Memoria práctica MAD

Graña Domínguez, Marcos

Dorrego Díaz, Diego

Figueiras Martínez, José Antonio

Índice

Introducción

Se presenta la documentación correspondiente a la práctica de MAD

1. Arquitectura Global

La arquitectura global se compone de los siguientes elementos:

- Dentro del proyecto Model:

1. Los paquetes correspondientes a los servicios.

   De manera general, estos paquetes contienen por una parte la interfaz del servicio correspondiente y por otro la implementación de dicha interfaz. A continuación, se enumeran los paquetes y se añaden comentarios en aquellos que presentan contenido adicional:

   1.1. CardServiceImp
   1.1.1. DTOs: se incluye un DTO para introducir los datos de la tarjeta.

   1.2. ClientServiceImp
   1.2.1. DTOs: se incluyen dos DTO: uno para representar a un cliente y otro para los datos de login.
   1.2.2. Exceptions: se incluye una excepción específica para este servicio.
   1.2.3. Útil: se incluyen métodos de utilidad para encriptar y desencriptar.

   1.3. CommentServiceImp
   1.3.1. DTOs: se incluyen los DTO para representar un comentario y la paginación de comentarios
   1.3.2. Exceptions: excepciones relacionadas con el servicio de comentario.

   1.4. LabelServiceImp
   1.4.1. DTOs: se incluye un DTO que representa una etiqueta sin la lista de comentarios.

   1.5. ProductServiceImp
   1.5.1. DTOs: se incluye un DTO con los datos fundamentales de un producto. También se proporciona en este paquete un mapper para realizar la conversión.
   1.5.2. Cache: se incluye una clase para gestionar la caché y otra con funciones útiles.

   1.6. SaleServiceImp
   1.6.1. DTOs: se incluyen los DTO necesarios para las entidades Sale y ShoppingCart.
   1.6.2. Exceptions: se incluyen excepciones específicas para este servicio.

2. El paquete DAO

   El paquete DAO contiene los paquetes DAO de todas las entidades del modelo. Dentro de cada uno de ellos se encuentra la interfaz del DAO y su correspondiente implementación.

3. El paquete SQL

   Dentro del paquete SQL se hallan dos scripts SQL: uno para la creación de las tablas y otro para la creación de la base de datos.

- Dentro del proyecto de pruebas Test

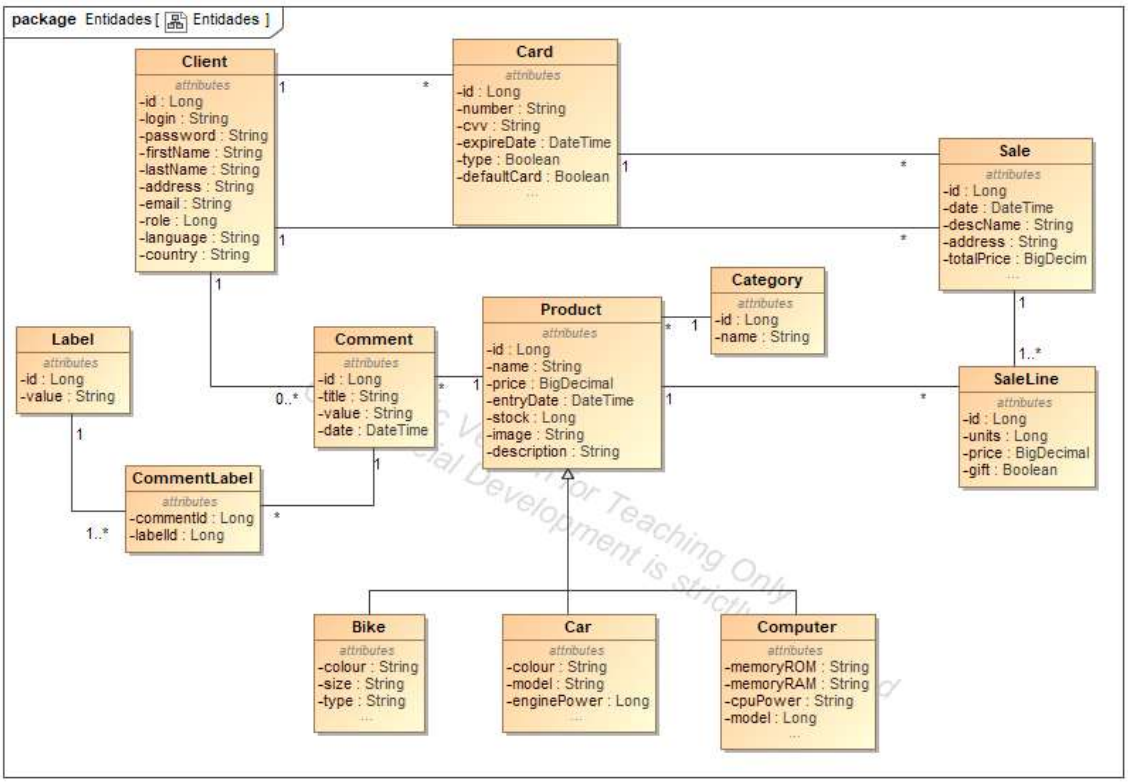
    1. Por cada uno de los servicios del modelo se crea un paquete para realizar las pruebas de dicho servicio. En estos paquetes aparece una única clase con los métodos de prueba correspondientes.
    2. Paquete sql: Script de creación de tablas y base de datos para los tests.

- Dentro del proyecto Web

1. App_GlobalResources: Contiene entradas para la internacionalización para toda la App.
2. App_LocalResources: Contiene entradas para la internacionalización de la MasterPage.
3. Css: Contiene el archivo .css que se usa para dar estilo a las páginas de la aplicación.
4. HTTP
    4.1. Session: Contiene utilidades para el manejo de la sesión y cultura del usuario.
    4.2. Util: Contiene utilidades para el manejo de cookies y dependencias de ninject.
    4.3. View: Contiene utilidades de i18n.
5. Pages: Contiene paquetes con las páginas que aparecerán en el navegador, que componen el proyecto web. De manera general, cada uno de estos contendrá su propio App_LocalResources para almacenar las entradas para la internacionalización de las páginas.
6. Amazonia.Master: Pagina plantilla de la aplicación.
7. Global.asax: Maneja eventos generales de la aplicación.

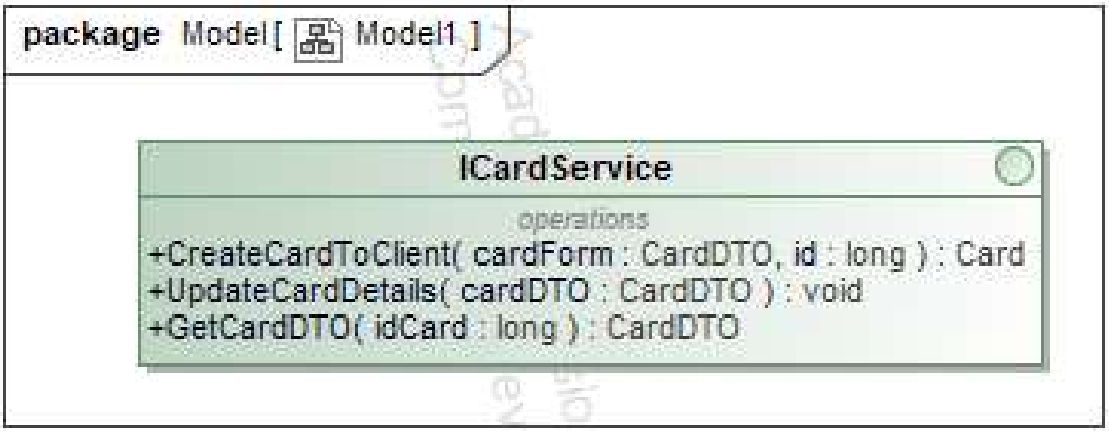
2. Modelo

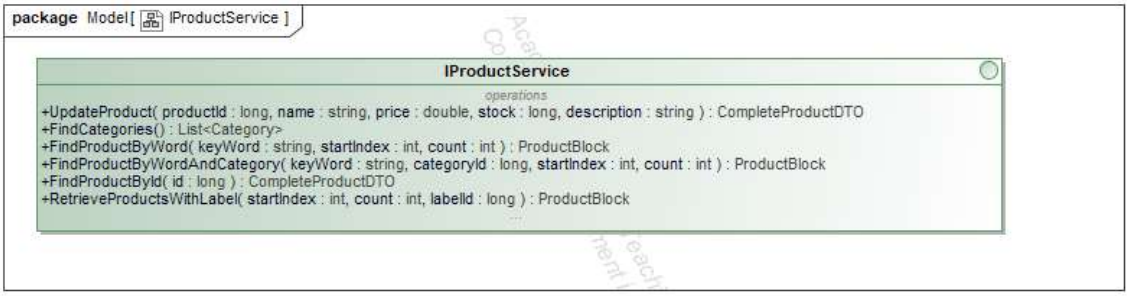
2.1. Clases persistentes



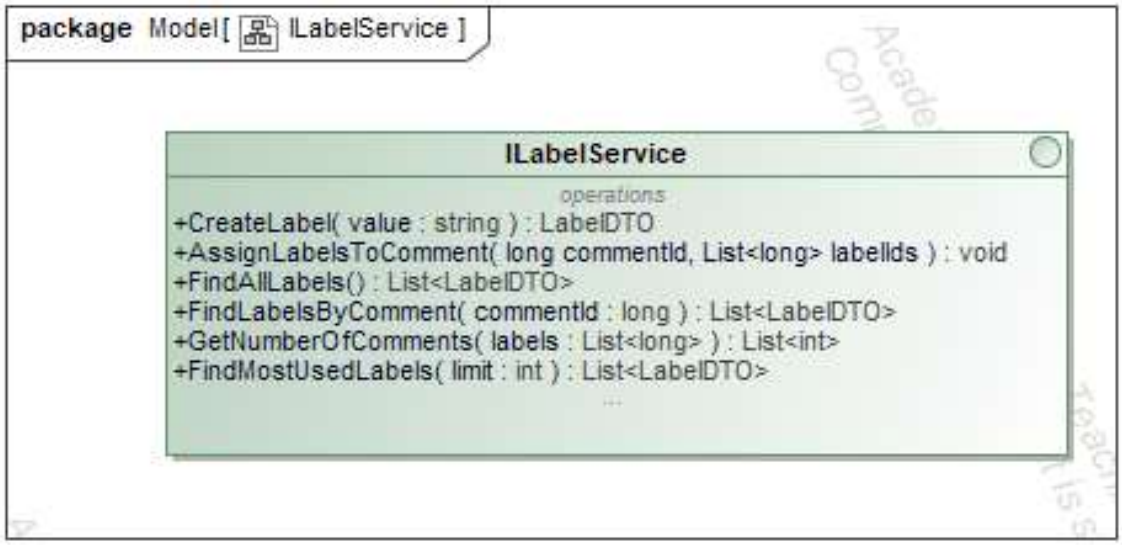
2.2. Interfaces de los servicios ofrecidos por el modelo

Los métodos se sitúan en el servicio de aquella entidad que se considera que tiene una mayor relación con su operativa.



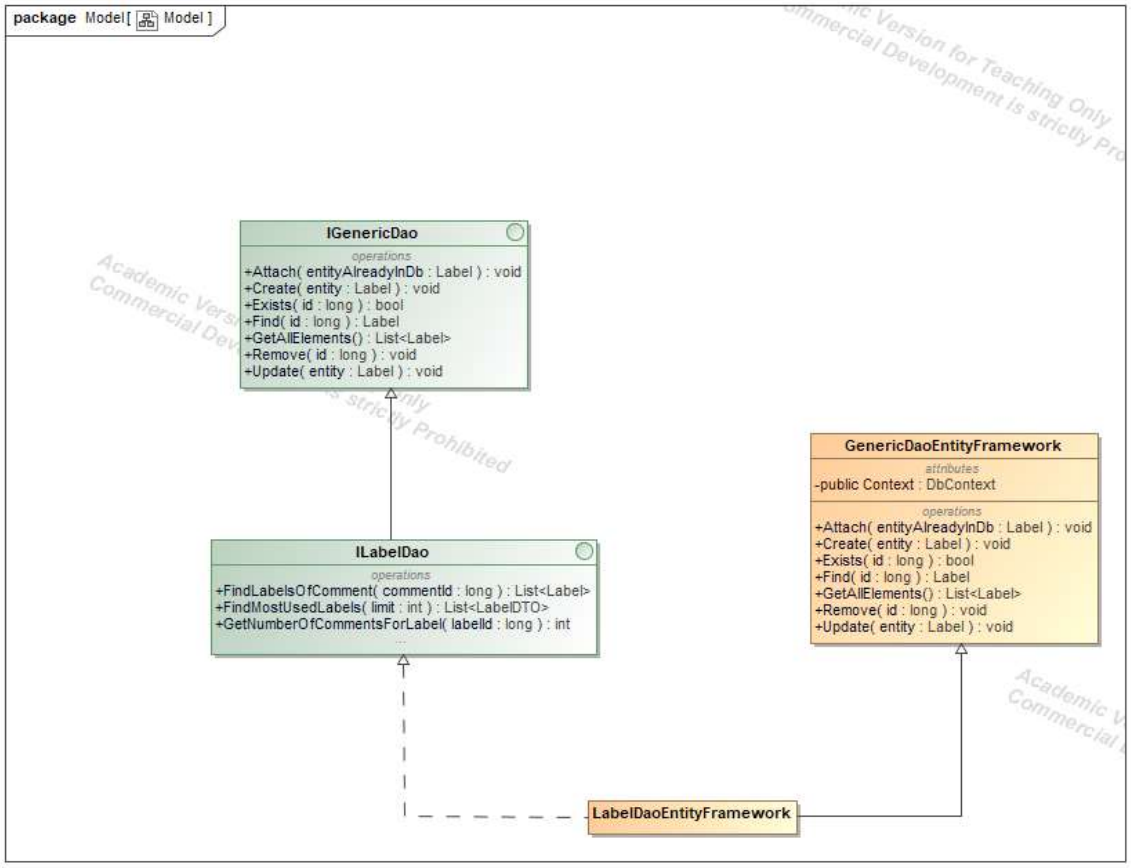



package Model[ICommentService]
ICommentService
operations
+AddComment(title : string, value : string, productId : long, clientId : long) : long
+RemoveComment(commentId : long, clientId : long) : void
+FindCommentsOfProduct(productId : long, startIndex : int, count : int) : CommentBlock
+FindCommentsOfProductAndClient(productId : long, clientId : long) : List<CommentDTO>
+FindCommentsByLabel(labelId : long) : List<Comment>
+ChangeComment(commentId : long, title : string, value : string, clientId : long) : Comment


package Model[ILabelService]
ILabelService
operations
+CreateLabel(value : string) : LabelDTO
+AssignLabelsToComment(long commentId, List<long> labelIds) : void
+FindAllLabels() : List<LabelDTO>
+FindLabelsByComment(commentId : long) : List<LabelDTO>
+GetNumberOfComments(labels : List<long>) : List<int>
+FindMostUsedLabels(limit : int) : List<LabelDTO>
...


package Model[ISaleService]
ISaleService
operations
+AddToShoppingCart(shoppingCart : ShoppingCart, productId : long, units : long, gift : boolean) : ShoppingCart
+DeleteFromShoppingCart(shoppingCart : ShoppingCart, productId : long) : ShoppingCart
+ModifyShoppingCartItem(shoppingCart : ShoppingCart, productId : long, units : long, gift : boolean) : ShoppingCart
+Buy(shoppingCart : ShoppingCart, descName : String, address : String, cardId : long, clientId : long) : long
+ShowClientSaleList(clientId : long, startIndex : int, count : int) : SaleBlock
+ShowSaleLines(saleId : long) : List<SaleLineDTO>
...

2.3. Diseño de un DAO



2.4. Diseño de un servicio del modelo

Diagrama de clases del servicio de etiqueta:

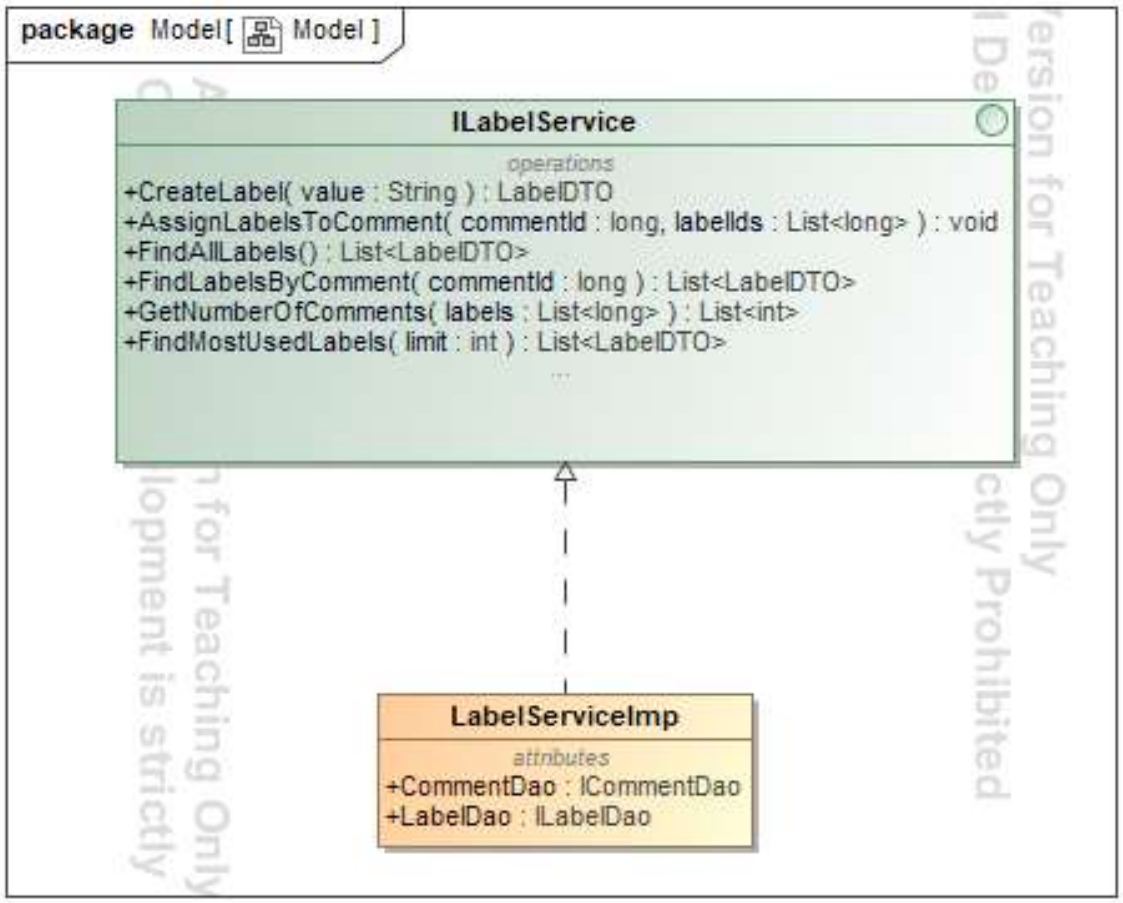
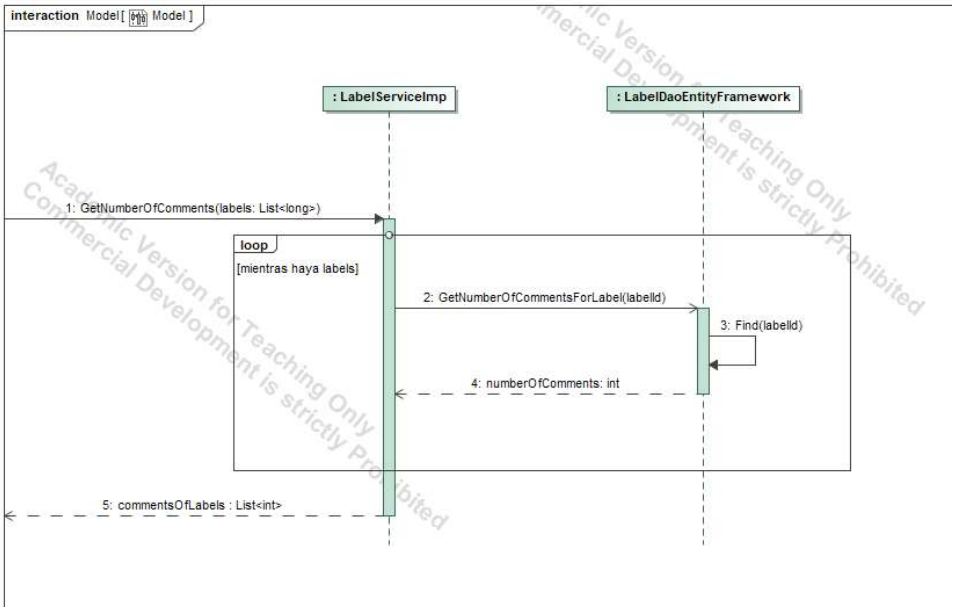


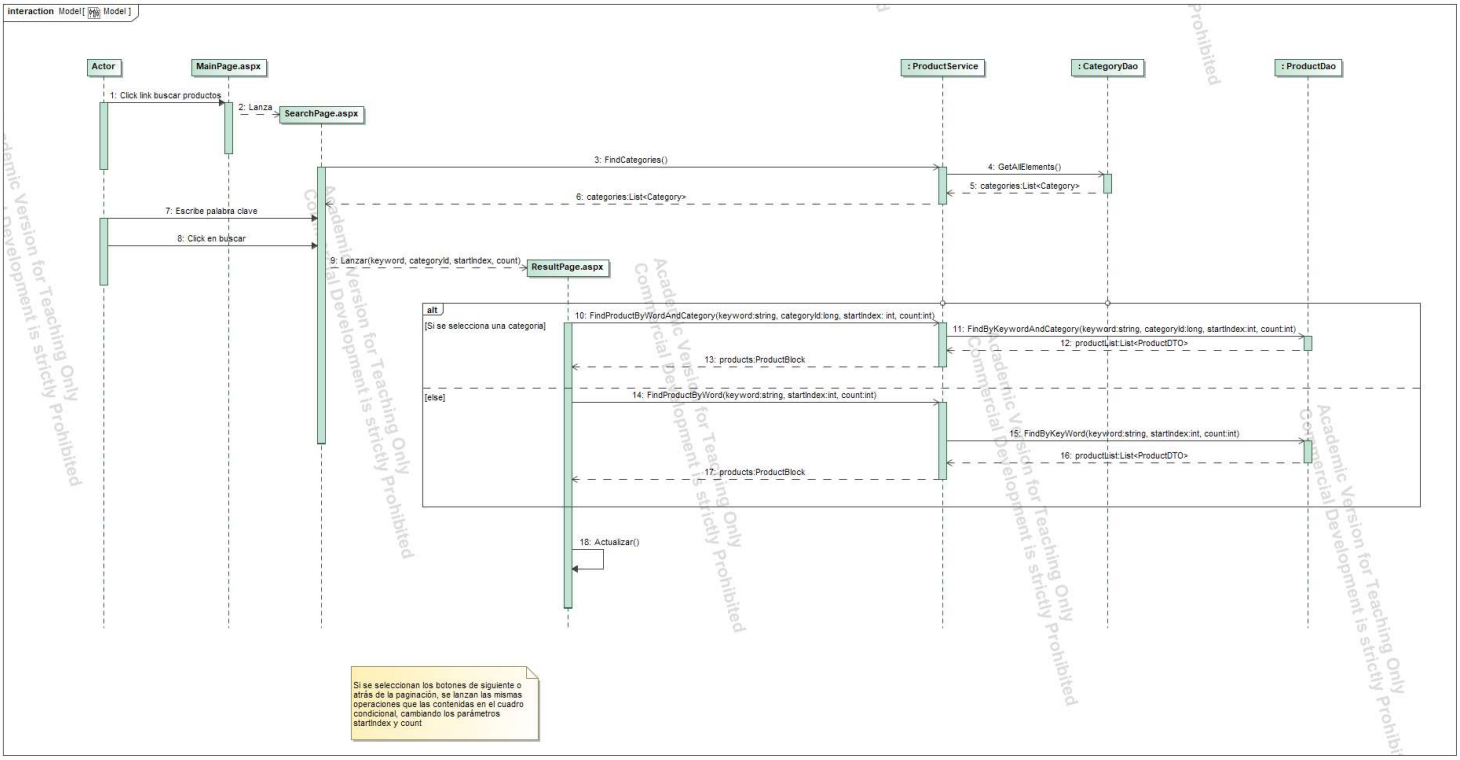
Diagrama de secuencia método recuperar número de comentarios de una lista de etiquetas:



2.5. Otros aspectos

N/A

3. Interfaz Gráfica



- La operación FindCategories lanzada por SearchPage está contenida dentro del Page_Load de su correspondiente code behind.

- Las operaciones de búsqueda tanto con categoría como sin ella están contenidas dentro del Page_Load del code behind de ResultPage.

- Tanto SearchPage cono FindCategories cuando reciben los datos de los servicios actualizan estructuras visuales: en el caso de SearchPage se actualiza un DropDownList con las categorías recuperadas, en el caso de ResultPage se actualiza un GridView con los datos de los productos.

- A mayores, en el caso de ResultPage se establecen los botones de paginación con sus respectivos enlaces y se añaden dos botones para poder ver el detalle de un producto en concreto y otro para poder añadirlo al carrito.

4. Partes Opcionales

4.1. Comentario de productos

Para realizar esta parte, en primer lugar, se añade en la base de datos una nueva tabla "Comment" relacionada con multiplicidad N a 1 tanto con "Product" como con "Client".

Para cumplir con esta parte opcional se crea un nuevo servicio "CommentService" con las siguientes operaciones:

- AddComment: Permite a un usuario añadir comentarios a productos. Asegura que un usuario solo puede comentar una vez en un mismo producto.
- ChangeComment: Permite a un usuario cambiar tanto el título como el contenido de un comentario, siempre y cuando el usuario que lo solicita sea el autor original del comentario.
- DeleteComment: Permite a un usuario eliminar sus propios comentarios.
- FindCommentsOfProduct: Devuelve los comentarios asociados a un producto. Es necesario crear una operación en el DAO para buscar solo los comentarios asociados a un producto en concreto, y a mayores, especificar que estos vengan ordenados por fecha.

En el proyecto web esta funcionalidad se puede acceder desde la página de detalle de un producto: tanto crear un nuevo comentario como modificar uno existente.

4.2. Etiquetado de comentarios

Para realizar esta parte, en primer lugar, se añade en la base de datos una nueva tabla "Label". Como un comentario puede tener varias etiquetas, se crea también una nueva tabla "CommentLabel" para representar esta relación N a M entre "Comment" y "Label".

Para cumplir con esta parte opcional se crea un nuevo servicio "LabelService" con las siguientes operaciones:

- CreateLabel: Permite crear una nueva etiqueta y asignarla a un comentario.
- AssignLabelsToComment: Permite asignar múltiples etiquetas a un comentario.
- FindAllLabels: Recupera todas las etiquetas.

  Los siguientes tres métodos sirven para implementar la nube de etiquetas:

- FindMostUsedLabels: Recupera las N etiquetas más usadas. Serán las que se muestren en la nube de etiquetas.
- FindLabelsByComment: Recupera todas las etiquetas asociadas a un comentario. Cada string de la etiqueta en la nube de etiquetas será un enlace que devuelva los comentarios de dicha etiqueta, por eso es necesario este método.
- GetNumberOfComments: Recupera el número de comentarios con el que están asociadas múltiples etiquetas. Sirve para establecer el tamaño de las etiquetas en función del número de comentarios con el que están asignadas.

La nube de etiquetas se implementa en la MasterPage para que esté visible en todas las páginas de la aplicación. Los métodos de créate label, assign label to comments se pueden acceder desde la página de añadir un comentario y modificar un comentario.

4.3. Cacheado de búsquedas

En esta parte se ha implementado el funcionamiento de una memoria caché que almacena el resultado de las cinco últimas búsquedas que se han realizado para proporcionar una respuesta más rápida en caso de que se repitan estas búsquedas almacenadas.

En primer lugar, se añaden dos clases dentro del paquete **ProductService**:

- Una clase **CacheContainer** que contiene la memoria caché en la que se almacenaran los resultados de las búsquedas. También contiene un atributo **List<String>** que se usa para llevar el control de las 5 entradas actualmente almacenadas en caché. Este atributo es necesario debido a la ausencia de un método en MemoryCache que nos permita consultar las entradas almacenadas. Sus métodos se describirán a continuación. Esta clase se implementa de forma que solo exista una instancia de la misma, para que todas las instancias de IProductService accedan a la misma caché.
- Una clase **CacheUtils** que, en nuestro caso, contiene un método **FormatSearch** que en función de los parámetros de búsqueda crea un string formateado que será la clave en la entrada de la cache. Esto se usa tanto para almacenar como para consultar si una búsqueda se encuentra en la cache.

La clase **CacheContainer** tiene los siguientes métodos:

- **GetCacheContainer:** Metodo que devuelve la instancia única de la clase.
- **AddToCache**: En este método se encuentra la lógica de añadir un elemento a la caché. Si la memoria contiene menos de 5 elementos, se añade directamente. En caso contrario, se emplea la lista para saber cuál es la entrada más antigua y eliminar esta para añadir la nueva entrada.
- **GetEntrie**: Recibe un string de una búsqueda formateada y devuelve el contenido de la caché para esta.
- **IsInCache**: Recibe un string de una búsqueda formateada y devuelve un booleano que representa si esa entrada se encuentra almacenada en la caché o no.

Por último, en ambas funciones de búsqueda, se añade una condición que comprueba si la búsqueda se encuentra en la caché, en cuyo caso ya se devuelve el resultado. Si no se encuentra en la caché, se realiza la búsqueda normal, y antes de devolverla se llama a AddToCache para introducir esta nueva búsqueda en la memoria.

5. Compilación e instalación

N/A

6. Problemas conocidos

Se tiene constancia de que no se muestran los atributos específicos de producto en la interfaz.