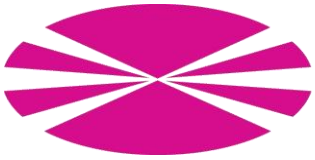


UNIVERSIDADE DA CORUÑA

Memoria final

GEI – MARCOS DEL DESARROLLO,
GRUPO_PRÁCTICAS: MAD06

Integrantes:

Alonso de San Segundo, Lucas
González Fernández, Jorge
Viera Rodríguez, Roxana

Contenido

1. ARQUITECTURA GLOBAL

Dentro del conjunto del proyecto encontramos una serie de paquetes principales dentro de los 3 proyectos (Model, Test y Web) que conforman nuestra aplicación.

Dentro de la capa **Modelo**, en la raíz nos encontramos con los archivos de configuración básicos, como son el *app.config* y el *package.config* donde especificaremos ajustes de configuración sobre la ejecución de esta capa y la gestión de los paquetes que se verán involucrados en ella. Además, en esta capa tenemos los paquetes de los servicios: UserService, ProductService, ShoppingService y CommentService, donde cada uno contiene los siguientes archivos:

- **UserService**:
    - Una carpeta para las excepciones propias del UserService.
    - Un archivo para encriptar las contraseñas (PasswordEncrypter) que guardamos en la carpeta Util.
    - LoginResult.
    - UserDetails.
    - IUserService (interfaz del servicio de Usuario).
    - UserService (implementación del servicio Usuario).

- **ProductService**:
    - Una carpeta para las excepciones propias del ProductService
    - ProductBlock: clase creada para obtener los productos de forma paginada.
    - ProductSummary: clase que utilizamos para mostrarle los atributos que creemos oportunos al usuario tras realizar una búsqueda.
    - IProductService (interfaz del servicio de Producto).
    - ProductService (implementación del servicio de Producto).

- **ShoppingService**:
    - Una carpeta para las excepciones propias del ShoppingService.
    - ShoppingCartItem: clase creada para representar los elementos que pertenece al ShoppingCart.
    - ShoppingCart: clase creada para representar el carrito de la compra, el cual gestionaremos en memoria.
    - OrderBlock: clase creada para obtener los pedidos de forma paginada.
    - OrderDetails: clase creada para poder obtener únicamente los detalles sobre el pedido que queremos mostrarle al usuario.
    - ProductDetailsOrderLine: clase creada para obtener únicamente los detalles sobre las líneas de pedido que queremos mostrarle al usuario.
    - IShoppingService (interfaz del servicio de Shopping).
    - ShoppingService (implementación del servicio de Shopping).

- **CommentService**:
    - CommentBlock: clase creada para obtener los comentarios de forma paginada.
    - CommentDetails: clase creada para poder obtener únicamente los detalles sobre un comentario que queremos mostrarle al usuario.
    - TagDetails: clase creada para poder obtener cuantas veces se ha utilizado un Tag.
    - ICommentService (interfaz del servicio de Comment).
    - CommentService (implementación del servicio de Comment).

Además, también tenemos una carpeta DAO para cada clase: BookDao, CategoryDao, CommentDao, CreditCardDao, MovieDao, OrderDao, OrderLineDao, ProductDao, TagDao y UserDao. Cada carpeta de DAO contiene dos archivos una de interfaz (e.g. ICategoryDao) y otro de implementación (e.g. CategoryDaoEntityFramework).

Para finalizar, en esta capa encontramos los scripts SQL (creación de la base de datos y creación de tablas) y el EDMX, que crearán la base de datos y las clases de forma automática respecto a los valores introducidos en la base de datos.

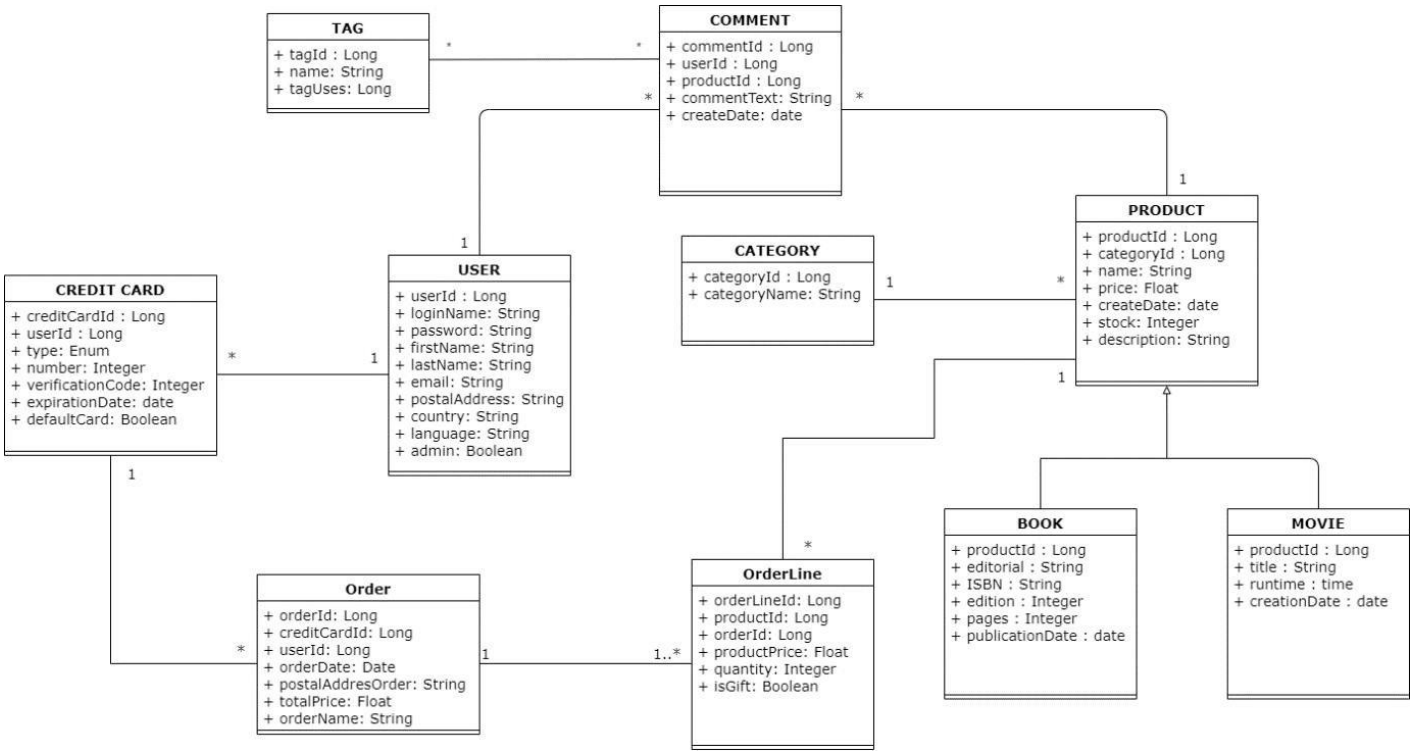
Para la capa de **Test** nos encontramos una estructura similar a la del Modelo, teniendo un paquete con el script SQL necesario para ejecutar los Test, y con los archivos de configuración como *app.config* y *package.config*. Encontraremos varios archivos **.cs** referentes a las distintas pruebas de cada uno de los servicios y los archivos de configuración de las mismas. También tenemos una clase *TestManager.cs* que configura y llena el kernel Ninject para poder realizar las pruebas posteriormente.

Por último, en la capa **Web** tenemos también un archivo de configuración *Web.config* junto a *Global.asax* necesario para la correcta ejecución de la aplicación. También tendremos el *MyMasterPage.master* que funcionará como página maestra para el resto de las páginas que podemos encontrar en la carpeta *Pages*. Junto a esta carpeta tenemos carpetas con archivos *Resources*, necesarios para las traducciones, y la carpeta HTTP utilizada para la gestión de la sesión del usuario, gestión de las peticiones hacia la capa modelo, control del idioma de la sesión, y gestión de las cookies de la web. También se incluye en la carpeta HTTP un fichero *IoCManagerNinject.cs* que lo que hace es configurar y llenar el kernel Ninject para poder utilizar los métodos de los servicios posteriormente.

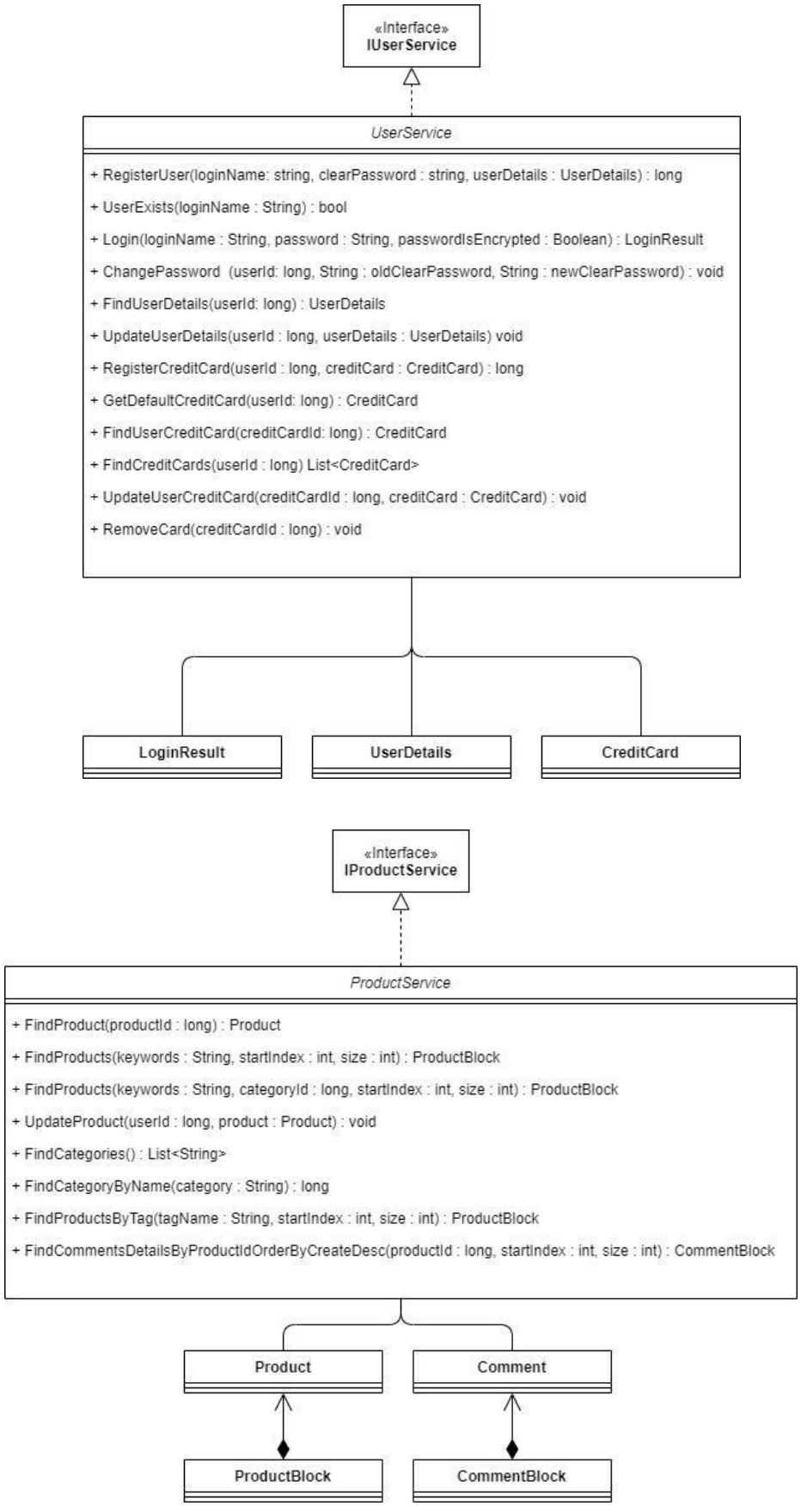
2. MODELO

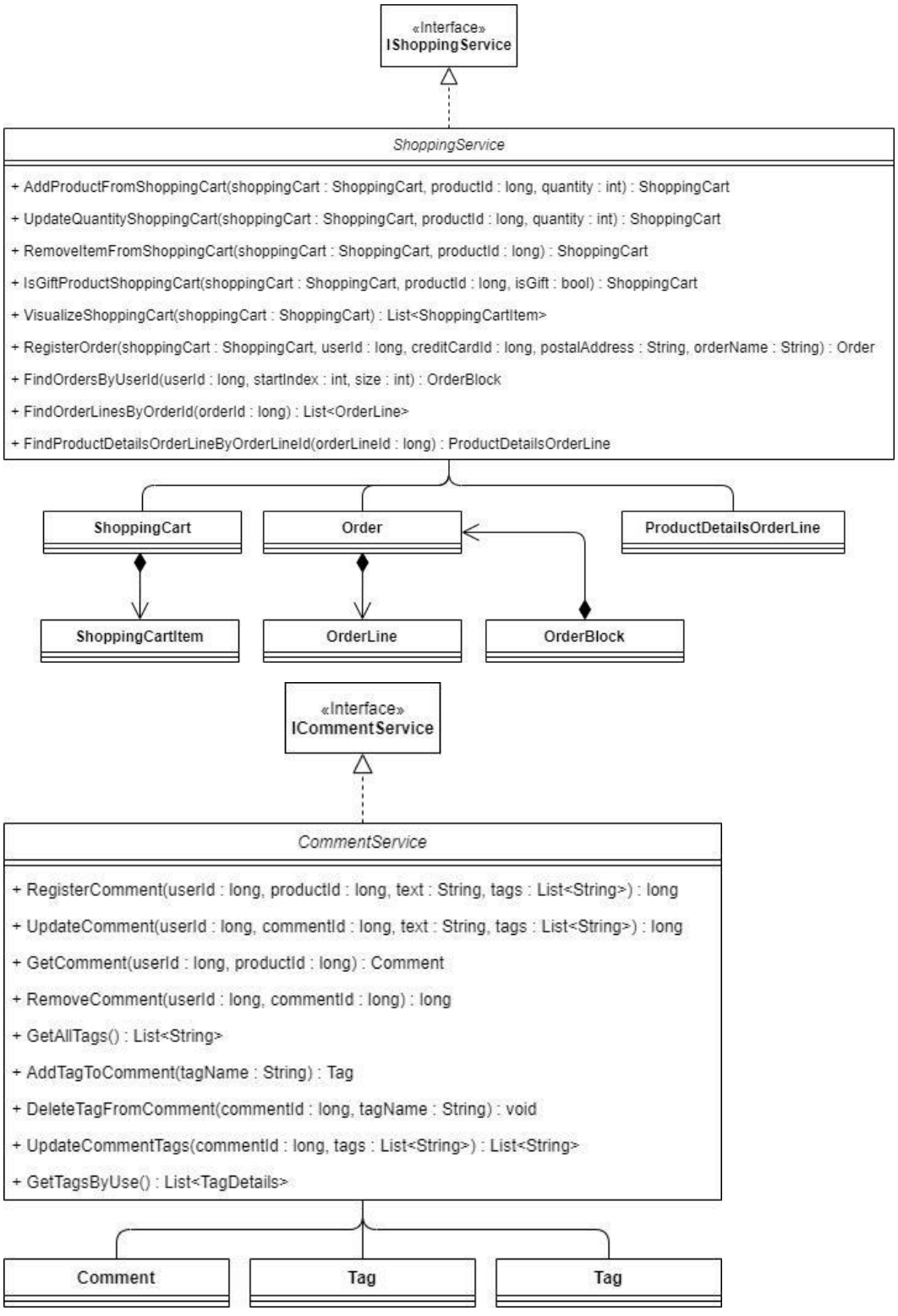
2.1. Clases persistentes

A través del siguiente diagrama podemos observar las clases persistentes del proyecto, así como las relaciones que hay entre ellas. Podemos ver como de la clase *Product* heredan tanto la clase *Book* como *Movie,* destacar que nos decantamos por la opción de *'Tabla por Tipo'* por lo que tenemos dicha estructura en esa parte del diagrama.

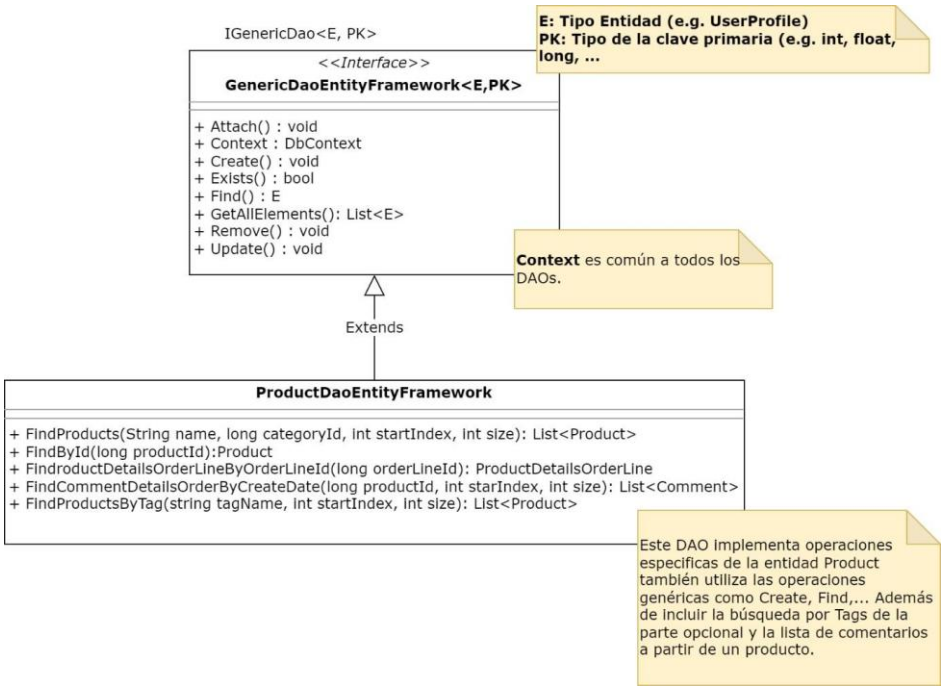


2.2. Interfaces de los servicios ofrecidos por el modelo

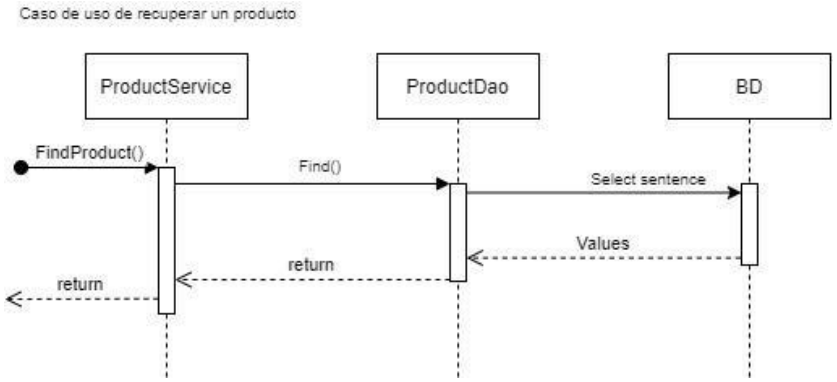



«Interface»
IShoppingService
ShoppingService
+ AddProductFromShoppingCart(shoppingCart : ShoppingCart, productId : long, quantity : int) : ShoppingCart
+ UpdateQuantityShoppingCart(shoppingCart : ShoppingCart, productId : long, quantity : int) : ShoppingCart
+ RemoveItemFromShoppingCart(shoppingCart : ShoppingCart, productId : long) : ShoppingCart
+ IsGiftProductShoppingCart(shoppingCart : ShoppingCart, productId : long, isGift : bool) : ShoppingCart
+ VisualizeShoppingCart(shoppingCart : ShoppingCart) : List<ShoppingCartItem>
+ RegisterOrder(shoppingCart : ShoppingCart, userId : long, creditCardId : long, postalAddress : String, orderName : String) : Order
+ FindOrdersByUserId(userId : long, startIndex : int, size : int) : OrderBlock
+ FindOrderLinesByOrderId(orderId : long) : List<OrderLine>
+ FindProductDetailsOrderLineByOrderLineId(orderLineId : long) : ProductDetailsOrderLine
ShoppingCart
Order
ProductDetailsOrderLine
ShoppingCartItem
OrderLine
OrderBlock
«Interface»
ICommentService
CommentService
+ RegisterComment(userId : long, productId : long, text : String, tags : List<String>) : long
+ UpdateComment(userId : long, commentId : long, text : String, tags : List<String>) : long
+ GetComment(userId : long, productId : long) : Comment
+ RemoveComment(userId : long, commentId : long) : long
+ GetAllTags() : List<String>
+ AddTagToComment(tagName : String) : Tag
+ DeleteTagFromComment(commentId : long, tagName : String) : void
+ UpdateCommentTags(commentId : long, tags : List<String>) : List<String>
+ GetTagsByUse() : List<TagDetails>
Comment
Tag
Tag

2.3. Diseño de un DAO



2.4. Diseño de un servicio del modelo



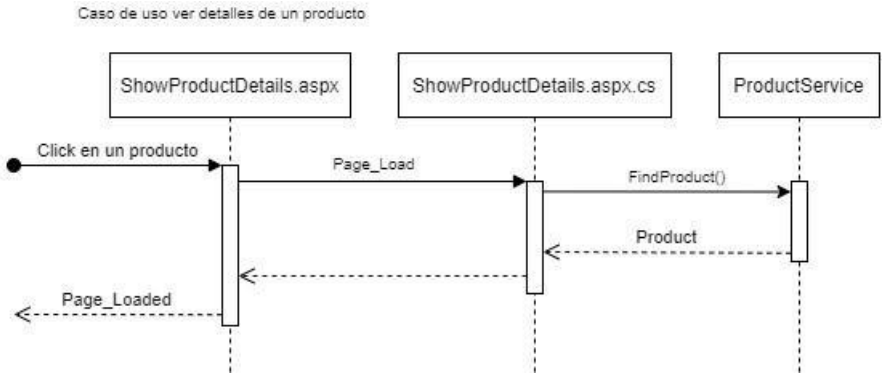
2.5. Otros aspectos.

Es importante destacar sobre el carrito (*ShoppingCart*) y sus elementos (*ShoppingCartItems*) que no lo guardamos a nivel de BBDD, sino que simplemente los gestionamos en memoria.

Hemos hecho uso de clases del estilo *X-Details* (e.g. *OrderDetails*) las cuales utilizamos como vehículo entre el Modelo y Web, para únicamente hacerle llegar al usuario aquello que creemos conveniente.

También mencionar que hemos implementado clases del estilo *X-Block* (e.g. *ProductBlock*) que nos permiten convertir una lista de elementos en bloques paginados, siéndonos esto útil para poder visualizar elementos de forma paginada en la parte web (e.g. en la búsqueda de productos).

3. INTERFAZ GRÁFICA



4. PARTE OPCIONAL

4.1. Comentario de productos

Es importante destacar que la parte de los comentarios de un producto (crear, editar, eliminar y ver) solamente tienen aparición en la página de los detalles de un producto.

En el caso de que un usuario no autenticado trate de registrar un comentario, se le redirigirá a la página de autenticación. Una vez haya iniciado sesión se le redirigirá a la página de detalles del producto que estaba anteriormente, conservando el texto del comentario en el caso de haberlo.

Un usuario solamente puede tener un comentario por producto, para ello hemos habilitado un *textbox* y un botón para registrar un comentario en el caso de que dicho usuario no comentara anteriormente.

En el caso de que dicho usuario comentara con anterioridad sobre dicho producto lo que hemos implementado es que le aparezca el mismo *textbox* con el contenido de su comentario junto a dos botones, los cuales le permiten al usuario actualizar o eliminar el comentario.

Un usuario independientemente de que esté *logueado* o no podrá ver los comentarios de un producto, los cuales aparecerán en la parte derecha de la página en la que se muestran los detalles de dicho producto.

4.2. Etiquetado de comentarios

Los etiquetados de los comentarios en cuanto a creación solamente tienen aparición en la pantalla de detalles de un producto. En esta página también podemos observar los tag's que tiene asociados un producto.

Aunque no estemos autenticados podremos tratar de añadir diferentes tag's, que de hecho aparecerán de forma visual en dicha página, pero el etiquetado de comentarios se llevará a cabo realmente cuando clickemos en el botón relativo a los comentarios, ya sea registrar, actualizar o eliminar. Por lo que, en el caso de no estar autenticados se nos mandará a la página de autenticación y tras haber iniciado sesión se nos redigirá a la página que estábamos anteriormente.

En la lista de comentarios que aparece en la pantalla de ver el producto en detalle podemos observar en el lado derecho que además del texto de los comentarios aparecen los tags asociados a los comentarios.

Finalmente, debajo de la barra de menú principal de la aplicación aparecerá una nube de tag's, si hay tag's creados. Si hacemos *click* en uno de esos tag's hará un filtrado de búsqueda que mostrará todos los productos que tengan ese tag asociado mediante los comentarios de dicho producto.

5. COMPILACIÓN E INSTALACIÓN DE LA APLICACIÓN

Para la correcta puesta en funcionamiento de la práctica, debemos de seguir unos sencillos pasos; teniendo en consideración que se dispone del entorno correctamente configurado (SQL Server Express y Visual Studio).

Primero deberemos compilar la solución entera para cargar todos los componentes, necesarios para el correcto funcionamiento de la aplicación. Después, para preparar el entorno sobre el que trabajar, deberemos ejecutar desde la capa modelo, los scripts SQL que se almacenan en la carpeta SQL de la capa correspondiente, primero creando la Base de Datos y después ejecutando el script de creación de tablas. En nuestro caso debido a un error en la creación de la base de datos original, tenemos que lanzar solo un script de base de datos: sqlServerCreateDatabaseAndTables.sql.

Adicionalmente, con fin de comprobar los test de la aplicación, tendremos que crear la base de datos de pruebas, cuyo script se encuentra en la carpeta Scripts de Test, ejecutándola. Podremos ejecutar los test desde el explorador de pruebas proporcionado por el Visual Studio.

Cabe resaltar, que las bases de datos empleadas a lo largo de la ejecución de la aplicación poseen un *path* de creación, que puede ser modificado antes de ejecutar para localizar la base de datos a la hora de crearla. En nuestro caso, el path de creación es: `'C:\SourceCode\DataBase\'`

Para la ejecución a nivel web, tendremos que acceder a la carpeta ***Pages*** desde el proyecto Web, la cual contiene todas las páginas ***.aspx*** del proyecto. De ahí haremos *click* derecho sobre el archivo ***Index.aspx*** y seleccionamos "Ver en explorador" para ejecutar la aplicación.

6. PROBLEMAS CONOCIDOS

Uno de los problemas más habituales que nos ha sucedido durante el desarrollo de la práctica está relacionado con los DAO's a la hora de hacer uso de sus funcionalidades por defecto, tales que Create, Update y Remove. Dicho error dice:

"The operation failed: The relationship could not be changed because one or more of the foreign-key properties is non-nullable. When a change is made to a relationship, the related foreign-key property is set to a null value. If the foreign-key does not support null values, a new relationship must be defined, the foreign-key property must be assigned another non-null value, or the unrelated object must be deleted."

En resumen, este error nos transmite que una o varias de las claves foráneas que es o son *"not null"* se está estableciendo a *null*. Para intentar solucionar dicho error hemos depurado para ver si realmente le llegaba algún valor a *null*, y eso es algo que no sucede. También hemos estado revisando los scripts de la BBDD y en ellos no hemos encontrado nada raro.

Finalmente, después de hacer varias comprobaciones hemos probado a copiar el script de la base de datos de la capa de test en la capa modelo, cambiándole el nombre a la base de datos, necesitando para la creación e inicialización de la base de datos de nuestra aplicación, un único script con el nombre: sqlServerCreateDatabaseAndTables.sql

Otro problema que nos surgió después de corregir el error que se producía por la base de datos, fue un error que se producía en dos casos diferentes pero que seguían los mismos pasos a la hora de probarlos y se producía el mismo error: el primero, fue al tener dos tarjetas de crédito, modificar la tarjeta que tenemos *por defecto*, e intentar eliminar la tarjeta que no es la de *por defecto*; y el otro caso es en los comentarios, después de modificar un comentario ya creado, e intentar eliminarlo se producía el error.

"Es.Udc.DotNet.ModelUtil.Exceptions.InternalErrorException: Adding a relationship with an entity which is in the Deleted state is not allowed"

Tras intentar depurar el código para intentar corregir el error, lo único que hemos podido observar es que se produce en ambos casos al intentar llamar al método *Remove()* del DAO.

Otro pequeño bug que tenemos es al añadir productos al carrito, que si estabas en una página de la lista de productos distinta a la página principal (p.ej. la página 3 de la lista de productos), vuelve a la página principal en vez de a la página que estabas antes.