

# Tecnologías Web

(71023097)

Práctica – Curso 2018 / 2019

“Creación de un portal para la gestión de una tienda de electrodomésticos online”

- M E M O R I A -



Uned – C.A. Bergara

**Pablo Uzquiano** – puzquiano3@alumno.uned.es

**Mikel García** – mgarcia8739@alumno.uned.es

**Enrique Pérez** – eperez1184@alumno.uned.es

<b>1. - Introducción</b>	<b>3</b>
<b>2.- Composición de grupo y roles</b>	<b>4</b>
<b>3.- Plan de trabajo</b>	<b>5</b>
<b>4.- Especificación de requerimientos</b>	<b>7</b>
4.1.- Identificación de los requisitos funcionales	7
4.2 Casos de uso	8
<b>5.- Diseño del modelo de datos y JPA.</b>	<b>13</b>
5.1.- JPA Java persistence API.	14
5.2.- Decisiones sobre las relaciones.	14
5.2.1 Relación Promociones-productos:	14
5.2.2 Relación Categorías-productos:	16
5.2.3 Relación Usuario-Lineas de carrito- producto:	16
<b>6.- Solución técnica</b>	<b>18</b>
6.1.- Spring Framework	18
6.2.- Spring Boot	19
6.3.- Base de Datos H2	19
6.4.- Arquitectura MVC con capa de Servicio	20
6.5.- Inyección de Dependencias	20
<b>7.-Capa “Modelo” del patrón MVC</b>	<b>22</b>
<b>8.- Capa “Servicios” del patrón MVC.</b>	<b>24</b>
<b>9.-Capa “Controlador” del patrón MVC</b>	<b>27</b>
9.1.-Securización de funcionalidades según roles:	27
9.1.1.- Admin	27
9.1.2.- Cliente.	28
<b>10.-Capa “Vista” del patrón MVC</b>	<b>29</b>
<b>11.- Patrones de diseño de navegación.</b>	<b>31</b>
<b>12.- Conclusiones.</b>	<b>33</b>
<b>13.- Anexos.</b>	<b>333</b>
13.1.- Tecnologías empleadas para el trabajo en grupo.	333

# 1. - Introducción

Objetivos cumplidos tras la realización de la práctica:

- La primera y nueva para los tres integrantes del grupo era el poder trabajar como un equipo y podemos decir que la solidaridad de cada miembro, la responsabilidad y el buen hacer ha sido la nota predominante durante el desarrollo de la práctica.
- A nivel técnico, hemos ido desarrollando e implementado cada uno de los requisitos que se nos especificaba en el enunciado. También hemos aplicado el uso de patrones de diseño tal como se nos pedía.
- Hemos ido cumpliendo progresivamente las etapas programadas de forma paulatina tal como estaban diseñadas al comienzo del trabajo. Para nosotros era importante respetar los tiempos de cada hito y así ha sido. El utilizar una metodología “iterativa” ha supuesto un desarrollo más ágil de lo planeado.
- Hemos llevado a cabo cada parte del trabajo en tiempo y forma tal como lo teníamos diseñado, respetando escrupulosamente tanto los requisitos funcionales, como los técnicos y los de despliegue.

## 2.- Composición de grupo y roles

Los miembros del grupo son:

- Pablo Uzquiano
- Mikel García
- Enrique Pérez

Todos somos del Centro Asociado de Bergara. Para la asignación de roles nos hemos basado en criterios como la experiencia y conocimiento de cada miembro del grupo.

Matriz miembros / roles

<b>Roles</b>	<b>Pablo</b>	<b>Mikel</b>	<b>Enrique</b>
Jefe de proyecto		X	
Analista	X	X	X
Desarrollador	X	X	
Diseño	X		X
Pruebas	X	X	X
Documentación			X

### 3.- Plan de trabajo

La propuesta inicial que hemos realizado, sobre las etapas o hitos que debemos cumplir son:

<b>Etapas</b>	<b>Descripción</b>	<b>Inicio</b>	<b>Fin</b>
1	Requisitos del sistema	12/2	28/2
2	Análisis del sistema	16/2	6/3
3	Diseño del sistema	6/3	18/4
4	Implementación del sistema	10/4	2/5
5	Plan de pruebas	18/4	10/5
6	Manual de usuario	10/5	29/5

#### Etapas 1.

En un primer momento estudiaremos y elegiremos las herramientas para desarrollar el proyecto. Definiremos un sistema de trabajo, distribuyendo roles. Elegiremos una metodología de desarrollo, analizaremos las necesidades de tiempos para cada etapa y organizaremos la estructura de trabajo.

#### Etapas 2.

En este caso, estudiaremos y analizaremos los casos de uso del sistema. Desarrollaremos los diferentes modelos y diagramas para entender y explicar cómo se relacionan los diferentes actores con el sistema. Utilizaremos el sistema UML para mostrarlo.

- Creamos la capa de persistencia

#### Etapas 3.

Una vez que hemos definido el modelo, diseñaremos la lógica de negocio, según los requerimientos del sistema, con los diferentes tipos de funcionalidades que exige el enunciado.

- Creamos la capa de servicios

#### Etapas 4.

Implementaremos el modelo y la lógica de negocio, junto con el diseño y la maquetación de la capa “vistas”.

- Creamos la capa de vistas

#### Etapas 5.

Una vez implementado todo el conjunto será tiempo para realizar las primeras pruebas unitarias, test y ensayos que nos permitan conseguir un producto entregable y preparado para su despliegue.

- Creamos pruebas de la aplicación y entregable

#### Etapa 6.

Finalmente se elaborará un manual de usuario, con las especificaciones de acuerdo a la solución implementada, así como todos los detalles funcionales, técnicos y de uso que corresponda.

- Creamos el manual de usuario junto con el javadoc.

## 4.- Especificación de requerimientos

### 4.1.- Identificación de los requisitos funcionales

	Nombre	Descripción
1	Acceso usuario (anónimo)	El sistema debe permitir por defecto el acceso libre de usuarios sin registrarse (anónimos) a la web.
2	Búsqueda productos	El sistema permitirá la búsqueda de productos según: Categoría, Marca o puntuación de los mismos a cualquier usuario sea registrado (cliente) o no (anónimo).
3	Visualizar ficha productos	El sistema permitirá la visualización de los productos con todas las características de compra de los mismos a cualquier usuario sea registrado (cliente) o no (anónimo)
4	Consultar servicios	El sistema debe permitir consultar los servicios que ofrece la tienda.
5	Darse de alta un usuario	El sistema debe permitir darse de alta a un usuario sin registrar (anónimo) a través de un formulario dispuesto al efecto accesible desde cualquier punto de la web.
6	Evaluar producto por un cliente	El sistema debe permitir sólo a los clientes la posibilidad de evaluar un producto puntuándolo a través de un sistema de estrellas (otorgando valores del 1 al 5)
7	Comprar online	El sistema debe permitir sólo a los clientes la posibilidad de realizar compras en la web
8	Consultar histórico de compras	El sistema debe permitir a los usuarios registrados (clientes) consultar su histórico de compras
9	Darse de baja un cliente	El sistema debe permitir a un usuario registrado (cliente) la posibilidad de darse de baja del sistema a través de un sencillo formulario accesible desde su panel de cliente
10	Figura de la tienda	El sistema contará con la figura de un administrador de tienda que podrá realizar las funciones propias de una tienda señaladas en el enunciado.
11	Añadir productos al catálogo	El sistema debe permitir sólo al administrador de una tienda la posibilidad de añadir electrodomésticos al mismo
12	Editar productos del catálogo	El sistema deber permitir sólo al administrador de una tienda la posibilidad de editar cada uno de los productos en catálogo.
13	Establecer promociones	El sistema deber permitir sólo al administrador de una tienda la capacidad para crear y establecer las diferentes promociones publicadas en la web.
14	Devolver productos al inventario	El sistema debe permitir sólo al administrador de una tienda la posibilidad de devolver un producto al inventario si un cliente decide devolver el producto comprado

15	Generar informes de ventas	El sistema debe permitir sólo al administrador de una tienda la posibilidad de generar informes de ventas, mediante un listado, de cualquier cliente suyo, según un determinado día.
----	----------------------------	--

## 4.2 Casos de uso

Actor: Usuario (no registrado)

	Nombre	Descripción
1	Buscar un producto	<ul style="list-style-type: none"> <li>El actor selecciona el tipo de búsqueda que quiere realizar según la categoría, la marca o la puntuación de un producto</li> <li>El sistema muestra por pantalla los productos a través de un listado de acuerdo a la opción de búsqueda seleccionada</li> </ul>
2	Visualizar ficha producto	<ul style="list-style-type: none"> <li>El actor una vez realizada la búsqueda, desde el listado que tiene en pantalla puede acceder a cada producto, haciendo click en la descripción de mismo.</li> </ul>
3	Consultar promociones	<ul style="list-style-type: none"> <li>El actor desde cualquier parte de la web puede acceder a la opción “Promociones activas” situado en el menú principal.</li> </ul>
4	Consultar servicios	<ul style="list-style-type: none"> <li>El actor desde cualquier parte de la web puede acceder a la opción “Servicios” situado en el menú principal.</li> </ul>
5	Darse de alta en el sistema	<ul style="list-style-type: none"> <li>El actor desde el menú principal puede a través de la opción “Área de cliente”, acceder a un formulario, donde introduciendo sus datos personales puede darse de alta en el sistema.</li> </ul>



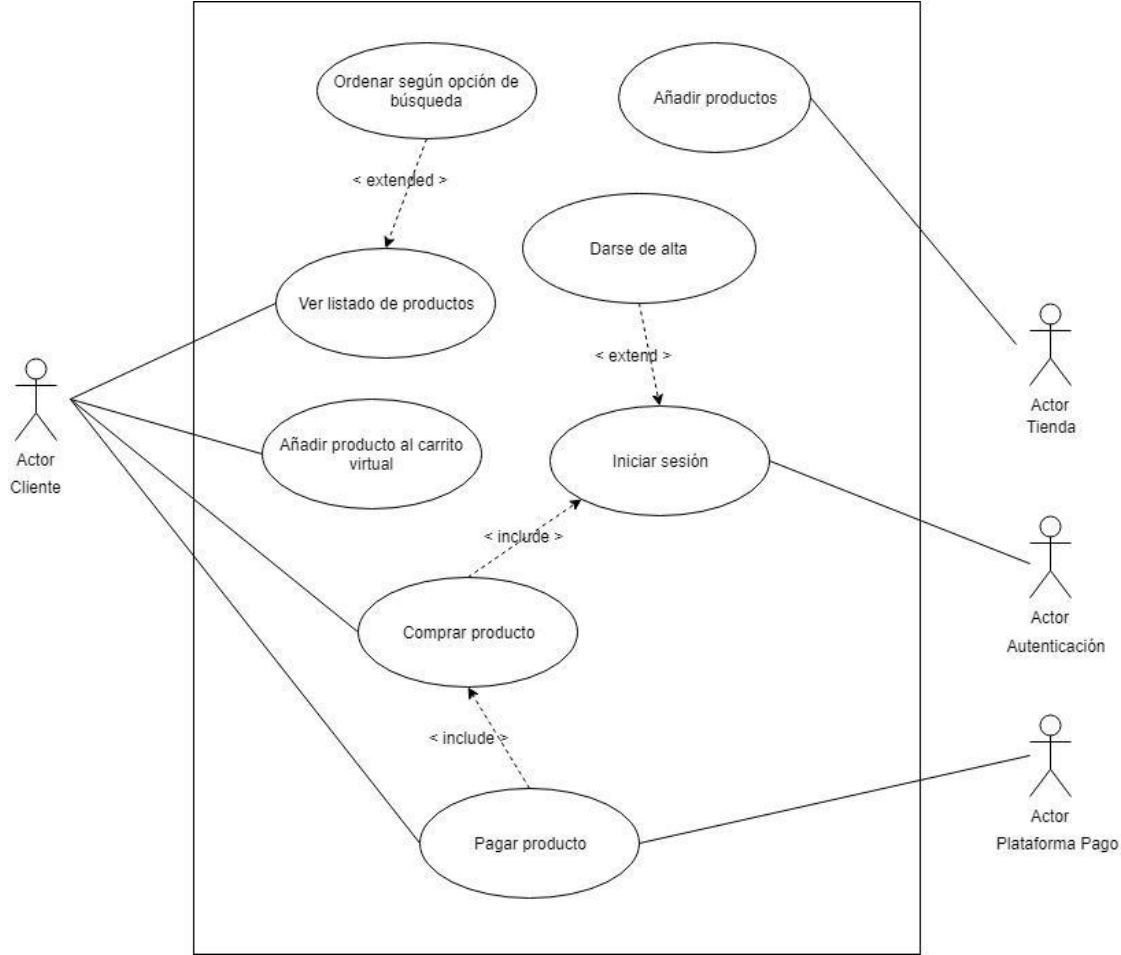
Actor: Cliente (usuario registrado)

	Nombre	Descripción
1	Buscar un producto	<ul style="list-style-type: none"><li>● El actor selecciona el tipo de búsqueda que quiere realizar según la categoría, la marca, la promoción activa o la puntuación de un producto</li><li>● El sistema muestra por pantalla los productos a través de un listado de acuerdo a la opción de búsqueda seleccionada</li></ul>
2	Visualizar ficha producto	<ul style="list-style-type: none"><li>● El actor una vez realizada la búsqueda, desde el listado que tiene en pantalla puede acceder a cada producto, haciendo click en la descripción de mismo.</li></ul>
3	Consultar promociones	<ul style="list-style-type: none"><li>● El actor desde cualquier parte de la web puede acceder a la opción “Promociones activas” situado en el menú principal.</li></ul>
4	Consultar servicios	<ul style="list-style-type: none"><li>● El actor desde cualquier parte de la web puede acceder a la opción “Servicios” situado en el menú principal.</li></ul>
5	Puntuar un producto	<ul style="list-style-type: none"><li>● El actor desde la ficha del producto puede asociar una puntuación del 1 a 5 estrellas a la valoración del mismo.</li></ul>
6	Consultar histórico de compras	<ul style="list-style-type: none"><li>● El actor desde su perfil, puede acceder de forma directa a la posibilidad de consultar un histórico mediante una lista con todas las compras realizadas hasta entonces.</li></ul>
7	Darse de baja en el sistema	<ul style="list-style-type: none"><li>● El actor desde su perfil, puede acceder a la posibilidad de darse de baja confirmándolo mediante un simple click.</li></ul>
8	Comprar online	<ul style="list-style-type: none"><li>● A continuación se describirá con mayor detalle este caso de uso por entender que es el caso de uso más importante dentro de la web.</li></ul>

Actor: Tienda (debe de estar registrado en el sistema como administrador)

	Nombre	Descripción
1	Añadir un producto	<ul style="list-style-type: none"><li>● El actor a través de su cuenta y mediante un sencillo formulario podrá ir añadiendo al catálogo de la BBDD en modo local los productos que considere oportunos, siempre ciñéndose al inventario disponible en su tienda.</li></ul>
2	Editar las características de un producto	<ul style="list-style-type: none"><li>● El actor podrá editar los productos que tenga en el portal a través de un listado accesible desde su cuenta.</li></ul>
3	Establecer promociones de productos	<ul style="list-style-type: none"><li>● El actor podrá establecer las promociones y ofertas que considere oportunas a través de cada una de las fichas de los productos que tiene en la web.</li><li>● Se considera que un producto está en promoción si tiene un precio rebajado y por tanto tiene un concepto de ahorro en su ficha.</li></ul>
4	Generar informes de ventas	<ul style="list-style-type: none"><li>● El actor podrá generar informes de ventas para un determinado día, tanto de las ventas por clientes, así como de su importe y facturación total</li></ul>
5	Devolver producto inventario	<ul style="list-style-type: none"><li>● Si un cliente devuelve un producto, el actor deberá actualizar el inventario, añadiendo dicho producto a la oferta de la tienda.</li></ul>
6	Eliminar histórico ventas de un cliente	<ul style="list-style-type: none"><li>● Si un cliente devuelve un producto, el actor tras actualizar el inventario, deberá eliminar del histórico de ventas del cliente dicha compra.</li></ul>

**Caso de uso:** Compra online por el actor Cliente.



Caso de uso – compra online

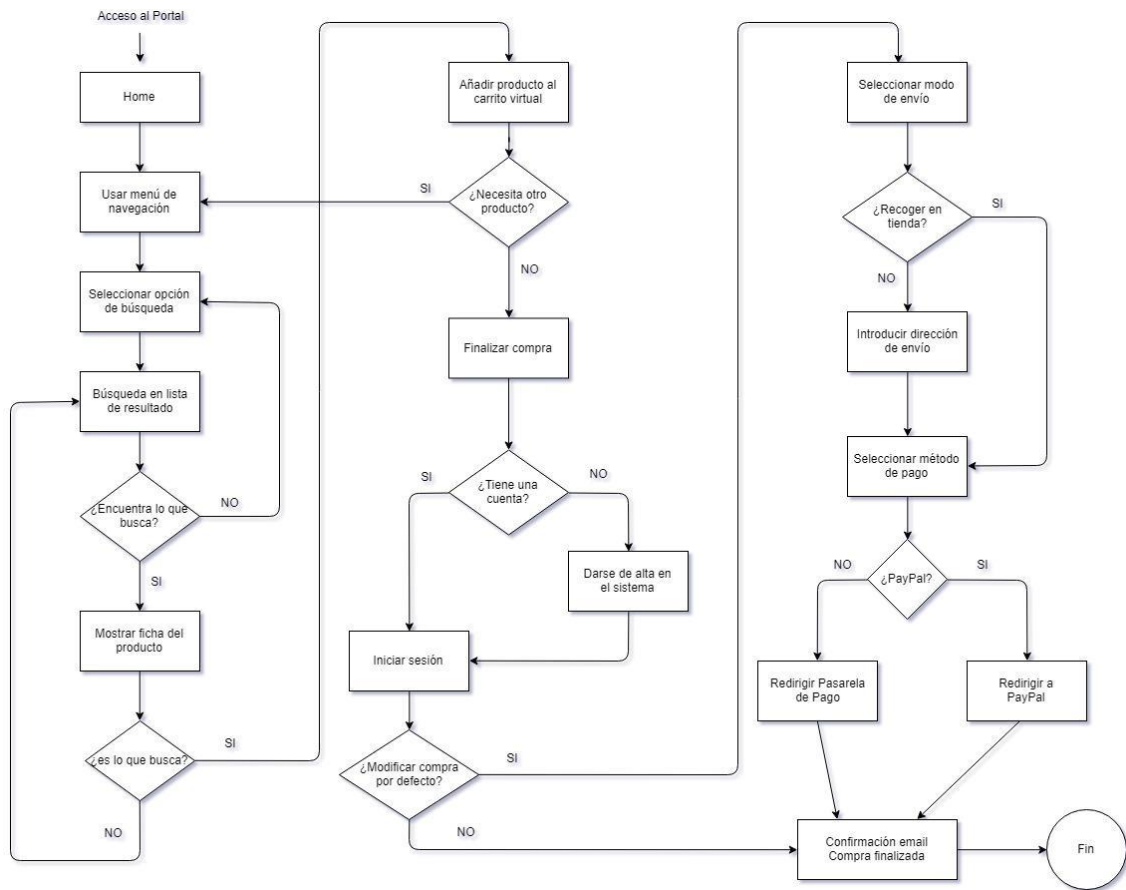


Diagrama de flujos – compra online

## 5.- Diseño del modelo de datos y JPA.

Tras analizar los requisitos del sistema el siguiente paso consiste en diseñar correctamente el modelo de datos a implementar. Para ello se realiza un diagrama entidad relación que a su vez representa el esquema de la base de datos.

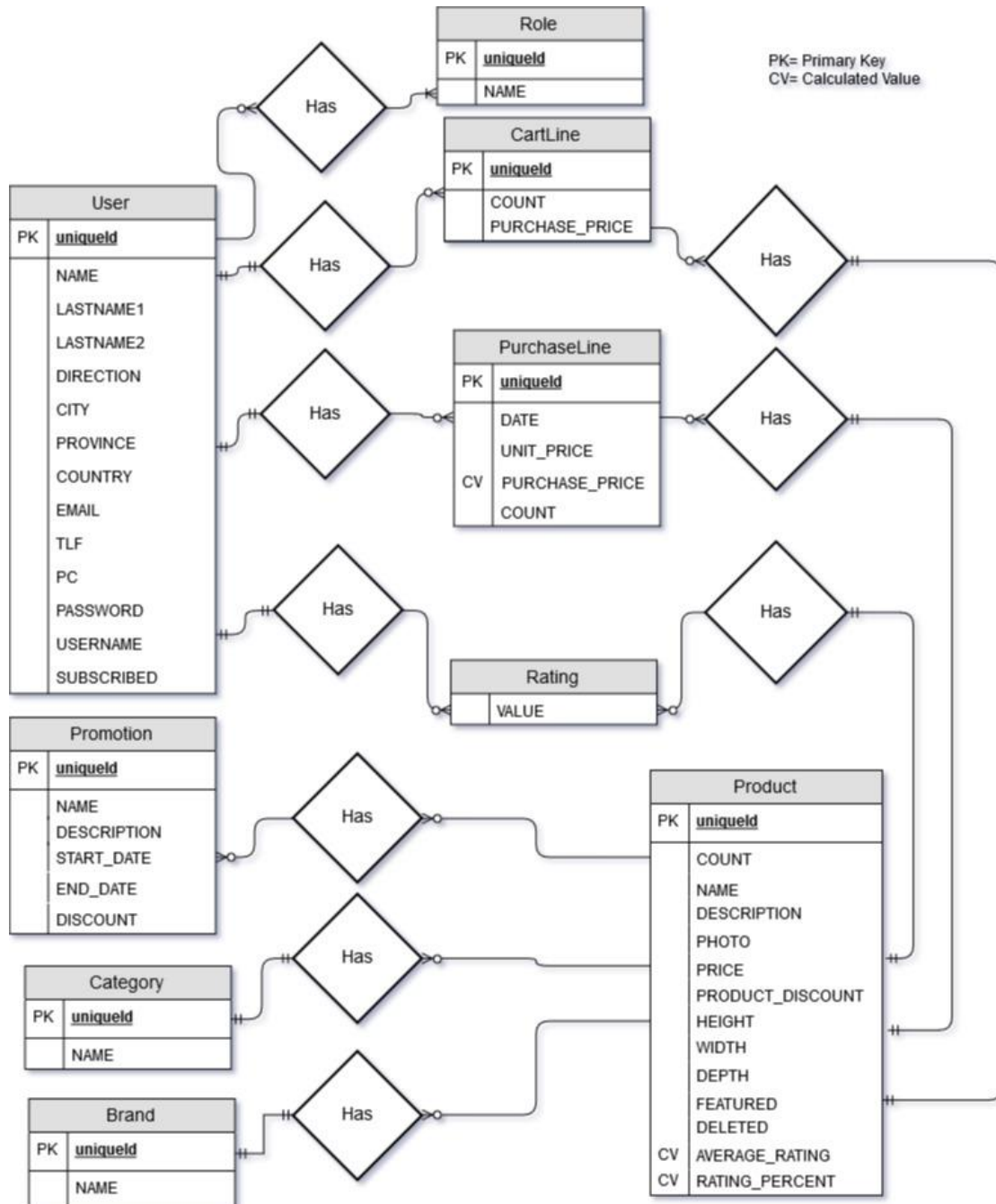


Diagrama E-R del sistema-esquema de base de datos

Desde el punto de vista del diseño de la lógica de negocio, este podría ser uno de los pasos más significativos para una correcta definición de los objetivos y una buena distribución de tareas para los integrantes del equipo. En este punto se definen

claramente las entidades que van a componer el sistema, las relaciones entre ellas y la cantidad y los nombres de sus atributos. El diagrama E-R se puede considerar como una hoja de ruta para el equipo, especialmente para los responsables de la implementación de la parte lógica.

## 5.1.- JPA Java persistence API.

Para llevar a cabo la persistencia de nuestro modelo hemos utilizado la API JPA, lo cual ha agilizado enormemente el correcto mapeo de nuestros objetos y la implementación de las relaciones de nuestro diagrama E-R.

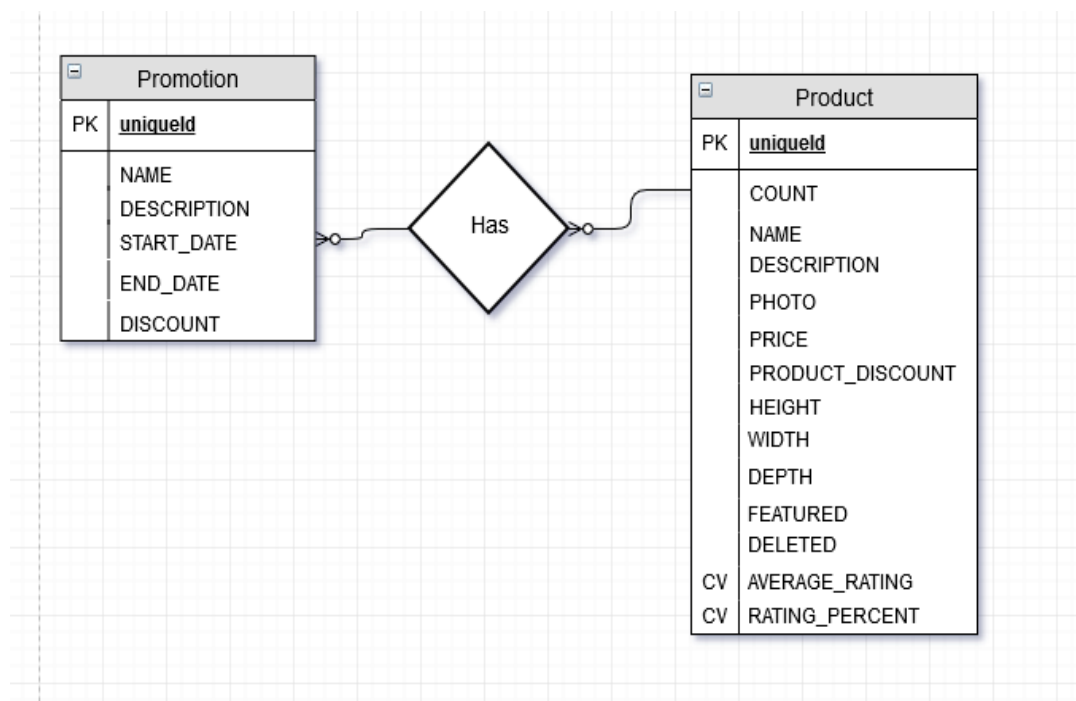
## 5.2.- Decisiones sobre las relaciones.

Este se podría considerar uno de los temas más ampliamente debatidos. Un buen punto de partida para empezar con la lógica pero a la vez un tema controvertido en el que se ha discutido ampliamente sobre las decisiones a tomar.

A continuación se detallan algunas de las decisiones importantes que se toman en este sentido.

### 5.2.1 Relación Promociones-productos:

Una promoción puede contener 0 a N productos y un producto puede tener de 0 a N promociones asignadas.



*Diagrama parcial Promociones-Productos*

```

@Entity
public class Promotion {

    @Id
    @GeneratedValue
    private long id;
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Calendar startDate;
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Calendar endDate;
    private String name;
    private String description;
    private int discount;
    @ManyToMany(mappedBy = "promotions", fetch = FetchType.EAGER)
    private Set<Product> products = new HashSet<>();

    @Transient
    private boolean active;
}

```

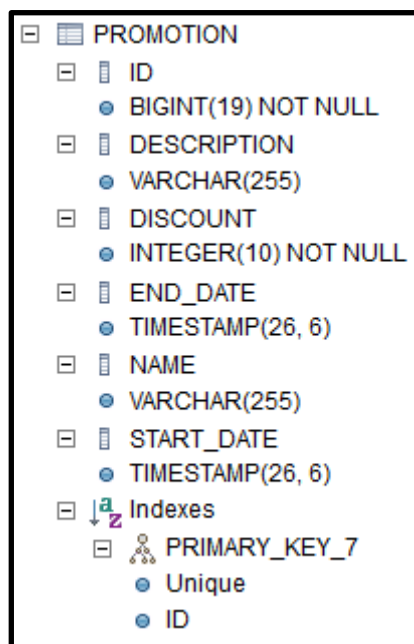
*Implementación del Promotion.java con anotaciones JPA*

```

@ManyToMany(fetch = FetchType.EAGER)
private Set<Promotion> promotions;

```

*Relación inversa en Product.java con las promociones*



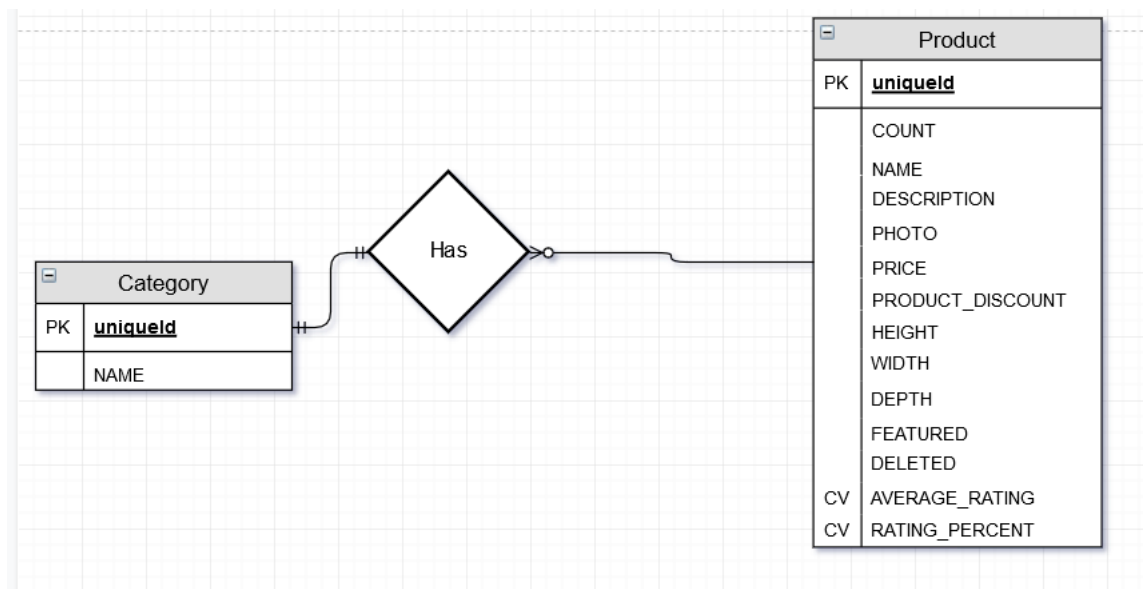
*Entidad promoción generada en Base de datos.*

### 5.2.2 Relación Categorías-productos:

Respecto a esta relación se debatieron diferentes opciones como la posibilidad de tener categorías anidadas, categorías padre, hijas, hermanas etc.

La decisión tomada ha sido la representada en el diagrama. Un solo nivel de categorías y cada producto pertenece a una sola categoría.

Una categoría puede contener de 0 a N productos y un producto pertenece a una categoría.



*Diagrama parcial Categorías-Productos*

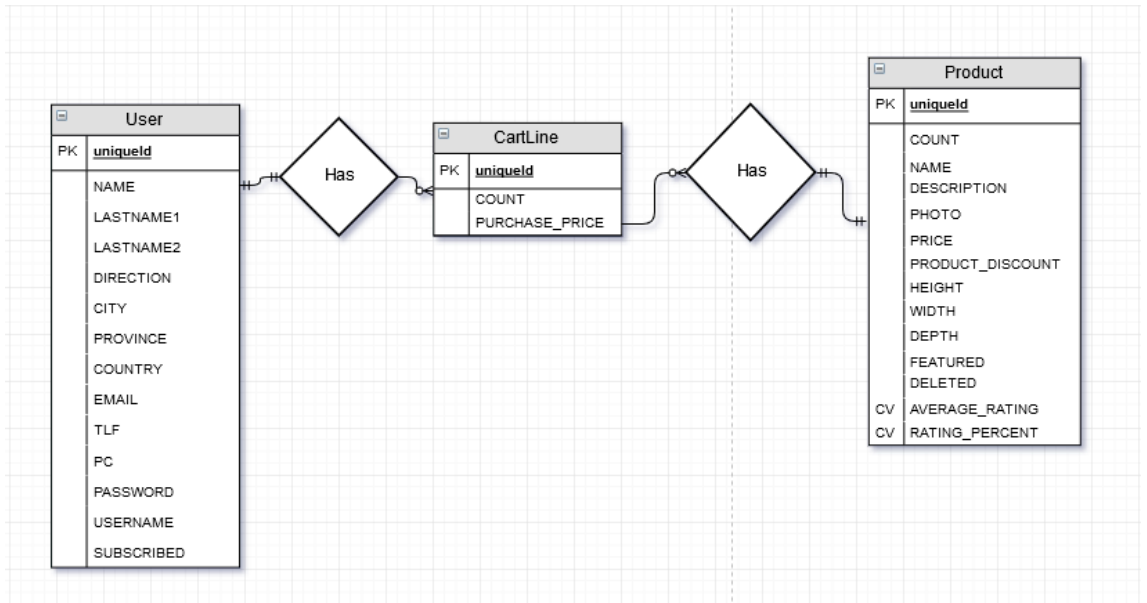
### 5.2.3 Relación Usuario-Líneas de carrito- producto:

Un usuario puede tener de 0 a N líneas de carrito, una línea de carrito puede tener 1 único producto. La cuestión en esta relación es que la línea de carro solo guarda relación con un único producto pero teniendo en cuenta que tiene una propiedad "cantidad" que se refiere a la cantidad de productos a comprar.

En el sentido inverso:

Un producto puede pertenecer de 0 a N líneas de carrito y una línea de carrito pertenece a un único usuario.





*Diagrama parcial Usuario-Líneas de carro-Producto*

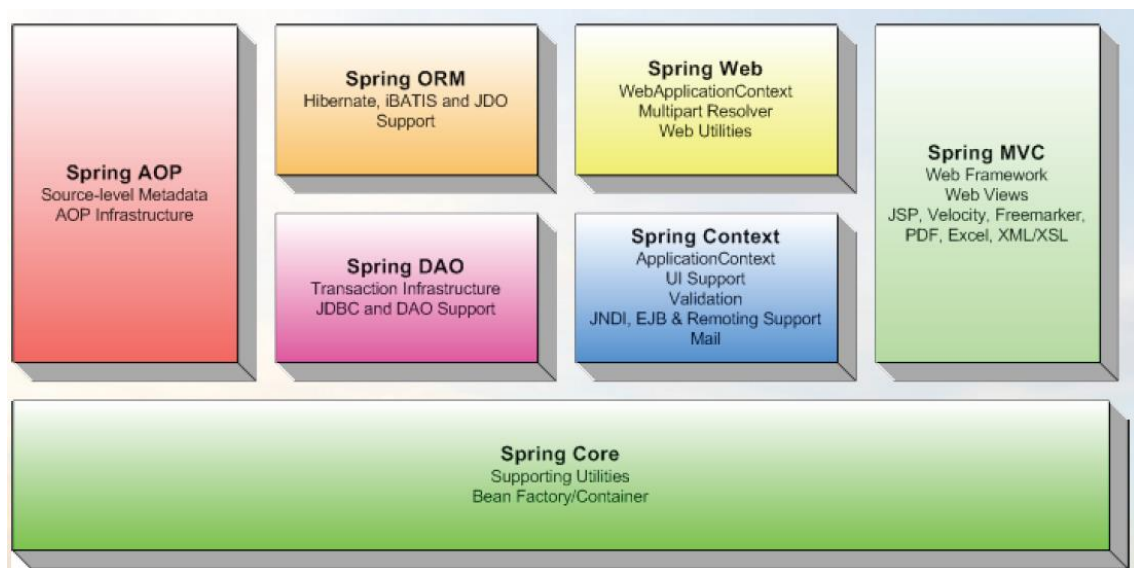
## 6.- Solución técnica

Siguiendo los requisitos técnicos especificados en el enunciado de la práctica, hemos decidido apostar por Spring Framework (y alguna otra tecnología que comentaremos en los siguientes apartados) para montar nuestro proyecto Web.

Hemos escogido estas tecnologías puesto que nos ofrecen un conjunto de herramientas más modernas para construir aplicaciones en Java. Estas herramientas, además, son ampliamente utilizadas en entornos empresariales, lo cual nos ayudará a entender mejor cómo se crean proyectos a día de hoy con Java.

### 6.1.- Spring Framework

Spring Framework ofrece como elemento clave el soporte de infraestructura a nivel de aplicación, brindando un completo modelo tanto para la configuración como para la programación de aplicaciones empresariales desarrolladas bajo Java, sin discriminación en cuanto al despliegue de la plataforma.



Todo esto trae consigo una gran ventaja, ya que permite que los equipos de desarrollo puedan enfocarse directamente en la lógica empresarial que requiere la aplicación, haciendo el proceso más corto, rápido y eficaz, ahorrando líneas de código evitando tareas repetitivas.

Spring Framework da soporte a varios frameworks como: Hibernate, Struts, Tapestry, EJB, JSF, entre otros.

## 6.2.- Spring Boot

Spring Boot facilita la creación de aplicaciones basadas en Spring, autónomas y del nivel de producción que "simplemente se ejecutan".

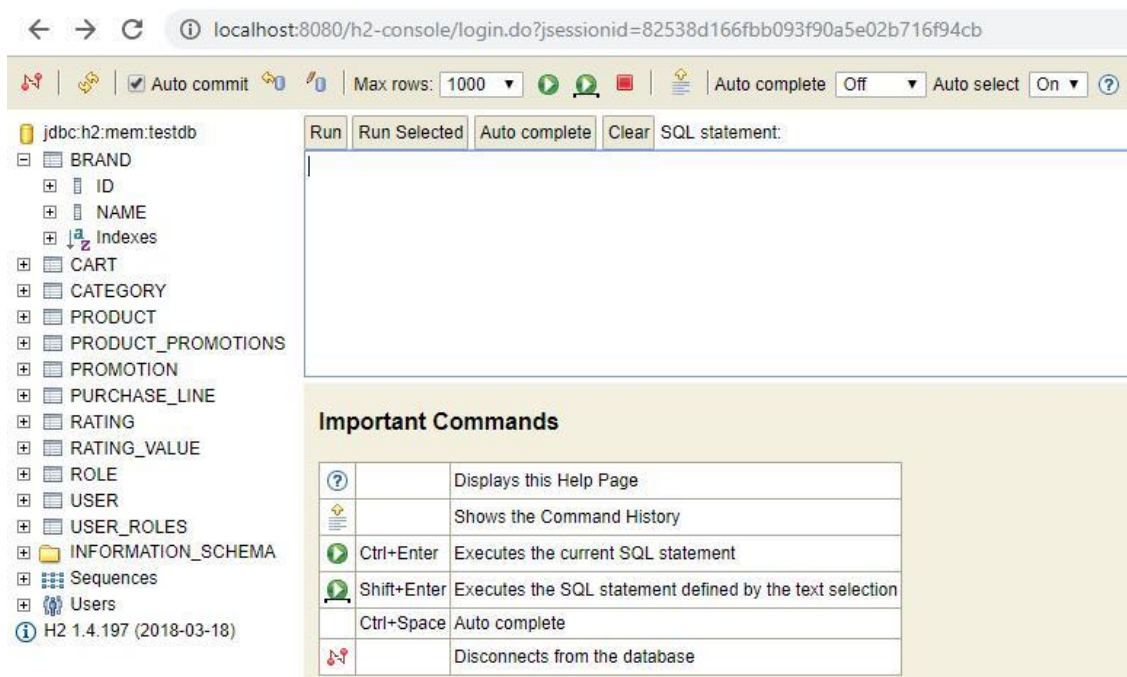
Esto significa que es posible poner en funcionamiento una aplicación de Spring con muy poca configuración. La poca configuración que se necesita está en forma de anotaciones, evitando las configuraciones en XML.

Entre las principales características de Spring Boot se encuentran:

- Contenedores Java embebidos: Tomcat o Jetty
- Soporte para la automatización con Maven y Gradle
- Configuración sugerida para iniciar rápidamente con un proyecto (Starters)
- Configura automáticamente Spring, cuando sea posible
- Características listas para producción: métricas, seguridad, verificación del estatus, externalización de configuración, etc.
- No genera código y no requiere configuración XML

## 6.3.- Base de Datos H2

Hemos escogido H2 como motor de base de datos en memoria dada su sencillez a la hora de integrarse con Spring, además de que es un motor muy rápido y ligero.



H2 nos da la posibilidad, además, de desplegar una pequeña consola con la que poder visualizar la base de datos, realizar consultas y ver su contenido.

La consola de H2 está accesible una desplegado el WAR en la dirección <http://localhost:8080/h2-console>

English Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded) Save Remove

---

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:testdb

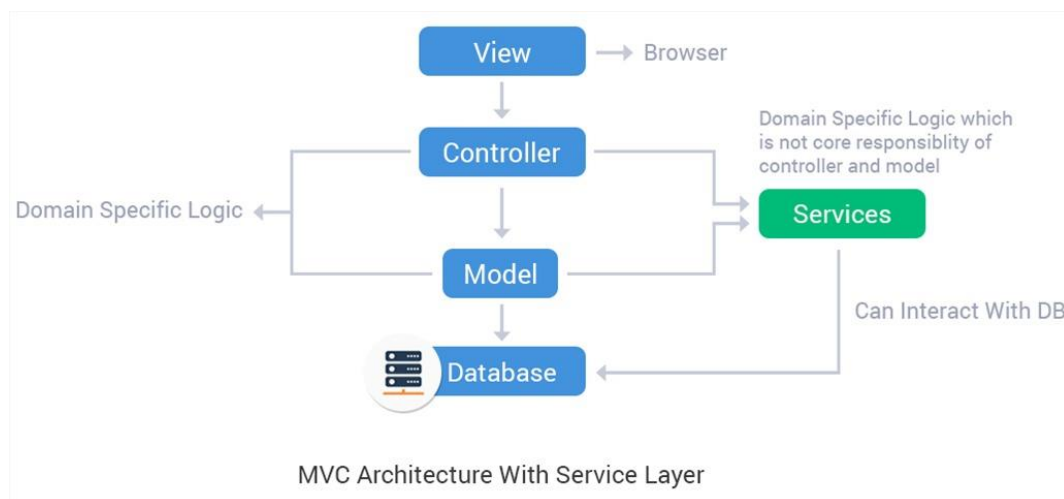
User Name: sa

Password:

Connect Test Connection

## 6.4.- Arquitectura MVC con capa de Servicio

La arquitectura de nuestro proyecto se basa en el patrón MVC (Modelo Vista Controlador), con una peculiaridad: la capa de servicio.



En nuestra capa de servicio tratamos de encapsular la lógica de negocio con la intención de que los controladores tengan la mínima responsabilidad.

## 6.5.- Inyección de Dependencias

Otro de los mecanismos ampliamente conocidos de Spring Framework es la inyección de dependencias.

El patrón de Inyección de Dependencias, también conocido como de Inversión de Control es un patrón que tiene como finalidad conseguir un código más desacoplado, facilitando las futuras modificaciones de nuestro código fuente.

```
private ProductService productService;
private CategoryService categoryService;
private BrandService brandService;
private PromotionService promotionService;
private RatingService ratingService;

@Autowired
public ProductController(ProductService productService,
    this.productService = productService;
    this.categoryService = categoryService;
    this.brandService = brandService;
    this.promotionService = promotionService;
    this.ratingService = ratingService;
}
```

*Inyección de dependencias con la anotación @Autowired*

Siguiendo el patrón de Inyección de Dependencias (DI, Dependency Injection) los componentes declaran sus dependencias, pero no se encargan de conseguirlas, ahí es donde entra el Contenedor de Spring, que en nuestras aplicaciones de Spring será el encargado de conseguir e inyectar las dependencias a los objetos.

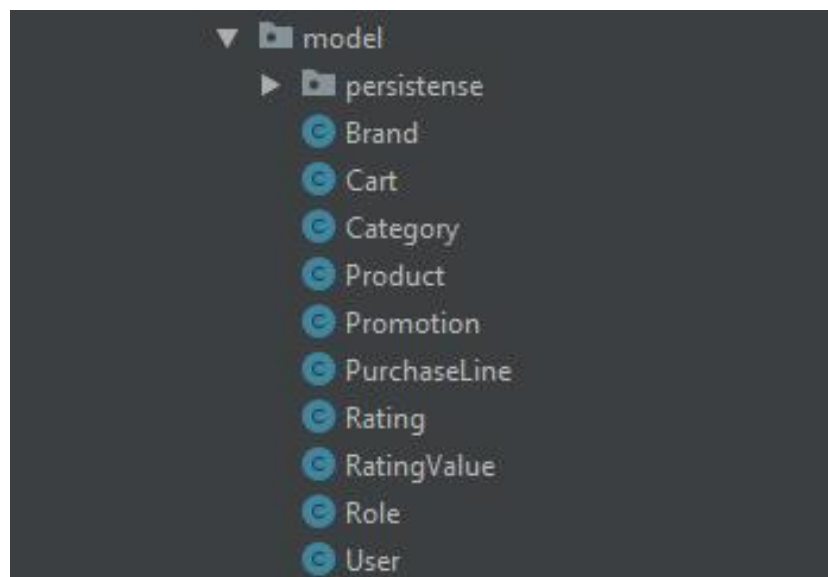
En nuestro proyecto se puede ver cómo se inyectan, por ejemplo, los servicios en los controladores con la anotación *@Autowired*.

## 7.-Capa “Modelo” del patrón MVC

La capa modelo del proyecto está directamente relacionada con el modelo de datos comentado en el apartado 5. Es una capa bien diferenciada en el proyecto y está implementada en el paquete *uned.webtechnologies.shop.inmemorydb.\**.

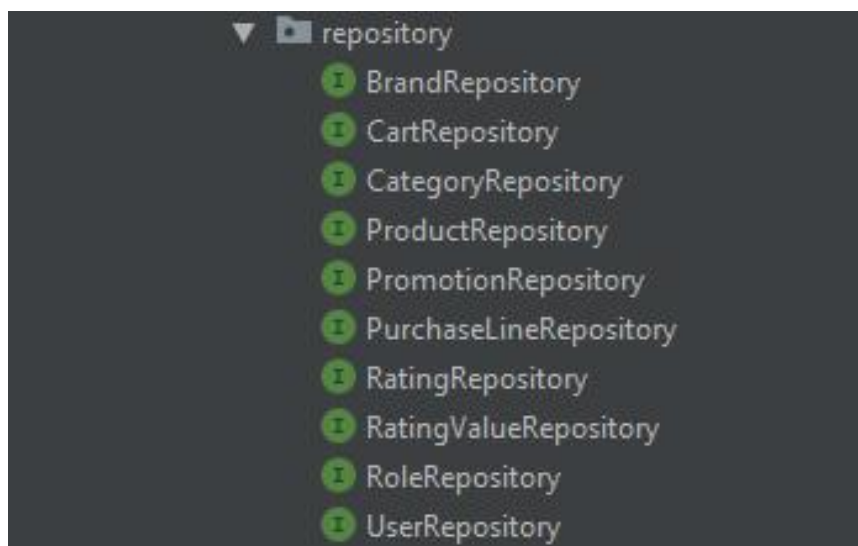
En dicho paquete se encuentran tres paquetes con responsabilidades bien diferenciadas.

En el paquete *uned.webtechnologies.shop.inmemorydb.model* se encuentran todas las clases del modelo.



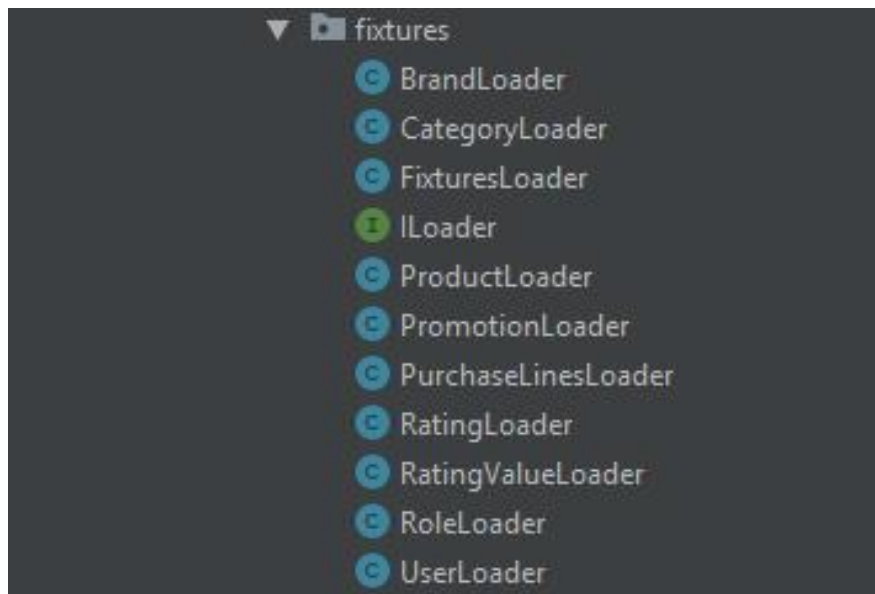
*Clases del modelo*

En el paquete *uned.webtechnologies.shop.inmemorydb.repository* se encuentran las interfaces encargadas de montar los repositorios de las entidades del modelo.



*Interfaces del repositorio*

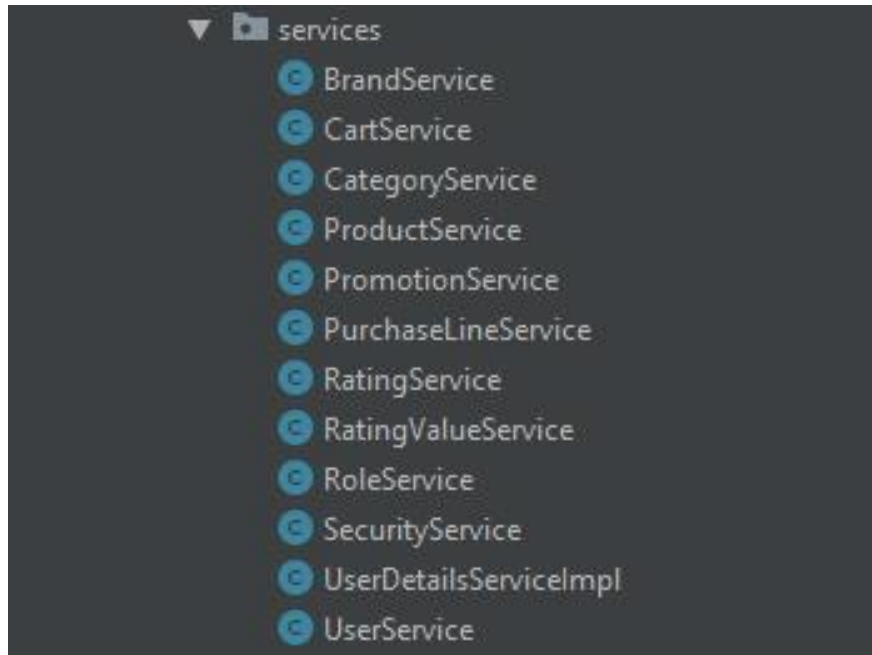
En el paquete *uned.webtechnologies.shop.inmemorydb.fixtures* se encuentran las clases encargadas de cargar datos en la base de datos cuando arrancamos la aplicación.



*Clases Loader*

## 8.- Capa “Servicios” del patrón MVC.

Tal y como se ha descrito en el apartado 6.4, en nuestro patrón MVC hemos añadido una capa “servicios” que se encarga de la lógica de negocio y están en el paquete *uned.webtechnologies.shop.services*.



*Clases Services*

Los controladores de nuestra arquitectura deberán solicitar y enviar la información a esta capa para que se encarga de interactuar con el modelo.

Hemos creado un servicio para cada modelo de nuestro sistema para poder diferenciar correctamente las responsabilidades de cada uno de los servicios, haciendo así más mantenible nuestro código. Estas decisiones de diseño nos han agilizado significativamente el mantenimiento del código, entendiendo como “mantenimiento” la detección de errores y ampliación-modificación de funcionalidades.



La mayoría de las clases de esta capa se encargan de solicitar a los correspondientes repositorios los datos necesarios y de guardar los objetos tal y como se reciben desde la vista, pero hay algunas en particular que resulta interesante mencionar en este apartado ya que son un buen ejemplo de aplicación de lógica un poco más elaborada.

Una de ellas es la clase *PurchaseLineService* ya que es una de las clases de esta capa que implementa más lógica.

Esta clase se encarga de recibir desde la vista un carrito de la compra lleno que pertenece a un usuario y lo convierte en una compra. Hay que mencionar que un carrito de la compra se compone de una o más líneas de carrito y que cada línea de carrito tiene uno o más productos.



## Orden de compra

Producto	Descripción	Precio	Cantidad	Importe	Resumen del pedido	
	Lavadora 3TS775BE (7 kg - 1200 rpm - Blanco)	399.99 €	1	399.99 €	Total (IVA incluido)	959.97 €
	Lavavajillas 3VS502IP (12 cubiertos - 60 cm - Inox)	279.99 €	2	559.98 €	Envío y manipulación	0 €
					<b>Total pedido</b>	<b>959.97 €</b>

Dirección de entrega

Orden de pago

*Ejemplo de carrito lleno.*

El *PurchaseLineService* debe guardar las líneas de carrito como compras pero para ello debe realizar las siguientes acciones (“lógica de negocio”):

- Comprometer la transacción. Para ello se utiliza la anotación JPA *@Transactional* que debe lanzar una excepción en tiempo de ejecución en caso de que ocurra algún problema en la transacción e impidiendo la transacción a cualquier otro usuario conectado.
- Por cada línea de carrito:
  - Comprobar la disponibilidad de las unidades de cada línea de carrito lanzando una excepción en caso de no haber suficientes y abortando dicha transacción.
  - Fijar la fecha de la compra a la fecha en la que se realiza la transacción
  - Pedir al *CartService* que borre el cart que se ha guardado.

*@Transactional*

```
public void saveCarts(List<Cart> carts) {  
    PurchaseLine purchase;  
    for (Cart cart : carts  
    ){  
        purchase = new PurchaseLine();  
        purchase.setProduct(cart.getProduct());  
        purchase.setUnitPrice(cart.getUnitPrice());  
        purchase.setCount(cart.getCount());  
        purchase.setPurchasePrice(cart.getCartPrice());  
        purchase.setUser(cart.getUser());  
        this.save(purchase);  
        this.cartService.removeCart(cart);  
    }  
}
```

```

@Transactional
public void save(PurchaseLine purchaseLine) {
    Product product=purchaseLine.getProduct();
    int count=purchaseLine.getCount();
    int productCount=product.getCount();

    checkQuantity(product,count);
    Calendar today=new GregorianCalendar();
    today.getTime();
    purchaseLine.setDate(today);
    product.setCount(productCount-count);
    purchaseLineRepository.save(purchaseLine);
}

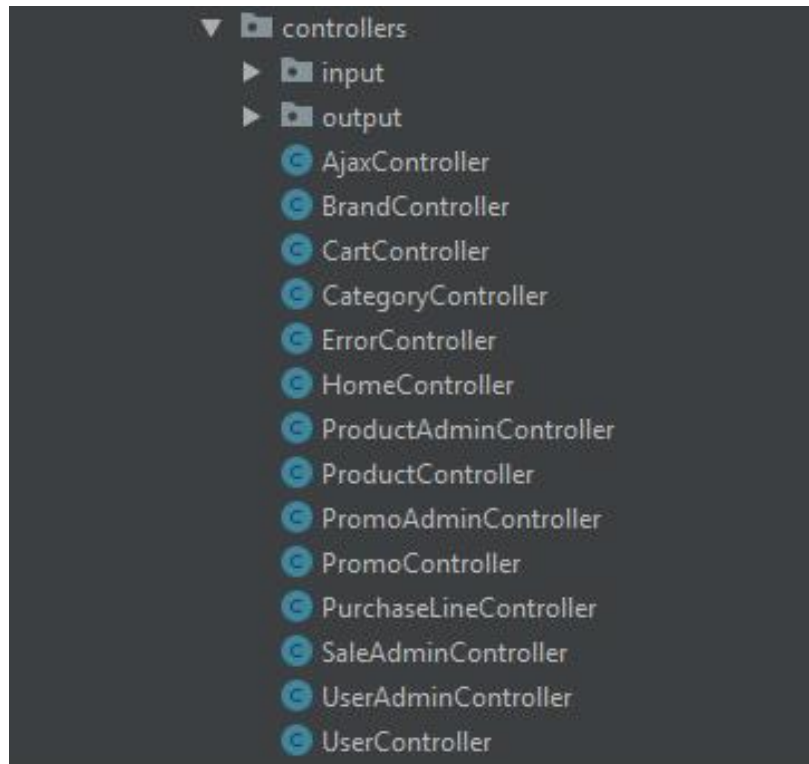
private void checkQuantity(Product product,int count) throws
RuntimeException{

    if(product.getCount()<count) throw new RuntimeException("No hay
suficientes unidades del producto : "+product.getName()+"\n" +
        "Unidades solicitadas =" +count+"\n" +
        "Unidades disponibles = "+product.getCount());
}

```

## 9.-Capa “Controlador” del patrón MVC

La capa “controlador” de la arquitectura de nuestro diseño está ubicada en el paquete *uned.webtechnologies.shop.controllers*.



*Clases Controller*

Al igual que con los servicios, hemos tratado de diferenciar bien las responsabilidades de los controladores, teniendo en cuenta que en esta capa, los controladores están haciendo de interlocutores entre las diferentes vistas y servicios de nuestra arquitectura.

Los controladores básicamente se encargan de responder a las solicitudes de las vistas poniendo a disposición de las mismas los datos del modelo necesarios para su renderizado y en el sentido contrario se encargan también de recoger los datos de las vistas (“generalmente de formularios”) para enviarlos a los servicios y delegar la responsabilidad de su tratamiento en estos últimos.

Nuevamente Spring facilita de forma considerable estas tareas gracias a las anotaciones que nos ofrece para este fin. @Get, @Post, @RequestParam, etc.

### 9.1.-Securización de funcionalidades según roles:

#### 9.1.1.- Admin

Para la gestión de nuestra aplicación web según los roles de usuario hemos utilizado Spring security.

Para limitar las funcionalidades de los usuarios con roles diferentes a los del administrador hemos creado unos controladores específicos para administradores que responden de manera que nadie pueda acceder a los métodos de dichos controladores gracias a la anotación `@Secured("ROLE_ADMIN")`

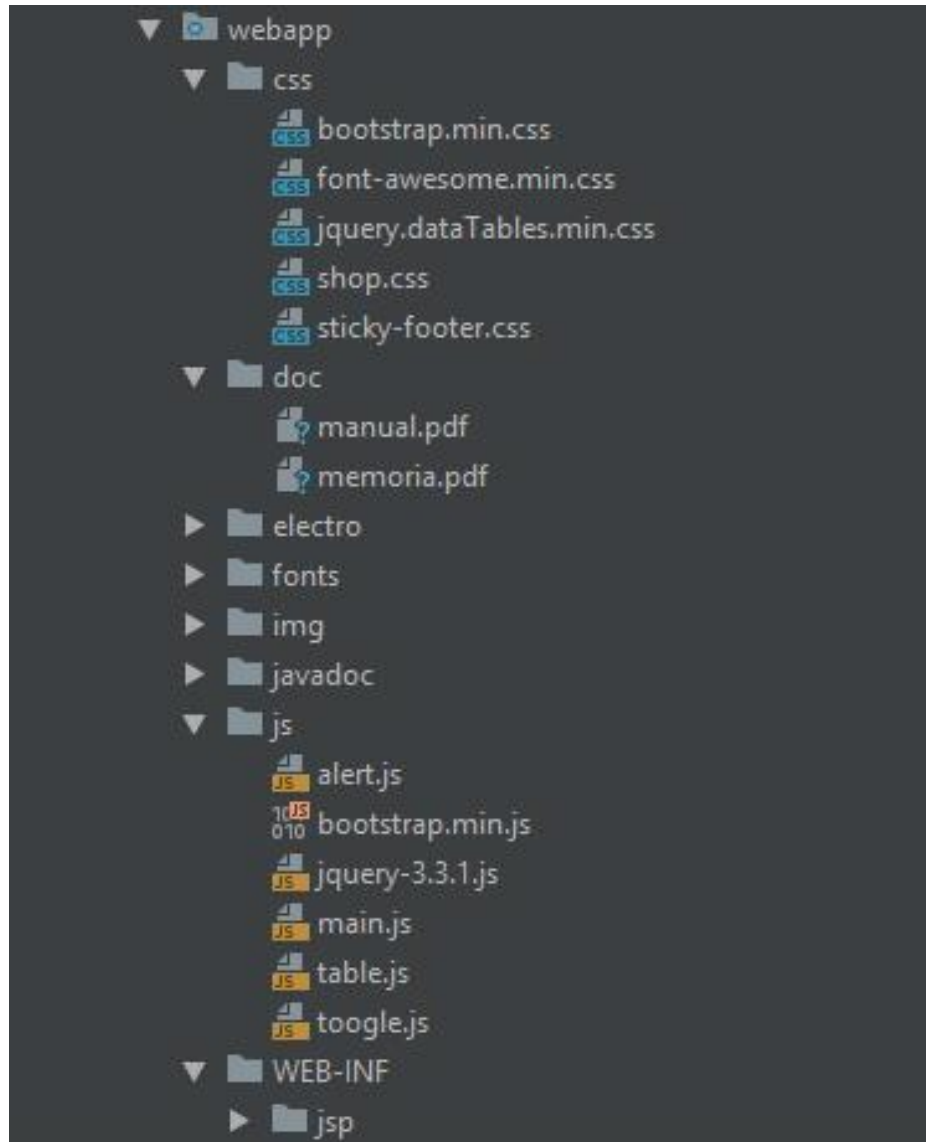
Un ejemplo de ello es el `ProductAdminController.java` que brinda al administrador las funcionalidades de creación y edición de productos.

#### **9.1.2.- Cliente.**

Para el cliente no es necesario implementar controladores específicos ya que los métodos dirigidos a clientes reciben como argumento el usuario autenticado gracias a la anotación `@AuthenticationPrincipal` y spring se encarga del resto.

## 10.-Capa “Vista” del patrón MVC

Para la capa de vista se han utilizado páginas JSP utilizando la librería de tags JSTL gracias a la cual se pueden generar unos archivos JSTL más legibles de cara a los desarrolladores web gracias a su sintaxis basada en etiquetas de marcado.

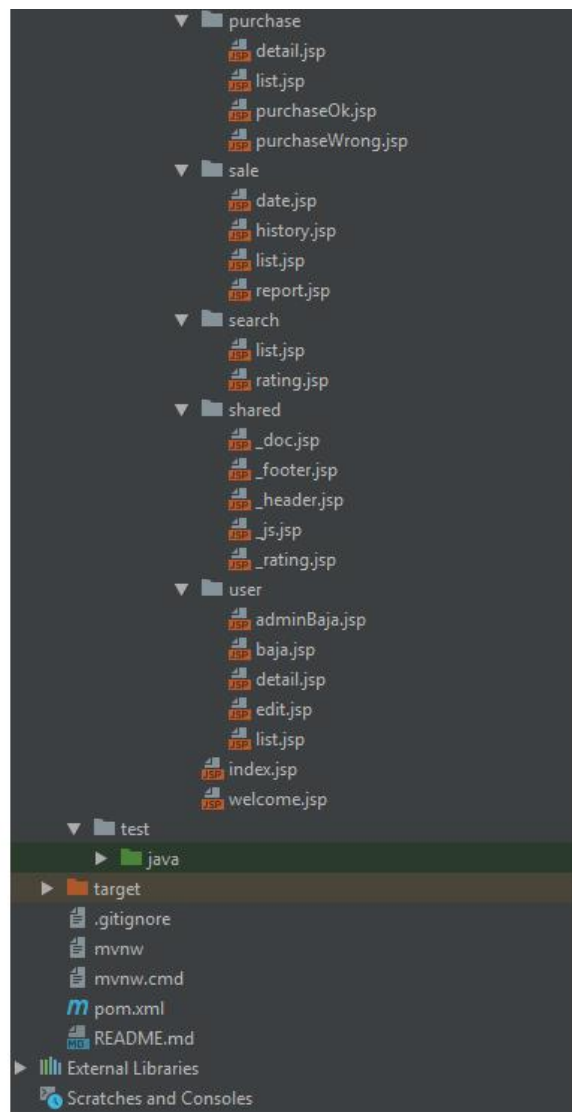
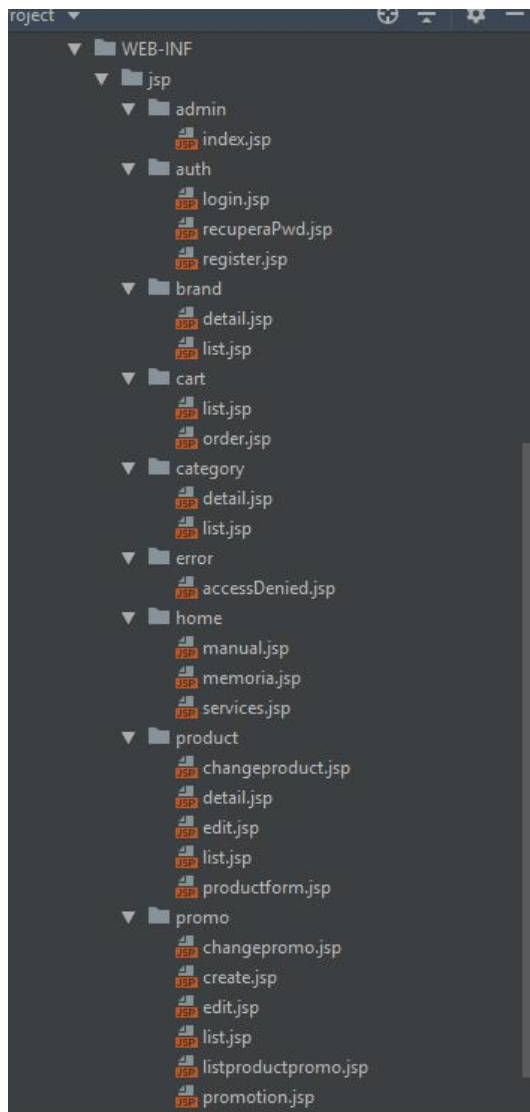


*Árbol con los ficheros de la “vista”*

Para el renderizado de las vistas se ha utilizado BOOTSTRAP 4.

Hemos tratado de utilizar el mínimo de lógica posible en esta capa, centrando su funcionalidad lo más estrictamente posible en la representación gráfica de los datos y en la funcionalidad de interacción con el usuario.

Las vistas son las encargadas de recoger la información del usuario pasándoselos al controlador y por otro lado, de renderizar los datos recibidos desde controlador.



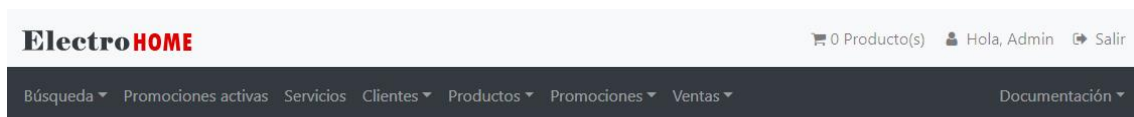
Árbol con la disposición de carpetas y ficheros \*.jsp

# 11.- Patrones de diseño de navegación.

Hemos definido y desarrollado un diseño de navegación por la aplicación web que depende claramente de las funcionalidades que tiene cada perfil de usuario que accede a la misma.



*Menú principal para usuarios no registrados y clientes*



*Menú principal para el perfil tienda o administrador*

Hemos usado para ello varios tipos de patrones de diseño que nos ha ayudado a que la navegación del usuario por la aplicación sea más sencilla y usable.

- El logotipo de la aplicación está localizado en la parte superior izquierda. Es de los patrones más comunes. Haciendo click en el mismo, mediante un hipervínculo nos llevará siempre a la “homepage” o página de inicio.



*Logotipo situado en la parte superior izquierda de la página*

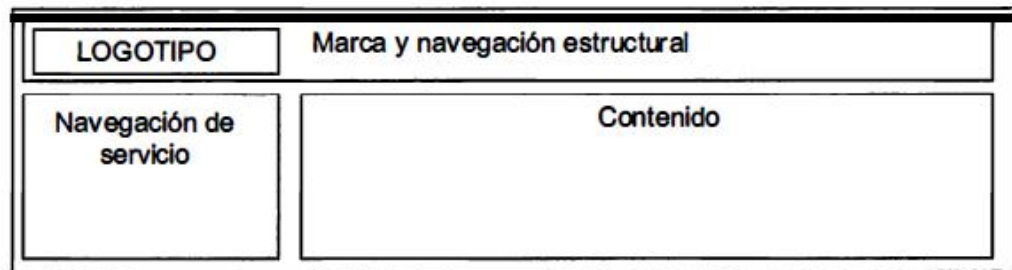
De esta forma el usuario puede acceder de forma rápida y desde cualquier página a la de inicio.

- Hemos usado otro de los patrones más comunes y conocidos como es el patrón “Barra de navegación”. Para el menú principal hemos dispuesto todas las opciones o funciones de forma horizontal y fácilmente accesible. Para que además sea más fácil su identificación hemos empleado un color gris oscuro que hace que resalte aún más su posición dentro de la página con respecto al resto de elementos.

Este patrón barra de navegación incluye el patrón del logotipo en la parte superior izquierda anteriormente comentado y que funciona como enlace a la página de inicio.

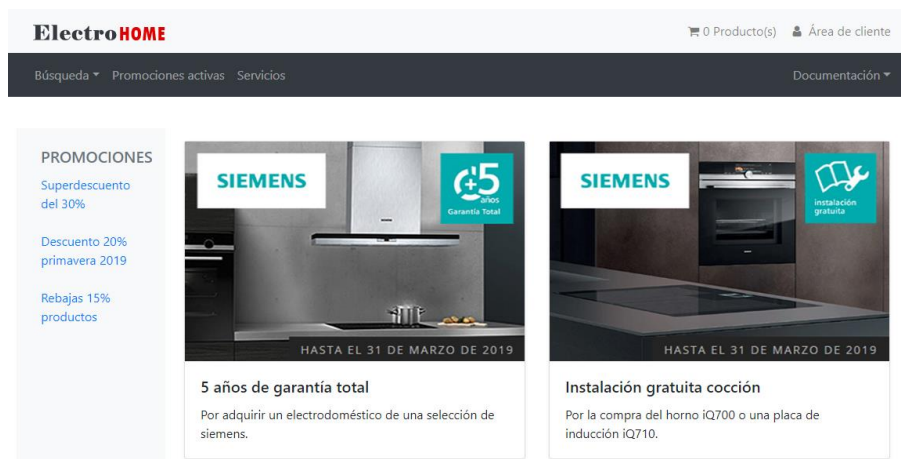
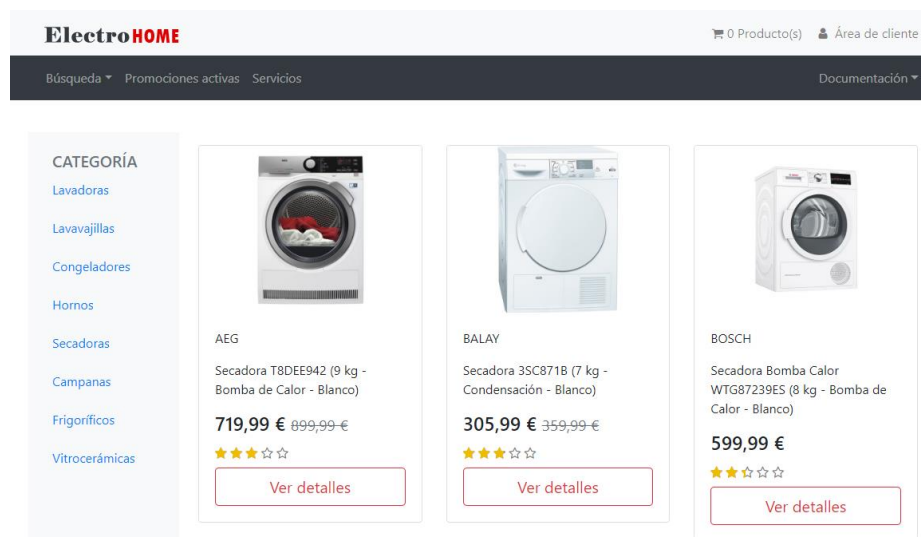
Además en este caso e independiente del tipo de perfil del usuario hemos colocado un acceso a la documentación de la aplicación para que resulte más fácil su acceso sin tenernos que identificar en ningún momento.

- Por último para algunas secciones de la aplicación web hemos usado el patrón “diseño en tres regiones”. Para ello hemos usado los dos anteriores patrones añadiendo una tercera región lateral donde hemos situado la navegación de servicio.



*Esquema patrón “diseño en tres regiones”*

Ejemplo de este uso, es en las secciones de búsqueda de la aplicación. Es precisamente en esta zona lateral donde hemos dispuesto las categorías, las promociones, las marcas o las diferentes valoraciones de los productos.



*Ejemplos de uso del patrón “diseño en tres regiones”*



## **12.- Conclusiones.**

La presente práctica no habría sido posible de no haber sido por el buen equipo que hemos formado. El trabajo en equipo ha sido algo nuevo para la mayoría de los integrantes del equipo y ha sido una experiencia enriquecedora.

Las nuevas tecnologías constantemente emergentes pueden resultar en ocasiones abrumadoras, hecho que puede llevar al desinterés por utilizarlas debido al esfuerzo que supone el aprendizaje de las mismas, pero esta práctica es la prueba evidente de que merece la pena hacer un esfuerzo inicial por decidir correctamente las tecnologías a emplear y aprender a utilizarlas, ya que tiene su recompensa a corto plazo en cuanto a rapidez y facilidad de mantenimiento.

## **13.- Anexos.**

### **13.1.- Tecnologías empleadas para el trabajo en grupo.**

#### **13.1.1 .- GitHub.**

Como sistema de control de versiones, hemos utilizado Github, una herramienta desconocida para dos de los integrantes del grupo y que ha resultado increíblemente útil.

#### **13.1.2 .- Whatsapp**

Para la comunicación constante hemos utilizado whatsapp por su simplicidad y popularidad aunque en discusiones complejas hemos tenido que acudir a la tradicional llamada de teléfono.

#### **13.1.3.- GoogleDrive**

Para compartir los documentos de texto y diagramas así como cualquier otro tipo de archivo hemos utilizado google drive.