Marcos de Desarrollo
Curso 2020/21

Tienda de Comercio Electrónico

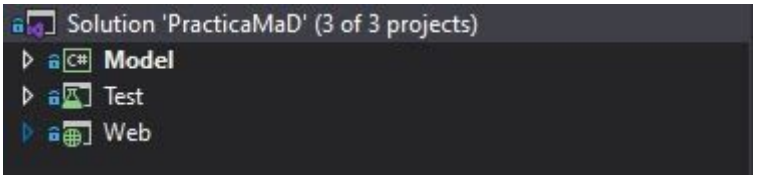
Memoria de la práctica

Grupo 09
Xaquín Pérez Bermo (xaquin.perez1)
Santiago Barreiro Hermida (santiago.barreiro)
Mauro Paredes Romero (mauro.paredes)

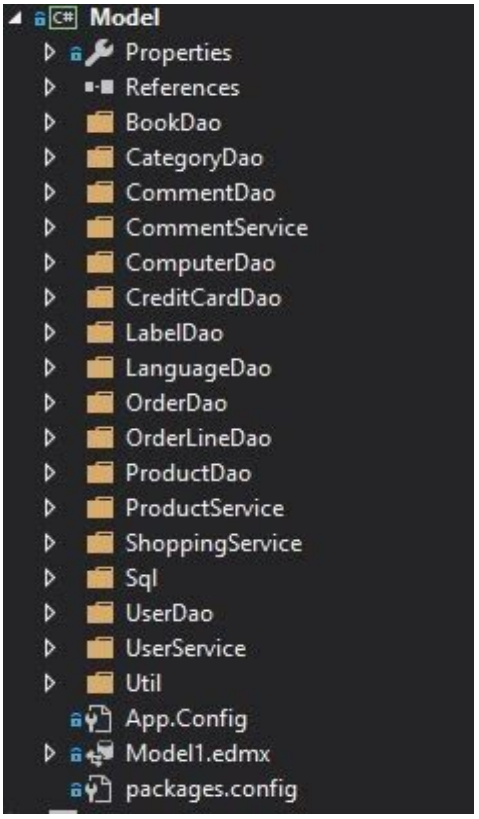
UNIVERSIDADE DA CORUÑA

1. Arquitectura global

A continuación mostraremos los distintos criterios que hemos seguido para la organización de nuestra aplicación, apoyándonos de imágenes:



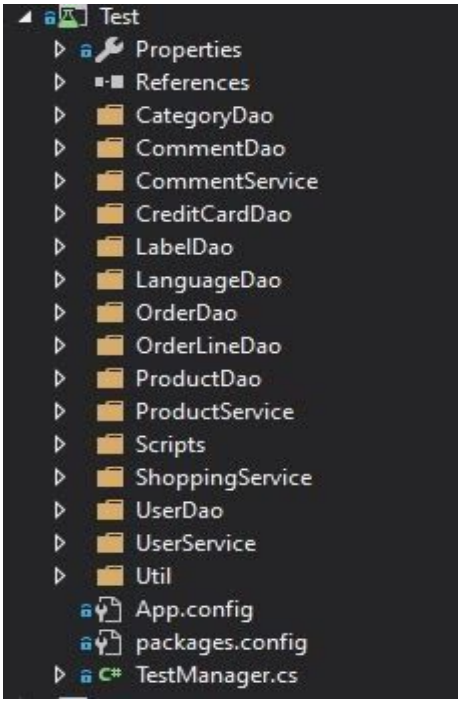
Desde una vista general, nuestra solución se compone de 3 proyectos: uno que representa el modelo (base de datos, clases persistentes, DAO's, etc.), los test asociados al modelo (comprobando funcionamiento de los DAO's y servicios) y la parte Web (vistas, CSS, imágenes, etc.).



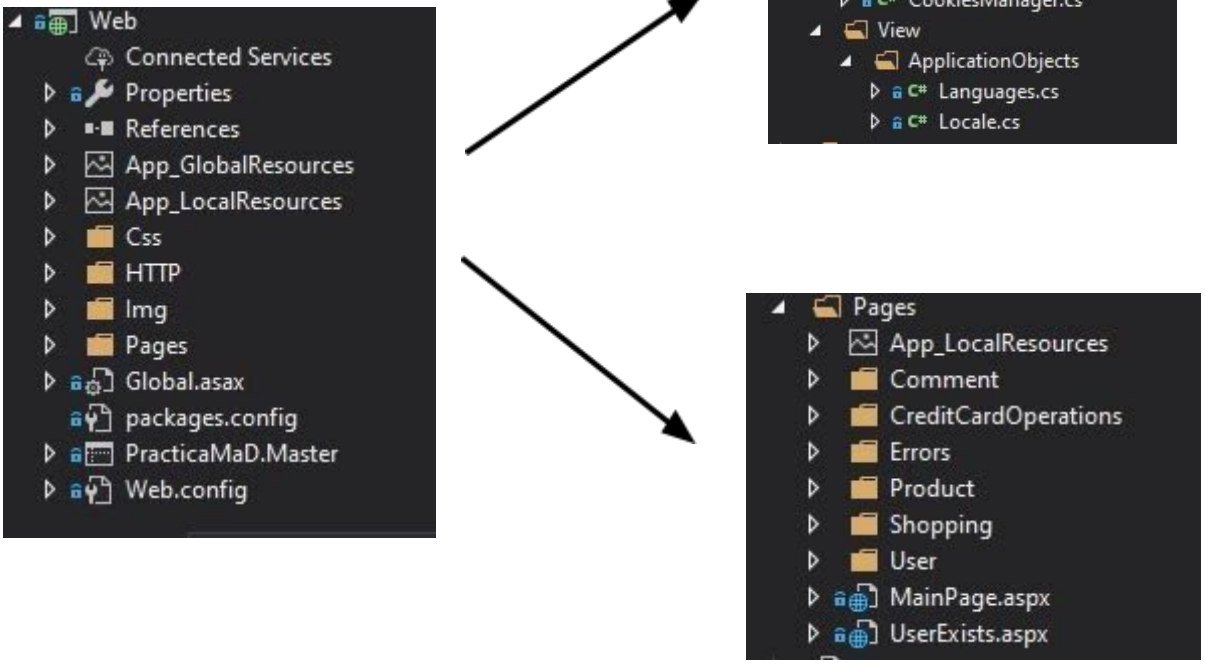
Para la parte del modelo, se pueden distinguir 4 categorías de directorios, los distintos DAO's (recoge tanto la interfaz como la implementación del mismo, p.ej: BookDao), los servicios (interfaz, implementación y DTO's y Blocks, en caso de que hagan uso de ellos), scripts de la BD y Util.

En este último directorio hemos almacenado la clase CacheUtil.cs, que contendrá funciones para la parte opcional de cacheado que comentaremos en el apartado 4.3 de este documento.

Para la parte de test, nos encontramos con una organización similar al modelo, guardando en los distintos directorios las clases de test correspondientes a los DAO's y servicios.



También hemos creado un directorio con el script de creación de la BD para los tests y otro para funciones comunes que crearán objetos que serán empleados en los distintos casos de prueba (creación de un User, un Book, etc…).



Finalmente, para la parte Web creamos 4 directorios:

- **Css:** contiene el archivo global CSS que contiene los distintos estilos que se usarán en las distintas vistas, se trata del archivo PracticaMaD.Styles.css.

- **HTTP:** contará con los subdirectorios Session, Util y View. En el caso de Session contiene clases encargadas del correcto funcionamiento de cada sesión que es creada en la parte Web. Mantiene el idioma del usuario, crea un carrito vacío, contiene parámetros de usuario logeado, etc. Util contiene funcionalidades para controlar las cookies, así como gestionar las dependencias con los distintos DAO's y servicios del modelo.

  Finalmente View para controlar la selección de idioma del usuario.

- **Img:** para las imágenes empleadas en las distintas vistas.

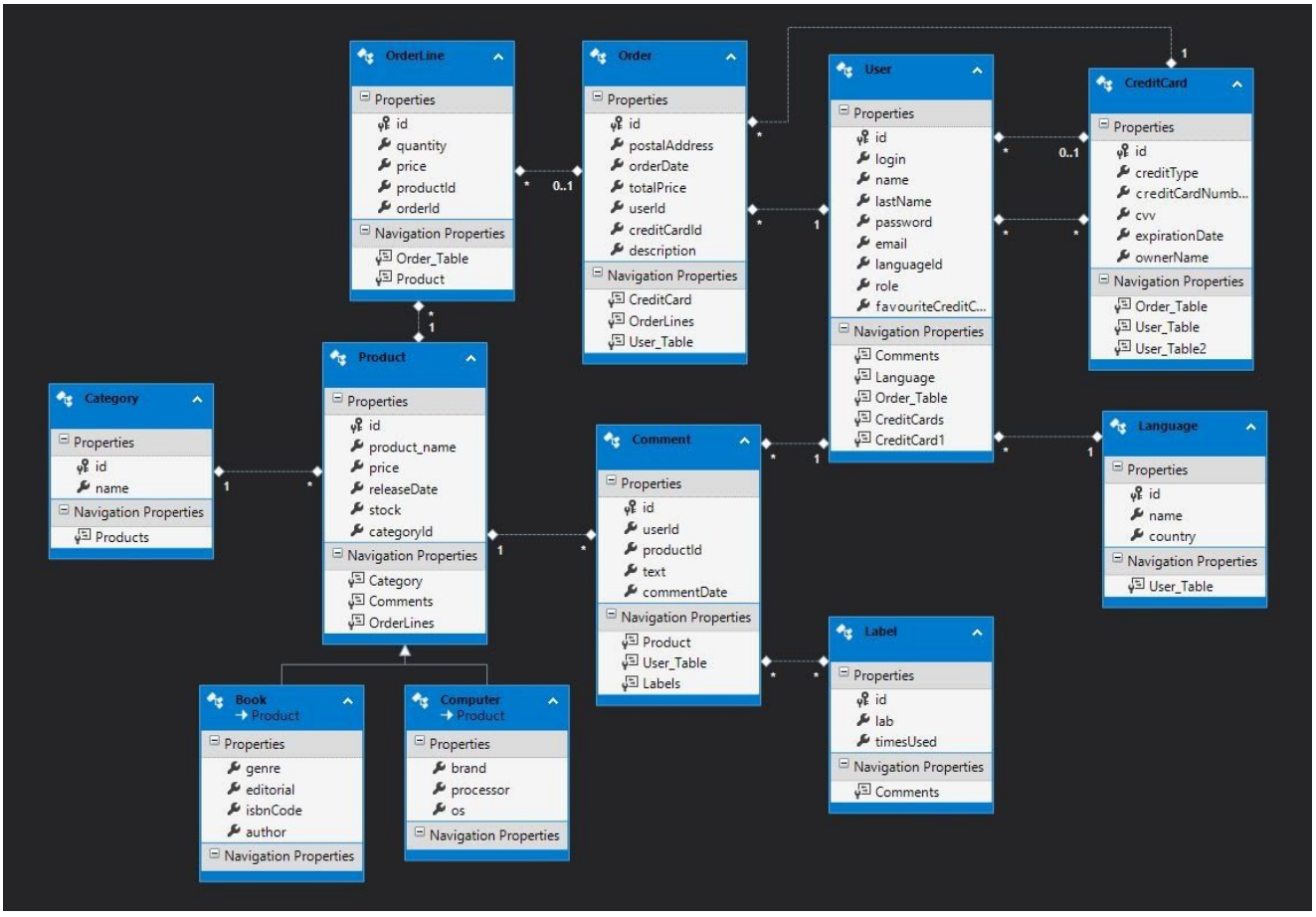
- **Pages:** para las vistas, el cual contiene una serie de subdirectorios. Estos agrupan las distintas vistas con sus correspondientes code-behind. Su organización es prácticamente la misma a la que empleamos para crear los distintos servicios (explicación en el apartado 2.2), a excepción de CreditCardOperations, que creamos a mayores para guardar los casos de uso de mostrar y añadir métodos de pago.

  También existe un directorio Errors con vistas para distintos errores que podamos recibir del modelo, como puede ser una tarjeta de pago caducada, solicitar más stock del disponible para un producto, etc.

  Dentro de cada una de ellas se encuentra la carpeta App_LocalResources con los archivos de internacionalización.

2. Modelo

2.1 Clases Persistentes



En el diagrama Entidad-Relación podemos observar las distintas clases persistentes en nuestra aplicación.

Como se puede observar hemos creado dos tipos de productos específicos que heredan las propiedades de su clase padre y amplían con características propias, se trata de las clases Book y Computer.

Cabe destacar, que como decisión de diseño hemos decidido establecer dos relaciones N-N entre usuarios y tarjetas de crédito y entre comentarios y etiquetas. Consideramos que un usuario puede tener N

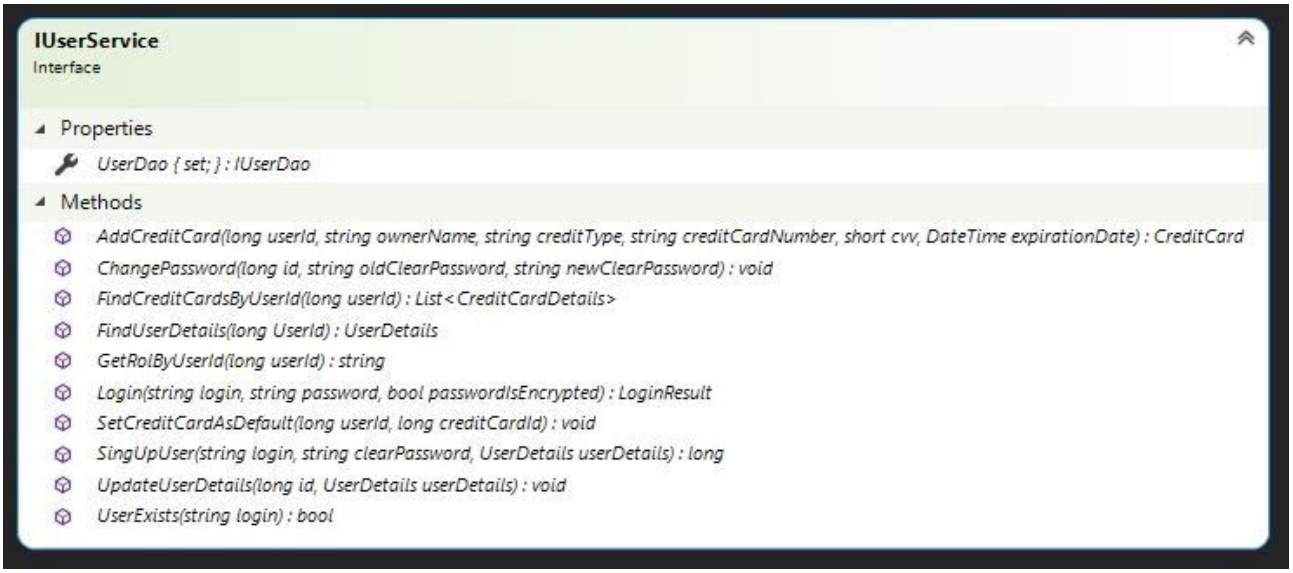
tarjetas, pero a su vez estas pertenecer a N usuarios. El resto consisten en relaciones 1-N o variantes (p.ej: 0..1-N).

En el caso de usuario y tarjeta existe una relación más que representa el método de pago que un usuario marca por defecto, lo cual facilitará al usuario final a la hora de realizar una compra.

2.2 Interfaces de los servicios ofrecidos por el modelo

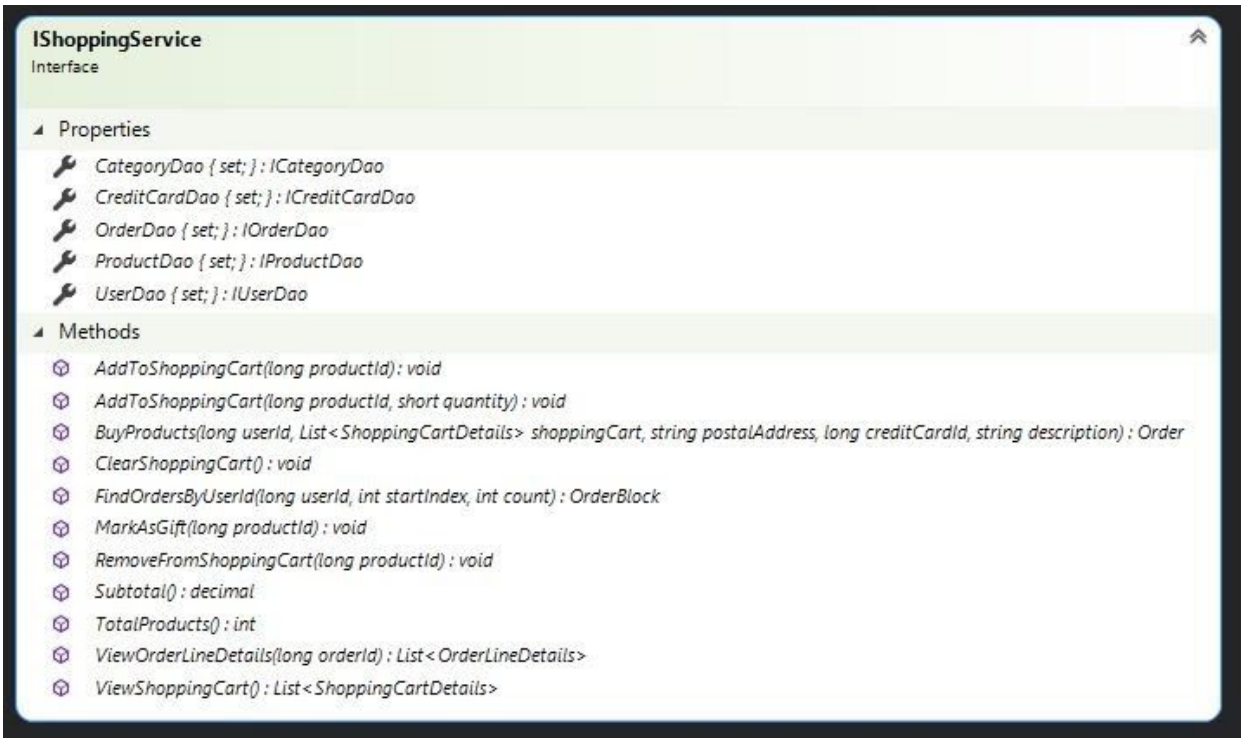
Para la agrupación de los distintos casos de uso nos hemos decantado por organizarlo en base a operaciones que pertenecen a un mismo área.

- **User:** contiene todas las operaciones relacionadas con una cuenta de usuario (autenticarse, cerrar sesión, cambiar datos, etc.) así como el control de sus métodos de pago (añadir tarjeta, recuperar tarjetas, etc.). Tras debatir si incluir las operaciones sobre los métodos de pago en este servicio o crear uno propio para ello, nos decantamos por la primera opción porque lo consideramos como algo propio de un usuario.



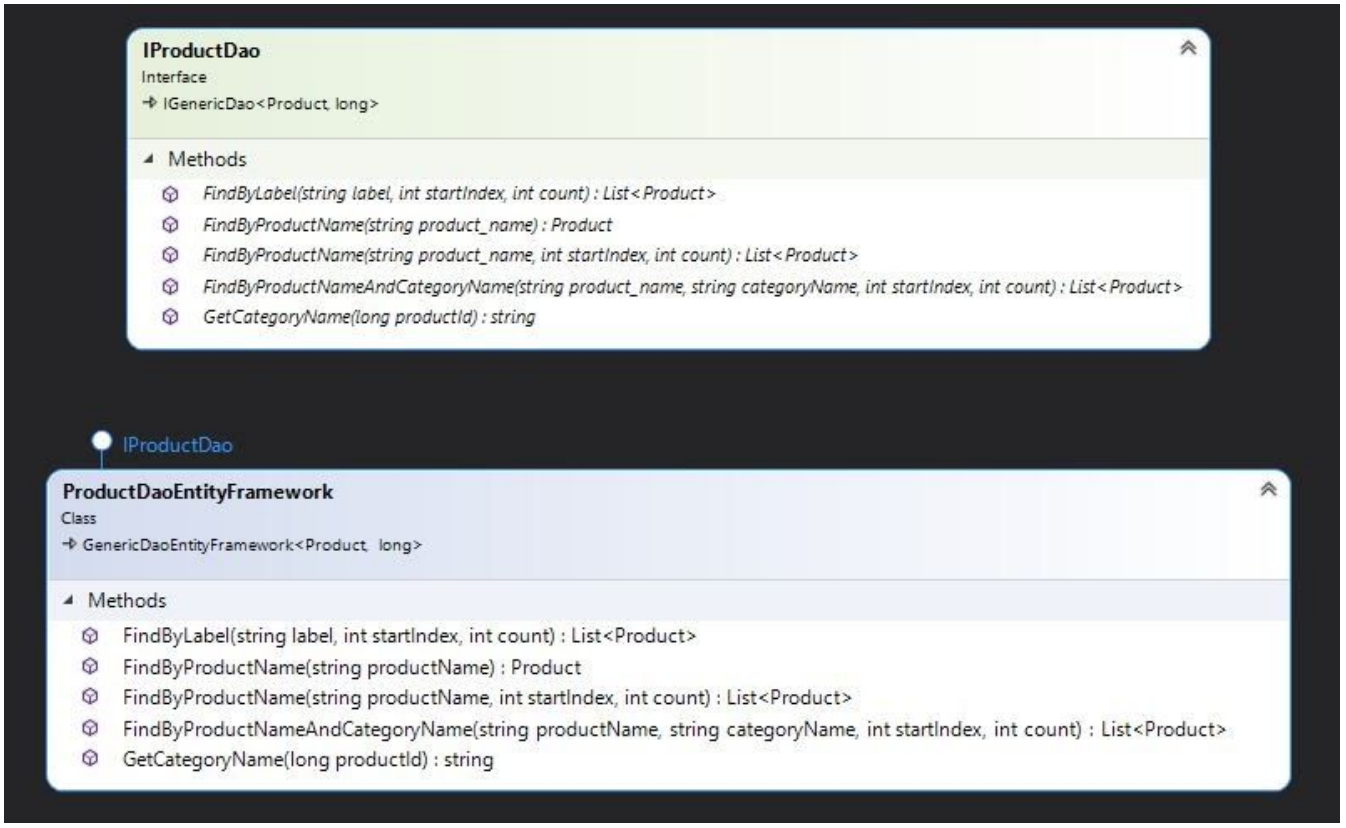
- **Product:** contiene las operaciones relacionadas con recuperar datos de productos, ya sea el catálogo completo o los detalles de un ordenador / libro, así como actualizarlos. También se encuentra la operación ViewProductsByLabel, que es parte del etiquetado de comentarios que se explica en el apartado 4.2, recuperando productos de la etiqueta seleccionada de la nube de etiquetas.

- **Shopping:** contiene tanto las operaciones del carrito (ver, añadir / borrar unidad, marcar como regalo, etc.), comprar y recuperar el histórico de compras.



- **Comment:** contiene los distintos casos de uso requeridos para implementar las partes opcionales de comentario de productos y etiquetado de comentarios, ya que ambas clases persistentes y DAO's están relacionados de forma directa. En este caso, también nos hemos planteado la opción de crear un nuevo servicio para las etiquetas, pero pensamos que esto agrupaba las operaciones de forma mucho más clara.

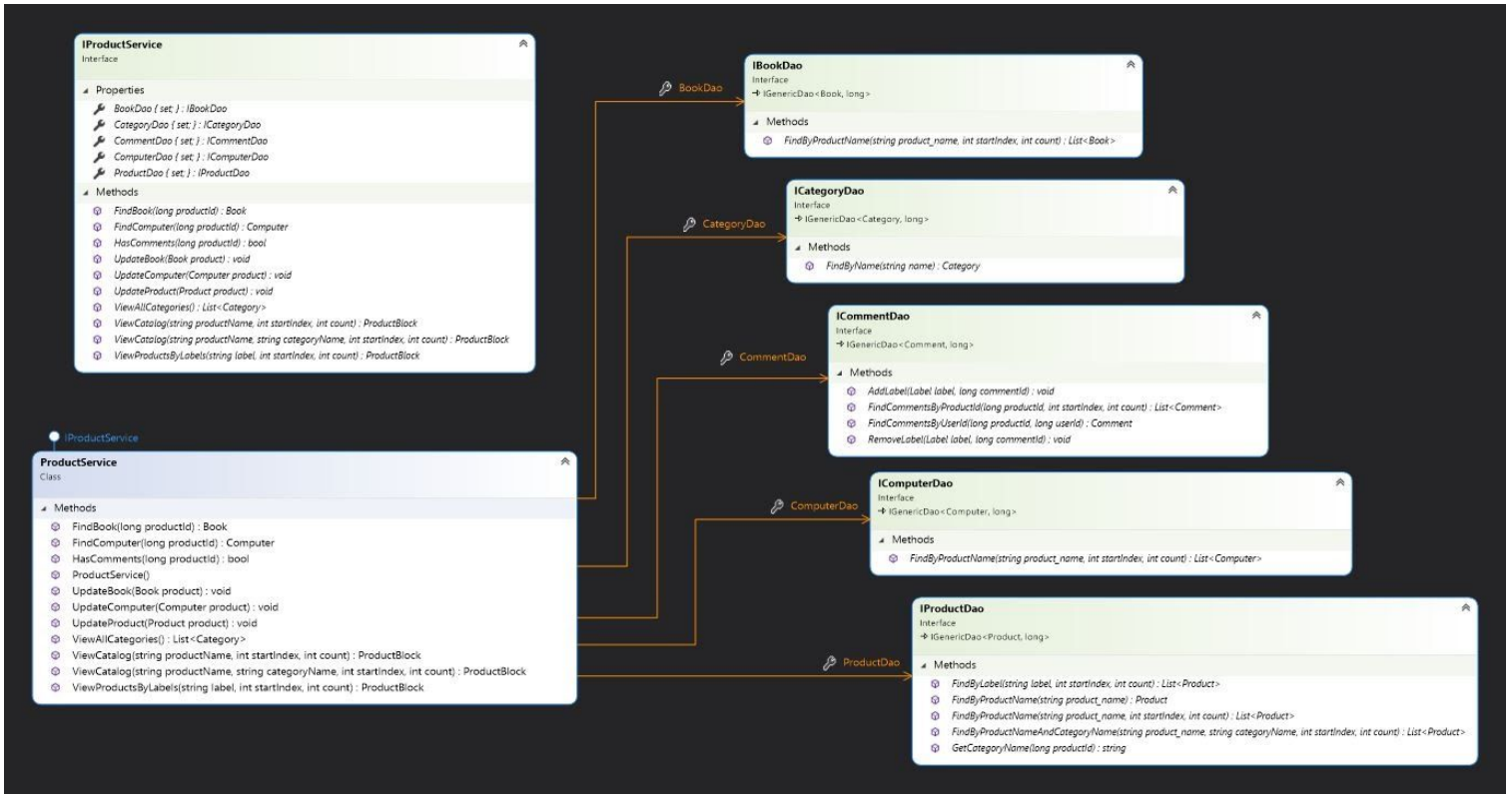
2.3 Diseño de un DAO



Para representar el funcionamiento de uno de nuestros DAO's, hemos decidido seleccionar ProductDao. Al igual que el resto de DAO's del modelo, la interfaz extiende de IGenericDao, pero en este caso con la clase Product y como tipo de clave long.

Por parte de su implementación, siguiendo la misma nomenclatura que el resto (ProductDaoEntityFramework), implementa la interfaz IProductDao y extiende de GenericDaoEntityFramework.

2.4 Diseño de un servicio del modelo



Para este apartado emplearemos como caso de uso para su explicación “Ver catálogo”. Para ello, es necesario mostrar el servicio representado con la fachada IProductService.

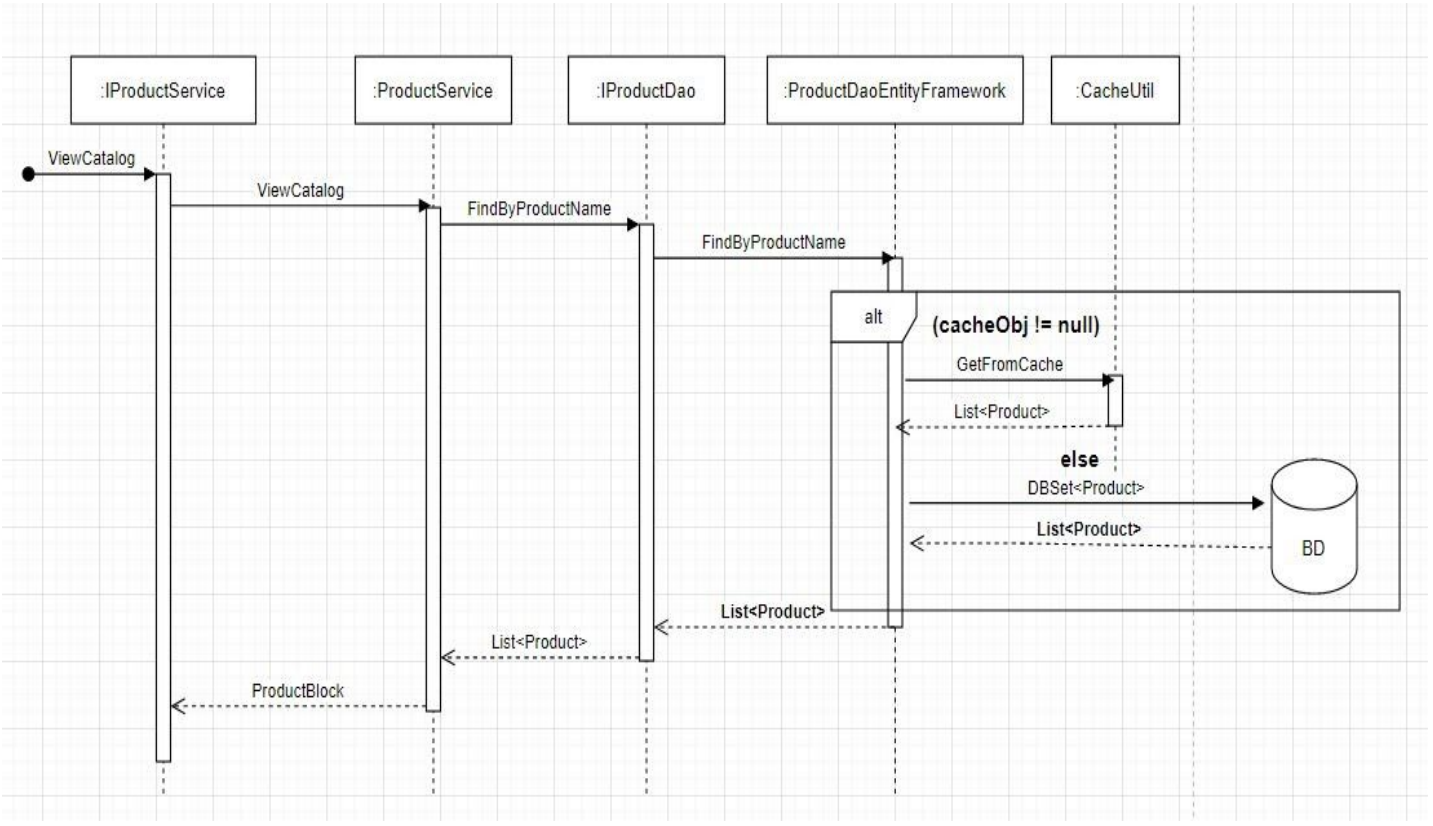
Como se puede ver en la imagen, esta interfaz es implementada por la clase ProductService, haciendo uso de los distintos DAO’s mostrados (ProductDao, BookDao, ComputerDao, CommentDao y CategoryDao).

En el diagrama de secuencia que mostraremos más abajo se puede ver con claridad el funcionamiento del caso de estudio, desde su llamada a la función ViewCatalog hasta el acceso a la base de datos para realizar la consulta select (en caso de ser necesario).

No será necesario escoger una de las dos operaciones que emplean la sobrecarga, puesto que el funcionamiento que queremos representar

tendría el mismo resultado si lo plasmamos en un diagrama de secuencia.

La diferencia entre ambas es que la primera no filtra por categorías y por lo tanto, llama a una función distinta de IProductDao que no realiza esa comprobación.

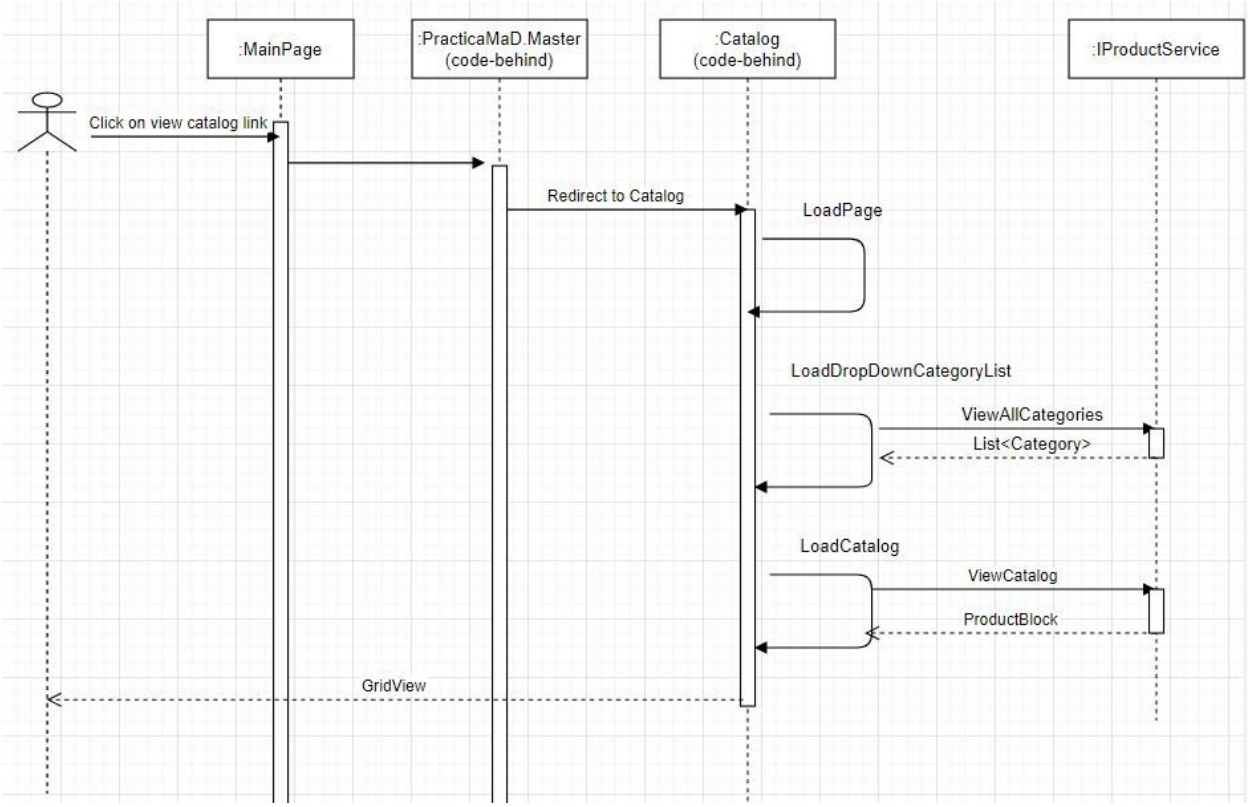


En el diagrama también se muestra el acceso a la caché en caso de que el objeto esté almacenado en ella. Su funcionamiento se explica con más detalle en el apartado 4.3.

2.5 Otros aspectos

Para la parte de internacionalización empleamos los idiomas gallego, castellano e inglés de Estados Unidos .

3. Interfaz gráfica



El diagrama de secuencia muestra el funcionamiento del caso de uso “Ver catálogo”, desde el click de un usuario en el enlace para ver el catálogo, hasta su llamada al modelo, concretamente IProductService.

El procesamiento de la llamada al servicio está simplificada, para verla en detalle, dirigirse al apartado 2.4 de este documento.

4. Parte opcional

4.1 Comentario de productos

- **Diseño**:
    - Para la base de datos, almacenamos todos los comentarios que se hagan sobre un producto, para ello guardamos la referencia al producto comentado, el usuario que lo ha realizado, el texto del comentario y la fecha de realización.

    - Para agrupar las operaciones realizadas sobre los comentarios y las etiquetas (apartado 4.2) hemos creado el servicio CommentService. Este agrupa operaciones como inserción, actualización y eliminación de comentarios, recuperación de comentarios asociados a un producto, de las etiquetas más usadas, de los detalles de un comentario en concreto y una última operación encargada de devolver información acerca de si un usuario ha realizado un comentario sobre un producto.

- **Aclaración**: Intentado imitar a las grandes redes sociales hemos hecho que si posteriormente se modifica cualquier aspecto del comentario no actualizamos la fecha, siempre mostramos la de creación. En un futuro se podría indicar la modificación de diversas formas, por ejemplo, con algún tipo de distintivo visual.

  Para recuperar los comentarios de un producto empleamos paginación.

4.2 Etiquetado de comentarios:

- **Diseño:**
  - Creamos una tabla (“Label”) en la base de datos que tiene como finalidad almacenar todas las etiquetas que actualmente están siendo usadas. En cuanto una etiqueta no está asignada a ningún comentario ésta desaparece de la base de datos.

  - La otra tabla (“Label_comment”), que surge de la relación N-N entre comentarios y etiquetas, tiene como función persistir en memoria todas las relaciones entre una etiqueta y los distintos comentarios.

  - Las operaciones para el etiquetado de comentarios, se encuentra en el servicio CommentService.

  - En cuanto a la nube de etiquetas visible en todas las páginas de la aplicación, la hemos implementado en el code-behind de la página maestra de PracticaMaD.Master. Esta recupera a través de la llamada al servicio ViewMostUsedLabels un número n=5 (puede cambiarse en cualquier momento) de etiquetas más usadas en los distintos comentarios sobre productos.
    El tamaño “x” de cada una de las entradas de la nube de etiquetas irá en base a la función f(x)=Math.Log(timesUsed) + 10.
    Conseguimos así no deformar mucho la nube, ya que una etiqueta que aparece en un comentario tendrá tamaño 10 y si, por ejemplo, aparece un millón de veces, tamaño 23.

4.3 Cacheado de búsquedas

- **Diseño**: hemos decidido implementar el cacheado de búsquedas en la capa de acceso a datos (DAO) ya que de esta forma podemos aprovechar el uso la información almacenada en la caché por distintos servicios que utilicen nuestra capa DAO, pudiendo incluso pertenecer a otra aplicación distinta.

- **Clase utilizada**: Para la implementación del cacheado hemos utilizado la clase MemoryCache, perteneciente al paquete System.Runtime.Caching, ya que nos permite acceder al mismo sistema de almacenamiento de caché desde todas las clases de la capa de acceso a datos.

- **Almacenamiento y recuperación de datos en la caché**: Hemos situado dos funciones en la clase "*Model/Util/CacheUti.cs*":

    - **public static T GetFromCache<T>(string key):** nos permite recuperar cualquier tipo de dato u objecto almacenado en caché simplemente pasando por parámetro el nombre con el que lo hemos guardado y el tipo de dato que debe usar.

    - **public static void AddToCache<T>(string key, T value)**: nos permite almacenar cualquier tipo de dato u objeto pasando como parámetro su nombre y valor.

- **Políticas de la caché:**
    - El tiempo máximo por el que un elemento en caché es válido es de 1 minuto, con esto intentamos tener en cuenta que en una aplicación similar a la nuestra que opere en el mundo real los datos cambiarán muy frecuentemente y será necesario actualizar la caché cada cierto tiempo.

    - Los nombres con los que guardamos los elementos siguen un único patrón basado en el estándar de las peticiones HTTP que nos permite diferenciarlos fácilmente. Un ejemplo del funcionamiento puede ser: FindByProductName?productName=ElHobbit?startIndex=0&count=5.

- **Operaciones en las que se usa el almacenamiento en caché:**

    - Debido a que en esta práctica no se pide tener en cuenta si se han insertado elementos nuevos en la base de datos hemos tenido que limitar el uso de la caché a operaciones que devuelven el mismo conjunto de datos (recuperación de operaciones de búsqueda en el catálogo y recuperación de idiomas).

    - También por decisión de diseño no almacenamos información personal en la memoria caché, esto es, conjuntos de tarjetas de débito/crédito, lista de pedidos realizados, etc. De esta forma evitamos llenar la memoria caché con información que sólamente podrá recuperar un usuario en concreto y, a su vez, aumentamos el nivel de seguridad.

- **Modo de utilización:** en las funciones elegidas para cachear sus resultado hacemos los siguiente:

    - Antes de realizar la consulta en la base de datos comprobamos si tenemos el resultado almacenado en la memoria caché y si está, devolvemos. En caso contrario, realizamos la consulta en la base de datos e introducimos el resultado en la caché.

5. Compilación e instalación

En el proyecto “Web”, añadir la referencia a:
Microsoft.CodeDom.Providers.DotNetCompilerPlatform (roslyn)

6. Problemas conocidos

En principio, no conocemos ningún error en el funcionamiento de la aplicación.