

# Marcos de Desarrollo

Diego García López y David Sineiro Barreiro

## 1. Arquitectura global por paquetes

La estructura del proyecto se caracteriza por tres grandes bloques desde los cuales sale todo lo demás. Estos bloques son Model, que incluye toda la lógica de negocio e interacción con base de datos, Test, que incluye las pruebas de la capa anterior, y Web, que incluye todo lo relacionado con el funcionamiento de la capa web. Aparte de los archivos de configuración, propiedades y referencias de cada uno de estos proyectos, lo que tendremos dentro será lo siguiente:

- En el caso de Model, nos encontraremos con una serie de paquetes. Estos paquetes son de tres tipos principalmente: DAO, DTO y Service:
  - Los paquetes DAO incluyen la interfaz y la implementación concreta de una clase, ya sea Tag, Comment, Product...
  - Los paquetes Service incluyen la interfaz, implementación y otras clases necesarias para el funcionamiento correcto de dicho servicio. Hay 4: Valoracion, Tag, User y Product
  - Los paquetes DTO incluyen la implementación de un Data Transfer Object, es decir, un objeto modificado para su manejo más cómodo en la aplicación web. Existen DTOs para las clases Comentario, Producto, Usuario y Valoracion.

A mayores de estos paquetes principales, hay un paquete Sql que incluye los archivos de generación de la base de datos del proyecto, así como el Entity Data Model, generado desde dicha base de datos, que genera las clases con las que hemos trabajado en todo el proyecto.

- En el caso de Test, nos encontraremos con paquetes que incluyen los tests de todos los servicios y algunos de los DAOs. Además, tenemos el archivo de TestManager, donde se inicializa el kernel para permitir la inyección de dependencias a través de Ninject.
- En el caso de Web, además de los archivos de configuración propios de un proyecto web, tenemos los siguientes paquetes:
  - App\_GlobalResources incluye recursos que utilizarán la mayoría de páginas, como por ejemplo textos internacionalizados.
  - Http contiene archivos que se encargan de gestionar los aspectos relacionados con las sesiones y las cookies.
  - Css incluye los archivos de estilo de las páginas
  - Pages incluye las páginas de servicio activas, con una carpeta de Usuario dentro con las páginas relacionadas al usuario.
  - A mayores, existe una carpeta App\_LocalResources en cada nivel del directorio, de tal forma que cada página en dicho nivel pueda acceder a su archivo de recursos. Este archivo contiene los textos internacionalizados.

## 2. Modelo

### 2.1 Clases Persistentes

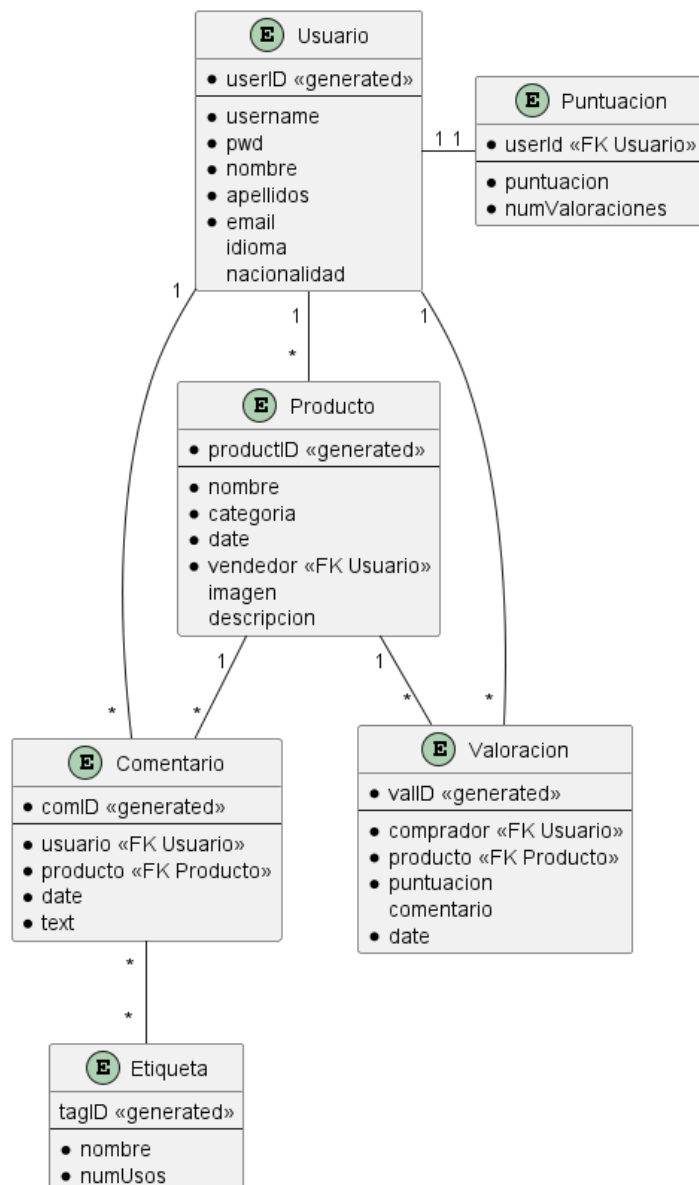
El proyecto incluye una arquitectura de base de datos que consta de las siguientes tablas:

- **Usuario:** incluye la información relacionada con un usuario registrado en la aplicación. Entre sus atributos están el ID único y generado automáticamente por base de datos, el nombre de usuario, nombre, apellidos, contraseña y email. Todos estos atributos son únicos y no nulos. A mayores, se puede introducir el idioma y nacionalidad como atributos extra. Su clave primaria es el identificador autogenerado.
- **Producto:** incluye la información relacionada con un producto concreto. Sus atributos incluyen un ID único generado automáticamente, un nombre, una categoría, una fecha y un vendedor no nulos. Este vendedor es un id que se usa como Foreign key conectada a un usuario que posee el objeto y lo quiere vender. A mayores, se puede incluir una imagen y una descripción, y la clave primaria es el ID autogenerado.
- **Comentario:** Incluye la información relacionada con un comentario. Tiene su propio ID único y generado, así como el ID de un usuario que lo publica, el ID de un producto sobre el que comenta, una fecha y el texto. Todos sus atributos son no nulos. Incluye dos Foreign key, entre el ID de usuario y la tabla usuario y el id de producto y la tabla producto, y su clave primaria es su ID autogenerado.
- **Valoración:** Incluye la información relacionada con una valoración concreta de un producto. Tiene su propio ID, así como el ID del usuario que la realiza, el ID del producto sobre la que realiza, la puntuación indicada como un float, un comentario que puede ser nulo y una fecha. Existen dos foreign key similares a las de comentario que le conectan con el usuario que la realiza y el producto a valorar. Su clave primaria es su ID y tiene una comprobación sobre la puntuación para que tome un valor de 0 a 5.
- **Etiqueta:** La etiqueta incluye su propio identificador generado, así como su nombre y un contador del número de veces que se ha usado. Su clave primaria es su ID.
- **TagCom:** TagCom es una clase débil que sirve como núcleo de conexión entre las etiquetas y los comentarios. Almacena el ID de una etiqueta y el ID de un comentario, de manera que ambos son referencia a cada una de estas tablas. Su clave primaria es la combinación de ambas.
- **Puntuación:** Puntuación es otra clase que sirve para almacenar la puntuación concreta de las valoraciones de un usuario. Toda la información aquí almacenada se podría incluir directamente dentro de usuario, pero nos pareció más visual y ordenado separar estos valores de la clase de Usuario propia. Esta clase incluye el ID de un usuario que actúa de foreign key, así como su puntuación media almacenada como float y el número total de valoraciones realizadas hasta el momento. De esta manera, cuando una nueva valoración entra, se puede recalcular la media y actualizar.

En cuanto a decisiones de implementación a nivel de base de datos a destacar, primeramente tenemos lo explicado relacionado con la clase de Puntuación. No es completamente necesario, pero sí permite mantener el orden en el proyecto.

Por otra parte, como vemos no es necesario que una valoración incluya a qué usuario vendedor se está valorando, ya que al incluir el producto, es este quien incluye la información sobre quién lo vende. Además, en el caso de las etiquetas y la puntuación de un usuario, se decidió almacenar el número de usos, valoraciones y puntuación media como atributo en base de datos en lugar de calcularlo cada vez que sea necesario, ya que consideramos más eficiente la modificación de estos valores ante un nuevo uso de etiqueta o una nueva valoración que su cálculo cada vez que se deba mostrar en pantalla.

Aquí tenemos el diagrama Entidad-Relación de la base de datos:



Comentando brevemente las relaciones de cardinalidad entre las tablas, podemos apreciar que:

- Un usuario puede tener publicados múltiples productos, comentarios y valoraciones. Aún así, cada una de estas solo pertenece a un usuario.
- Un usuario tiene una sola puntuación y dicha puntuación pertenece a un solo usuario.
- Un producto puede tener múltiples comentarios y valoraciones, pero cada comentario y cada valoración solo hacen referencia a un producto.
- Finalmente, una etiqueta puede aparecer en múltiples comentarios, y un comentario puede tener múltiples etiquetas. De esta relación n-n es de donde surge la necesidad de la tabla extra TagCom, que guarda la información de las relaciones entre estas dos tablas.


## 2.2 Interfaces de los servicios ofrecidos por el modelo

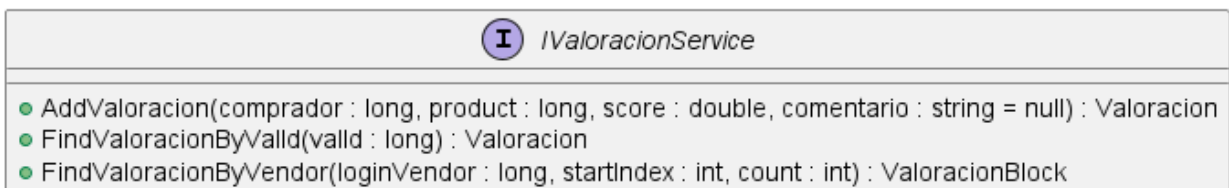
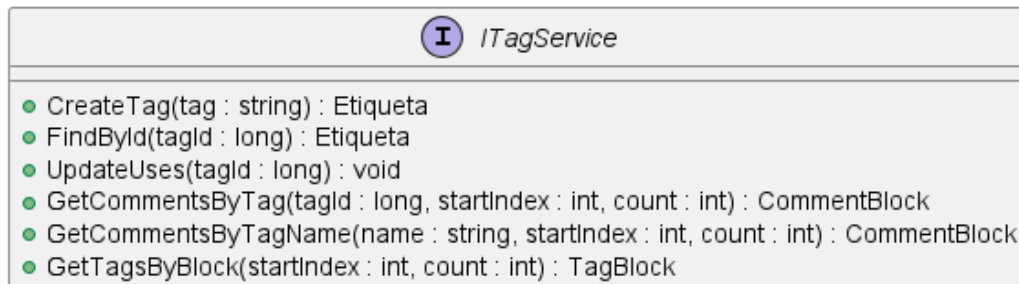
El proyecto incluye un total de cuatro servicios que se dividen de la siguiente manera:

1. UserService: servicio dedicado a operaciones de usuario: registro, identificación, modificación...
2. ValoracionService: servicio dedicado a operaciones relacionadas con la valoración de un producto.
3. ProductService: este servicio incluye operaciones relacionadas con encontrar productos y comentarlos. Se trata del servicio más denso ya que también incluye toda la lógica para crear comentarios, actualizarlos, borrarlos, mostrar los de un producto concreto, etc.
4. TagService: servicio extra dedicado a las operaciones relacionado con etiquetar comentarios.

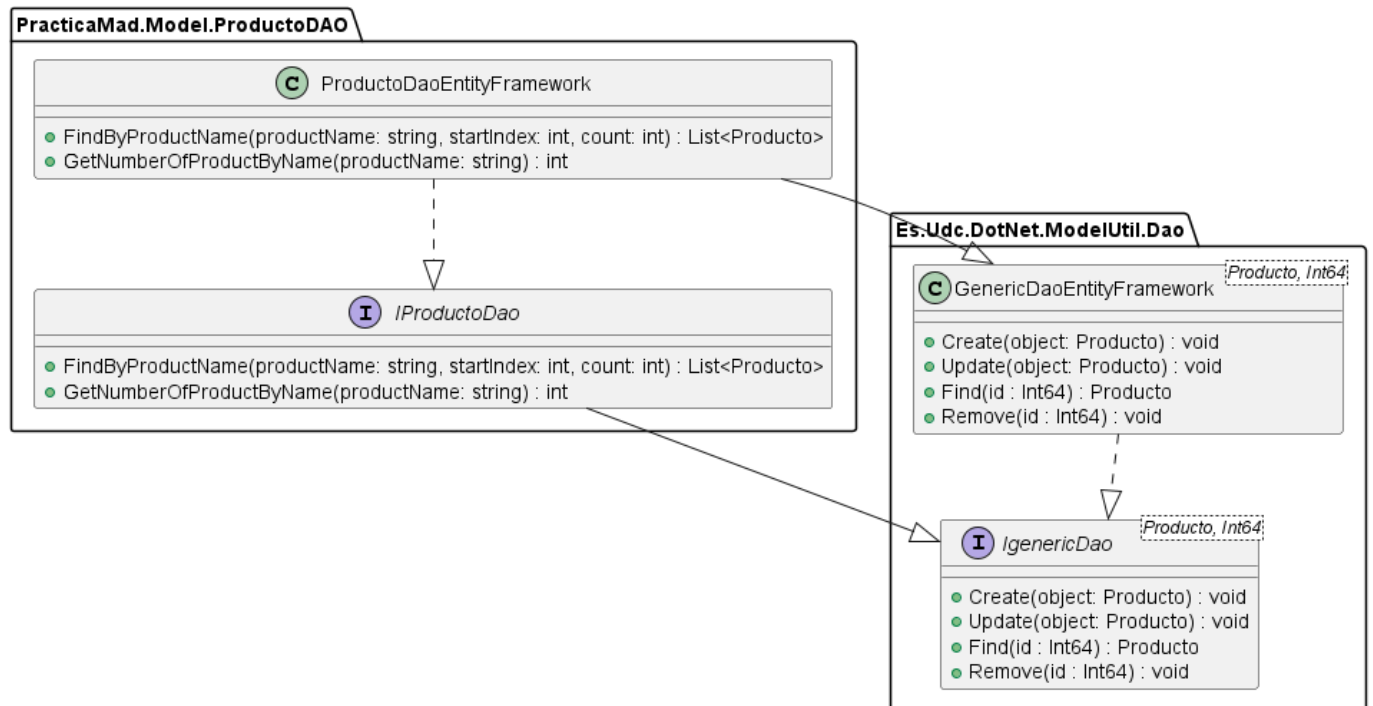
La distribución de las funcionalidades en los servicios es relativamente equitativa, a excepción del productService. Esto se debe a que en un primer momento infravaloramos la densidad de funcionalidad que podría conllevar la búsqueda de productos con toda la infraestructura de comentarios, ya que operaciones como crear producto y borrarlo no existen en el proyecto. Si bien no puede resultar del todo intuitivo a primera vista que las operaciones de comentario formen parte del servicio del producto, creemos que esto es preferible a separar el servicio de producto del de comentario, dejándonos con un servicio de producto demasiado pequeño.

Aquí podemos ver en detalle las interfaces de cada uno de estos servicios con sus métodos públicos, parámetros de entrada y salidas:

 IUserService
<ul style="list-style-type: none"><li>● RegisterUser(username: string, password: string, userProfileDetails: UserProfileDetails) : Usuario</li><li>● UpdateUser(userId : long, userProfileDetails : UserProfileDetails) : void</li><li>● LoginUser(username: string, password : string, passwordIsEncrypted : bool) : LoginResult</li><li>● UpdateUserScore(username : string, score : double) : void</li><li>● Find(userId : long) : UserDTO</li><li>● FindByUsername(username : string) : UserDTO</li><li>● FindUserProfileDetails(userId : long) : UserProfileDetails</li><li>● ChangePassword(userId : long, oldClearPassword : string, newClearPassword : string) : void</li></ul>
 IProductService
<ul style="list-style-type: none"><li>● FindProductByName(productName : string, startIndex : int, count : int) : ProductBlock</li><li>● FindComentario(comid : long) : Comentario</li><li>● CommentProduct(usuario : long, producto : long, text : string, tagNames : HashSet&lt;string&gt; = null) : Comentario</li><li>● UpdateComment(comment : Comentario) : void</li><li>● RemoveComment(commentid : long) : void</li><li>● SeeProductComments(productId : long, startIndex : int, count : int) : CommentBlock</li><li>● Find(productId : long) : ProductoDTO</li><li>● GetNumberOfProductsByName(productname : string) : int</li><li>● GetNumberdOfCommentsByProdId(productId : long) : int</li></ul>



## 2.3 Diseño de un DAO

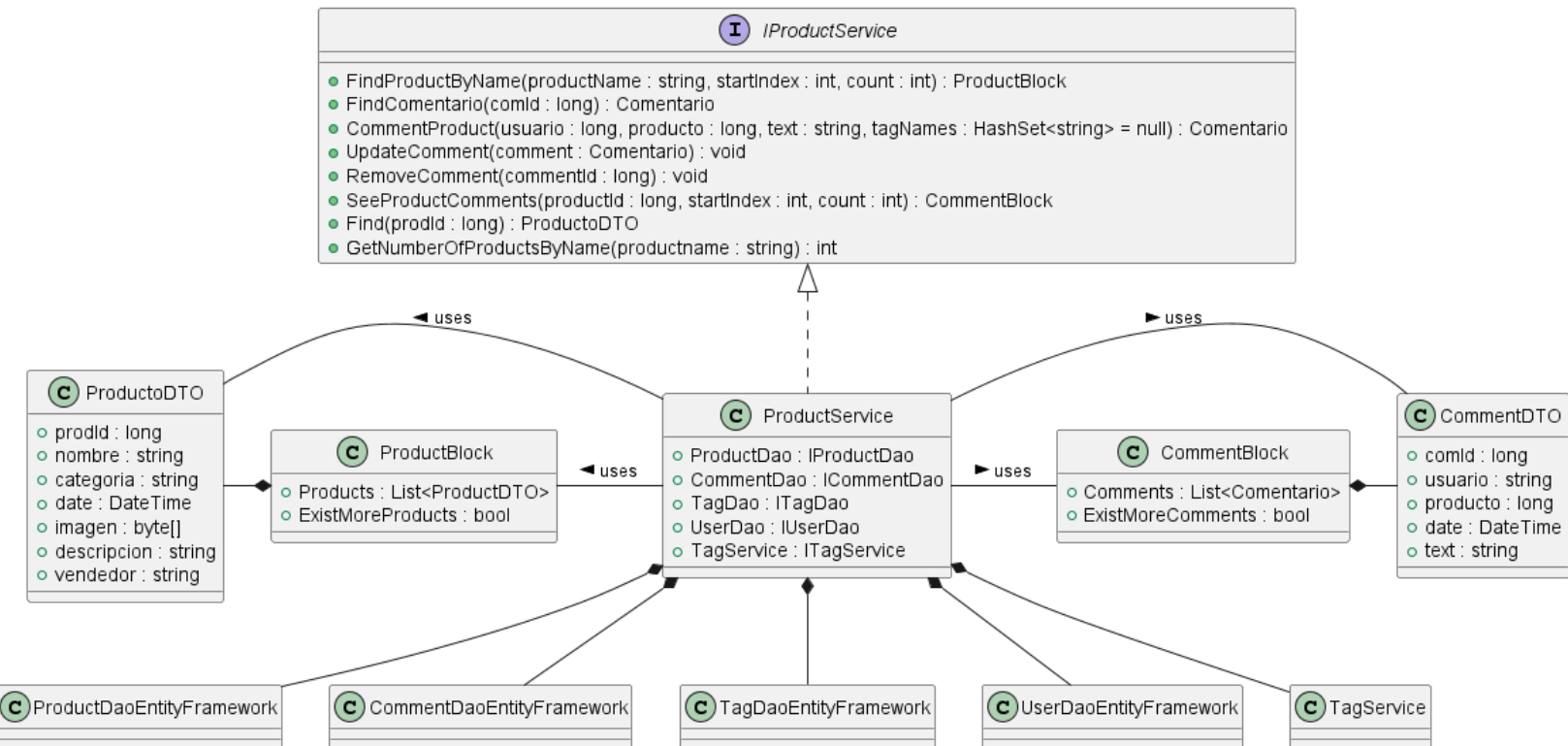


Para este ejemplo, hemos utilizado el DAO de Producto, aunque cualquiera ilustra correctamente el funcionamiento. Como vemos en el diagrama, la implementación de un DAO siempre está formada por la interacción de dos paquetes: uno principal, que incluye la nueva funcionalidad implementada durante la práctica y uno auxiliar proporcionado como base que forma parte del paquete ModelUtils. Lo que tenemos exactamente en el DAO son dos interfaces y dos clases:

- La interfaz `IGenericDao` proporcionada por el paquete `ModelUtils` se encarga de incluir el funcionamiento básico de un DAO implementado de manera genérica para cualquier tipo. En este ejemplo se usa para objetos de tipo `Producto` cuya clave es un `Int64`, es decir, un `long`. Es por eso que esta interfaz incluye las operaciones básicas de trabajo con base de datos: `Create`, `Update`, `Find` y `Delete`. Todas ellas con sus tipos modificados para objeto(`Producto`) y clave(`long`).
- La clase `GenericDaoEntityFramework` es la implementación concreta de la interfaz previa. Incluye las llamadas y métodos explicados previamente implementados a través de la tecnología de `EntityFramework` y, en este caso concreto, para objetos de tipo `Producto` con claves de tipo `long`.
- La interfaz `IProductDao` incluye las operaciones formalizadas por nosotros para este DAO a mayores de las básicas. Se trata de una extensión de `IGenericDAO`, de manera que cualquier clase que la implemente debe implementar también las de esta otra interfaz. Los métodos a mayores son `FindByProductName`, que recibe una `string` con el nombre a buscar, así como un valor de inicio y uno de `count` para indicar de qué número a qué número de respuesta queremos (si hay 7 productos con dicho nombre, `startIndex 2 count 3` nos devolvería las respuestas del índice 2 al 4). Este método devuelve una lista de `Productos`. Por otra parte, la función `GetNumberOfProductByName`, recibe una `string` con el nombre y devuelve el total de productos que contienen ese nombre.
- La clase `ProductDaoEntityFramework` se encarga de implementar las funcionalidades indicadas en la interfaz `IProductDao` utilizando la tecnología de `EntityFramework`. A mayores, esta clase debe implementar los métodos de `IGenericDao`, ya que `IProductDao` es una extensión de dicha interfaz. En lugar de eso, `ProductDaoEntityFramework` extiende a la clase `GenericDaoEntityFramework`, en quien deposita la responsabilidad de la implementación de dichos métodos.

## 2.4 Diseño de un servicio del modelo

Los servicios del modelo son el punto donde se une todo el conjunto del proyecto. Esto los convierte en los puntos de mayor complejidad de la lógica, ya que interactúan con los DAOs para generar datos que persisten en la base de datos al mismo tiempo que ofrecen salidas procesadas para que la interfaz pueda trabajar cómodamente. El ejemplo que veremos es el servicio del producto, ProductService. Aquí tenemos su diagrama de clases:

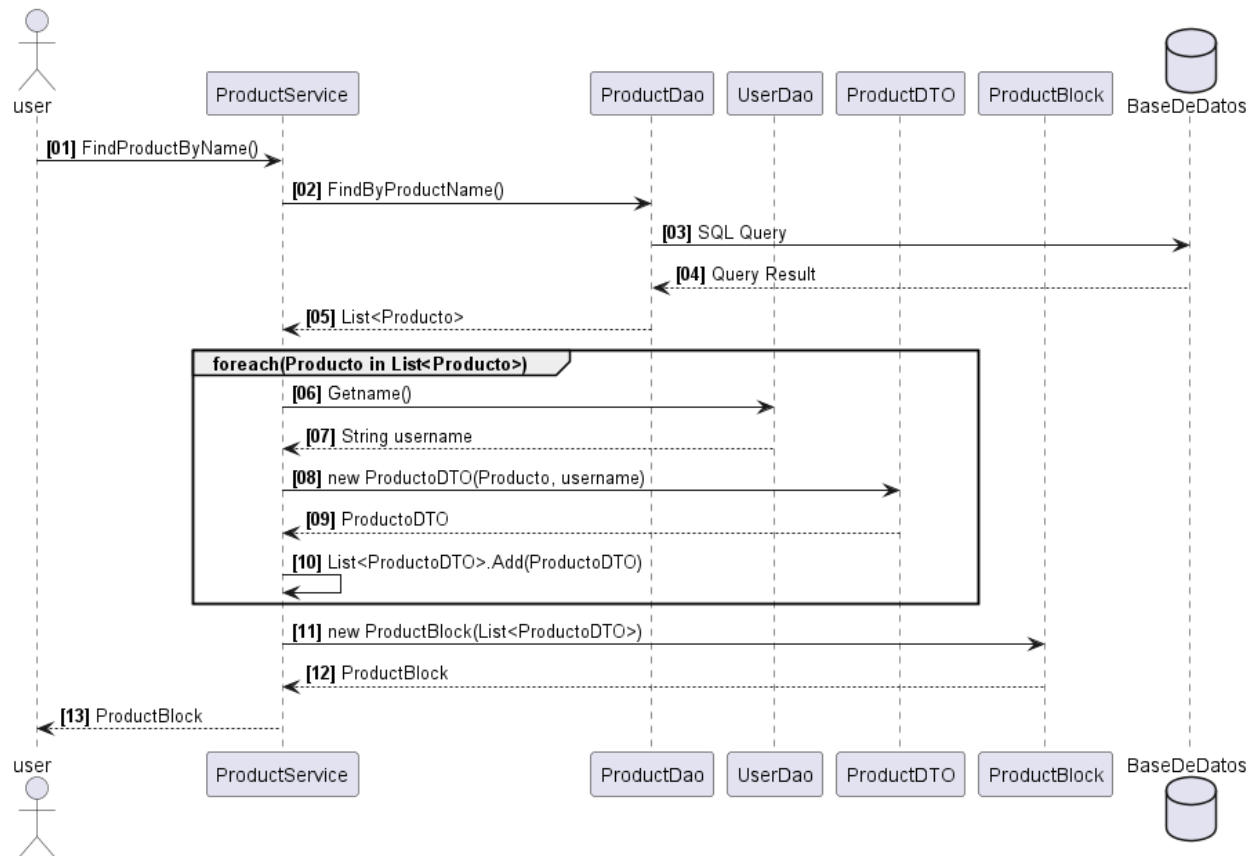


Como vemos, ProductService es una clase que interacciona con gran parte del modelo. Primeramente vemos que existe una interfaz IProductService que implementa. Aunque por claridad para el diagrama no se hayan incluido las funciones en su clase, obviamente ProductService incluye implementaciones para cada uno de estos métodos. Lo que sí se incluye en su definición es una serie de atributos (o propiedades, como se denominan en C#), mediante las cuales ProductService accede a otras clases del modelo. Las instanciaciones de dichas propiedades son encargadas a Ninject, quien ofrece las clases concretas que implementan la interfaz que necesita el servicio para trabajar. De esta manera, aparecen esas relaciones de agregación que vemos abajo que conectan con los DAOs de tecnología EntityFramework para las clases Product, Comment, Tag y User. Además, ProductService hace uso de otro servicio, TagService, para simplificar algunos métodos que ya están implementados en dicho servicio.

Por otra parte, hay relaciones menos obvias que han sido representadas como una relación de uso. ProductService hace uso de las clases ProductBlock y CommentBlock para

ofrecer bloques de productos y comentarios que sean más fáciles de usar en la interfaz para el proceso de paginación y visualización en lista. Por otra parte, la clase ProductoDTO no es más que una pequeña encapsulación de la clase Product, mediante la cual se modifica el id del vendedor por su nombre de usuario, facilitando así el acceso a este elemento, que tiene importancia en la interfaz gráfica. Finalmente, el propio bloque de productos tiene una relación de agregación con el ProductoDTO, ya que la lista de productos que posee es de este tipo. Lo que ocurre con ProductoDTO ocurre exactamente igual con CommentDTO, que existe para cambiar el id del usuario por su nombre.

Veamos ahora un diagrama de secuencia del caso de uso FindProductByName(), que recibe un nombre de producto con unos índices y devuelve un ProductBlock:

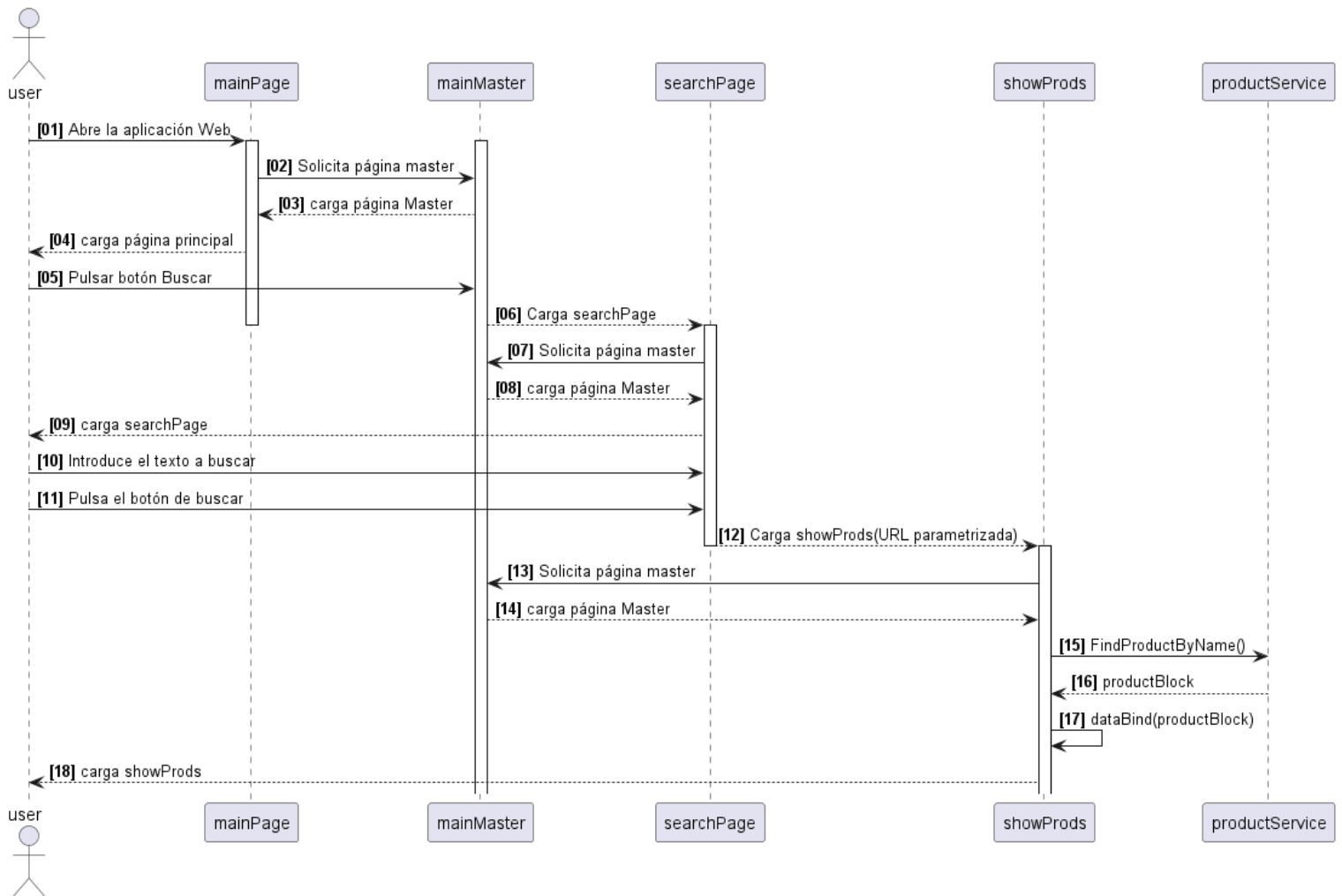


Como vemos, comienza con el usuario llamando al servicio. Este primeramente hace la petición al `ProductDao`, quien tras realizar la query a base de datos devuelve una lista de productos sin procesar. Después, para cada uno de los productos de esa lista, se deberá hacer una llamada a `UserDao` para conseguir el nombre del usuario. Una vez recibido, se instancia a través de `ProductoDTO` un nuevo objeto que se almacena en una lista local en `ProductService` de `ProductoDTO`s. Posteriormente, se instancia un `ProductBlock` con esa lista de `ProductoDTO`s y ese `ProductBlock` es el resultado final del método que llega al usuario.



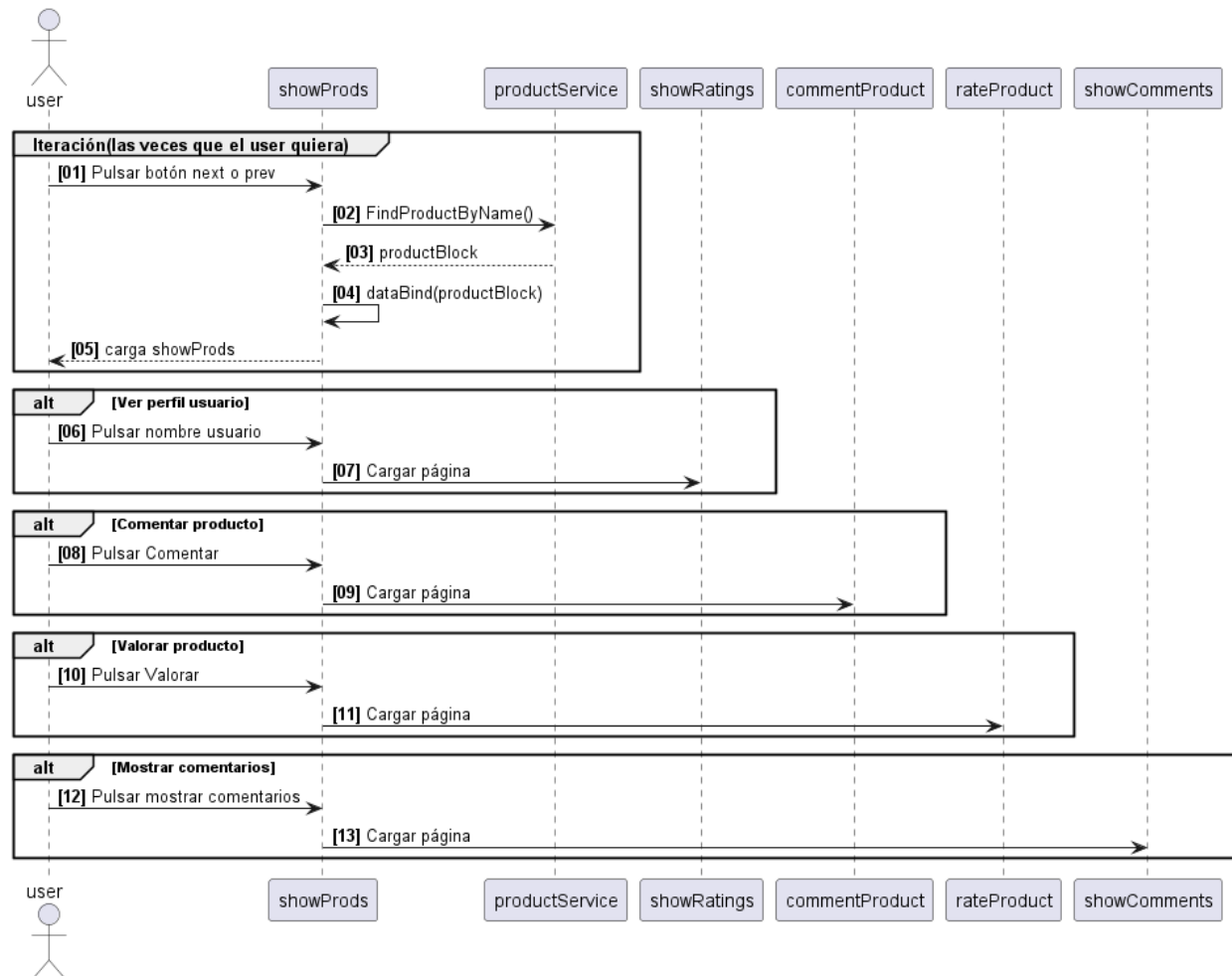
### 3. Interfaz Gráfica

La interfaz gráfica está formada por webforms, los cuales contienen tanto un archivo de configuración a nivel gráfico como un archivo que contiene la lógica de la página web. Es en ese archivo donde, a través de inyección de dependencias se accede a los diferentes servicios ofrecidos por la capa de negocio. En el siguiente diagrama podemos ver ilustrado uno de los casos de uso principales de la aplicación: la búsqueda de productos. Este caso de uso se caracteriza por iniciar en la página principal y dar acceso a otros casos de uso como, por ejemplo, mostrar comentarios, añadirlos, valorar productos o ver las valoraciones de un usuario concreto.



En este primer diagrama, vemos el proceso inicial desde que el usuario abre la aplicación web hasta que se muestra por primera vez la lista de productos. Primeramente, al abrir la aplicación web, la página principal se comunicará con su página maestra para traer los recursos que necesita. Una vez los cargue, cargará la página principal al usuario. En ella, el usuario podrá navegar a través de un botón Buscar. Al pulsarlo, la lógica de la página maestra se encargará de dar la orden de cargar la página searchPage, quien a su vez solicitará a la página maestra

los recursos necesarios. Una vez los tenga, la cargará y el usuario podrá introducir el texto a buscar. Una vez hecho y pulsado el botón de Buscar, la página de búsqueda dará la orden de cargar la página showProds, indicando a través de una URL parametrizada el nombre a buscar. Esta página será la que se comunique con el servicio del producto, recupere un ProductBlock y lo muestre en una tabla. Una vez que tiene la tabla lista, carga la página al usuario y se la muestra.



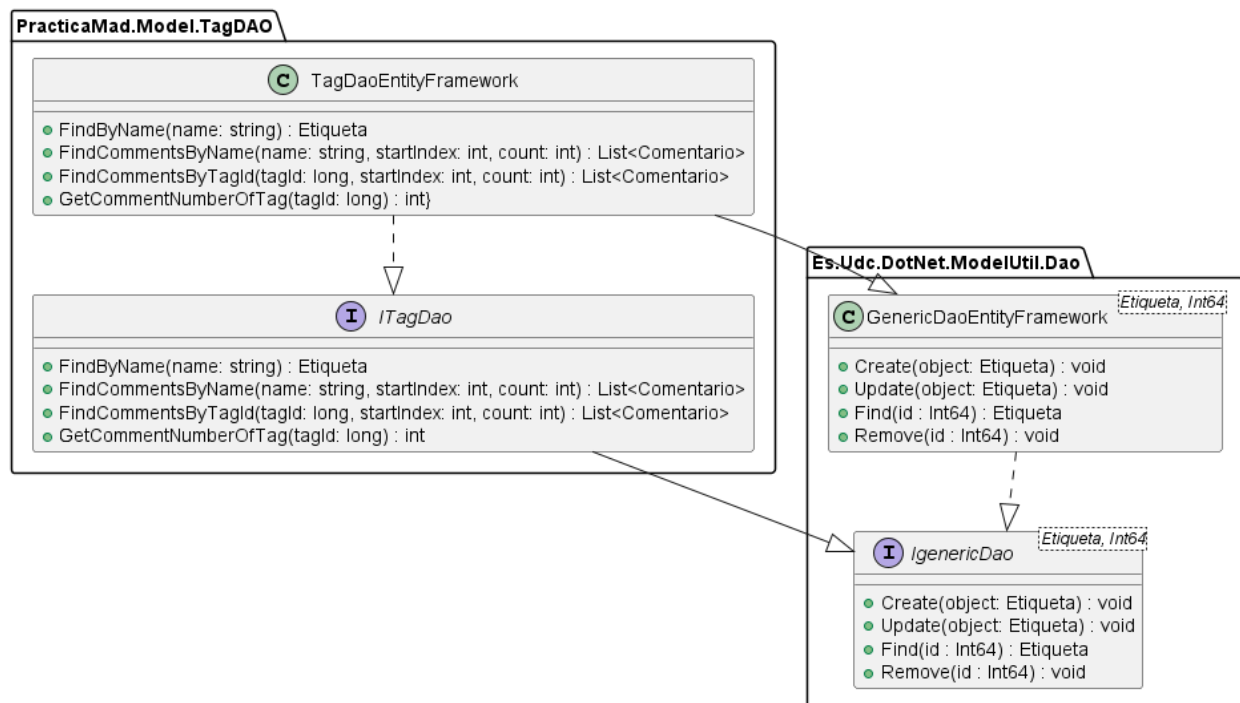
En este segundo diagrama, vemos parte de la funcionalidad accesible desde esta página showProds. En ella, el usuario puede navegar libremente por la lista de productos paginada utilizando los botones next y prev, que harán que la página llame de nuevo al productService solicitando un nuevo ProductBlock que mostrar y cargando de nuevo la página. Además, puede acceder a otros casos de uso a través de los links relacionados con cada producto, que lanzan los casos de uso y las páginas para cada una de ellas como podemos ver en el diagrama.

#### 4. Etiquetado de comentarios

El etiquetado de comentarios es un apartado incluido en el desarrollo de nuestra práctica. En este apartado, comentaremos cómo se realizó su implementación a lo largo de las diferentes partes del proyecto.

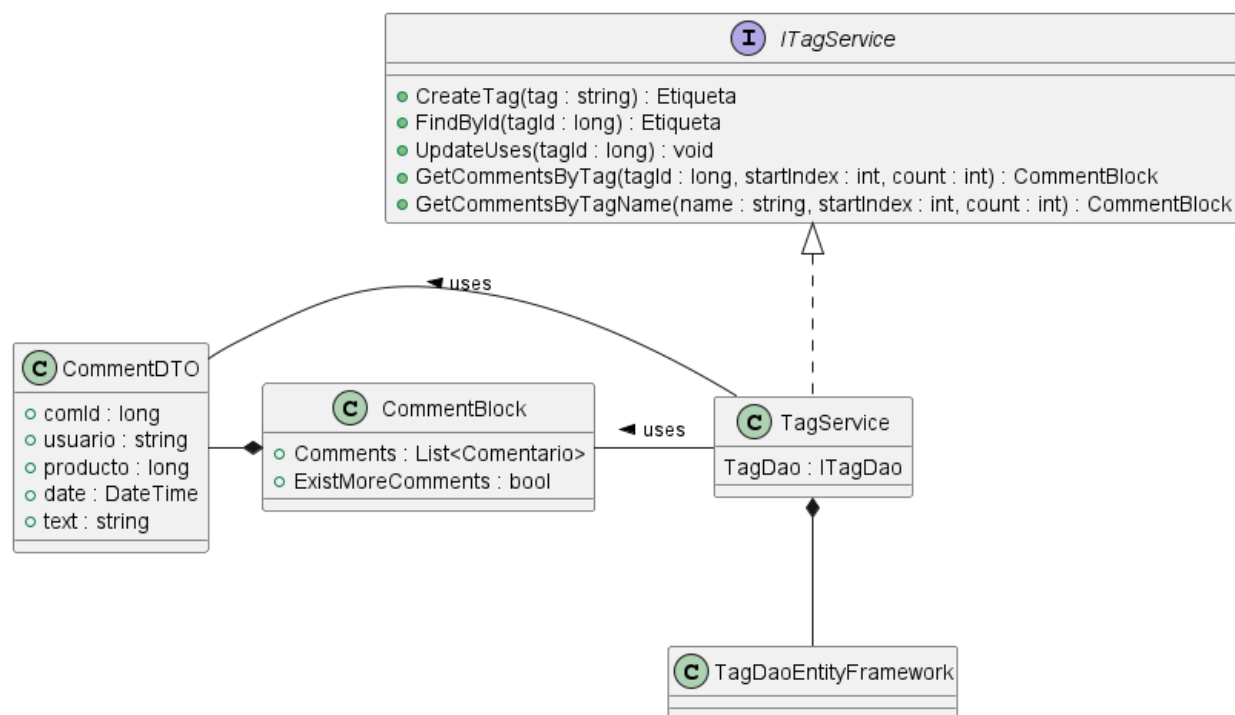
Primeramente, se incluyeron dos tablas en base de datos relacionadas con esta implementación. La primera de ellas, Etiqueta, se encarga de almacenar la información de las etiquetas como tal. La otra, la tabla TagCom, se trata de una entidad débil cuya labor es almacenar relaciones entre etiquetas y comentarios. Esto se puede ver en el apartado 2.1 de esta práctica, por lo que consideramos que no es necesario volver a incluir dicho diagrama en este apartado.

A continuación, fue necesaria la creación de un Dao propio de las Etiquetas que permitiese la elaboración de operaciones contra dicha tabla. Para ello, se implementaron una serie de clases que detallamos en el siguiente diagrama:



Como vemos, la implementación es muy similar a la ya vista en el apartado 2.3. Los cambios son, evidentemente, en los métodos declarados por la interfaz de **ITagDao** e implementados por **TagDaoEntityFramework**, que en este caso están orientados a las necesidades del modelo y la interfaz respecto a las etiquetas.

En cuanto al servicio de Etiquetas, nos encontramos con lo siguiente:



Este servicio es considerablemente más sencillo que **ProductService**, el que ya vimos en el apartado 2.4. El funcionamiento es muy similar. **TagService** implementa una interfaz llamada **ITagService** y ofrece sus métodos. Tiene definido como propiedad un **ITagDao**, que se consigue mediante Ninject, lo que genera una relación de agregación con la clase **TagDaoEntityFramework**. Por otra parte, utiliza **CommentBlock** para generar salidas más cómodas para el uso en la interfaz, ya que posee métodos para recuperar listas de comentarios en base al tag que ha sido introducido. De la misma manera que en **ProductService**, **CommentBlock** posee **CommentDTO** en su lista y **TagService** hace uso de ellos para las respuestas.

Mediante un menú de navegación introducido en la página maestra, se implementó una nube de tags, presente en todas las páginas. Este menú, que implementa paginación, permite seleccionar cualquier tag y ver los comentarios etiquetados correspondientes en una tabla.

## 5. Compilación e instalación de la aplicación.

Para la compilación del proyecto será necesario instalar a través de Nuget los paquetes de EntityFramework, Ninject, Ninject.Extensions.Interception y MsTest.TestFramework. También será necesario cambiar la cadena de ubicación de la base de datos situada en el archivo `SqlCreateDatabase.sql` del paquete `Model/Sql`, así como configurar correctamente la cadena de conexión a base de datos en los archivos `appConfig`. Una

vez hecho esto, se podría compilar el proyecto. Para ejecutar las pruebas bastaría con abrir el explorador de pruebas y pulsar en el botón de ejecutar todas. En el caso del proyecto web, deberá seleccionar el proyecto Web como principal y el IDE ya ofrecerá la posibilidad de ejecutarlo en la barra superior. En caso de que su navegador no permita la conexión, es posible que tenga que modificar las flags para permitir certificados SSL desde localhost(en el caso de google chrome, por ejemplo).

## **6. Problemas conocidos**

La funcionalidad de Remove para Comentarios no pudo ser implementada debido a un problema desconocido relacionado con la interacción del Dao con las funciones generadas a través del archivo MaD.tt del EDMX.