E-COMMERCE

Marcos de Desarrollo

Autor Ricardo Sánchez Arias (ricardo.sanchez1@udc.es)

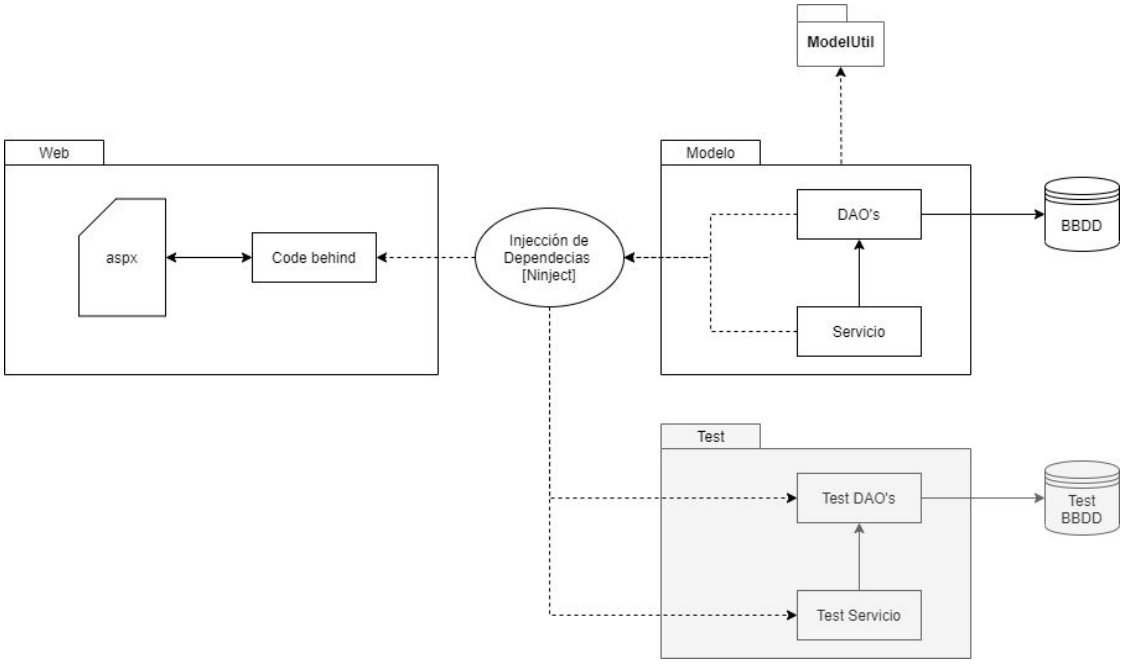
Autor Adrián García Vázquez (adrian.gvazquez@udc.es)

Autor Pablo Hermida Mourelle (pablo.hermidam@udc.es)

Repositorio https://github.com/ricardos6/mad2021

Fecha 09/01/2021

1. Arquitectura global



Diferenciamos entre los 3 bloques(proyectos en la solución) que tenemos. La parte Web está escrita con la tecnología ASP.NET que se conecta con la capa Modelo gracias al inyector de de dependencias de Ninject.

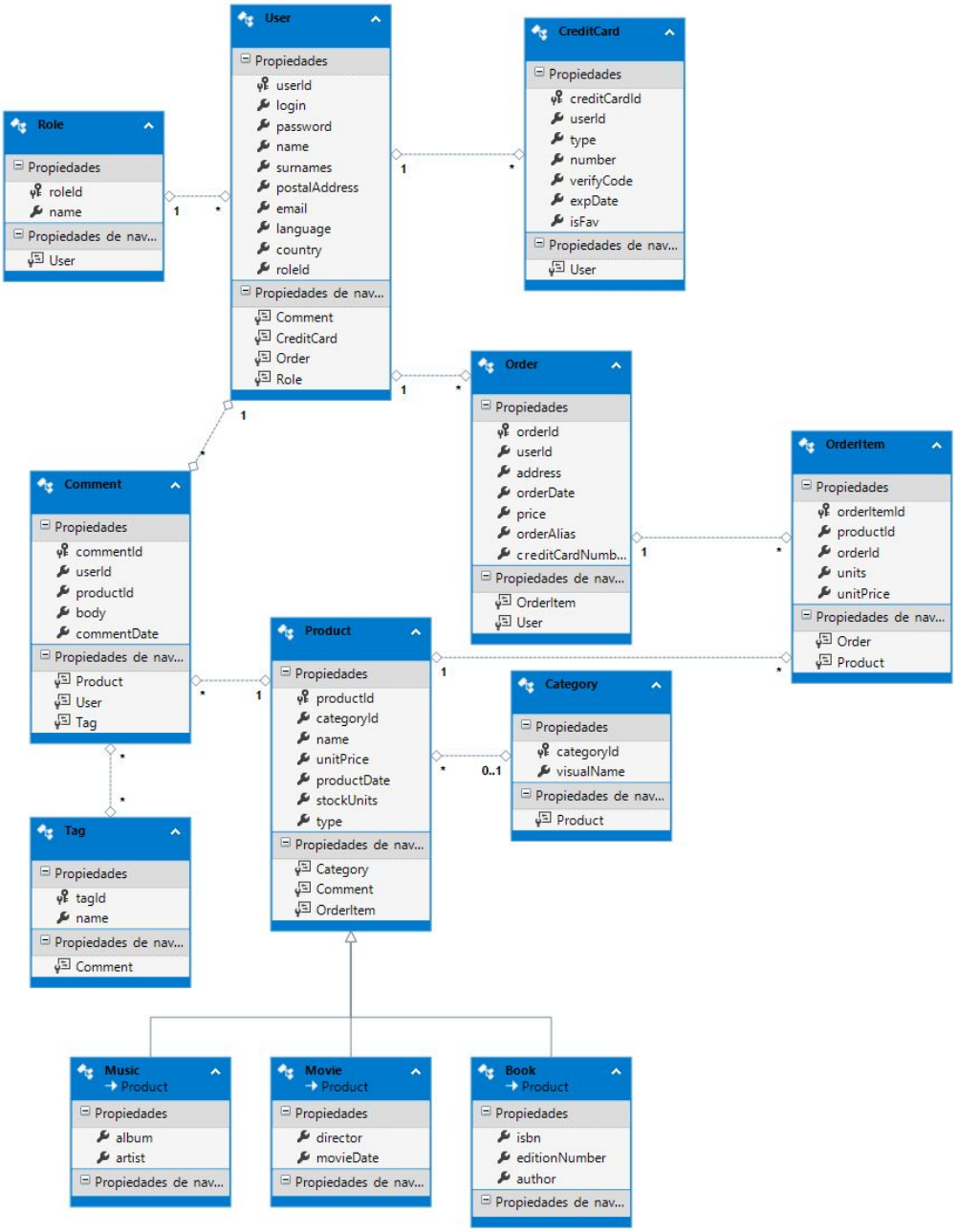
Tanto el Modelo como el Test han sido implementados con C#. En el proyecto Modelo aislamos la implementación de la base de datos con los DAOS, que hacen uso de ModelUtil para generar los métodos necesarios para acceder a la base de datos.

En el proyecto Test tenemos todos los test necesarios para la capa modelo. De forma general, solo testeamos el uso correcto de los métodos. Además tenemos configurada una base de datos de test para no interferir con los datos reales.

2. Modelo

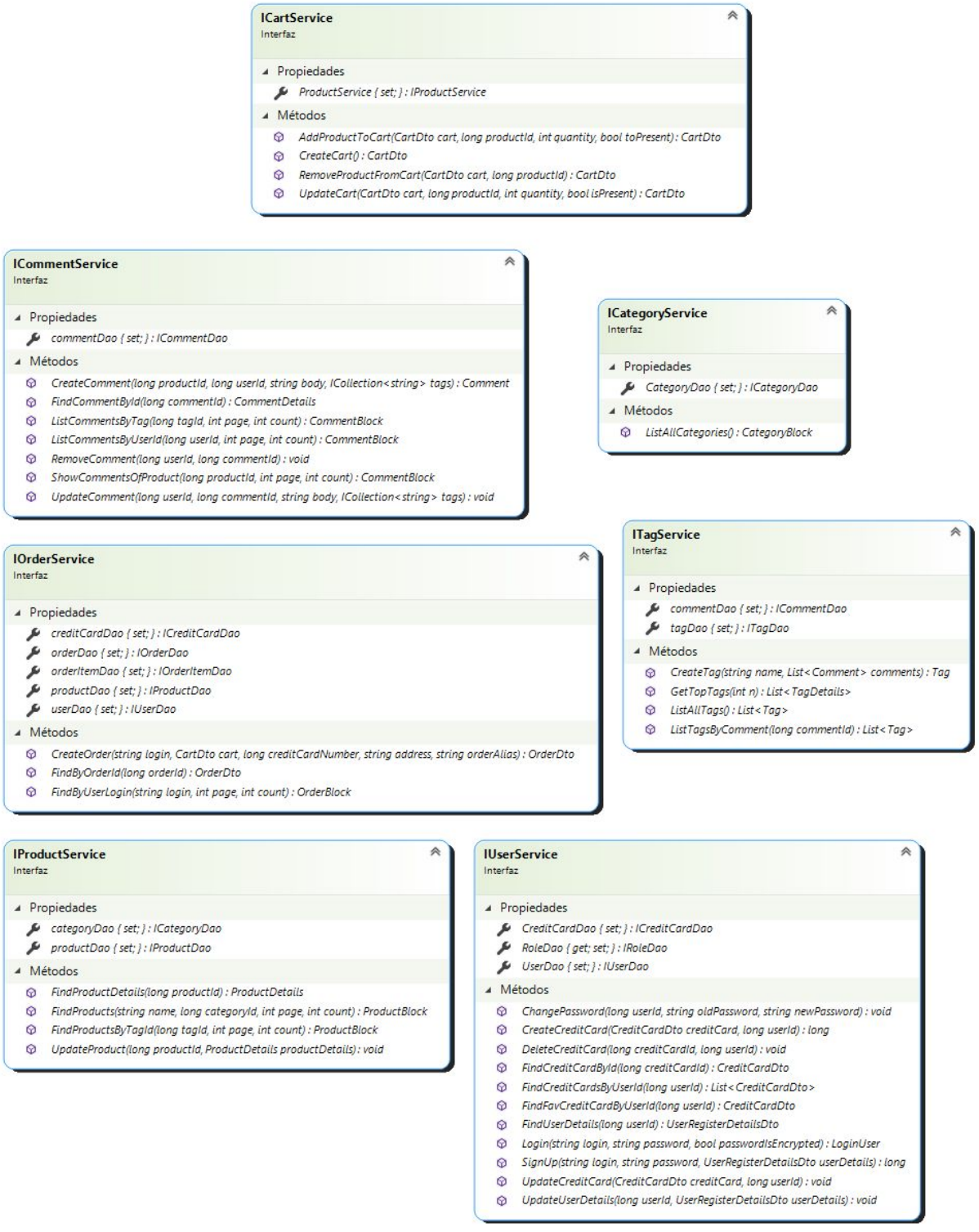
2.1 Clases persistentes

Este es el diseño de base de datos que hemos utilizado en la aplicación. Destacamos la herencia entre producto y sus especificaciones (música, película y libro), esta herencia se denomina TPT(Tabla Por Tabla), cada tabla hija contiene los campos propios, el ID de la tabla padre y no contienen ID propio dado que tienen una clave foránea apuntando al padre.



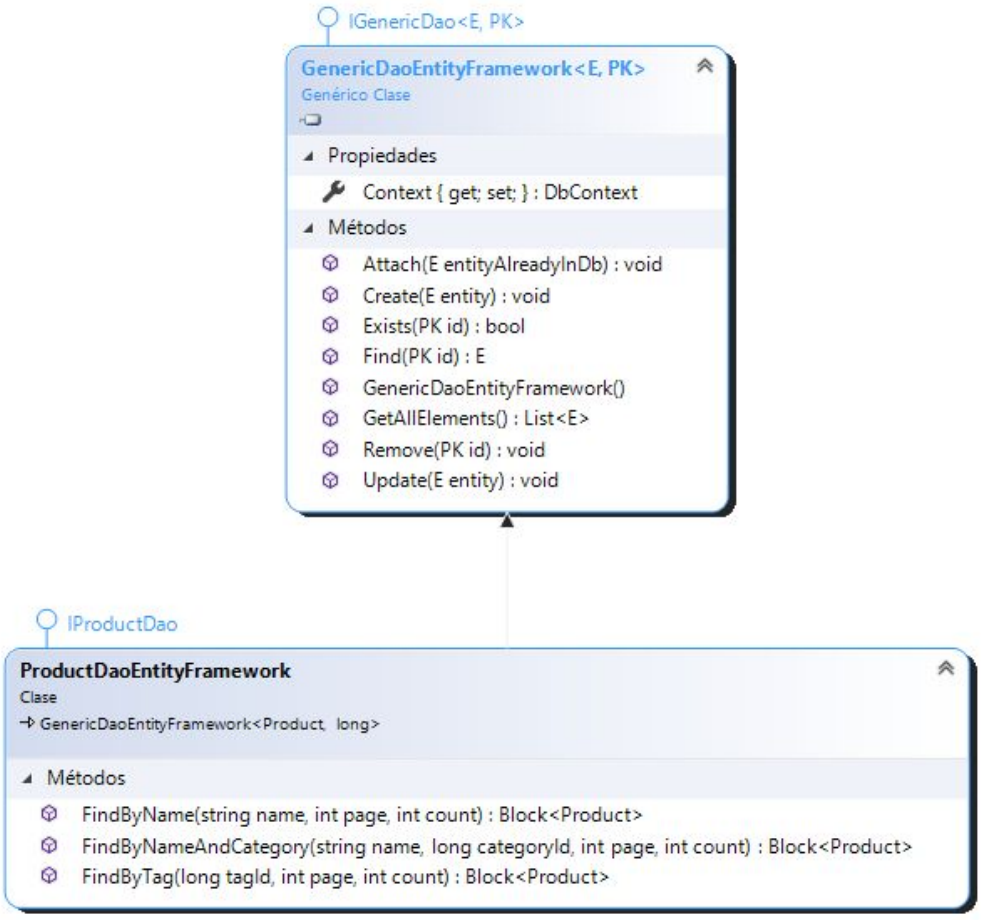
2.2 Interfaces de los servicios ofrecidos por el modelo

Hemos separado los casos de uso en siete servicios, uno por cada entidad principal. Cabe destacar que el único servicio que no accede a la base de datos sería el servicio de Carrito, dado que no tenemos una entidad y lo gestionamos a nivel de memoria.



2.3 Diseño de un DAO

Este DAO hereda de GenericDaoEntityFramework para obtener un CRUD básico, este clase se encuentra en el ModeUtil que usamos como módulo auxiliar.



2.4 Diseño de un servicio del modelo

Diagrama de clases del servicio de **OrderService.**

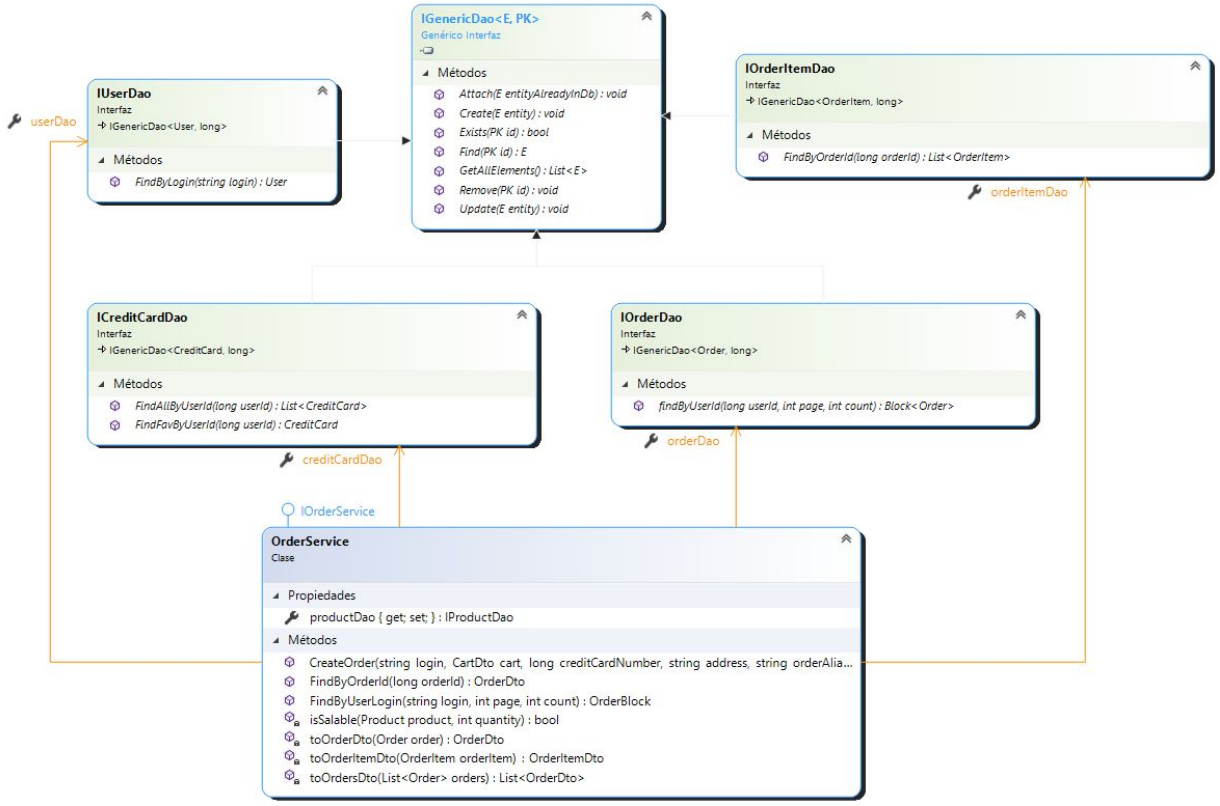
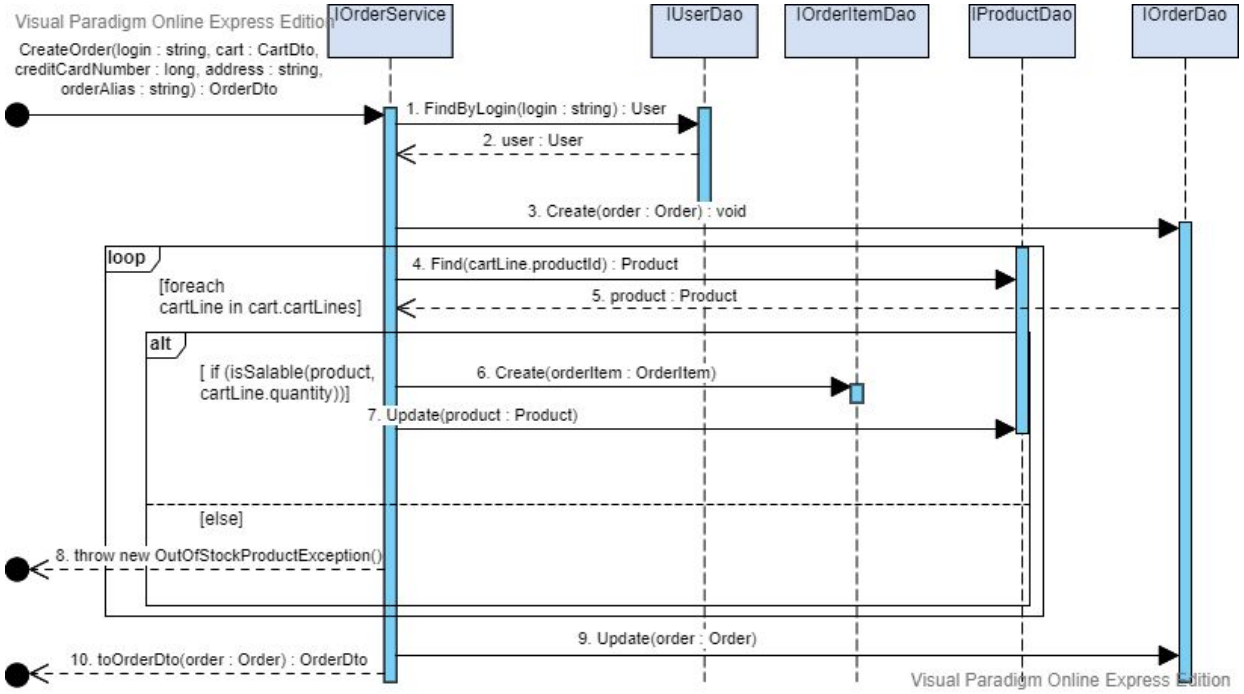


Diagrama de secuencia de caso de uso: **Crear un pedido**.



2.5 Otros aspectos

NInject

Para la correcta comunicación de las clases que interactúan entre sí en la capa modelo, hacemos uso de la herramienta **NInject**. Esta herramienta es un contenedor de **inversión de control** el cual nos permite **inyectar las dependencias** de cada módulo. De esta forma podemos hacer que nuestro código sea más **mantenible, extensible y testeable**.

Contextos en los DAO's

A la hora de hacer peticiones a la base de datos, optamos por dos estrategias distintas en función del tipo de petición que realizamos:

Contexto de vida corta: Para aquellas consultas que nos devuelven datos con los que no vamos a interactuar(búsqueda de productos, listas las tarjetas de crédito, etc.)

Contexto de vida larga: El resto de consultas que suelen estar vinculadas a la edición y creación de datos.

3. Interfaz Gráfica

NInject

Al igual que en la capa modelo, para **inyectar las dependencias** de dicha capa en la interfaz gráfica, hacemos uso de la herramienta NInject. De esta forma ambas capas se implementan de forma totalmente **independiente**.

Autenticación

Utilizamos **autenticación basada en formularios**. Mediante el **SessionManager** gestionamos las sesiones del usuario y el carrito. En caso de que las cookies estén activadas, tenemos el **CookieManager** para almacenar durante más tiempo la sesión del usuario mediante cookies.

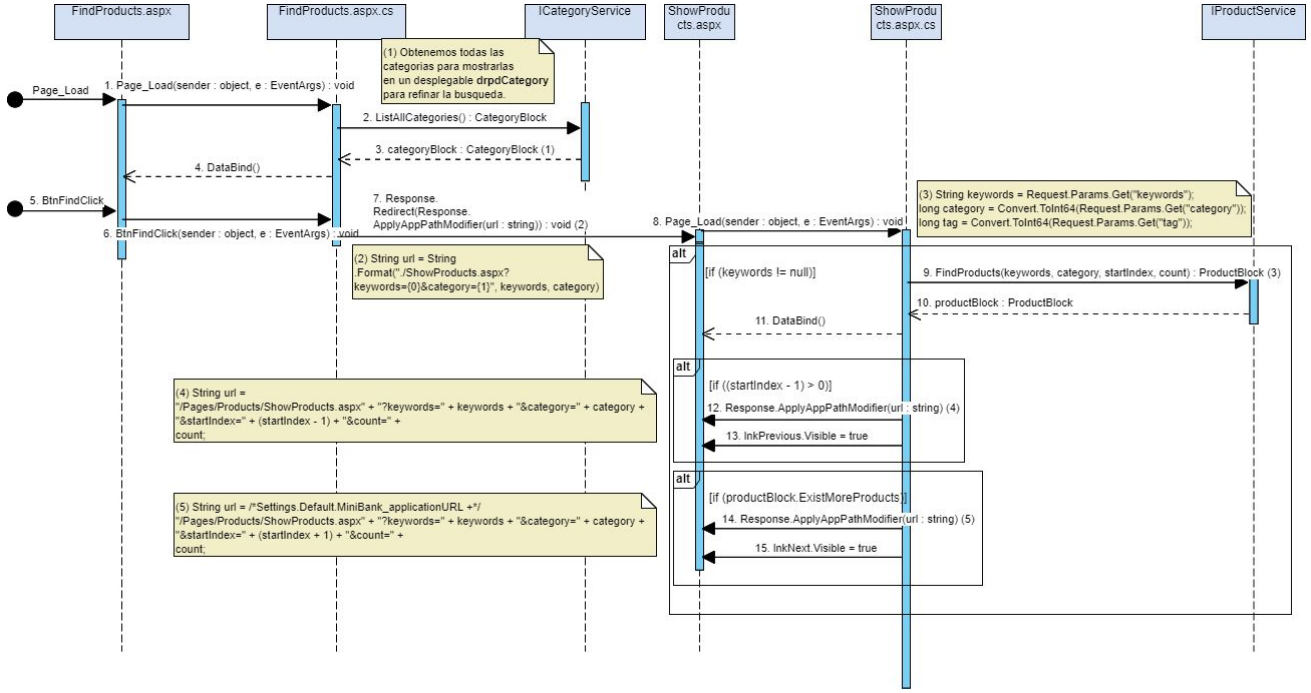
Autorización

En la configuración del proyecto Web, localizamos el archivo **Web.config** en la raíz. En dicho archivo tenemos las configuraciones básicas del proyecto y es donde configuramos la autorización. Optamos por la **política de acceso permitiva**, denegando así el acceso únicamente a las páginas que requieren de la autenticación de un usuario.

Carrito

Para la implementación del carrito, hemos utilizado el **SessionManager** guardando el **carrito en sesión**. Nunca se guarda en base de datos y desaparece al mismo tiempo que la sesión del usuario.

Caso de uso: **Buscar productos por keywords y categoría:**



4. Partes Opcionales

4.1. Comentario de productos

Creamos una entidad Comment (Ver diagrama) relacionándola N-1 con User y N-1 con Product. Implementamos un Dao para la persistencia.

Añadimos un servicio CommentService con métodos para:

- **Crear comentario:** CreateComment(long productId, long userId, string body, ICollection<string> tags) : Comment

- **Buscar comentario por id:** FindCommentById(long commentId) : CommentDetails

- **Actualizar información de un comentario:** UpdateComment(long userId, long commentId, string body, ICollection<string> tags) : void

- **Borrar comentario:** RemoveComment(long userId, long commentId) : void

- **Listar comentarios de un producto:** ShowCommentsOfProduct(long productId, int page, int count) : CommentBlock

- **Listar comentarios realizados por un usuario:** ListCommentsByUserId(long userId, int page, int count) : CommentBlock

En la parte web, creamos una serie de paginas .aspx para realizar los casos de uso:

- **AddComment.aspx:** Formulario para rellenar la información necesaria para crear un comentario sobre un producto. Se accede a través de un botón “Comment” que aparece al visualizar los detalles de un producto.

- **EditComment.aspx:** Formulario donde aparece la información actual del comentario, pudiendo editar cualquiera de sus valores. Solo se pueden editar tus propios comentarios. Se accede a través de un enlace “Edit” que aparece al visualizar a un lado de cada comentario mostrado en las listas.

- **RemoveComment.aspx:** Al cargar esta página se borra el comentario que tiene el ID que se pasa como parámetro en la url. Solo se pueden borrar tus propios comentarios. Se accede a través de un enlace “Edit” que aparece al visualizar a un lado de cada comentario mostrado en las listas.

- **ShowCommentsByProductId.aspx:** Esta página muestra los comentarios de un producto. Aquí aparecerán las opciones de borrado y editado de comentarios que estén realizados por el usuario loggeado. Se accede a través de un botón “Show Comments” que aparece al visualizar los detalles de un producto.

- **ShowCommentsByUserLogin.aspx**: Esta página muestra los comentarios realizados por el usuario loggeado. También muestra la opción de borrado y editado. Se accede a través de un enlace en la barra superior “My Comments”.

4.2. Etiquetado de comentarios

Creamos una entidad Tag (Ver diagrama) relacionándola N-N con Comment. Implementamos un Dao para la persistencia.

Añadimos un servicio TagService con métodos para:

Obtener los n tags más usados: List<TagDetails> GetTopTags(int n) : List<TagDetails>

Crear un Tag relacionado con varios comentarios (solo se usa para Test): CreateTag(string name, List<Comment> comments) : Tag

Listar todos los tags disponibles: ListAllTags() : List<Tag>

Listar todos los tags relacionados con un comentario: ListTagsByComment(long commentId) : List<Tag>

A la hora de **crear y editar un Comment**, añadimos como parámetro una lista de Tag para poder especificar qué tags relacionamos con el comentario que estamos creando/editando.

En la parte web, añadimos a las páginas de de **AddComment.aspx** y **EditComment.aspx**, una entrada de formulario para añadir tags que ya han sido creados para otros comentarios y otro formulario para añadir comentarios nuevos, en esta última entrada se insertaran los comentarios deseados separados con “,”.

4.3. Cacheado de búsquedas

Hemos creado una clase SearchCache que maneja la caché de la aplicación en la carpeta Util del Modelo, en esta clase contiene un ObjectCache, que nos permite almacenar CacheItems a nivel de caché de aplicación. Cada CacheItem contiene una clave y un valor, en la clave guardamos el nombre de cada llamada al Dao que se ha realizado con sus parámetros y en el valor guardamos lo que devolvió dicha consulta. Cabe destacar que utilizamos el cacheado de búsquedas en las cuales los elementos están paginados.

En esta clase SearchCache también hemos añadido una cola que contiene las claves de los 5 últimos CacheItems para asegurarnos de solo cachear las últimas 5 peticiones que se han realizado a la base de datos.

La clase también tiene 3 métodos para poder usar dicha caché, estos métodos serían crear elemento en la caché, obtener elemento de la caché y limpiar toda la caché. Estos dos primeros métodos se usarán a nivel de DAO, cada vez que se llame a un DAO este comprobará si existe la consulta en la caché; si existe devuelve el valor almacenado en la caché, si no existe ejecuta la consulta a base de datos mientras guarda el valor en la caché para futuras peticiones. El método de borrar caché sólo se utiliza a nivel de test para evitar que se utilicen llamadas cacheadas.

5. Compilación e instalación de la aplicación

En las propiedades de configuración de la solución (Accediendo Botón Derecho en la solución > Propiedades) cambiamos la configuración de Debug a Release y Aceptamos. Compilamos la solución y en la consola de salida nos dirá donde almacenó los 3 .dll, que son los archivos que necesitaremos para utilizar la aplicación. El Modelo lo podremos usar en otros proyectos como usamos el ModelUtil, y la parte Web la podremos desplegar en un IIS Express.

6. Problemas conocidos

- Muy ocasionalmente, se producía un error en la inicialización de los test por problemas de concurrencia en la base de datos. Cabe destacar que últimamente no se produjo más: *System.Data.Entity.Infrastructure.DbUpdateConcurrencyException: Store update, insert, or delete statement affected an unexpected number of rows (0)*
- Tal y como está indicado en el enunciado, no contemplamos que ninguna de las entradas de la caché se actualiza después de la edición de las entidades correspondientes.
- Después de editar una entidad, si queremos borrarla nos mostraba este error: *Adding a relationship with an entity which is in the Deleted state is not allowed*. Para solucionarlo borramos el método GetHashCode generado automáticamente en el edmx.