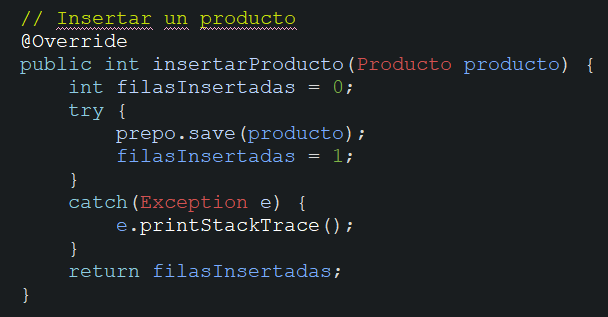


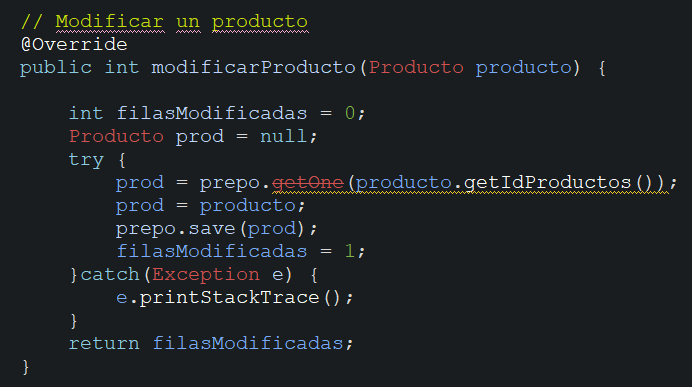
Una vez creado el repositorio creamos el service donde declaramos los métodos que vamos a necesitar para las funciones que vamos a realizar con los productos, entre los mas importantes estan eliminar, insertar y modificar un producto.



Seguidamente creamos la clase que implementa este service, esta lleva la lógica de cada método para aplicar las funcionalidades deseadas, en la siguiente imagen se muestra la implementación de los métodos definidos en el service.



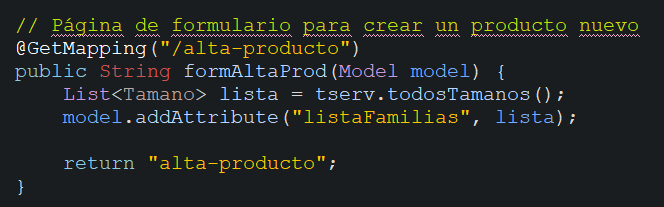




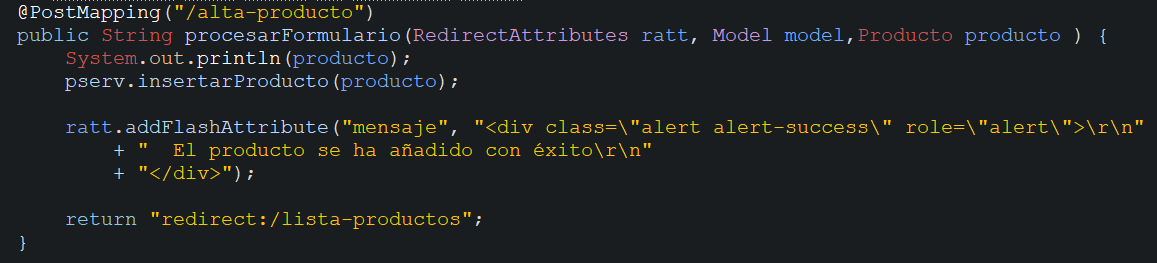
Finalmente en el controller de los productos se crean los métodos que hacen uso de toda la lógica creada anteriormente y conecta esa información del back-end para ser mostrada en el front-end a través de los JSP.

* **Alta de un producto**

Para dar de alta un producto vamos con la ayuda de la anotación @GetMapping vamos a mostrar un formulario, que en este caso estará definido en el JSP ‘alta-producto.jsp’, donde se podrán insertar los datos necesarios para registrar un nuevo producto.

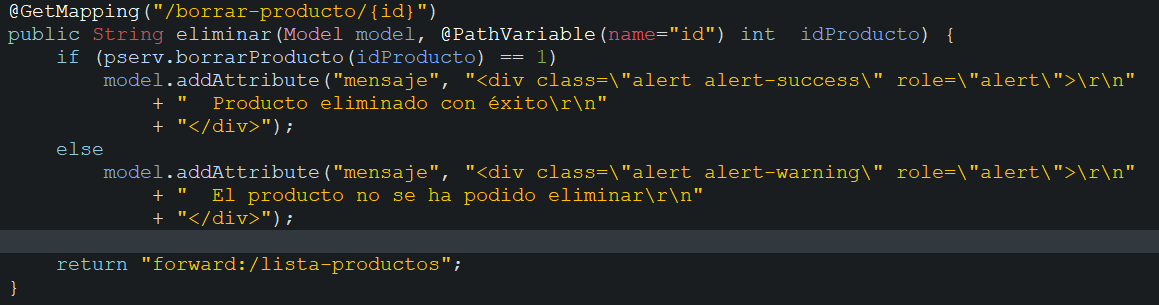


Posteriormente con la anotación @PostMapping y haciendo uso de la lógica creada en pasos anteriores, procesamos los datos obtenidos con el formulario lo añadimos a la base de datos. El usuario en el front verá un mensaje con la información de que su producto se ha añadido correctamente.



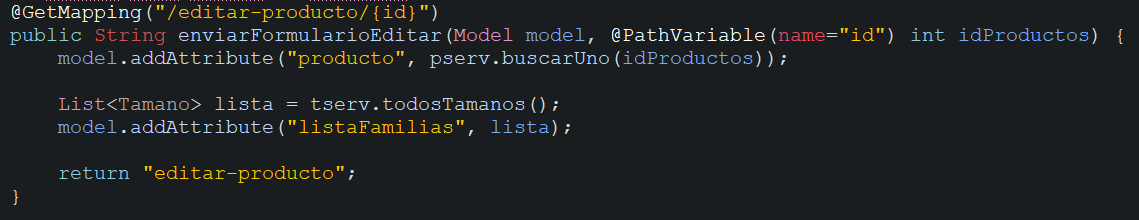
* Eliminar un producto

Para eliminar un producto hemos agregado un botón, que sólo verá el usuario con el rol de administrador, que aparecerá en cada producto de la lista de los mismos y que al pulsarlo el método que se muestra a continuación se encargará de obtener el id del producto en cuestión y aplicar la lógica creada el pasos anteriores para eliminar de la base de datos el producto.

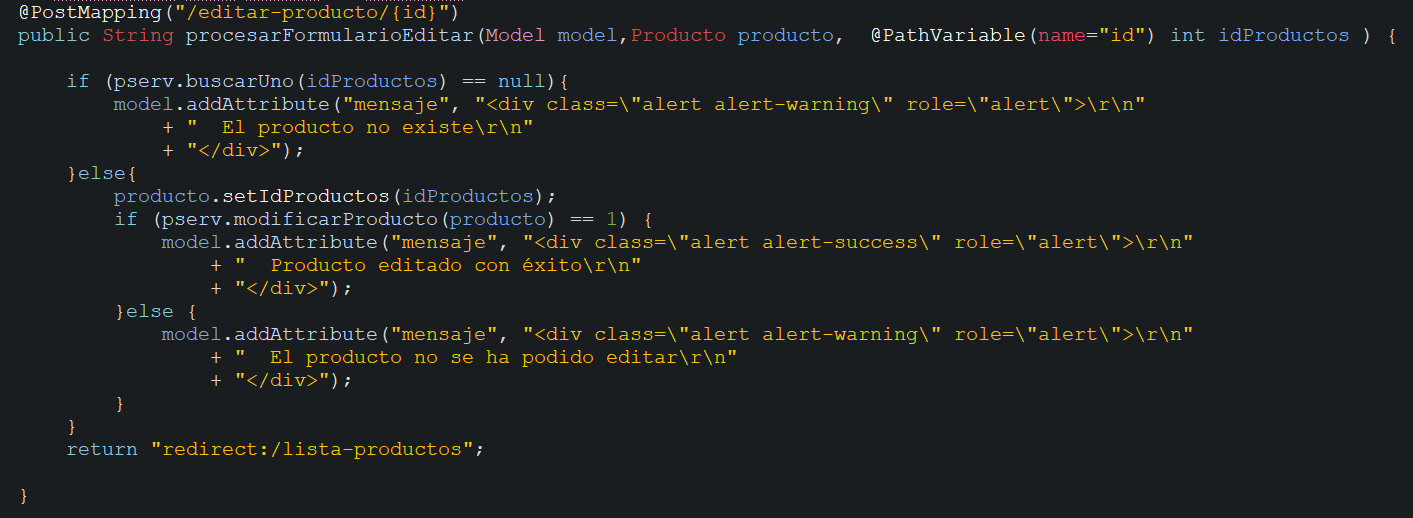


* Modificar un producto

Para la modificación de un producto primeramente mostramos un formulario, que será un jsp llamado ‘editar-producto’, que contiene la toda la información del producto en ese momento, cuyos campos se pueden editar con la información que se necesite.



Dicho formulario tiene un botón que al pulsarlo se pondrá en funcionamiento la lógica que contiene el siguiente método que se muestra a continuación.



Este método se encargará de sustituir la información que ingrese el administrador en el producto que desea modificar.