UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

# Desarrollo de una tienda online para la venta de artículos de ferretería

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Carceller Llorens, Fernando

Tutor/a: Vidal Oriola, Germán Francisco

Cotutor/a externo: CARCEL SERRANO, JOSE MIGUEL

CURSO ACADÉMICO: 2021/2022

# Resumen

Con este trabajo de fin de grado se pretende diseñar e implementar una aplicación web para la venta de productos de ferretería de la empresa Enrique Llorens Ciurana S.L. Esta empresa, a pesar de que se dedica en gran parte a la programación y desarrollo de almacenes y cadenas logísticas, también tiene una subvertiente que consiste en una tienda de ferretería y menaje en el pueblo donde se localiza la empresa, Buñol. La aplicación consistirá en una única aplicación web que contendrá un sistema para la administración y gestión de ésta por los empleados de la empresa y un modelo de aprendizaje automático para recomendar productos a los usuarios. Los usuarios podrán comprar productos, consultar catálogos, ponerse en contacto con la empresa desde los contactos que encuentren allí, etc. Por otra parte los administradores podrán añadir modificar y eliminar artículos, añadir y eliminar catálogos, seleccionar productos destacados que aparecerán en la portada y gestionar los pedidos que vayan realizando los usuarios.

**Palabras clave:** Aplicación, web, tienda, ferretería, ASP.NET MVC, aprendizaje automático

# Abstract

This TFG is focused on designing and implementing a web application with the main interest being a web store for Enrique Llorens Ciurana S.L. bussines hardware products. Even though this company is centered around software development for warehouses and logistic chains, it also has a section dedicated to selling hardware and DIY products in Buñol, the village where the main company hub is located. This software will consists of a single web application containing a management system in which the employees will be able to manage the available products and an AI that will deal with the user recomendation. Furthermore, the users will be able to buy products, search catalogs and contact the company using the provided information. On the other hand, administrators will be able to add, modify and remove products and catalogs, select distinguished products for them to appear on the main page and manage the orders requested by the users.

**Keywords :** Application, web, store, hardware store, ASP.NET MVC, Machine Learning.

# Índice general

# Índice de figuras

# Índice de tablas

# Índice de fragmentos de código

# CAPÍTULO 1
# Introducción

## 1.1 Motivación

Hoy en día, las empresas necesitan una exposición en internet si quieren crecer y prosperar. Esto se puede conseguir con un buen posicionamiento en las redes sociales más utilizadas, con anuncios o con páginas o aplicaciones web. Además, en general, cuanta más exposición tenga la empresa en internet, mejores resultados tendrá en sus negocios. También cabe destacar el auge de las tiendas virtuales, las cuales han transformado la compraventa de cualquier tipo de productos en una tarea tan simple como hacer un par de clics y que, en contraposición con los comercios tradicionales, permiten a las personas disponer de un inmenso catálogo de productos a cualquier hora del día, en cualquier día de la semana y con entrega a domicilio, cada vez en menos tiempo. Este auge es cada vez mayor debido a plataformas como Amazon o AliExpress, las cuales venden sus propios productos y a la vez permiten a empresas externas vender en su tienda virtual sus productos, siempre que cumpla las condiciones de la plataforma, ya que estas son las que gestionarán todo el proceso de ventas y devoluciones.

Teniendo en cuenta lo anterior, muchos negocios contratan empresas dedicadas al desarrollo de páginas web para que creen su página web. Esto, sin embargo, puede no ser la mejor de las soluciones, ya que estas empresas no suelen considerar el mantenimiento de la página web. Por ello puede ocurrir que se quede desactualizada con facilidad o que el encargado o encargados de mantener la página dentro de la propia empresa tengan que desperdiciar tiempo estudiando el código que han escrito las personas de dichas empresas. Otra alternativa podría ser crear una web con un gestor de contenidos como WordPress, aunque esta solución tampoco es idónea debido a la poca flexibilidad de las plataformas, que están pensadas para poder crear webs muy generales y poder abarcar un mayor mercado. Descartadas las opciones anteriores, la empresa solicitó la implementación de una tienda virtual donde ellos pudieran tener el control del código y el mantenimiento de la aplicación, permitiéndoles realizar cambios cuando ellos deseen y no dejando la web desactualizada en ningún momento.

Por todo ello, nace la idea de este proyecto: la creación de una aplicación web para la empresa Enrique Llorens Ciurana S.L. y , más concretamente, para su subvertiente enfocada a la venta de productos de ferretería, fontanería, hogar y eléctricos, entre muchos otros, que permita darle una mayor visibilidad al comercio más allá del pueblo donde se sitúa y que incluya la gestión de productos del propio comercio y la venta de dichos productos creando así una tienda virtual de la empresa.

## 1.2 Objetivos

En este trabajo se estudiará el panorama actual de las aplicaciones web dedicadas a la venta de productos, tanto de nuestro dominio (para poder encontrar una forma de diferenciarnos), como de otras áreas (para buscar similitudes y encontrar inspiración en ellas). De este modo, se aspira a implementar una aplicación web para Enrique Llorens Ciurana S.L. que permita dar una mayor visibilidad a su comercio (Elecfer), facilitando algunos contactos con la empresa, exponiendo el catálogo de productos de los que puede disponer el cliente en esta empresa e incluso implementando una tienda virtual para que los clientes puedan comprar los productos desde casa y recibirlos en un plazo normalmente no superior a 2-3 días.

Más concretamente, se pretende desplegar esta aplicación en el dominio elecfer.com, el cual pertenece a la empresa Enrique Llorens Ciurana S.L., a modo de MVP [1] e ir mejorándola con el paso del tiempo hasta lograr una tienda virtual competitiva (al menos, dentro del mercado español). Para ello también se estudiarán funcionalidades que puedan ayudar a nuestro comercio a destacar y posicionarse por delante de otros comercios para los clientes. También formará parte de este trabajo implementar funciones de administración de la tienda, como la administración de productos, catálogos y pedidos, dentro de la propia web creando un subsistema en ésta que solo será accesible para personas con rol de administrador.

## 1.3 Impacto Esperado

El impacto de este trabajo lo podremos apreciar tanto en los clientes como en los administradores:

- Por parte de los clientes, se espera que estos empiecen a utilizar la aplicación web para realizar las compras que suelen realizar en la propia tienda de una manera cómoda y segura y que les permita obtener sus productos con la mayor prontitud posible. Además, estos clientes podrán revisar los catálogos en formato PDF, que se irán actualizando cada cierto tiempo, y consultar los productos disponibles en la tienda desde la web. Por último, podrán acceder a las redes sociales de la empresa y consultar otras formas de contactar con la empresa.

- Por parte de los administradores y empleados de la tienda, se pretende que puedan realizar una correcta gestión de los productos, catálogos y pedidos de la forma más fácil posible, siendo la gestión de pedidos una de las partes más importantes.

Se espera que la aplicación web pueda atraer nuevos clientes a la empresa desde cualquier parte de España, que de otra forma no podrían ser clientes potenciales debido a la localización de una única tienda física en Buñol. De esta forma se quiere extender el negocio a todo el territorio español y continuar creciendo en el ámbito del comercio.

## 1.4 Estructura de la memoria

Los capítulos que se listan a continuación serán los que conformarán la memoria y por tanto los que documentarán todo el proceso de desarrollo del trabajo:

- **Capítulo 2. Estudio estratégico:** En este capítulo realizaremos un estudio del mercado actual tanto de comercios online que se dedican exclusivamente al mismo dominio de productos que nosotros como de comercios o *marketplaces* [2] más grandes y con un dominio más amplio.

- **Capítulo 3. Metodología:** Se expondrán las herramientas y metodologías utilizadas durante el desarrollo del proyecto.

- **Capítulo 4. Especificación de requisitos:** En este capítulo profundizaremos en la parte de especificación de requisitos del proyecto.

- **Capítulo 5. Diseño de la solución:** Se presentarán las decisiones tomadas en el diseño de la aplicación, las tecnologías, lenguajes y *frameworks* [3], destacando por qué se descartaron el resto de las opciones.

- **Capítulo 6. Desarrollo de la solución propuesta:** Aquí comentaremos las partes que componen nuestra aplicación web, destacando su comportamiento y funcionamiento, siendo el apartado más técnico de la memoria.

- **Capítulo 7. Pruebas:** En esta parte se describe el proceso de pruebas que se ha realizado sobre nuestro software para comprobar que todo está correcto y listo para pasar a producción.

- **Capítulo 8. Despliegue:** Aquí se comentará el proceso de implantación que se ha seguido para desplegar la aplicación web y que sea accesible para los usuarios finales.

- **Capítulo 9. Conclusiones y trabajos futuros:** En este apartado se pueden encontrar las conclusiones posteriores al desarrollo, prueba y despliegue del proyecto. Por último, también comentaremos las tareas que quedan pendientes tras este trabajo y las mejoras que se pretenden implementar en el propio software.

- **Anexos:** Aquí podremos encontrar información de referencia que no aparce en los capítulos anteriores pero que puede resultar útil para el lector.

# CAPÍTULO 2

# Estudio estratégico

## 2.1 Comercio de ferretería - Elecfer

Enrique Llorens Ciurana S.L. fue fundada en 1962, pero no fue hasta 2006 cuando se decidió crear un nuevo apéndice en la empresa que se dedicaría a la venta de productos de ferretería, enfocada a una clientela residente en Buñol, pueblo en el que se sitúa la empresa, y en localidades adyacentes.

Desde el momento de su apertura ha resultado un éxito y, a pesar de las varias mudanzas que ha tenido la empresa, la clientela no ha hecho más que crecer, lo que ha permitido a los administradores y empleados de la propia empresa el plantearse expandir sus fronteras e ir más allá de la comarca en cuanto a clientela se refiere, Para ello, la apertura de una tienda virtual es una de las mejores opciones.

Obviamente, no es lo mismo triunfar en un comercio de pueblo que en una tienda virtual, donde la competencia no es un número reducido de comercios con oportunidades similares a las nuestras, sino que existe competencia con prácticamente todo el mundo. Así, habrá comercios o empresas que tengan una ventaja de partida sobre nosotros, ya sea por ser empresas ya implantadas desde hace mucho tiempo y que son fáciles de reconocer para los clientes o por negocios de éxito que puedan abarcar nuestro catálogo de productos dando una mejor oferta de estos productos al cliente. Para que este proyecto no termine en fracaso, se debe realizar un estudio de mercado donde observaremos a empresas similares a la nuestra y a empresas más grandes que abarquen un mayor dominio de productos. De este modo, buscaremos referencias a la hora de crear nuestra aplicación, para intentar diferenciarnos y hacernos un hueco en este mercado.

## 2.2 Comercios de ferretería en la era digital

Desde la llegada de internet a nuestras vidas, las empresas han intentado aprovecharlo de distintas maneras: dar visibilidad a la empresa, buscar nuevas formas de llegar a los clientes, anunciarse en diversas plataformas, … Pero la forma más común de utilizarlo es creando su propia página web. Según el tipo de empresa la web puede servir como un expositor de la empresa a todo el mundo. Pero, además, las empresas que poseen comercios son capaces de implementar tiendas dentro de sus webs, haciendo que el cliente ni siquiera tenga que moverse de su silla para poder comprar sus productos. Por otro lado, estos productos cada vez llegan más rápido a casa del cliente, convirtiéndose en uno de los modelos de negocio para los comercios más exitosos, sobre todo entre la gente joven, más familiarizada con las nuevas tecnologías e internet.

Para que nuestra aplicación web tenga éxito en este panorama de tiendas virtuales deberemos realizar un estudio de algunas de estas páginas web, lo que nos permitirá conocer las fortalezas y debilidades de

cada una, pudiendo guiarnos para la especificación de requisitos del proyecto. También analizaremos algunas webs de empresas más grandes que incluyan nuestro dominio, aunque no sea su prioridad.

### 2.2.1 Ferretería Dols

La tienda virtual de ferretería Dols fue uno de los que nos indicaron los administradores como referencia a la hora de realizar el proyecto.

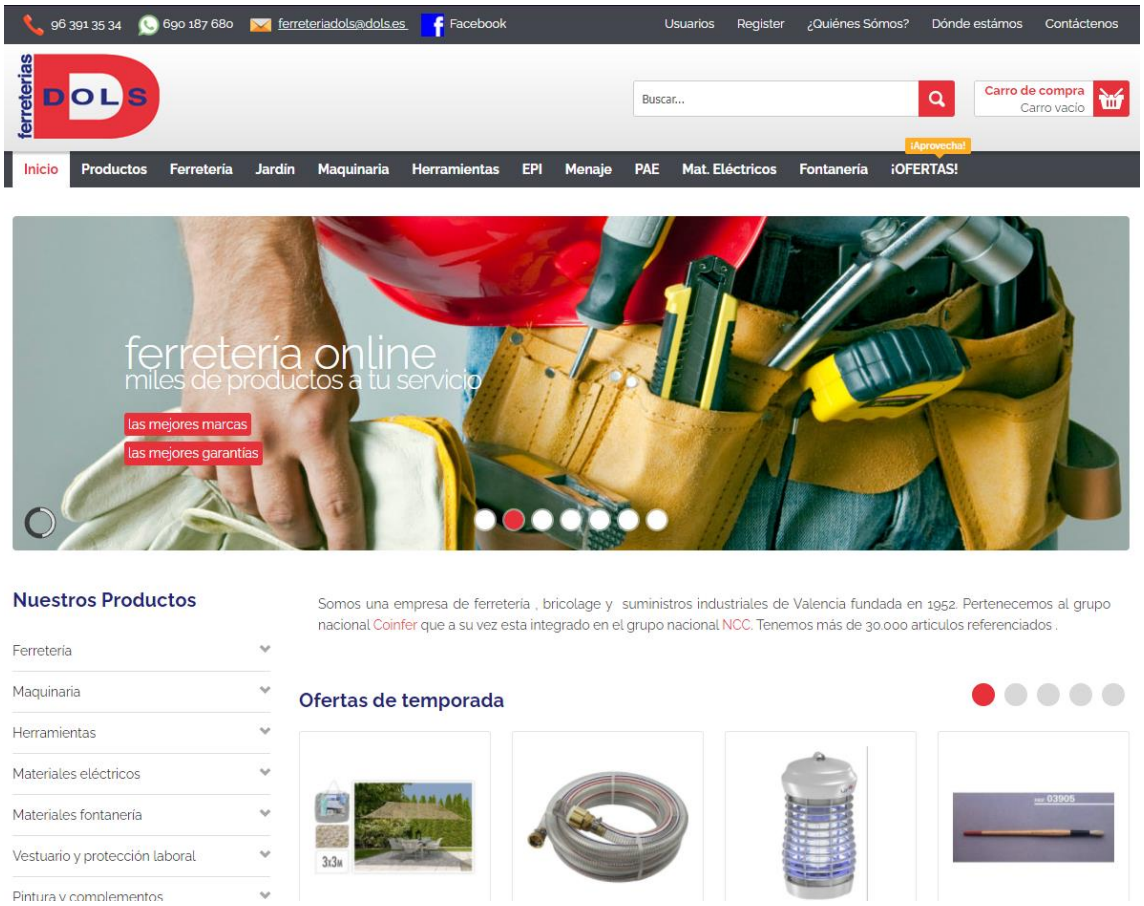


**Figura 2.1:** Pantalla de inicio ferretería Dols

Lo importante de esta web es la distribución de los artículos en familias bien estructuradas, lo que nos permite encontrar los productos de una forma sencilla sin la necesidad de usar un buscador. De hecho, el buscador es posiblemente una de las funcionalidades menos detalladas de la web, ya que no muestra nada de *feedback* [4] durante la escritura de la búsqueda que queremos realizar. En otras tiendas se puede observar un autocompletado del nombre de productos relacionados o incluso un desplegable con artículos relacionados con la búsqueda, sin la necesidad de redirigirnos a la búsqueda entera (ver figura 2.1).

Por último, podemos destacar la falta de recomendaciones reales, puesto que en la web se pueden observar ofertas de temporada y artículos destacados, pero estos no dependen en absoluto de nuestras visitas a productos, compras o referencias. Aunque no son siempre los mismos productos los de estas secciones, parece que van rotando productos de una lista preseleccionada para ambos campos.

En cuanto a los métodos de pago, solo se pueden realizar pagos mediante PayPal o transferencia bancaria, algo que a priori parece un poco pobre, ya que no incluye siquiera la forma más habitual de pago online: la tarjeta de crédito.

### 2.2.2 Ferreteria.es


**Figura 2.2:** Pantalla de inicio ferreteria.es

Aunque a priori puede parecer que tiene una organización más simple que la web anterior, dentro de cada familia podemos encontrar más subdivisiones de productos, teniendo una organización más completa. Podemos encontrar un buscador que nos ofrece gran cantidad de productos para acceder que aparecen en pantalla según se va escribiendo, el inconveniente de esto, sin embargo, es que son demasiados y puede saturar al usuario.

Por otro lado, no hay unas recomendaciones reales. En la pantalla principal aparecen dos carruseles de artículos que contienen siempre los mismos productos y en cada una de las subdivisiones aparece una lista de productos pertenecientes a esa división que tampoco varían entre sesiones (ver figura 2.2).

En cuanto a los métodos de pago que se pueden utilizar, encontramos tarjeta de crédito, transferencia, Bizum y PayPal, con lo que cubren un amplio rango de métodos de pago, dándole distintas posibilidades al cliente para realizar sus compras.

### 2.2.3 ManoMano

ManoMano es una de las tiendas online más conocidas de productos de ferretería y relacionados en España. Esta popularidad se debe probablemente a una serie de campañas de marketing que se realizaron hace unos años y a la gran variedad de productos con los que trabaja (ver figura 2.3).

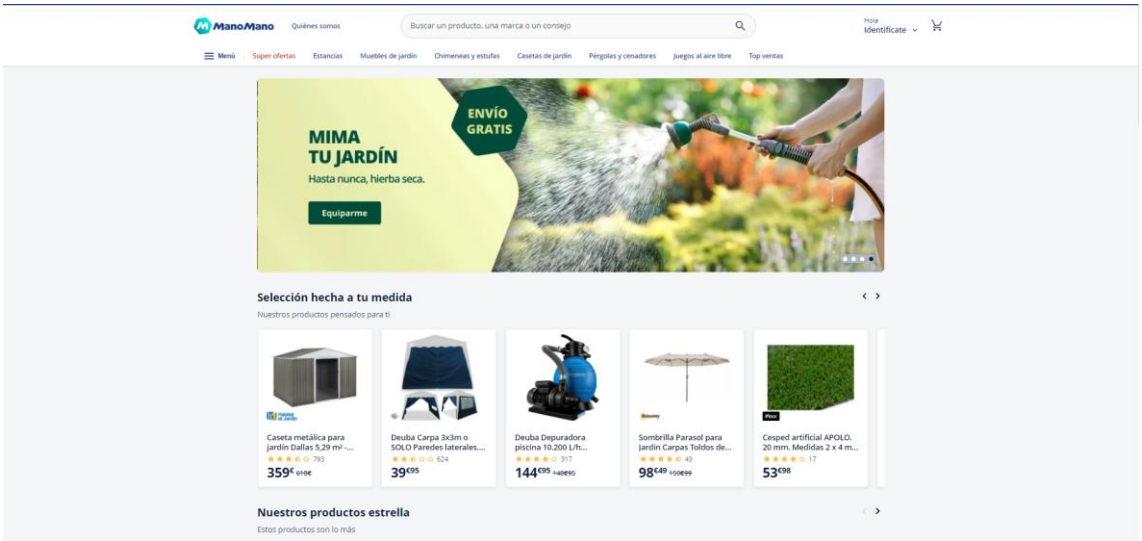


**Figura 2.3:** Pantalla de inicio de ManoMano

Como ya hemos comentado, en ManoMano podemos encontrar una gran cantidad de productos bien estructuradas en diversas familias y subfamilias dentro de estas. La web nos proporciona un buscador que da *feedback* en forma de un autocompletado de nuestra búsqueda que no solo se centra en artículos, sino también en familias y subfamilias.

En este caso, sí que encontramos recomendaciones reales bajo la sección “Selección hecha a tu medida” (ver figura 2.3), distintas para cada usuario de la web y, además, unas secciones con ofertas y productos destacados dentro de la misma página.

Aparte de la página web, también existe una aplicación móvil para Android y iOS con la misma funcionalidad que la web.

ManoMano nos permite realizar pagos mediante tarjetas de crédito o mediante PayPal, dos de los pagos más recurrentes por los comercios online debido a que son los más usados por los usuarios.

### 2.2.4 Amazon

Amazon es una de las empresas más conocidas alrededor del mundo y, aunque es una obviedad que no se le puede hacer competencia a una empresa de este calibre, (además de que no está centrada en ferretería, es interesante analizarla, aunque sea superficialmente para tomar referencias y extraer requisitos (ver figura 2.4).

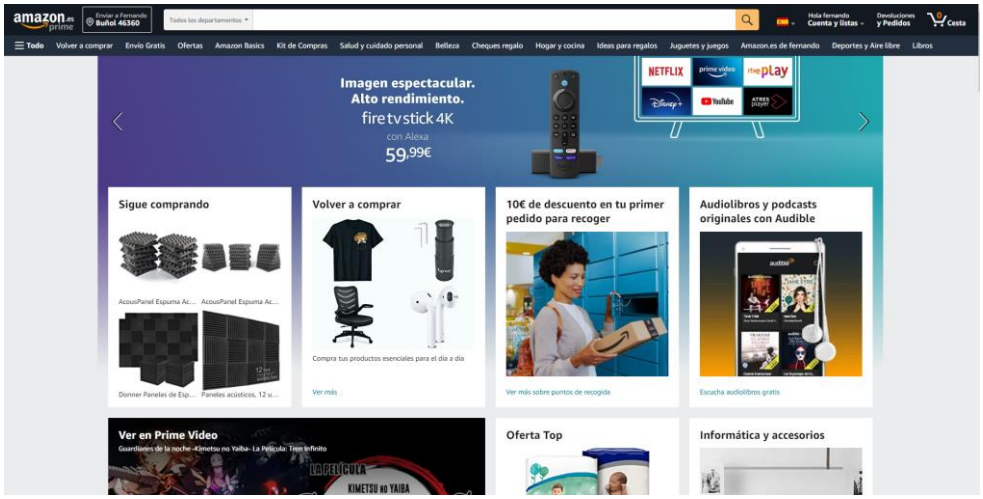


**Figura 2.4:** Pantalla de inicio de Amazon

En Amazon podemos encontrar recomendaciones según las compras y recomendaciones según las preferencias de los usuarios. Además, encontramos varias secciones de productos destacados, productos visitados pero no comprados, ofertas y secciones que más visita el usuario, etc...

El buscador es similar al de ManoMano, siendo este todavía más completo, debido a la amplitud de productos que ofrece Amazon, donde se nos proporciona un *feedback* autocompletando por productos, grupos de productos, secciones e incluso temáticas, además de mostrarnos búsquedas que ya realizamos en otras ocasiones.

En términos de pagos, Amazon siempre ha apostado por una única forma de pago con tarjeta de crédito, lo que tiene sentido al ser la forma más utilizada por los usuarios y la más común de encontrar en estos tipos de comercios, aunque realmente sería interesante añadir nuevos métodos de pago que han ido surgiendo y otras tiendas virtuales están adoptando poco a poco, como podrían ser PayPal o Bizum.

Hay que destacar que Amazon es una plataforma de ventas, por lo que no todos los productos son vendidos directamente por Amazon, sino que Amazon permite que otras empresas vendan en su web mientras cumplan unas condiciones que plantea la empresa.

Por último, cabe destacar que Amazon no solo consta de esta tienda, sino que dentro de sus servicios podemos encontrar Amazon Music un servicio de *streaming* [5] de música, Amazon Prime Video, un servicio de *streaming* de series y películas, etc. Todos estos servicios se reúnen bajo una suscripción llamada Amazon Prime, que además de darnos beneficios en estas plataformas adyacentes, nos permiten recibir descuentos en gran cantidad de productos ciertas veces al año, recibir la mayoría de productos vendidos directamente por Amazon en un plazo de 24 horas y con envíos gratuitos.

## 2.3 Propuesta

Una vez explicado un poco el contexto de la empresa y observadas las funcionalidades de algunos de las tiendas virtuales más interesantes relacionados con la temática de la ferretería, se tendrán en cuenta los siguientes aspectos a la hora de la creación de nuestro producto:

- Se diseñará una aplicación web similar a las que hemos visto en el apartado anterior, pero por el momento no se considera la creación de una aplicación móvil.

- Se realizará un diseño básico pero efectivo en las interfaces de usuario en una primera instancia, algo que permita una funcionalidad completa, y este se irá mejorando poco a poco con siguientes entregas.

- Se trasladarán la gran mayoría de productos de la tienda física, ordenándolos en familias y subfamilias de forma similar a las webs vistas anteriormente, facilitando así la búsqueda de productos.

- La aplicación contará con un buscador de productos que nos proporcionará *feedback* en forma de un desplegable con algunos artículos relacionadas con la búsqueda, desde las cuales podremos acceder directamente a esos productos, permitiéndonos hacer una búsqueda más intensiva que arroje todos los resultados encontrados.

- La aplicación contará con un motor de aprendizaje automático [6] que revisará las visitas de los usuarios a los artículos en relación con el histórico de productos visitados por los usuarios, seleccionando los productos más recomendables para el usuario y mostrándolos en la página principal.

- Se incluirá de momento el pago por tarjeta de crédito, ya que aparte de ser el más utilizado por este tipo de comercios y por el usuario, es uno de los servicios que cobra menos comisiones en comparación (siendo uno de los que más PayPal, por lo que este último de momento queda descartado).

- Se ha observado que en muchas de las webs se utiliza una ventana modal para avisar de que se ha añadido un producto al carrito. Se intentará crear un carrito de compra dinámico que cuando se añadan productos se despliegue mostrando el contenido, lo que puede ser una mejor solución.

- En la página principal habrá una sección dedicada a los productos destacados, los cuales serán elegidos por los administradores.

- Se implementará un subsistema de gestión, solo accesible por los administradores de la web (los empleados de la tienda), para que estos puedan realizar la preparación, envío y facturación de los pedidos de una forma simple. Además, también podrán añadir y modificar los artículos, catálogos y artículos destacados, incluso eliminarlos.

# CAPÍTULO 3
# Metodología

A la hora de tomar la decisión sobre qué tipo de metodología deberíamos adoptar para la realización de este proyecto, se decidió elegir un enfoque ágil.

Esto se debe a que el uso de metodologías tradicionales [7] conlleva un flujo secuencial en el cual pasaríamos por fases de análisis, diseño, implementación, pruebas y mantenimiento (aunque cabe destacar aun así que, en metodologías tradicionales cíclicas, es viable realizar proyectos incrementales). El mayor problema que comportan este tipo de metodologías es su rigidez. En estas metodologías, tanto los requisitos del sistema como el tiempo de desarrollo tienen que estar muy bien definidos. Además, es común que en la mayoría de estas metodologías solo se vean los resultados en las fases finales, debido a este flujo secuencial que comentábamos anteriormente. Por último, los administradores de la empresa no tienen grandes nociones del desarrollo de software y los empleados que sí las tienen están envueltos en otros proyectos. Por lo tanto, un enfoque tradicional sería difícil de implementar al no tener un gerente del proyecto que marque unas pautas y estimaciones sobre éste.

Por otra parte, al elegir una metodología ágil [8] se nos permite de una manera sencilla realizar un proyecto incremental, es decir, seguir un concepto de MVP en nuestro proyecto, para que desde lo más temprano posible tengamos una aplicación con las principales funciones y podamos ir ampliando la funcionalidad con el tiempo. También es una ventaja la flexibilidad que comportan este tipo de metodologías, lo que nos permitirá no perder tanto tiempo al principio del proyecto realizando estimaciones sobre el tiempo o especificando requisitos. Por último, al tener una progresión constante de la aplicación, dividiéndolo en *sprints* [9], al final de los cuales tenemos una nueva versión de la aplicación. Esto puede resultar realmente útil para los administradores de la empresa, ya que les permite saber cómo está evolucionando el proyecto cada poco tiempo, resultando una forma más fácil para ellos de gestionar el proyecto.

Por todo esto se ha decidido utilizar SCRUM [10]. El proyecto se implementará de manera iterativa dividiendo el trabajo en diversas iteraciones o *sprints* (como ya habíamos mencionado). Estos *sprints* tendrán una duración de tres semanas, durante las cuales se trabajará en las funcionalidades que se acordarán previamente con los administradores de la empresa, que actuarán de *Product Owners* [11]. En estas reuniones, aparte de nuestro rol como equipo de desarrollo, donde deberemos ayudar a los *product owners* a elegir qué funcionalidades restantes deberían de ser las primeras en implementarse, también tendremos que tomar el rol de *scrum master* [12], donde nuestro papel consistirá en ayudar a entender a los *product owners* el funcionamiento de los procesos ágiles y cualquier duda que les surja relacionada con SCRUM.

Al final de cada uno de estos *sprints* se realizará una demo con los *product owners* para que puedan ver y probar la aplicación y se concretarán las funcionalidades que se van a programar en el siguiente sprint. Las funcionalidades que se eligen para los *sprints* han sido previamente acordadas en una reunión con los *product owners*, recogidas en el *backlog* [13] y ordenadas de más prioritarias a menos.

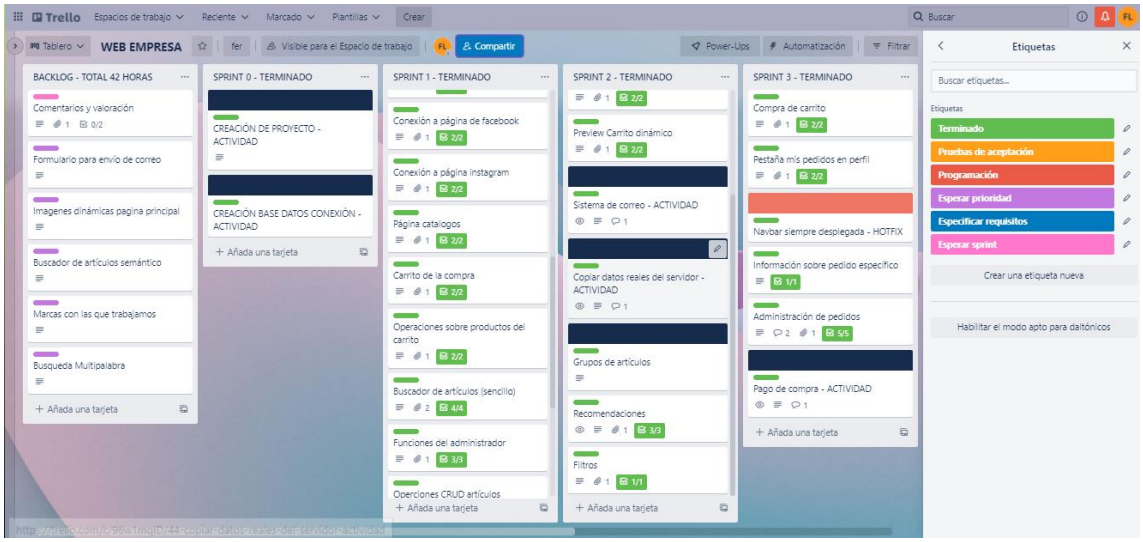


**Figura 3.1:** Tablero de Trello

Para la organización del trabajo se planteó la creación de un tablero de Trello, ya que lamentablemente la empresa no tiene acceso a licencias de programas de gestión de proyectos como Jira o Worki. Como podemos observar en la figura 3.1, Trello nos permite organizar varias tablas con múltiples tarjetas que podemos marcar con etiquetas e incluir información en su interior. En nuestro caso, se crearon 4 *sprints*, además del *backlog* donde se guardan todas las funcionalidades sin implementar. Para poder marcar en qué parte del desarrollo se encuentra cada una de las funcionalidades implementamos un sistema de etiquetas (ver figura 3.1, parte derecha), con las cuales se puede indicar en qué parte del *workflow* [14] se encuentra cada funcionalidad: Esperar *sprint*, Especificar requisitos, Esperar prioridad, Programación, Pruebas de aceptación y Terminado. Además, también añadimos cabeceras de color azul oscuro a las actividades (funcionalidades o tareas que se tienen que realizar para que funcione alguna parte de la aplicación, pero que no tienen interfaz de usuario o directamente son externas al propio proyecto) y de color rojo a los *hotfix* (problemas que se encuentran al hacer pruebas de regresión o implementar nuevas funcionalidades y que se han de corregir, pero que no ocupan mucho tiempo).

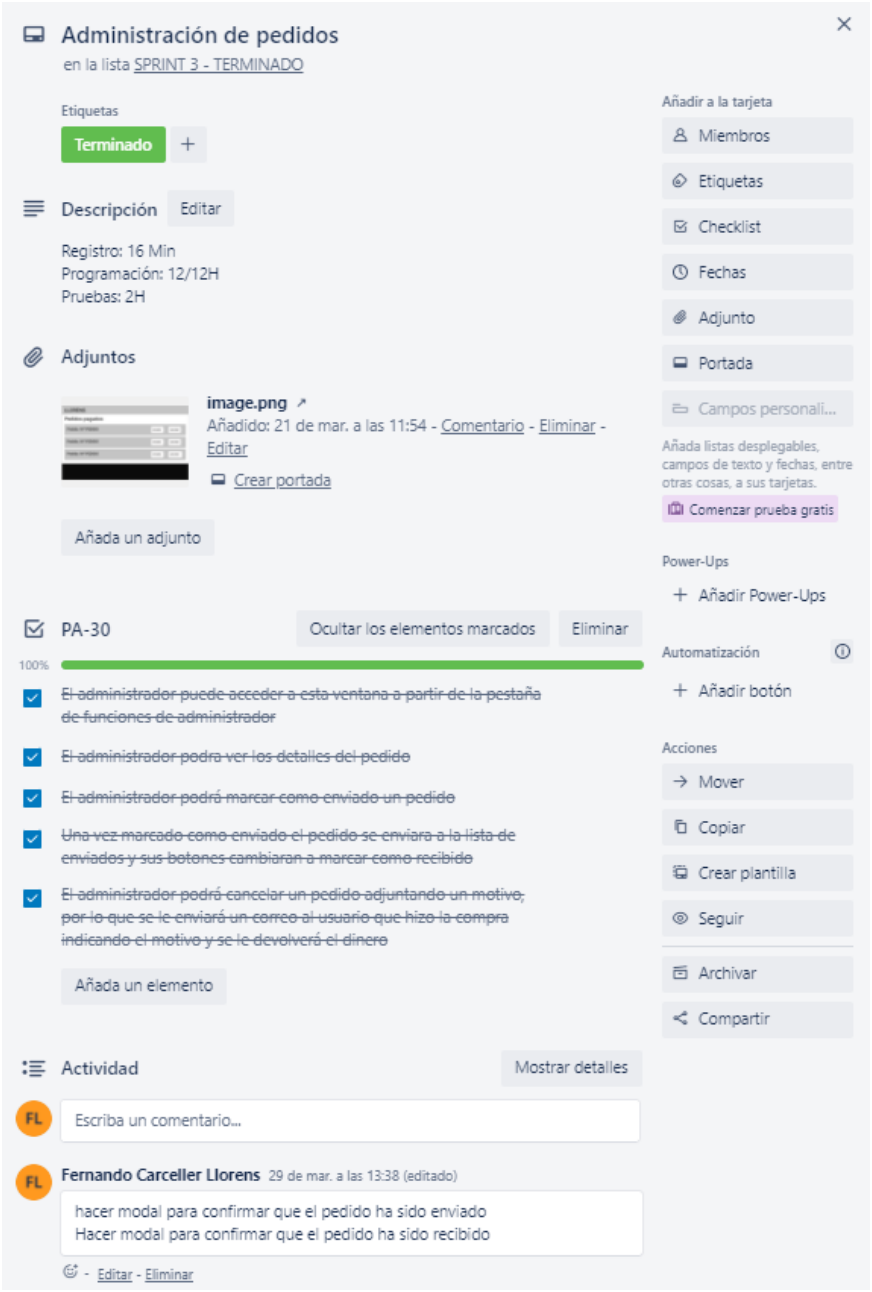


**Figura 3.2:** Modal de una tarjeta de Trello

Dentro de cada una de las tarjetas (ver figura 3.2) hemos añadido un *mockup* [15] de la interfaz de usuario asociada a la funcionalidad que queremos implementar, una descripción (donde podemos computar las horas dedicadas al registro y el diseño del mockup, la programación y las pruebas) y una *checklist* con pruebas de aceptación que debe cumplir la parte de la aplicación que se quiere implementar. Por último, utilizamos la sección de comentarios del final para añadir recordatorios de funcionalidades que faltan por implementar.

# CAPÍTULO 4

# Especificación de requisitos

A la hora de especificar los requisitos para el proyecto, al haber adoptado una metodología ágil y disponer de una comunicación tan estrecha como los *stakeholders* [16], se ha decidido utilizar una técnica informal de especificación utilizando lenguaje natural, es decir, se definirán los requisitos de manera que puedan ser entendidos tanto por los programadores como por los *stakeholders* que no estén familiarizados con prácticas o lenguajes más formales de especificación. De la misma forma, tomando la ventaja de que podemos mantener una comunicación estrecha y directa en cualquier momento con gran parte de los *stakeholders*, la elicitación de requisitos se basará completamente en esta comunicación, mediante reuniones y entrevistas.

## 4.1 Diagrama de contexto y modelo de dominio de la aplicación

En primer lugar, es interesante presentar el diagrama de contexto de la aplicación, que consiste en un sistema principal, la aplicación web con la tienda, que contendrá un subsistema de administración al que solo podrán acceder usuarios con el rol de administrador (ver figura 4.1).

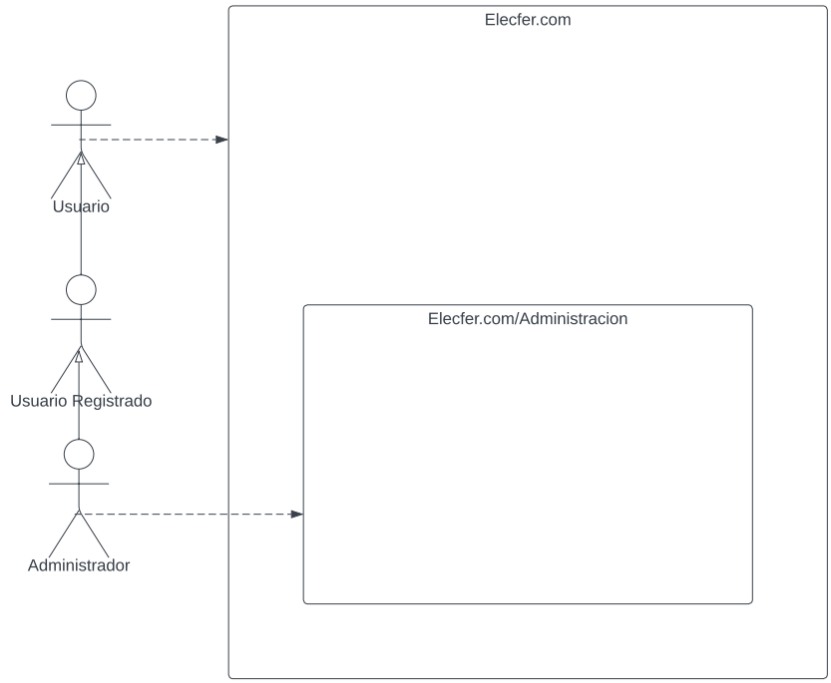


**Figura 4.1:** Diagrama de contexto de la aplicación

Además, también se realizó un modelo de dominio que relacionará los principales conceptos de la aplicación web (ver figura 4.2). En este modelo de dominio podemos observar las principales relaciones que tendrá la aplicación. Los administradores podrán subir tanto catálogos como artículos a la página web mediante las funciones de administración. Los usuarios podrán crear una cuenta en la web con la que podrán añadir artículos al carrito, realizar pedidos y pagarlos y añadir valoraciones a los productos.

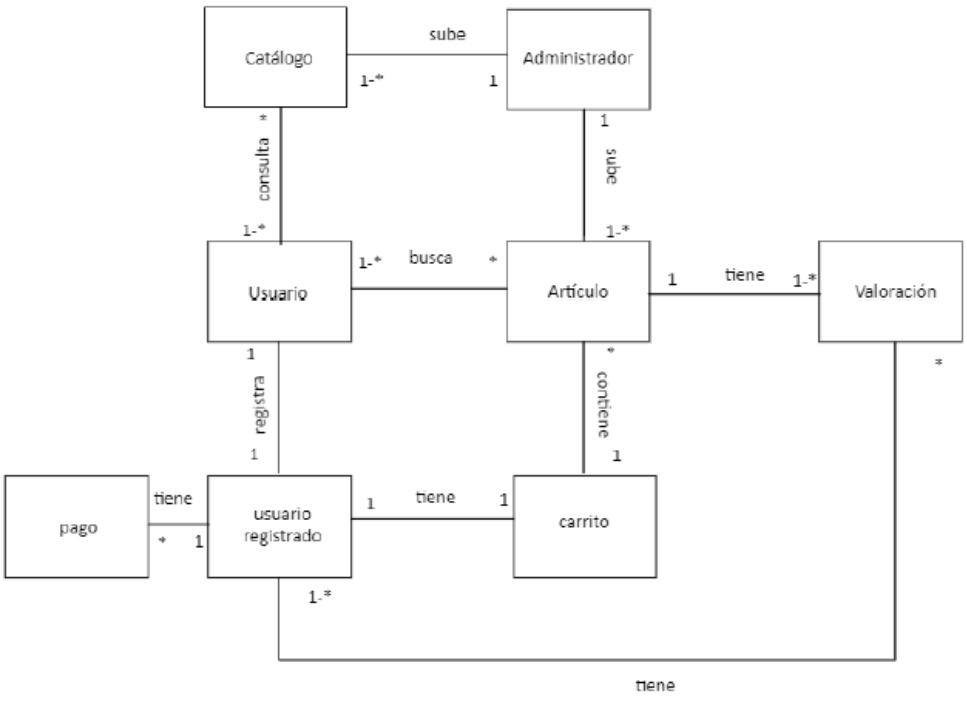


**Figura 4.2:** Modelo de dominio de la aplicación

## 4.2 Elicitación de requisitos

Como comentábamos anteriormente, la elicitación de requisitos se realizará mediante métodos que permitan la comunicación directa con los *stakeholders*, como pueden ser las entrevistas o las reuniones. Disponer de esta comunicación debe considerarse como una gran ventaja, puesto que nos va a permitir tener un *feedback* constante de los *stakeholders*, haciendo que nuestro producto este enfocado correctamente desde el principio y que todos los requisitos que definamos tengan la confirmación de éstos, no dejando nada a la imaginación.

Los principales *stakeholders* de este proyecto son:

- **Administradores de la empresa:** En términos de SCRUM son los *product owners*, por lo que son los que nos deben dar un mayor *feedback*, indicando en cada reunión si los

requisitos obtenidos del resto de *stakeholders* tienen más o menos prioridad, y serán los que indiquen los requisitos principales que quieren que tenga la aplicación.

- **Empleados de la tienda:** Serán los encargados de proporcionarnos los requisitos para la administración de la aplicación, puesto que ellos serán, junto a los administradores de la empresa, los únicos que podrán acceder a este apartado para gestionar los principales aspectos de la web.

- **Clientes de la tienda:** Son los futuros usuarios finales de la aplicación. Éstos nos pueden ayudar a encontrar nuevos requisitos que faciliten las compras en la tienda online o el uso de la aplicación web.

## 4.3 Definición de requisitos

Como hemos remarcado anteriormente, la especificación de requisitos se realizará en lenguaje natural, y no en lenguaje formal o semiformal, ya que de esta manera no se requiere de un entrenamiento especial para entenderlos, facilitando así la comunicación en las reuniones con los *stakeholders*, evitando malentendidos y permitiéndonos llegar a un consenso de cuáles son los requisitos más importantes y cuales los menos importantes.

Para ello, marcaremos una serie de pautas que deben de cumplir todos los requisitos que se definan para que, de esta manera, sea más fácil su lectura para todos:

- En cada una de las definiciones de los requisitos solo se incluirá únicamente la información de dicho requisito.

- Cada requisito tendrá un identificador único, que dependerá de la parte de la aplicación a la que se refiera.

- Se indicará si los requisitos son funcionales [17] o no funcionales [18].

- Se indicará un número dependiendo de la prioridad de los requisitos y el orden en el que aparecerán en el *backlog* (siendo los números más pequeños los más prioritarios).

### 4.3.1 Gestión de usuarios

| Id | Descripción | Tipo | Prioridad |
|---|---|---|---|
| GU01 | El usuario podrá crear una cuenta en la web mediante un formulario de registro. Se le enviará un correo para verificar el correo electrónico introducido. | Funcional | 10 |
| GU02 | El usuario podrá iniciar sesión en la cuenta una vez creada a través de un formulario de inicio de sesión. | Funcional | 20 |
| GU03 | El usuario contará con una pantalla donde se mostrarán los datos de su cuenta. | Funcional | 160 |
| GU04 | El usuario podrá añadir, modificar o eliminar direcciones de envío desde su perfil de usuario. | Funcional | 170 |
| GU05 | El usuario podrá cambiar su contraseña desde el perfil rellenando un formulario con su contraseña actual y su nueva contraseña. | Funcional | 250 |
| GU06 | El usuario podrá recuperar su contraseña si no se acuerda siempre y cuando su cuenta de correo haya sido confirmada. Para ello se le mandará un correo al usuario con un formulario de cambio de contraseña. | Funcional | 260 |
| GU07 | El sistema tendrá que comprobar que la persona que entra al cambio de contraseña posee un token válido y no es una falsificación. | No Funcional | 270 |

**Tabla 4.1:** Requisitos de gestión de usuarios

### 4.3.2 Administración

| Id | Descripción | Tipo | Prioridad |
|---|---|---|---|
| AD01 | Los administradores podrán modificar los artículos, eliminarlos o incluso añadir nuevos desde las funciones de administración. | Funcional | 120 |
| AD02 | Los administradores podrán añadir nuevos catálogos o eliminarlos desde las funciones de administración. | Funcional | 130 |
| AD03 | Los administradores podrán seleccionar cuáles de los artículos serán destacados. Los artículos seleccionados aparecerán en la página principal. | Funcional | 140 |
| AD04 | Los administradores podrán consultar los pedidos desde las funciones de administración. Además, podrán cancelarlos, aportando un motivo y realizando una devolución, o marcarlos como enviados. | Funcional | 230 |

**Tabla 4.2:** Requisitos de administración

### 4.3.3 Compra de artículos

| Id | Descripción | Tipo | Prioridad |
|---|---|---|---|
| CA01 | Los usuarios podrán acceder a las fichas de los artículos y, siempre que estén registrados, añadir el artículo a su carrito de la compra, al cual podrán acceder desde la barra superior. | Funcional | 70 |
| CA02 | Los usuarios podrán eliminar y modificar la cantidad de los artículos que se encuentran en su carrito de la compra. | Funcional | 90 |
| CA03 | Los usuarios podrán buscar por referencia o por descripción los artículos que deseen gracias a un buscador en la barra superior. Este buscador proporcionará *feedback* al momento de escribir mediante un desplegable que mostrará algunos artículos que coincidan con la búsqueda. | Funcional | 100 |
| CA04 | La web contendrá todos los artículos que los usuarios podrían encontrar en la tienda física. | No Funcional | 180 |
| CA05 | Los usuarios podrán filtrar los artículos por grupos de artículos en las páginas de las subfamilias. | Funcional | 200 |
| CA06 | El usuario podrá comprar los artículos que hay en su carrito de la compra. Si el pago se realiza de forma correcta se generará un pedido y se notificará por correo al usuario. | Funcional | 220 |
| CA07 | El usuario podrá revisar los pedidos que ha realizado y su estado en su perfil. | Funcional | 240 |

**Tabla 4.3:** Requisitos de compra de artículos

### 4.3.4 Diseño de la web

| Id | Descripción | Tipo | Prioridad |
|---|---|---|---|
| DW01 | La web tendrá una barra superior desde la que los usuarios podrán navegar al resto de páginas y un *footer* donde podrán encontrar información legal y algunos atajos a funciones de la web. | Funcional | 0 |
| DW02 | La web tendrá una página principal donde el usuario entrará en primera instancia. En esta se mostrarán imágenes e información de la empresa. | Funcional | 30 |
| DW03 | La web tendrá una página de productos donde podremos ver las principales familias de productos. | Funcional | 40 |
| DW04 | Los productos estarán organizados en familias de productos que a su vez contendrán subfamilias. Cada una de éstas tendrá una interfaz accesible donde se pueden ver las subfamilias o artículos que engloban. | Funcional | 50 |
| DW05 | Los artículos dispondrán de una ficha donde se muestra una imagen, la información de dicho artículo y el precio. | Funcional | 60 |
| DW06 | La página web tendrá una página donde se mostrarán los principales contactos de la empresa | Funcional | 80 |
| DW07 | Los administradores podrán acceder a una página que mostrará todas las funciones de administración (Administración de artículos, administración de catálogos, administración de pedidos y administración de destacados). | Funcional | 110 |

| Id | Descripción | Tipo | Prioridad |
|---|---|---|---|
| DW08 | La página dispondrá de un menú lateral donde se nos permitirá navegar entre las familias de productos y mostrará el último catálogo en orden. | Funcional | 150 |
| DW09 | La web contará con un sistema de envío de correos para las confirmaciones de los correos o los avisos sobre pedidos entre otras cosas | No Funcional | 190 |
| DW10 | La web poseerá un sistema de recomendaciones basado en los artículos que visita cada usuario, y mostrará dichas recomendaciones en la página principal si el usuario tiene la sesión iniciada. | Funcional | 210 |

**Tabla 4.4:** Requisitos de diseño de la web

# CAPÍTULO 5

# Diseño de la solución

## 5.1 Controlador de versiones: Git

Es innegable la presencia de esta herramienta en cualquiera de los proyectos de hoy en día, sea un proyecto colectivo o individual. Git nace como herramienta de gestión de versiones en 2007 y desde entonces se ha ido incorporando en el desarrollo de software hasta el punto de ser imprescindible. Git nos permite crear repositorios donde depositar nuestro código, que irá modificándose con el tiempo. Para hacer más fácil el trabajo en equipo incluye: *branching*, para ramificar las versiones del código y poder trabajar varias personas en el mismo proyecto, *merging*, para unir ramas distintas en una sola, etc.

En nuestro proyecto, al estar trabajando una sola persona, estas funciones no son muy útiles, pero nos permite tener guardado el código en un repositorio, hacer una rama por cada UT [19] del *sprint* para tener una mayor organización del trabajo, unir estas ramas en una rama principal conforme vayamos programando y arreglando los bugs que surjan con las UT o tener copias de seguridad de *sprints* o versiones anteriores alojadas en ramas, entre otras cosas.

Para nuestro trabajo se ha decidido utilizar un servidor central alojado en GitHub. Esto se debe a que ya estamos familiarizados con este servicio y ofrece algunas herramientas interesantes como GitHub Desktop (ver figura 5.1), que permite realizar todas las funciones de Git de forma sencilla mediante una interfaz de usuario, ver las modificaciones del código en tiempo real y gestionar las ramas de forma simple.

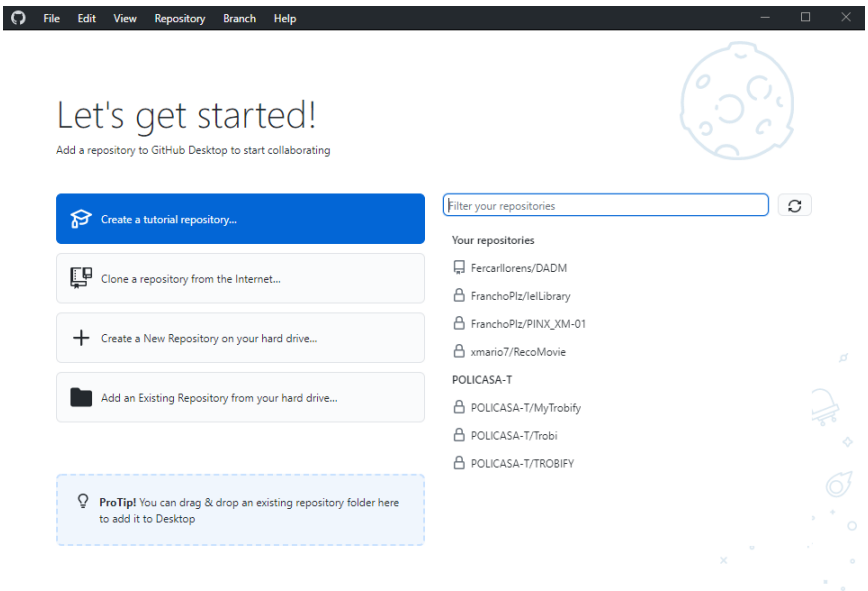


**Figura 5.1:** GitHub Desktop

## 5.2 Arquitectura: Modelo-Vista-Controlador

Nuestro proyecto va a utilizar una arquitectura llamada MVC [20] (Modelo-Vista-Controlador) bastante común en ASP.NET [21] (ver figura 5.2). Consiste en crear la estructura de datos por medio de los modelos, que serán las clases de C# que formarán las tablas de la base de datos *code-first* [22]. Las vistas, creadas con Blazor CSHTML, siguen una estructura de HTML a la que se le puede añadir código en C#, se le proporciona estilo con el CSS y se le dota de animaciones y algunos detalles con JavaScript. Por último, será el controlador (programado en C# también) el que dotará a las vistas de la lógica de la aplicación, utilizando los modelos también. Todo esto funcionando en conjunto permite tener una aplicación web con una funcionalidad completa.

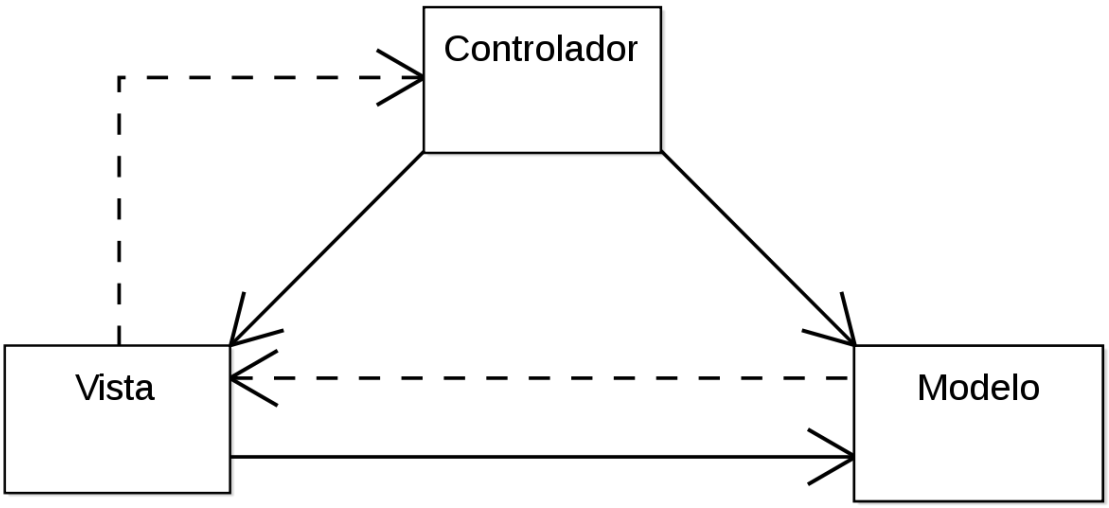


**Figura 5.2:** Arquitectura MVC

El proyecto se conectará a la base de datos mediante el DAL (*Data Acces Layer*), que como su nombre indica es una interfaz que implementa las definiciones de las funciones genéricas para actuar con la base de datos. Para dotar a esta interfaz de la lógica, crearemos una clase "EntityFrameworkDAL" que implemente las principales funciones de las operaciones con la base de datos. Por último, implementaremos mediante un patrón fachada (ver apéndice A) una clase ServicioFachada que nos proveerá de una instancia de conexión a la base de datos cuando la requiramos y que contendrá todas las operaciones específicas que queremos implementar para gestionar las tablas de la base de datos (Añadir, Eliminar, Modificar, …).

## 5.3 Estructura del código

El código de nuestro proyecto se estructurará todo en un mismo proyecto de Visual Studio que contendrá seis carpetas:

- **Models**: Carpeta dedicada a guardar los modelos de los objetos que manejaremos en la base de datos.

- **Views:** Vistas de las páginas en las que se dividirá nuestra web. Comporta el *frontend* del programa.

- **Controllers:** Es el *backend*, la lógica de la aplicación. Cada grupo de vistas tiene asociado un controlador.

- **Model-ML:** En esta carpeta encontramos el modelo de aprendizaje automático que utilizaremos para las recomendaciones de productos.

- **Service:** Carpeta que incluye la conexión a la base de datos y el patrón fachada con las operaciones CRUD [23] de las tablas.

- **Sources:** Incluye las imágenes y catálogos que se utilizarán en la web.

## 5.4 Base de datos

En cuanto a la base de datos, se decidió realizar una base de datos relacional, ya que esta nos permitiría hacer una correcta organización de artículos, imágenes y catálogos y se adaptaría a la perfección al dominio. Además, al utilizar ASP.NET MVC era recomendable utilizar una base de datos *code-first*, debido a que este *framework* nos proporciona los métodos para migrar estos modelos automáticamente y mantener un control de versiones de migraciones.

A la hora de seleccionar la tecnología, por estar más familiarizados con ella y porque era una para las que nuestro host ofrecía soporte, decidimos utilizar MSSQL, la tecnología de bases de datos relacionales basada en SQL de Microsoft.

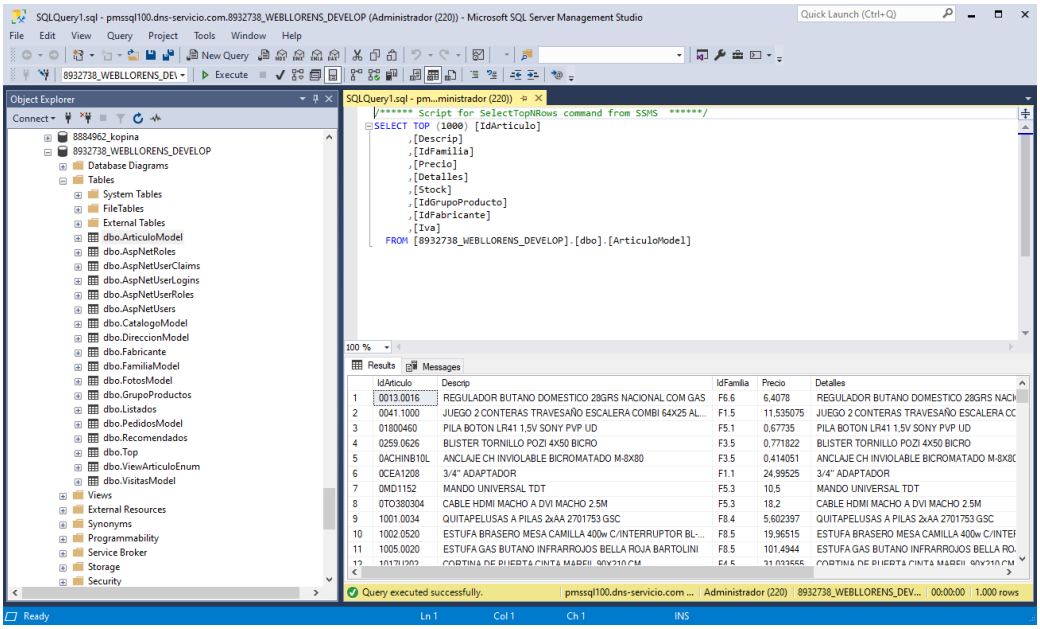


**Figura 5.3:** Microsoft SQL Server Management Studio

Para gestionar propiedades o realizar consultas que no dependan del código del proyecto decidimos utilizar Microsoft SQL Server Management Studio, la aplicación de escritorio para diseñar, migrar datos o hacer consultas (entre otras cosas) que ofrece Microsoft (ver figura 5.3).

## 5.5 Entorno de desarrollo: Visual Studio

Al estar utilizando ASP.NET, framework basado en C#, lo más lógico es programar el proyecto en el entorno propio de Microsft Visual Studio (en nuestro caso Visual Studio 2019).

Visual Studio nos permite la creación de gran variedad de aplicaciones, como pueden ser aplicaciones de consola, aplicaciones de Windows Forms o WPF (basadas en formularios), aplicaciones basadas en .NET [24] (como ASP.NET, ML.NET, ASP.NET Core, …) todas ellas en sus variantes de Visual Basic o C#.

El entorno nos facilita la conexión con un servidor de Git (como puede ser GitHub o Azure), lo que nos facilita la importación y conservación de nuestro proyecto, además del control de versiones propio de Git. En el caso de Visual Studio podemos prescindir de utilizar comandos básicos de Git en consola o GitHub desktop, ya que tiene un apartado propio para realizar estas funciones.

También son realmente sencillas las conexiones a bases de datos, puesto que desde el explorador de servidores se nos permite conectarnos a los servidores que queramos e incluso diseñar y gestionar las bases de datos que contengan sin necesidad de un programa como Management Studio.

Para la importación de librerías encontramos el gestor de paquetes NuGet, que nos permite encontrar e instalar los paquetes y librerías que necesitemos con gran facilidad, actualizar los ya incluidos en nuestro proyecto e incluso eliminar estos si ya no los vamos a utilizar.

## 5.6 Lenguaje del *backend*: C#

Para las partes del *backend* [25] de nuestro proyecto utilizaremos C#, el lenguaje de programación de Microsoft. Este es un lenguaje multiparadigma, pero principalmente lo usaremos como un lenguaje orientado a objetos debido al enfoque MVC que utilizaremos en la arquitectura.

C# es un lenguaje estandarizado de Microsoft como antes comentábamos que cuenta con una sintaxis que deriva de los leguajes C y C++ y que utiliza el modelo de objetos perteneciente a la plataforma .NET de Microsoft. Este lenguaje nace como una evolución de sus antecesores, C y C++, y su nombre es una metáfora que indica dicha mejoría. Al igual que C++ utilizaba una metáfora para indicar su mejora respecto a C (utilizando el símbolo de incremento '++'), C# utiliza una en términos musicales, tomando el símbolo '#' como un sostenido (que indica que es un semitono mayor). Por tanto, desde el principio se nos está planteando que el lenguaje deriva de C y C++, como comentábamos, pero con mejoras notables.

En el apartado de modelos utilizaremos C# para crear clases objeto con sus parámetros, que serán los encargados de manejar los datos en la aplicación, mientras que en los controladores y la conexión a la base de datos crearemos clases con funciones, que utilizarán dichos modelos como parte del código y formarán la lógica de la aplicación, es decir, el cómo actuará nuestra aplicación web en determinadas situaciones.

Por último, también utilizaremos C# para la creación de un motor de inteligencia artificial mediante la librería de ML.NET.

## 5.7 Lenguajes del *frontend*

### 5.7.1 Blazor – CSHTML

Antes de comenzar a explicar qué es Blazor, primero comenzaremos explicando qué es HTML.

HTML o *HyperText Markup Language* es, como su propio nombre indica, un lenguaje de etiquetado de texto para la elaboración de páginas web. Es el estándar que se ha impuesto y el que hoy en día utilizan todos los navegadores. HTML nos permite definir el contenido de nuestras páginas web y estructurarlo de manera sencilla y consistente, permitiendo un fácil entendimiento por parte de los humanos y la máquina.

Blazor nace con ASP.NET MVC 3 con la finalidad de crear vistas dinámicas que puedan usar código para gestionar la lógica. Utiliza CSHTML, un formato de HTML que permite el uso de código de C# (de ahí el nombre, C Sharp HTML) para gestionar la lógica de la vista desde el propio HTML. Para

incluir código de C# se marcarán las partes en las que vaya a aparecer dicho código con un carácter '@'. Gracias a esto podemos utilizar modelos en las propias vistas o incluso automatizar partes del HTML de una forma muchísimo más sencilla, permitiéndonos crear páginas web dinámicas con gran facilidad.

### 5.7.2 CSS

CSS (siglas de *Cascading Style Sheets*; en español Hoja de estilo en cascada) es un lenguaje de diseño gráfico para crear la presentación de documentos escritos en HTML, o XML entre otros. Junto con HTML y JavaScript, es uno de los principales lenguajes que podemos encontrar en cualquier página web. Es un lenguaje de marcado, y está diseñado para diferenciar entre el contenido del documento y la forma en que este se presentará. Una de sus características es que las referencias a elementos más concretas sustituirán a las más generales y las que aparecen más abajo en el fichero a las que aparecen más arriba.

En nuestro proyecto crearemos un CSS principal que afectara a todas las partes de la web para los elementos que se deben visualizar igual en cualquier momento, y un CSS por cada apartado de la web que presente elementos que necesitan una presentación en concreto solo en esa página.

Por último, cabe destacar que utilizaremos comprobaciones según el ancho de la pantalla para poder identificar cuándo se está ejecutando la web en un ordenador, una tablet o un móvil. De esta manera podremos organizar las presentaciones de los elementos según el dispositivo en el cual se ejecuten, creando vistas adecuadas para cada uno de estos dispositivos.

### 5.7.3 JavaScript

JavaScript es un lenguaje de programación orientado a objetos y débilmente tipado. Por lo general se implementa en el lado del cliente para crear páginas web dinámicas. JavaScript se basa en la sintaxis de C principalmente, aunque tiene algunas referencias de Java. Java y JavaScript sin embargo son para propósitos totalmente distintos y su nombre compartido se debe meramente al ámbito comercial debido a la compra de Netscape Navigator por Sun Microsystems (creador de Java). Éste es el único lenguaje de programación que entienden los navegadores de forma nativa.

La función de JavaScript en nuestra página web será para dotar de dinamismo a las páginas que componen nuestra web, utilizar alguna librería como jQuery o utilizar librerías que se basen en JavaScript como por ejemplo Bootstrap.

jQuery es una biblioteca de JavaScript creada inicialmente para simplificar la manera de interactuar con los elementos de HTML y desarrollar animaciones. Es interesante el uso de esta librería en nuestro caso para poder interactuar con el código de la parte del servidor desde la parte cliente en tiempo real mediante la técnica AJAX [26]. De esta manera, por ejemplo, podemos hacer en tiempo real que se añadan productos al carrito de la compra y se inserten a la base de datos en tiempo real o realizar búsquedas dinámicas que nos muestren resultados de la base de datos conforme vamos escribiendo.

Bootstrap es una biblioteca multiplataforma basada en HTML, CSS y JavaScript que, a diferencia de otros *frameworks*, solo se ocupa del *frontend* [27] de la aplicación. En esta librería podemos encontrar

plantillas de diseño, formularios, botones y otras muchas cosas que facilitan la implementación de elementos que, de otra forma, serían complejos y largos de programar.

## 5.8 Librerías utilizadas

En la parte del *backend* hemos utilizado algunas librerías instalándolas como paquetes NuGet desde el propio *marketplace* de Visual Studio o incluyendo el archivo ".dll" dentro de nuestro proyecto.

### 5.8.1 Entity Framework

Entity Framework es una librería que ofrece Microsoft para todas las aplicaciones de .NET. Ésta nos permite la conexión de nuestro código con una base de datos y el uso de las tablas de ésta como modelos de nuestro propio código. En nuestro proyecto utilizamos la versión de Entity Framework destinada a *code-first*, es decir, primero programamos los modelos y a partir de estos generaremos las tablas de la base de datos mediante migraciones.

### 5.8.2 NewtonSoft

NewtonSoft es una librería para C# dedicada exclusivamente a el manejo de datos JSON [28]. Para ello nos permite serializar objetos de C# como un *string* con el formato típico del JSON o deserializar *strings* con formato JSON y convertirlos en objetos de C#.

Esta librería resulta especialmente útil para introducir datos en formato JSON en la base de datos y ocupar menos espacio en algunos campos o para tratar datos o pasar datos a funciones de AJAX que se encuentren en la parte del cliente.

### 5.8.3 Redsys API

API desarrollado por la empresa Redsys para la implementación de una pasarela de pago en un comercio online. Redsys funciona independientemente del banco con el que trabaje la empresa, así que la forma de conseguir este API [29] es pidiéndolo en la oficina del banco. Una vez dado de alta en la plataforma requieren unos requisitos que debe seguir la página web. Una vez cumplidos los puntos propuestos te permiten descargar la API en formato ".dll" y acceder a la plataforma de gestión de la pasarela. Una vez implementado siguiendo el manual que podemos encontrar en su web deberemos realizar un par de pruebas y ya estaremos listos para la venta de productos online.

Además del pago con tarjeta nos permite implementar pagos por Bizum, que en nuestro caso ha sido descartado por el momento debido a las comisiones que tienen este tipo de ventas.

# CAPÍTULO 6

# Desarrollo de la solución

Durante este capítulo comentaremos cómo se programó nuestra aplicación web desde el punto de vista de cada una de las partes de la arquitectura propuesta y cómo se implementaron algunas de las funciones más importantes.

## 6.1 Capa modelo

Como hemos comentado en el capítulo anterior, en esta parte de nuestro código guardaremos los modelos con los que trabajaremos con la base de datos. Estos modelos son básicamente clases objeto de C#, con sus atributos, sus métodos y sus constructores.

A pesar de estar utilizando un modelo de base de datos *code-first*, que generara las tablas de nuestra base de datos y sus relaciones a partir de los modelos que creemos en esta capa, podemos indicar las propiedades necesarias de las tablas en el propio código utilizando "System.ComponentModel.DataAnnotations". Esta librería nos permite añadir anotaciones sobre nuestro modelo entre corchetes relacionadas con las propiedades que queramos que tengan los atributos. Esto hará que se cumplan estas propiedades en la base de datos cuando se realicen las migraciones a la base de datos (ver fragmento de código 6.1).

```
public class ArticuloModel
{
        [Key]
        public string IdArticulo {get; set;}
        [Required]
        public string Descrip {get; set;}
        //Resto del modelo…
}
```

**Fragmento de código 6.1:** Ejemplo "DataAnnotations"

Las relaciones de las tablas en la base de datos también se crearán de forma automática teniendo en cuenta los atributos de los objetos, sabiendo de qué tipo de relación se trata en cada momento por las multiplicidades de los atributos que se utilizan.

Para realizar las migraciones de las que hablábamos, al crear el proyecto se crea una carpeta "Migrations" que tiene esta finalidad. Dentro de esta podemos encontrar la clase "Configuration", donde podremos indicar que se activen las migraciones automáticas, de forma que detecten automáticamente los cambios que queremos realizar en la base de datos sin la necesidad de que hagamos nada. Dentro de esta clase también podemos introducir código dentro del método "Seed" que se ejecutará al hacer las migraciones y que funciona para poblar la base de datos en una primera instancia entre otras cosas. Para ejecutar las migraciones escribiremos en el terminal "add-migration".

Gracias a este método podemos crear una base de datos incremental que se irá ampliando a medida que vayamos añadiendo nuevos modelos a nuestro proyecto, lo que nos viene perfecto para aplicar una estrategia de MVP en nuestro producto, ya que no tenemos la necesidad de crear toda la base de datos final desde el principio, sino que podemos ir añadiendo tablas poco a poco conforme las vayamos necesitando (ver figura 6.1).

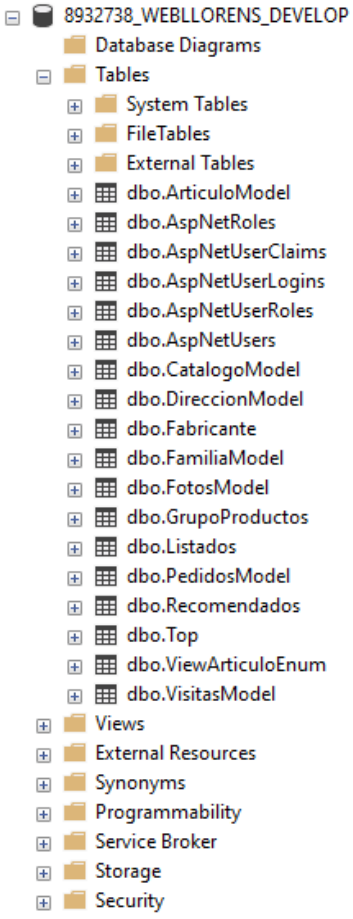


**Figura 6.1:** Base de datos al final del *sprint* 3

## 6.2 Capa vista

En esta capa encontraremos todas las vistas que formarán la interfaz de usuario de nuestra página web, a través de la cual los usuarios podrán interactuar con el sistema para realizar sus compras, consultar los catálogos o artículos o buscar nuestros principales contactos entre otras cosas.

La capa de vistas se divide en diversas carpetas según a que parte del sistema pertenecen. Las principales divisiones que hemos creado en esta capa son las siguientes:

- Home
- Productos
- Administración
- Account
- Shared

Vamos a comentar brevemente qué podemos encontrar dentro de cada una de estas divisiones y a mostrar algunos *mockups* básicos que se utilizaron para la implementación de estas vistas.

### 6.2.1 Carpeta Home

En esta carpeta podremos encontrar las vistas pertenecientes a la página de inicio (ver figura 6.2), la página para descargar los PDF's de los catálogos, la página de contactos, el carrito de la compra (ver figura 6.3), el perfil de usuario y los términos y condiciones de uso de la aplicación y toda su información legal.

Podemos encontrar en este apartado prácticamente todas las vistas (menos productos) a las que podemos acceder directamente desde el menú superior o el *footer*. La mayoría de estas vistas tienen poca carga en la parte lógica y se pueden crear prácticamente con los lenguajes del *frontend* (a excepción del carrito de la compra o el perfil de usuario que llamará a algunas funciones de Account).


**Figura 6.2:** *Mockup* página principal

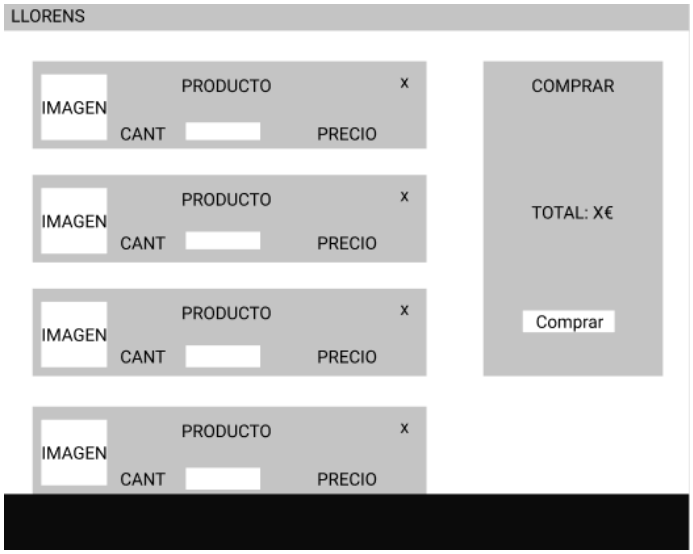


**Figura 6.3:** *Mockup* carrito de la compra

### 6.2.2 Carpeta Productos

En esta carpeta se incluyen todas las vistas relacionadas con las familias y subfamilias de los productos, además, por supuesto, de los productos. Esta carpeta incluye también las búsquedas. Esta subdivisión se ha realizado para separar posteriormente toda la lógica de los productos en el mismo controlador y para que los usuarios puedan encontrar todo lo relacionado con productos bajo la url "elecfer.com/Productos".

Estas vistas permitirán también al usuario poder añadir los productos a su carrito de la compra si tienen un usuario registrado y han iniciado sesión. Esto se realizará desde la vista de las fichas de productos (ver figura 6.4).


**Figura 6.4:** *Mockup* ficha artículo

### 6.2.3 Carpeta Administración

Aquí encontraremos reunidas todas las vistas accesibles solo por los usuarios con rol de administración. Entre estas podremos encontrar los apartados de administración de artículos (ver figura 6.5), de catálogos, de artículos destacados (ver figura 6.6) y de pedidos.

Esta carpeta tiene como finalidad separar la parte de administración del resto de la web, creando el subsistema de administración de la aplicación solo accesible para los usuarios con permisos (los empleados de la tienda). De esta manera no hace falta ningún conocimiento de informática para hacer cambios en los productos o catálogos de la web o para consultar los pedidos que se han realizado.


**Figura 6.5:** Administración de artículos

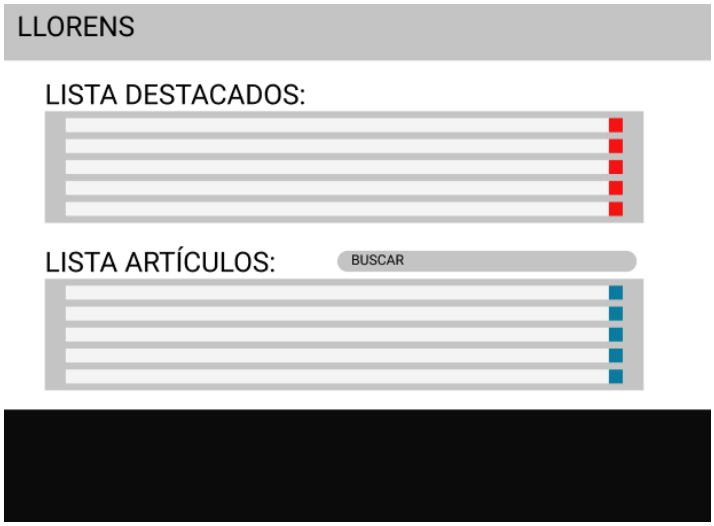


**Figura 6.6:** Administración de artículos destacados

### 6.2.4 Carpeta Account

Esta carpeta tiene contenidas las vistas que se utilizan para la gestión de las cuentas como el *log in* (ver figura 6.7), el registro (ver figura 6.8) o el cambio de contraseña.

El objetivo de esta carpeta es que todas las vistas relacionadas con cualquier gestión de la cuenta de usuario se encuentren bajo la url "elecfer.com/Account", separando en la parte del controlador su lógica, como veremos más adelante.

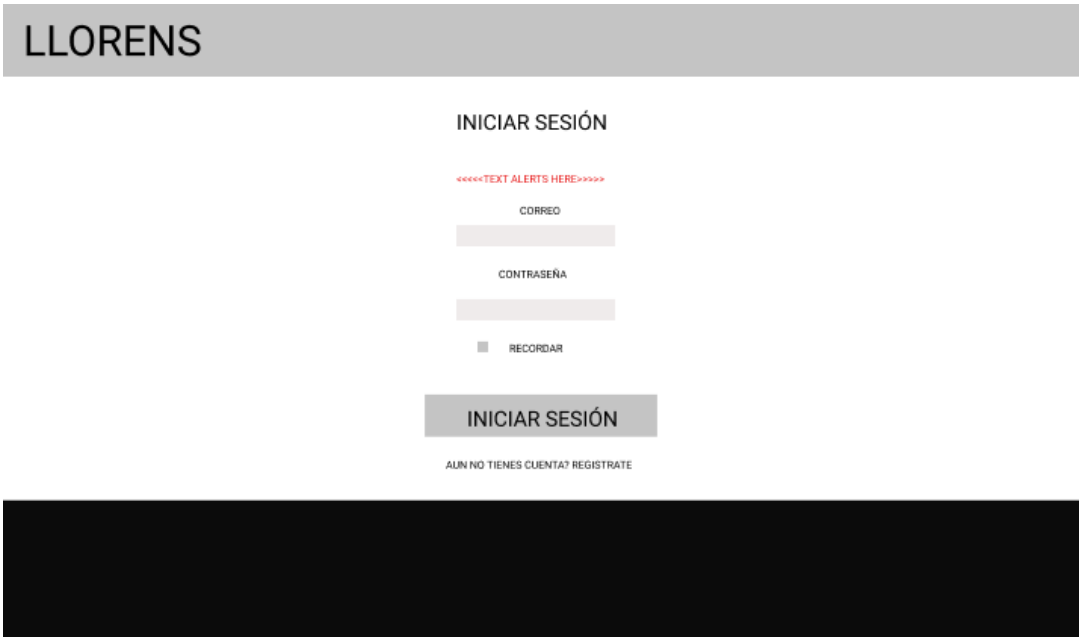


**Figura 6.7:** *Mockup* iniciar sesión

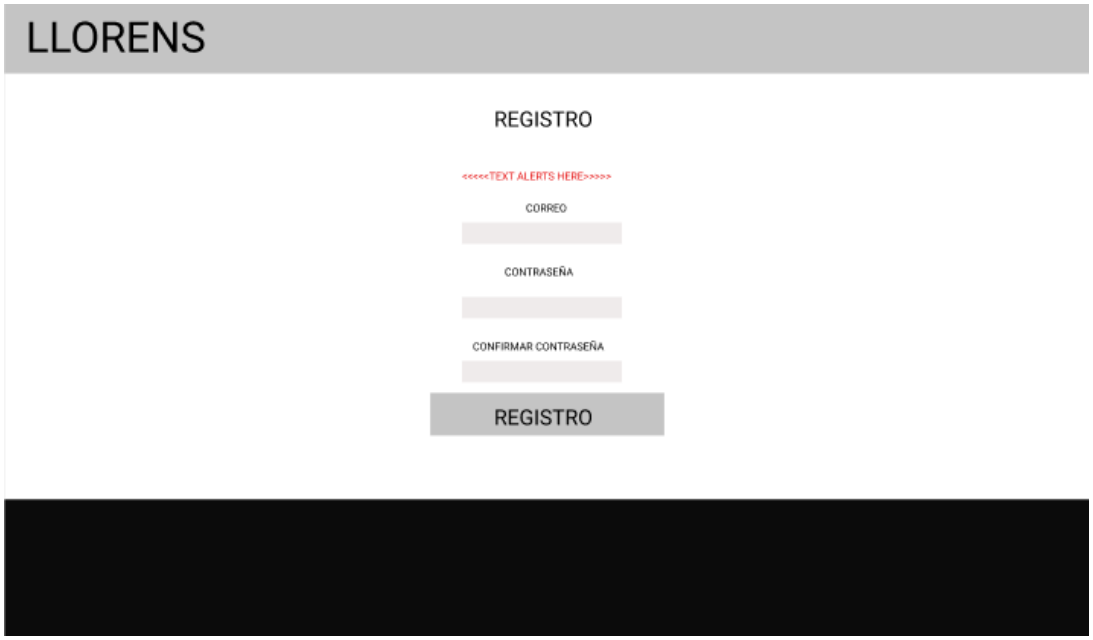


**Figura 6.8:** *Mockup* registro

### 6.2.5 Carpeta Shared

La carpeta "Shared" como su propio nombre indica incluye vistas que se comparten en el resto de apartados de la web. Es la única división que no tiene un controlador asociado, pues no lo necesita. Las vistas que podemos encontrar en esta carpeta son páginas de error que se pueden utilizar en cualquier momento llamadas por cualquier otro controlador o vistas parciales que se superponen sobre otras vistas como la barra de búsqueda, el menú superior, el menú lateral o el *footer*.

## 6.3 Capa controlador

Esta capa será la que contendrá toda lo lógica de la aplicación que se ejecutará en la parte del servidor y que conectará la interfaz de usuario de la aplicación con los modelos y, por ende, la base de datos.

Para ello la lógica se divide en tantos controladores como carpetas encontramos en la capa de vistas (excepto la carpeta "Shared" que no tiene controlador), puesto que cada controlador dotará de lógica a las vistas que contengan esa carpeta.

A continuación, describiremos algo más en detalle la lógica que contiene cada uno de los controladores.

### 6.3.1 HomeController

En este controlador podemos encontrar varios tipos de funciones que se aplicarán en el carrito de la compra y las páginas principales accesibles por el menú.

El primer tipo de funciones que podremos encontrar son funciones de navegación, cuya única finalidad es cambiar la página que se muestra en el navegador a otra (ver fragmento de código 6.2). En estas funciones se le puede pasar como parámetro a la vista una clase que denominaremos modelo y que podremos utilizar en la parte de la vista, obteniendo estos datos de forma mucho más fácil.

```
public ActionResult Catalogos()
{

   TraspasoCatalogos model = new TraspasoCatalogos();
   return View(model);

}
```

**Fragmento de código 6.2:** Función de navegación a catálogos

Otro tipo de funciones que podemos encontrar son operaciones CRUD de los modelos que se ven envueltos en este apartado, como pueden ser el carrito de la compra. Estas funciones nos permiten, haciendo uso de los modelos, realizar una modificación en la base de datos de estos datos desde la interfaz del usuario. Por ejemplo, la función “EditarLista” que nos permite cambiar la cantidad de artículos de un tipo que queremos incluir en el carrito de la compra. Para ello se elimina el objeto guardándolo previamente y se crea uno nuevo con la nueva cantidad para añadirlo en la base de datos (ver fragmento de código 6.3).

```
public ActionResult EditarLista(string ArticuloId, string cantidad)
{

   string UserId = User.Identity.GetUserId();

   if (ArticuloId == ""|| UserId == ""||cantidad == "")
   {
     return View("/Views/Shared/Error.cshtml");
   }

   int cant = Int32.Parse(cantidad);
   Listados articuloLista = servicio.GetListado(UserId, ArticuloId);
   servicio.Delete(articuloLista);
   servicio.Commit();

   articuloLista.cantidad = cant;
   servicio.AddListado(articuloLista);
   servicio.Commit();

   return RedirectToAction("ListaCompra", "Home");
}
```

**Fragmento de código 6.3:** Función EditarLista

También podemos encontrar funciones que se ejecutan a través de la llamada de otras funciones, ya sea en el propio controlador o con una función de AJAX desde el *frontend* con javascript. Por ejemplo,

"Sugerencias" que se llama desde el *frontend*, realiza una búsqueda de artículos en la base de datos, los transforma a un objeto adaptado para poder imprimirlo en la interfaz y se devuelve este objeto en formato JSON al *frontend* para que realice las funciones necesarias en la parte del cliente con el fin de que se muestren en pantalla (ver fragmento de código 6.4).

```
public JsonResult Sugerencias(string data) {

   List<ArticuloModel> lista = servicio.GetArticulosBusquedaPopUp(data).ToList();
   List<string> jsonLista = new List<string>();

   foreach (var a in lista)
   {
      TraspasoItemsBusqueda item = new TraspasoItemsBusqueda()
      {
         Id = a.IdArticulo,
         Nombre = a.Descrip,
         Foto = a.GrupoFotos.First().PathRelativo
      };

      jsonLista.Add(JsonConvert.SerializeObject(item));
   }

   var response = jsonLista;
   return Json(response);
}
```

**Fragmento de código 6.4:** Función sugerencias

### 6.3.2 ProductosController

En este controlador encontraremos funciones destinadas a proveer a el apartado de productos de su lógica, separándolo del resto de la aplicación.

En primer lugar, encontramos funciones de navegación como puede ser la función "ProductosFamilia" que recibe un parámetro con el identificador de la familia, comprueba que este parámetro no contenga una cadena vacía o que no exista una familia con ese id y devuelve la vista correspondiente pasándole un modelo (ver fragmento de código 6.5).

```
public ActionResult ProductosFamilia(string Id)
{

  string id;

  if (Id == null || Id == ""){ return View("/Views/Shared/Error.cshtml"); }
  else { id = Id; }

  FamiliaModel familiaCheck = servicio.GetFamiliaByID(id);

  if (familiaCheck == null) { return View("/Views/Shared/Error.cshtml"); }

  TraspasoSubfamlias model = new TraspasoSubfamlias(id);
  return View(model);

}
```

**Fragmento de código 6.5:** Función de navegación a subfamilias

También encontramos la función "AnadirLista" relacionada con las operaciones CRUD de los artículos del carrito de compra, siendo la función encargada de añadirlos en la base de datos. Para ello recibe como parámetros el id del artículo y la cantidad, crea un objeto listado y lo añade a la base de datos (ver fragmento de código 6.6).

```
public void AnadirLista(string IdArticulo, string cantidad)
{

   if (IdArticulo == "" || cantidad == "") { return ; }

   Listados articuloLista = new Listados()
   {
     articuloId = IdArticulo,
     cantidad = Int32.Parse(cantidad),
     userId = User.Identity.GetUserId(),
   };

   servicio.AddListado(articuloLista);
   servicio.Commit();


}
```

**Fragmento de código 6.6:** Función AnadirLista

Por último, encontramos funciones de navegación un tanto más complejas como puede ser "BuscaArticulo", que comprueba que la búsqueda y la página que se le pasa no sean cadenas vacías y crea el modelo que se enviará a la vista (ver fragmento de código 6.7). En el constructor del modelo que vamos a utilizar se hará una llamada a "GetArticulosBusqueda" de la parte del código que conecta con la base de datos. En esta función se convertirá la cadena introducida en mayúsculas con la función "ToUpper()" y con la cadena resultante se hará una búsqueda en la base de datos que devolverá una lista con todos los artículos

que contienen dicha cadena contenida en el nombre o la referencia del artículo (ver fragmento de código 6.8).

```
public ActionResult BuscaArticulo (string busqueda , string Pagina)
{

  if (busqueda == "" || busqueda == null  ) {
    return View("/Views/Shared/Error.cshtml");
  }

  int pagina;

  if (Pagina == "" || Pagina == null)
  {
    pagina = 1;
  }
  else {
    pagina = Int32.Parse(Pagina);
  }

  TraspasoArticulos model = new TraspasoArticulos(busqueda, pagina, true);

  ViewBag.texto = "Búsqueda";

  return View("/Views/Productos/Articulos.cshtml", model);
}
```

**Fragmento de código 6.7:** Función BuscaArticulo

```
public ICollection<ArticuloModel> GetArticulosBusqueda(string busqueda)
{

  string aux = busqueda.ToUpper();
  List<ArticuloModel> busquedaNombre = dal.GetAll<ArticuloModel>().Where(a=>
    a.IdFamilia.StartsWith("F") && (a.Descrip.Contains(aux)
    || a.IdArticulo.Contains(aux))).ToList();

  return busquedaNombre.OrderBy(a => a.Descrip).ToList();

}
```

**Fragmento de código 6.8:** Función GetArticulosBusqueda

### 6.3.3 AdministracionController

En este apartado de nuestro código encontramos la parte de la lógica que se refiere exclusivamente a la administración de la web. Estas funciones solo podrán ser ejecutadas por usuarios que tengan el rol de administrador.

Obviamente en este apartado podemos encontrar funciones de navegación como en los anteriores, pero las funciones más interesantes son las funciones destinadas a las operaciones CRUD de los objetos

más importantes de la web. Para dar un ejemplo de esto vamos a tomar el conjunto de funciones utilizadas para añadir un artículo.

Para empezar, tenemos “AnadirArticuloBase”, la función principal de esta operación. De primeras se comprobará si los parámetros necesarios son no nulos o vacíos (si alguno lo es devolverá la página de error). Tras esto se llaman a las funciones de “CrearArticulo” y “CrearFoto”, que no comentaremos debido a su simplicidad, pero que en resumidas cuentas comprueban algunos de los parámetros obtenidos para ver si son válidos y crean los objetos. Por último, se comprobará si en la base de datos ya existen el artículo o la foto con el id que queremos añadir, y según si existen o no se ejecutarán unas funciones otras (ver fragmento de código 6.9).

```
public ActionResult AnadirArticuloBase(HttpPostedFileBase file, string Id, string
Descrip, string Idfamilia, int Precio, int Stock, string Detalles, string Grupo,
string Fabricante)
{

  if (file == null || Id == "" || Descrip == "" || Idfamilia == "")
  {
    return View("/Views/Shared/Error.cshtml");

  }

   ArticuloModel articulo = CrearArticulo(Id, Descrip, Idfamilia, Grupo,
     Detalles,Precio, Stock);
   FotosModel foto = CrearFoto(file, Id);

   ArticuloModel testArt = servicio.GetArticuloById(articulo.IdArticulo);
   FotosModel testFoto = servicio.GetFotoById(foto.Id);

   if (testArt != null && testFoto == null)
   {
     return AnadirFotoAlArticulo(file, foto, articulo);
   }
   else if (testArt == null && testFoto != null)
   {
     return AnadirArticuloSinFoto(articulo);
   }
   else if (testArt != null && testFoto != null)
   {
     return RedirectToAction("AdministracionArticulos", "Administracion");
   }
   else
   {
     return AnadirArticuloYFoto(file, foto, articulo);
   }
}
```

**Fragmento de código 6.9:** Función AnadirArticuloBase

Si se da el caso de que se encuentra un artículo de mismo id pero no la foto se ejecutará la función "AnadirFotoAlArticulo" que básicamente subirá el archivo a la carpeta de imágenes y añadirá el objeto foto previamente creado al artículo y la base de datos (ver fragmento de código 6.10).

```
public ActionResult AnadirFotoAlArticulo(HttpPostedFileBase file, FotosModel
foto, ArticuloModel articulo)
{

   try
   {
      var contenido = new byte[file.ContentLength];

      foto.SubirArchivo(contenido);
      file.SaveAs(foto.PathCompleto);
   }
   catch (Exception e)
   {
      return View("/Views/Shared/Error.cshtml");
   }

   articulo.AddFoto(foto);
   servicio.AddFoto(foto);
   servicio.Commit();

   return RedirectToAction("AdministracionArticulos", "Administracion");
}
```

**Fragmento de código 6.10:** Función AnadirFotoAlArticulo

Por otra parte, si se encuentra una foto y no un artículo se ejecutará "AnadirArticuloSinFoto" que básicamente añadirá el artículo a la base de datos (ver fragmento de código 6.11).

```
public ActionResult AnadirArticuloSinFoto(ArticuloModel articulo)
{

  servicio.AddArticulo(articulo);
  servicio.Commit();

  return RedirectToAction("AdministracionArticulos", "Administracion");
}
```

**Fragmento de código 6.11:** Función AnadirArticuloSinFoto

En el caso en el que se encuentra tanto una foto como un artículo se devolverá al administrador a la página de administración de artículos pues el artículo y la foto que quiere añadir ya se encuentran en la base de datos y por tanto en la web.

Por último, si no se encuentran ninguna de las dos cosas se llamará a la función "AnadirArticuloYFoto" que realizará la funcionalidad de las dos funciones anteriores juntas (ver fragmento de código 6.12).

```
public ActionResult AnadirArticuloYFoto(HttpPostedFileBase file, FotosModel foto,
ArticuloModel articulo)
{

   try
   {
     var contenido = new byte[file.ContentLength];
     foto.SubirArchivo(contenido);
     file.SaveAs(foto.PathCompleto);
   }
   catch (Exception e)
   {
     return View("/Views/Shared/Error.cshtml");
   }


     servicio.AddArticulo(articulo);
     articulo.AddFoto(foto);
     servicio.AddFoto(foto);
     servicio.Commit();

     return RedirectToAction("AdministracionArticulos", "Administracion");
}
```

**Fragmento de código 6.12:** Función AnadirArticuloYFoto

### 6.3.4 AccountController

Este controlador se encargará de dotar de la parte de la lógica a las vistas relacionadas con la administración de la cuenta como el log in, el registro etc…

En éste podemos encontrar muchas funciones que vienen creadas por defecto al crear un proyecto nuevo de ASP.NET MVC como pueden ser el log in, el registro el cambio de contraseña… que están implementados por medio de llamadas a funciones del propio Microsoft que son más seguras que programarlo todo por nuestra cuenta.

También encontramos dos funciones que se llaman en el log in para entrenar el modelo de aprendizaje automático y para obtener recomendaciones usándolo, que se ejecutan cada cierto tiempo cuando unusuario inicia sesión (inicialmente cada mes).

La función "TrainModel", como su propio nombre indica, sirve para entrenar el modelo de recomendaciones, para ello esta función obtiene las visitas a productos del usuario, que es un JSON y las desserializa. Posteriormente, por cada entrada en visitas crea un objeto "ArticuloVisit" que es con el que trabaja nuestro modelo y llama a la función "Train" del modelo con estos datos (ver fragmento de código 6.13).

```
private void TrainModel(string email)
{

  Recommender recommender = new Recommender();
  VisitasModels visitasUsuario =
  servicio.GetVisita(servicio.GetUserByEmail(email).Id);
  List<Visita> visitas =
  JsonConvert.DeserializeObject<List<Visita>>(visitasUsuario.Visitas);
  List<ArticuloVisit> newData = new List<ArticuloVisit>();

  if (visitas.Count != 0) {

      foreach (var v in visitas)
      {
         ArticuloVisit aux = new ArticuloVisit()
         {
            userId = servicio.GetUserByEmail(email).Id,
            articuloId = v.articuloId,
            Label = (float) v.visitas
         };
         newData.Add(aux);
      }

      recommender.Train(newData);
  }
}
```

**Fragmento de código 6.13:** Función TrainModel

La función “GetRecomendaciones”, toma el usuario y su lista de visitas. A partir de ahí, se eliminan las recomendaciones anteriores y se toma de su lista de visitas los tres artículos con más visitas. A raíz de estos artículos, se toma una lista de artículos posiblemente recomendados (para cada uno de los artículos) por familia y grupo de producto y se transforman en una lista de “ArticuloVisit” y esto se pasa en la llamada a la función “recommend”. La lista que devuelve introducirá los artículos recomendados en una lista y calculará cuantos artículos faltan para llegar a 12. Por último, si al terminar las recomendaciones aún falta por completar huecos se añadirán algunos de los artículos destacados. Cabe destacar que, debido a la complejidad y el peso de la función, esta se ejecutará como un hilo en segundo plano, por lo que no obtendremos las recomendaciones nuevas nada más iniciar sesión, pero obtendremos un inicio de sesión rápido (ver fragmento de código 6.14).

```
private void GetRecomendaciones(Object email)
{
   string userId = servicio.GetUserByEmail(email.ToString()).Id;
   VisitasModels visitas = servicio.GetVisita(userId);
   List<Visita> visitaList =
      JsonConvert.DeserializeObject<List<Visita>>(visitas.Visitas);
   List<Visita> topVisitados = new List<Visita>();
   List<ArticuloModel> topArticulos = servicio.GetTop().ToList();
   List<ArticuloModel> recomendado = new List<ArticuloModel>();
   Recommender recomender = new Recommender();
   List<Recomendados> prevRecomendados = servicio.GetRecomendados(userId);
```

```
   foreach (var r in prevRecomendados) {
     servicio.Delete(r);
   }
   servicio.Commit();

   if (visitaList.Count != 0)
   {
     for (int i = 0; i < 3 && i < visitaList.Count ; i++) {
        int indexVisita = visitaList.IndexOf(visitaList.First(x =>
            visitaList.Max(v => v.visitas) == x.visitas));
        topVisitados.Add(visitaList.ElementAt(indexVisita));
        visitaList.Remove(visitaList.ElementAt(indexVisita));
     }

     foreach (var v in topVisitados)
     {
        if (recomendado.Count < 12) {
          ArticuloModel articulo = servicio.GetArticuloById(v.articuloId);
          List<ArticuloVisit> mayRecommend =
             servicio.GetAllArticulosByFamiliaID(articulo.IdFamilia).Where(a =>
             a.IdGrupoProducto != 0 && a.IdFamilia == articulo.IdFamilia &&
             a.GrupoFotos.Count != 0).Select(a => new ArticuloVisit()
             {
                articuloId  = a.IdArticulo,
                userId = userId
             }).Where(a => recomender.recomend(a)).ToList();

             if (recomendado.Count + mayRecommend.Count > 12) {
                int rest = 12 - recomendado.Count();
                mayRecommend = mayRecommend.Take(rest).ToList();
             }

             recomendado.AddRange(mayRecommend.Select(a =>
               servicio.GetArticuloById(a.articuloId)).Where(a =>
               !recomendado.Contains(a)));
         }
     }

  }
  else
  {
     recomendado.AddRange(topArticulos);
  }

  foreach (var r in recomendado) {
     Recomendados rec = new Recomendados()
     {
       articuloId = r.IdArticulo,
       userId = userId,
       fecha = DateTime.Now
     };
     servicio.AddRecomendado(rec);
  }
  servicio.Commit();
}
```

**Fragmento de código 6.14:** Función GetRecomendaciones

## 6.4 Operaciones sobre la base de datos

Como ya habíamos comentado anteriormente, la conexión con la base de datos se realiza mediante el uso de un patrón fachada (ver apéndice A). Todas las operaciones sobre las tablas de la base de datos se implementan en la clase "ServicioFachada". Podemos encontrar tres tipos de funciones: para obtener objetos (ver fragmento de código 6.15), para añadirlos (ver fragmento de código 6.16) y para borrarlos (ver fragmento de código 6.17).

```
public ICollection<FamiliaModel> GetAllFamilias() {
   return dal.GetAll<FamiliaModel>().ToList();
}
```

**Fragmento de código 6.15:** Método de obtención

```
public void Delete(ArticuloModel a) {
   dal.Delete<ArticuloModel>(a);
}
```

**Fragmento de código 6.16:** Método de borrado

```
public void AddArticulo(ArticuloModel articulo){
   dal.Insert<ArticuloModel>(articulo);
}
```

**Fragmento de código 6.17:** Método de añadido

## 6.5 Recomendaciones con aprendizaje automático

Para realizar el sistema de recomendaciones se decidió implementar un modelo de predicciones basado en aprendizaje automático, en concreto con el tipo de predicción por factorización matricial, que suele ser el más recomendado para las recomendaciones de productos debido a que tiene en cuenta las preferencias de los usuarios por algunos productos teniendo en cuenta los que ya ha comprado, permitiéndonos estimar de esta manera que, por ejemplo, si dos clientes han visitado el mismo producto, una pala, y uno de los dos también ha estado visitando otro producto, un rastrillo, el otro usuario que visito la pala podría estar interesado también en el rastrillo debido a que siguen un mismo perfil de intereses. Esto obviamente es un ejemplo muy simple. En la realidad este computo es mucho más complejo y las recomendaciones se calculan relacionando los gustos del usuario con el resto de usuarios, no con uno solo.

Para implementar este sistema de recomendaciones basado en aprendizaje automático teníamos varias opciones en la mesa, pero finalmente decidimos utilizar ML.NET [30], la librería de aprendizaje automático que ofrece Microsoft, ya que nos permitía una integración total en el proyecto, debido a que forma parte del ecosistema .NET de Microsoft (al igual que ASP.NET) y se programa en C#, evitando el cambio de lenguaje entre la parte de la lógica de la aplicación y el modelo de aprendizaje automático. De esta manera también evitamos tener que montar un programa a parte en una API o algo similar y tener que acceder a él por medio de una uri.

La implementación consta de dos clases objeto bastante sencillos y una clase que contiene todas las funciones para crear, entrenar y guardar el modelo, además de la función para recomendar los artículos.

Las clases objeto que hemos creado son "ArticuloVisit" y "ArticuloVisitPrediction". En el primero encontramos un objeto con los atributos que van a ser utilizados para relacionar los usuarios y artículos en la matriz de cómputo y un atributo *label* que es el que da el valor numérico de las visitas, que es lo que queremos predecir en los casos de las recomendaciones (ver fragmento de código 6.18). Por otra parte, en el segundo encontramos el objeto que nos devolverá la recomendación con la predicción de las visitas y una puntuación asociada según si la predicción es más o menos acertada (ver fragmento de código 6.19).

```
public class ArticuloVisit
{
        [LoadColumn(0)]
        public string userId;

        [LoadColumn(1)]
        public string articuloId;

        [LoadColumn(2)]
        public float Label;
}
```

**Fragmento de código 6.18:** Clase ArticuloVisit

```
class ArticuloVisitPrediction
{

        public float Label;
        public float Score;

}
```

**Fragmento de código 6.19:** Clase ArticuloVisitPrediction

Respecto a la parte de la lógica del modelo podemos encontrar dos funciones principales, la función de recomendación y la función de entrenamiento, las cuales contienen otro grupo de funciones internas.

La función de entrenamiento “Train” generará un nuevo “MLContext” y llamará por este orden a una función interna “LoadData” que cargará los datos que recibe por parámetros, una función interna “Train” que creará y entrenará el modelo y una función interna “SaveModel” que guardará el modelo en la carpeta del proyecto para utilizarlo en el futuro (ver fragmento de código 6.20).

```
public void Train(ICollection<ArticuloVisit> newData) {

   MLContext mlContext = new MLContext();

   IDataView training = LoadData(mlContext, newData);

   ITransformer newModel = Train(mlContext, training);


   SaveModel(mlContext, training.Schema, newModel);

   [Resto del código…]

}
```

**Fragmento de código 6.20:** Método Train

La función LoadData toma como parámetros el contexto y la lista de artículos que recibe el método principal “Train”. La función tomará estos datos y los introducirá en el documento csv de datos con el que entrenaremos en el modelo. Seguidamente toma todos los valores del documento y los transforma en una matriz para poder entrenar el modelo (ver fragmento de código 6.21).

```
IDataView LoadData(MLContext Context, ICollection<ArticuloVisit> list)
{

   using (StreamWriter writer =
      File.AppendText("C:/repos/WebLlorens/" +
      "Model_ML/Data/Training-data.csv")) {

      foreach (var l in list) {
         if (l.Label > 10)
         {
            writer.WriteLine(l.userId + ";" + l.articuloId + ";" + 10 + ";" +
              "nombreArticulo");
         }
         else
         {
            writer.WriteLine(l.userId + ";" + l.articuloId + ";" + l.Label + ";"+
              "nombreArticulo");
```

```
            }
        }
    }

    var trainingDataPath = "C:/repos/WebLlorens/" +
        "Model_ML/Data/Training-data.csv";
    IDataView trainingDataView =
        Context.Data.LoadFromTextFile<ArticuloVisit>(trainingDataPath, hasHeader:
        true, separatorChar: ';');


    return trainingDataView;

}
```

**Fragmento de código 6.21:** Función interna LoadData

Por su parte, en la función “Train” se mapean los datos devueltos por la función “LoadData” en una matriz de forma que estén preparados para el entrenamiento del modelo, tras esto se especifican las opciones del entrenamiento y acto seguido se crea y entrena el modelo (ver fragmento de código 6.22).

```
ITransformer Train(MLContext Context, IDataView trainingData) {

    IEstimator<ITransformer> estimator = Context.Transforms.Conversion
      .MapValueToKey(outputColumnName: "userIdEncoded", inputColumnName:
      "userId").Append(
      Context.Transforms.Conversion.MapValueToKey(outputColumnName:
      "articuloIdEncoded", inputColumnName: "articuloId"));

    var options = new MatrixFactorizationTrainer.Options
    {
        MatrixColumnIndexColumnName = "userIdEncoded",
        MatrixRowIndexColumnName = "articuloIdEncoded",
        LabelColumnName = "Label",
        NumberOfIterations = 100,
        ApproximationRank = 100
    };

    var trainerEstimator = estimator.Append(Context.Recommendation().Trainers
     .MatrixFactorization(options));

    ITransformer model = trainerEstimator.Fit(trainingData);

    return model;
}
```

**Fragmento de código 6.22:** Función interna Train

En último lugar, la función “SaveModel” toma la dirección de la carpeta en el proyecto y guarda el modelo ya entrenado para próximos usos (ver fragmento de código 6.23).

```
void SaveModel(MLContext Context, DataViewSchema trainingDataViewSchema,
ITransformer model)
{
   var modelPath = "C:/repos/WebLlorens/Model_ML/ArticuloRecommenderModel.zip";

   Context.Model.Save(model, trainingDataViewSchema, modelPath);
}
```

**Fragmento de código 6.23:** Función interna SaveModel

Para finalizar este apartado, La funcion “IsRecommend” recibe como parámetro un Artículo en el formato de la matriz usada para el entrenamiento, carga el modelo previamente guardado y llama a la funcion “UseModelForSinglePrediction”, que crea un motor de predicción a raíz del modelo entrenado y lo utiliza para calcular un “ArticuloVisitPrediction”, a través del cual, según el valor de su atributo score, se decide si se recomienda o no el artículo al usuario (ver fragmento de código 6.24).

```
Boolean IsRecommended(ArticuloVisit art) {

   MLContext mlContext = new MLContext();

   DataViewSchema modelSchema;


   ITransformer trainedModel =
      mlContext.Model.Load("C:/Users/ferna/source/repos/WebLlorens" +
      "/WebLlorens/Model_ML/ArticuloRecommenderModel.zip", out modelSchema);


   return UseModelForSinglePrediction(mlContext, trainedModel, art);

   bool UseModelForSinglePrediction(MLContext Context, ITransformer model,
      ArticuloVisit articulo)
   {
      var predictionEngine =
         mlContext.Model.CreatePredictionEngine<ArticuloVisit,
         ArticuloVisitPrediction>(model);

      var testInput = articulo;

      var articuloVisitPrediction = predictionEngine.Predict(testInput);

      if (articuloVisitPrediction.Score > 5)
      {
         return true;
      }
      else
      {
```

```
            return false;
        }
    }
}
```

**Fragmento de código 6.24:** Función IsRecommended

# CAPÍTULO 7

# Pruebas

En este apartado hablaremos de las opciones que teníamos para probar nuestro proyecto y las pruebas que se han realizado sobre el código de éste.

Para empezar, comentaremos que debido al tiempo con el que contábamos, además de que utilizábamos una metodología ágil como es SCRUM, no era recomendable implementar pruebas unitarias [31] o pruebas de integración [32] en su totalidad. En algún caso hubiera sido interesante para realizar alguna automatización de pruebas de aceptación [33] con pruebas unitarias (con XUNIT o MSTest), pero finalmente no se implementó ninguna. En su lugar, permitiremos a los empleados utilizar la aplicación una vez se pasen estas pruebas para ver posibles errores de ésta y encontrar formas de realizar un diseño más cómodo o nuevas funciones que no se han planteado y podrían ser útiles para los usuarios finales.

Principalmente, las pruebas que se han realizado sobre el proyecto son pruebas de aceptación, definidas previamente en la fase de especificación de requisitos de cada UT, debido al uso de SCRUM. Estas pruebas unitarias eran comprobadas en una primera instancia por el programador y, una vez comprobadas, se volverán a pasar por los empleados de la tienda, a los que se les dará también el control de la aplicación por un tiempo y se les observará para detectar posibles *bugs* que puedan surgir.

Cuando una de las pruebas de aceptación falla, debemos retroceder en el *workflow* hasta la actividad de programar y volver a computar las horas en esta actividad, poniendo todo nuestro esfuerzo en arreglar los problemas que han surgido durante la ejecución de estas pruebas de aceptación.

También se realizarán pruebas de regresión [34] al final del sprint, que consisten en volver a pasar todas las pruebas de aceptación de las UT's que se han programado hasta el momento para ver si al integrar las nuevas funcionalidades todo sigue funcionando como debería. Todos los *bugs* que se encuentran en estas pruebas de regresión se deben añadir al siguiente sprint como UT's de continuación y marcarlas como *hotfix* en el Trello para poder solucionarlas al siguiente sprint, o arreglarlas en el momento de las pruebas si se trata de un error crítico que afecte muy negativamente a la aplicación o si es un error que no lleva mucho tiempo en solucionar.

Además, al utilizar las pruebas de aceptación, nos aseguramos de que la interfaz de usuario funciona correctamente y podemos comprobar características como el tiempo de carga de las páginas o si éstas son *responsive* [35]. Estas propiedades son exclusivas de la interfaz de usuario, por lo que no podríamos comprobarlas con pruebas unitarias o cualquier otro tipo de pruebas.

Cabe destacar que este tipo de pruebas nos pueden resultar realmente interesantes cuando las realizan los trabajadores de la empresa para obtener *feedback* en ese mismo momento. Esto nos permitirá

realizar cambios más acertados en nuestra aplicación continuamente y nos ayudará a realizar cambios para crear una interfaz apropiada para los usuarios si la que habíamos creado en un principio no resulta fácil de utilizar, es poco intuitiva, …

Finalmente, este tipo de pruebas resulta muy interesante al utilizarla en conjunto con la metodología SCRUM y la idea del MVP, ya que por cada UT que realizamos podemos ver con nuestros propios ojos como la aplicación va tomando forma, integrándose con el resto de funcionalidades. Esto nos puede ayudar a cambiar a tiempo algunas de las interacciones de la página web que se planearon de una forma a la hora de especificar la UT y que por temas de diseño o funcionalidad es más interesante plantear esta interacción de otra manera, mejorando la funcionalidad de la aplicación, la estética o la facilidad de uso, entre otras cosas.

# CAPÍTULO 8

# Despliegue

En este capítulo hablaremos del despliegue de nuestra aplicación para que los usuarios (tanto trabajadores como clientes) puedan acceder a ella por internet. Para ello, la empresa decidió contratar un plan de *hosting* de la empresa Hostalia, que se dedica completamente a la venta de planes para hosting de PHP y .NET, clouding o implementación y despliegue de comercios virtuales. Esta decisión fue tomada por los administradores de Enrique Llorens Ciurana S.L. debido a que ya poseen una página web para el apartado de montaje de almacenes y cadenas logísticas alojada en Hostalia.

Hostalia ofrece diferentes planes para PHP y .NET, siendo para PHP *Standard*, *Unlimited* y *Unlimited Pro* y para .NET *Basic Windows*, *Standard Windows* y *Unlimited Windows.* Dentro de los planes de .NET, debido a que nuestra aplicación está programada en ASP.NET, decidimos contratar el plan intermedio *Standard Windows* que incluye: un dominio, alta en buscadores, herramientas de email marketing, posibilidad de alojar múltiples páginas web, 600MB de RAM en el servidor, 100GB de almacenamiento SSD, 20 cuentas de correo y posibilidad de almacenar hasta 10 Bases de datos MS SQL (ver figura 8.1).


**Figura 8.1:** Planes de hosting ASP.NET Hostalia

Otra de las opciones interesantes que ofrece Hostalia es la contratación de servidores *cloud* [36] de bastante buen rendimiento, pero debido a las dimensiones de nuestro proyecto y el precio ofertado para dichos servidores no resultaba del todo interesante en nuestro caso (ver figura 8.2).

Por último, existía la opción de montar la aplicación en el servidor físico de la empresa, pero debido a la conexión que dispone la empresa hoy en día (al estar situada en un polígono aún no disponen de fibra óptica, por lo que la conexión es bastante lenta e inestable) y a posibles brechas de seguridad que podrían comprometer los datos que se encuentran en el servidor, esta opción fue descartada desde el principio.


**Figura 8.2:** planes servidores *cloud* Hostalia

Además, debido a la metodología que seguimos, desde la finalización del primer *sprint* comenzó el despliegue de la aplicación subiendo una primera versión de esta, a modo de catálogo online y con las funciones principales, y poco a poco, tras cada *sprint*, se ha ido actualizando con las nuevas versiones y continuará actualizándose cuando se vaya alcanzando una nueva versión del producto. De esta manera

seguimos con la idea del MVP en nuestro proyecto y en cada final de *sprint* la aplicación se vuelve a subir a los servidores de Hostalia , que automáticamente queda disponible para todos los usuarios a través de la dirección www.elecfer.com.

# CAPÍTULO 9
# Conclusiones y Trabajo futuro

## 9.1 Relación con los estudios cursados

Es evidente que para la realización de este trabajo resultan indispensables gran cantidad de conocimientos que he adquirido durante la carrera y que, sin ellos, la realización del proyecto no hubiera sido posible. Para su realización se han requerido conocimientos que he ido adquiriendo en la carrera de ingeniería informática y, sobre todo, en la rama de Ingeniería de Software.

Para empezar, gracias a asignaturas como PSW (Proceso de software) o PIN (Proyecto de ingeniería de software) aprendimos el uso de metodologías ágiles como SCRUM, que es la metodología utilizada en nuestro proyecto. También de estas asignaturas proviene el uso de las pruebas de aceptación y pruebas de regresión, muy útiles en proyectos que siguen este tipo de metodologías.

A la hora de especificar los requisitos, hay que atribuirle los méritos, además de las dos asignaturas anteriores, a AER (Análisis y especificación de requisitos), asignatura en la cual se nos explicaron métodos de elicitación y especificación de requisitos que hemos utilizado en nuestro proyecto, como pueden ser el uso de entrevistas y reuniones con los *stakeholders* para recopilar requisitos, el uso del lenguaje natural para la definición de requisitos, el diagrama de contexto o el modelo de dominio entre otras cosas. Aunque en esta asignatura la especificación de requisitos estaba planteada para una metodología más tradicional ha resultado de gran ayuda junto con las asignaturas de PIN y PSW para la obtención de requisitos y la buena definición de estos en lenguaje natural, de manera que estos puedan ser entendidos por personas fuera del ámbito de la informática y el desarrollo de software.

Otra asignatura que ha resultado importante ha sido IEI (Integración e interoperabilidad), gracias a la cual hemos podido crear extractores de datos para poder introducir los datos de los artículos desde la base de datos de la empresa a la base de datos de la página web. También ha sido útil a la hora de crear *web-scrappers* para obtener información o fotos que faltaban de algunos de los artículos.

Hay que mencionar también la asignatura DDS (Diseño de software), gracias a la cual aprendimos a realizar *clean code* y *refactoring*, además de aprender patrones de diseño como el patrón fachada utilizado en nuestra aplicación. Relacionado con esto también esta la asignatura de MES (Mantenimiento y evolución de software), donde también aprendimos a arreglar errores del código y refactorizar partes de el para que sea más mantenible. También aprendimos a realizar una buena documentación del documento que hemos puesto en práctica en la aplicación, ya que nos ayudará a comprender el código con mayor facilidad en el futuro.

## 9.2 Conclusiones

En cuanto a las conclusiones, podemos afirmar que hemos conseguido llevar a cabo el objetivo principal del proyecto: desarrollar un MVP de una aplicación web para la venta de productos de ferretería para la empresa Enrique Llorens Ciurana S.L. Además, la aplicación web dispone de las características más importantes que nos habíamos planteado, como pueden ser el sistema de venta y pagos mediante un TPV virtual de Redsys, un modelo de aprendizaje automático para la recomendación de artículos a usuarios registrados y un subsistema de administración con funciones que ayudarán a la correcta gestión de la aplicación web por parte de los empleados.

La aplicación no se ha implementado de una vez, sino que a lo largo de los *sprints* que se han seguido (planteados en la metodología SCRUM) se obtenía un MVP con funcionalidades suficientes como para disponer de una aplicación funcional, aunque incompleta. Tras cada sprint conseguíamos una nueva versión de la aplicación con nuevas funcionalidades, implementando las más prioritarias primero (elegidas por consenso con los administradores en una reunión).

Ha sido necesaria la investigación del panorama actual de tiendas online, tanto de productos de ferretería, como de empresas más grandes que no solo se limitaban a este dominio (como puede ser Amazon). Gracias a esto y a las reuniones y entrevistas con los *stakeholders* se ha podido definir de manera sencilla los requisitos necesarios para la aplicación, los cuales pueden hacer que el producto destaque sobre el resto (o que el resto ya implementa pero nos resultan imprescindibles en una aplicación web como esta).

Durante el desarrollo de la aplicación se ha utilizado la arquitectura MVC haciendo una clara separación en los modelos de datos, las interfaces de usuario y la lógica y controladores de la aplicación. También se han investigado varias opciones de tecnologías de aprendizaje automático y se ha desarrollado un modelo capaz de recomendar productos a los usuarios según sus preferencias.

En cuanto a la persistencia de datos, se ha desarrollado una base de datos de MSSQL Server con un modelo *code-first* que nos permite generar un esquema en dicha base de datos a raíz de los modelos implementados en nuestra aplicación. Además, al igual que la aplicación, esta base de datos es incremental, puesto que a medida que avanza el desarrollo y necesitamos incluir nuevas tablas de datos al esquema de nuestra base de datos, simplemente con la implementación del modelo y una migración ejecutada en el entorno de desarrollo, nos permite añadir y modificar la estructura de dicho esquema y adaptarlo según nuestras necesidades.

Por último, toda la aplicación se ha probado tanto por el desarrollador como por algunos de los *stakeholders* (administradores y empleados de la tienda) mediante pruebas de aceptación y pruebas de regresión, comprobando de esta manera que tanto las interfaces de usuario, el código de la lógica de la aplicación, la integración de las distintas funcionalidades y las funciones sobre la base de datos funcionan de forma adecuada.

## 9.3 Trabajo futuro

Tras la finalización del proyecto, se han encontrado una serie de puntos que sería conveniente mejorar o incluir en la aplicación web con el fin de que esta tenga una mejor funcionalidad, sea más usable y amigable tanto para administradores como para clientes y nos permita destacar por encima del resto de aplicaciones web similares.

Lo primordial sería hacer un *restyling* de la interfaz de usuario, ya que la actual es muy básica y está pensada para ofrecer toda la funcionalidad al usuario. Tanto por motivos estéticos como por motivos de usabilidad se debería mejorar la interfaz para obtener una mejor experiencia para el usuario y que estos se encuentren cómodos a la hora de utilizar nuestra aplicación web.

También está planeado mejorar algunas de las funciones de administración para facilitar la gestión de la página web a los empleados de la tienda. Estas mejoras se irán incluyendo en futuros *sprints* según vayamos obteniendo el *feedback* de los empleados.

Otro de los factores a mejorar es el modelo de aprendizaje automático, al que deberíamos realizarle algunos retoques ya que, ahora mismo, los datos con los que se entrena este modelo son introducidos manualmente por nosotros para que recomiende productos relacionados con la familia. En su lugar esto se debería cambiar para que las recomendaciones se hagan en base al flujo de visitas de los usuarios a los artículos.

Finalmente, aunque se puede acceder a la web con el móvil y verla en vista móvil (aunque esta se mejorará con el futuro *restyling* de la web), sería interesante investigar e implementar una aplicación móvil para que los clientes puedan acceder a la tienda online desde su dispositivo móvil con mayor facilidad.

# Bibliografía


[1] Karla Sutil: What is an MVP in an Agile context? Boldare (2021). https://www.boldare.com/blog/what-is-mvp-in-agile/.

[2] Franc Vidal: What is a marketplace? Our understanding of multi-seller businesses. Shopery (2019). https://www.shopery.com/insights/what-is-a-marketplace.

[3] Ritesh Ranjan: What is a framework in programming & why you should use one. Net solutions (2021). https://www.netsolutions.com/insights/what-is-a-framework-in-programming/.

[4] Edgar Higuerey: La importancia del feedback para el éxito de una empresa (2018). https:/rockcontent.com/es/blog/que-es-feedback/.

[5] Brindalakshmi Rajkumar: What is video streaming & how does video streaming work? Vplayed (2022). https://www.vplayed.com/blog/what-is-video-streaming/.

[6] MathLab Team: What is machine learning? MathWorks. https://www.mathworks.com/discovery/machine-learning.html.

[7] Tefo Sekgweleo: Understanding traditional systems development methodologies.IJAME, vol. 4, pp 51-58 (2015). https://www.managementjournal.info/index.php/IJAME/article/viewFile/370/313.

[8] Radha Shankarmani, Renuka Pawar, S. S. Mantha: Agile methodology adoption: benefits and constraints. International Journal of Computer Applications 58(15), 31-37 (2012). https://www.researchgate.net/publication/261017281_Agile_Methodology_Adoption_Benefits_and_Constraints.

[9] Abraham Requena Mesa: Qué es un sprint de scrum. OpenWebinars (2018). https://openwebinars.net/blog/que-es-un-sprint-scrum/.

[10] Sean Peek: What is agile scrum methodology? Business News Daily (2021). https://www.businessnewsdaily.com/4987-what-is-agile-scrum-methodology.html.

[11] Ana Arboleda: Conoce las funciones de un producto owner y su importancia en los proyectos ágiles. Rockcontent (2020). https://rockcontent.com/es/blog/product-owner/.

[12] Leeron Hoory, Cassie Bottorff: What is a scrum master? Everything you need to know. Forbes (2021). https://www.forbes.com/advisor/business/what-is-a-scrum-master/.

[13] Rafael García Tamarit: ¿Qué es el backlog? Muy agile (2019). https://muyagile.com/que-es-el-backlog/.

[14] Dan Radigan: Uso de workflows por diversión y sus beneficios. Atlassian. https://www.atlassian.com/es/agile/project-management/workflow.

[15] Soledad Pagés: El mock up en programación: Que es, importancia y cómo hacer uno. Workana (2018). https://blog.workana.com/uncategorized/importancia-mock-up-proyectos-it/.

[16] Gustavo Sánchez: Agile: ¿Qué es un Stakeholder? Happy Devops (2020). https://happydevops.com/2020/01/09/agile-que-es-un-stakeholder/.

[17] Mathew Martin: What is a functional requirement in Software Engineering? Guru99 (2022). https://www.guru99.com/functional-requirement-specification-example.html.

[18] Paula Rome: What are non functional requirements? Perforce (2020). https://www.perforce.com/blog/alm/what-are-non-functional-requirements-examples.

[19] TUNE-UP Process Team: Trabajo básico con una UT. Worki-Ayuda. https://cliente.tuneupprocess.com/help/web/TrabajobasicoconunaUT.html.

[20] Miguel Ángel Álvarez: ¿Qué es MVC? Desarrolloweb.com (2020). https://desarrolloweb.com/articulos/que-es-mvc.html.

[21] Christina Tyler: What is ASP.NET? and it's architecture. Guru99 (2022). https://www.guru99.com/what-is-asp-dot-net.html.

[22] Abhimanyu K Vatsa: Code first Approach in Entity Framework. C# Corner (2020). https://www.c-sharpcorner.com/UploadFile/abhikumarvatsa/code-first-approach-in-entity-framework/.

[23] Joe Johnston: What is a CRUD app and how to buil one. Buildbase (2021). https://budibase.com/blog/crud-app/.

[24] Andrea Chiarelli: What is .NET? An overview of the platform. Auth0 (2021). https://auth0.com/blog/what-is-dotnet-platform-overview/.

[25] Chinmayee Deshpande: What is backend development: skills, salary, roles & more. Simplilearn (2021). https://www.simplilearn.com/tutorials/programming-tutorial/what-is-backend-development.

[26] Gustavo B.: ¿Qué es AJAX y cómo funciona? Hostinguer (2022). https://www.hostinger.es/tutoriales/que-es-ajax.

[27] Jessica Wilkins: Front end developer – What is front end development, explained in plain english. FreeCodeCamp (2021). https://www.freecodecamp.org/news/front-end-developer-what-is-front-end-development-explained-in-plain-english/.

[28] Jonathan Freeman: What is JSON? A better format for data exchange. Infoworld (2019). https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html.

[29] Chris Hoffman: What is an API, and how do developers use them? How-To Geek (2021). https://www.howtogeek.com/343877/what-is-an-api/.

[30] Mahes Chand. What is ML.NET? C# Corner (2020). https://www.c-sharpcorner.com/article/what-is-ml-net/.

[31] Navdeep Singh Gill. Unit testing techniques and best practices. Xenonstack (2021). https://www.xenonstack.com/insights/what-is-unit-testing.

[32] Thomas Hamilton. Integration Testing: What is, types, top down & bottom up. Guru99 (2022). https://www.guru99.com/integration-testing.html.

[33] Thomas Hamilton. What is User Acceptance Testing (UAT)? Guru99 (2022). https://www.guru99.com/user-acceptance-testing.html.

[34] Thomas Hamilton. What is regression testing? Definition, Test Cases. Guru99 (2022). https://www.guru99.com/regression-testing.html.

[35] Sol González. ¿Qué es un diseño web responsive? Cyberclick (2019). https://www.cyberclick.es/que-es/diseno-web-responsive.

[36] Eric Griffith. What is cloud computing? PC Mag (2022). https://www.pcmag.com/how-to/what-is-cloud-computing.

# APÉNDICE A

# Patrón Fachada

El patrón fachada es un patrón de diseño estructural utilizado para ofrecer una interfaz simplificada a una biblioteca, framework o cualquier otro grupo de clases. Por ello hemos decidido utilizarlo en conjunto con *Entity Framework*, de modo que tengamos una interfaz clara y accesible para realizar todas las operaciones necesarias para la creación, lectura, modificación y borrado de los objetos que persisten en la base de datos.

De esta manera, facilitamos el acceso a las operaciones CRUD de los objetos de la aplicación y resulta más sencillo implementar la gestión de la tienda, el carrito de la compra o la búsqueda de artículos, entre otras cosas.

Una analogía para entender este patrón con un entorno real podría ser la que se plantea en Refactoring guru (https://refactoring.guru/es/design-patterns/facade), donde dicen que cuando llamas a una tienda, la persona que te atiende sería la fachada que te ofrece toda la funcionalidad de la tienda (ver figura A.1).

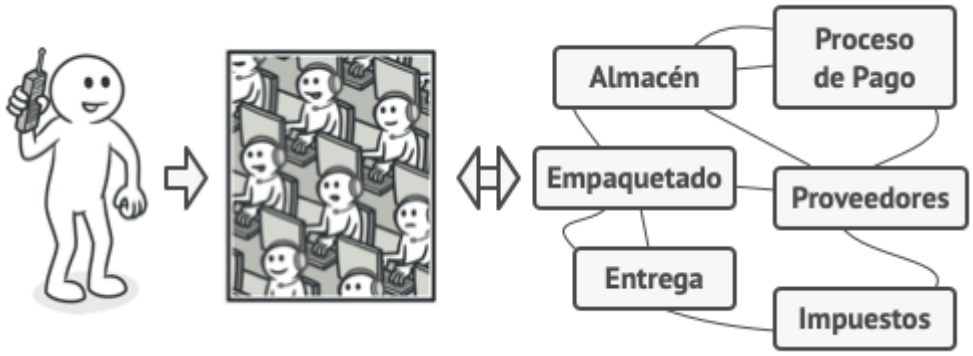


**Figura A.1:** Analogía del patrón fachada en un entorno real

También en esta web se expone un caso más ambientado al desarrollo software, como es una aplicación cuya clase "VideoConverter" se relaciona con un conjunto de clases creando dicho patrón fachada para poder encapsular las funciones de conversión de video en una sola clase más accesible (ver figura A.2).

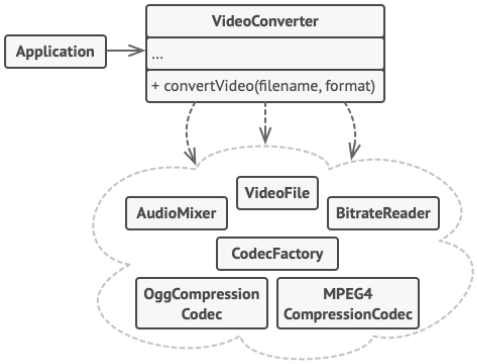


**Figura A.2:** Ejemplo de patrón fachada

En nuestro caso hemos creado una clase “ServicioFachada” que está relacionada con las interfaces de “IDAL” y “EntityFrameworkDAL”, Además de la clase “ApplicationDBContext” encargada de crear el contexto de la base de datos en nuestro proyecto para poder realizar las migraciones de los modelos. Dentro de la clase “ServicioFachada” encontramos funciones para la creación, modificación, lectura y borrado de los objetos que persisten en la base de datos, permitiéndonos acceder a ellas desde cualquier parte del código mediante una instancia de esta clase fachada (ver figura A.3).

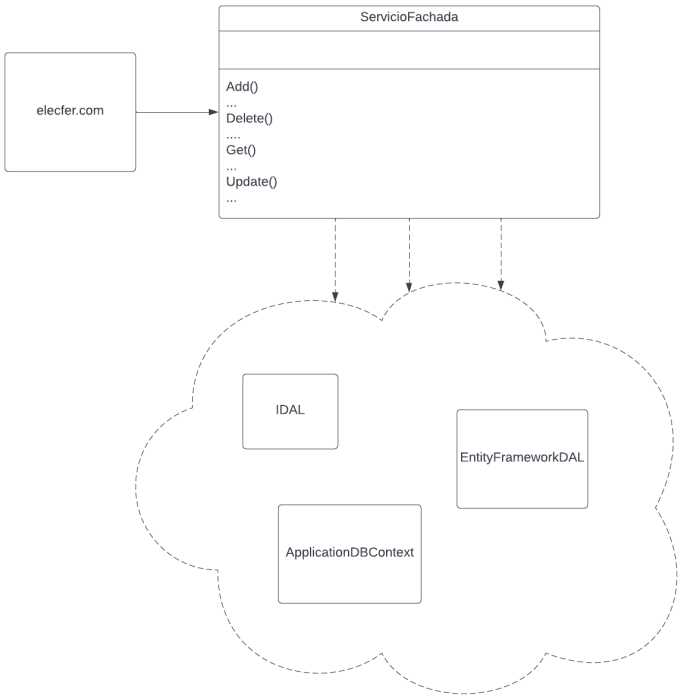


**Figura A.3:** Esquema del patrón fachada en nuestra aplicación

# APÉNDICE B

# Objetivos de desarrollo sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

| **Objetivos de Desarrollo Sostenibles** | **Alto** | **Medio** | **Bajo** | **No Procede** |
|---|---|---|---|---|
| ODS 1. **Fin de la pobreza.** | | | | **X** |
| ODS 2. **Hambre cero.** | | | | **X** |
| ODS 3. **Salud y bienestar.** | | | | **X** |
| ODS 4. **Educación de calidad.** | | | | **X** |
| ODS 5. **Igualdad de género.** | | | | **X** |
| ODS 6. **Agua limpia y saneamiento.** | | | | **X** |
| ODS 7. **Energía asequible y no contaminante.** | | | | **X** |
| ODS 8. **Trabajo decente y crecimiento económico.** | | **X** | | |
| ODS 9. **Industria, innovación e infraestructuras.** | | | | **X** |
| ODS 10. **Reducción de las desigualdades**. | | | | **X** |
| ODS 11. **Ciudades y comunidades sostenibles**. | | | **X** | |
| ODS 12. **Producción y consumo responsables.** | | | | **X** |
| ODS 13. **Acción por el clima.** | | | | **X** |
| ODS 14. **Vida submarina.** | | | | **X** |
| ODS 15. **Vida de ecosistemas terrestres.** | | | | **X** |
| ODS 16. **Paz, justicia e instituciones sólidas.** | | | | **X** |
| ODS 17. **Alianzas para lograr objetivos.** | | | | **X** |

**Tabla B.1:** Relación del trabajo con los ODS

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados:

Realmente nos encontramos ante un proyecto que no tiene mucha relación en cuanto a los objetivos de desarrollo sostenible, pues una tienda virtual de ferretería tampoco puede ayudar en objetivos como pueden ser “Fin de la pobreza”, “Hambre cero” o “Vida submarina”, puesto que no tienen ninguna relación con el objetivo del proyecto. Es evidente que no todos los desarrollos van a poder abarcar todos los ODS, habiendo proyectos que puedan abarcar más objetivos y otros menos, pero aun así, aunque no lo creamos, la gran mayoría de los desarrollos que se realizan contribuyen, aunque sea en menor medida, con algunos de los objetivos que podemos observar en la lista de los ODS. Teniendo en cuenta esto, podemos encontrar

relaciones evidentes entre algunos de los objetivos que se mencionan y nuestro proyecto, y es de interés comentarlas.

El objetivo principal y con el que más se relaciona el proyecto es “Trabajo decente y crecimiento económico”. Esto se debe obviamente a que el objetivo del proyecto es implementar una tienda virtual para la empresa Enrique Llorens Ciurana S.L., por tanto estaremos generando puestos de trabajo dignos tanto dentro de la empresa como fuera de esta, y por ende, un crecimiento económico en cierta medida.

Estos nuevos empleos que surgen dentro de la empresa estarán centrados principalmente en la administración y gestión de la página web mediante el apartado de administración que hemos implementado, haciendo que el catálogo de productos, los artículos destacados y los catálogos que podemos encontrar en la web nunca estén desactualizados. También dentro de la empresa encontramos puestos dedicado al mantenimiento de éste y futuros proyectos, haciendo que nunca se encuentren desactualizados, en cuanto a software se refiere. Además, necesitaremos personal encargado de preparar los pedidos, realizar los albaranes y facturas y tramitar los envíos de dichos pedidos.

Fuera de la empresa también incentivamos nuevos puestos de trabajo o afianzamos puestos ya existentes, debido a la necesidad de llegar a un acuerdo con una empresa de reparto para poder enviar los pedidos a cualquier parte de España en el tiempo establecido. Esto involucrará a repartidores de paquetes, camioneros para trayectos más largos y personal de los almacenes de dicha empresa de reparto, que deberán gestionar a que sucursal se envía cada paquete para que pueda ser repartido a su comprador.

Por último, en cuanto al objetivo “Ciudades y comunidades sostenibles”, este proyecto tiene como objetivo principal poder abastecer de productos de ferretería y menaje a personas que lo necesiten en todo el territorio español. Debido al reparto a domicilio que queremos ofrecer con nuestra tienda virtual estaremos evitando desplazamientos innecesarios a una tienda física, ya sea andando o con cualquier vehículo, aportando de esta manera nuestro granito de arena a crear ciudades más sostenibles.

En conclusión, podemos afirmar que nuestro proyecto se compromete con los Objetivos de Desarrollo Sostenible, teniendo una clara relación con los objetivos “Trabajo decente y crecimiento económico”, debido a que creamos nuevos puestos de trabajo y afianzamos puestos ya existentes, y con “Ciudades y comunidades sostenibles”, debido a que con nuestra tienda virtual contribuiremos a que las ciudades sean un poco más sostenibles permitiendo la compra de productos de ferretería sin necesidad de ir a una tienda física.