**Marcos de Desarrollo**
**2024/2025**

# Web de Restaurantes

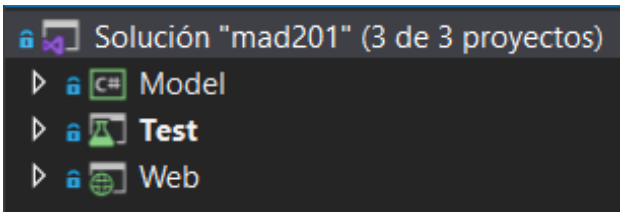
**Grupo 201**
Javier Rodríguez Rodríguez (j.rrodriguez1@udc.es)
Roberto Piñón Diaz (r.pinon.diaz@udc.es)
Angel Fraga Regueiro (angel.fraga.regueiro@udc.es)

# Índice

# 1. Arquitectura global

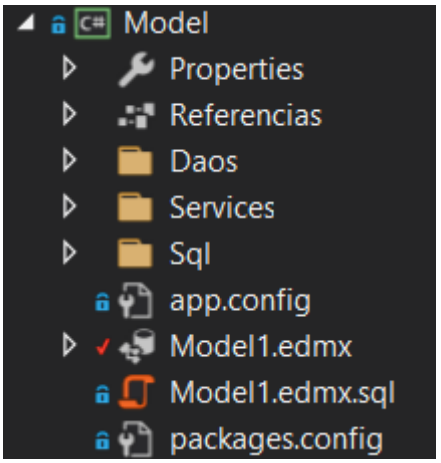
A continuación se explicará cómo hemos organizado los diferentes ficheros de nuestra aplicación, además de añadir imágenes para apoyar la explicación.

La solución “mad201” se compone de 3 proyectos: el Modelo, con su base de datos, clases persistentes, DAOs, …; los Test para las funciones del modelo, para comprobar su correcto funcionamiento; y la parte Web, con las vistas, imágenes, CSS, …. Se puede observar la arquitectura en la siguiente figura:



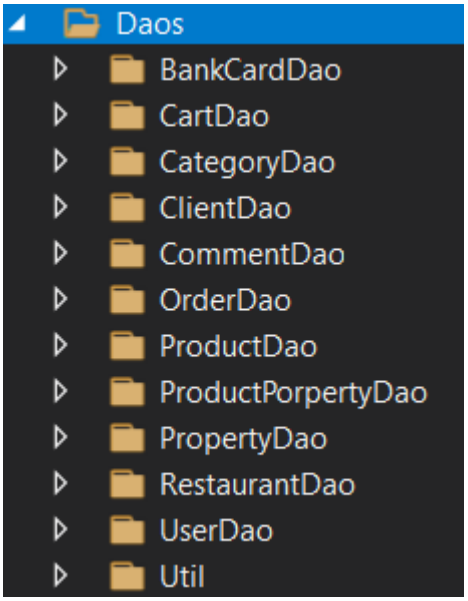

## 1.1. Model

En el proyecto “Model”, se encuentra la capa modelo dividida en 3 grandes subdirectorios, que son Daos, Services y Sql. El directorio “Sql” alberga scripts dedicados a la creación de la base de datos, junto con sus tablas e inserciones de datos, mientras que en el directorio “Daos” se almacenan los DAOs para interactuar con la base de datos y la creación de los servicios. Por último, en el directorio de Services se encuentran las interfaces y servicios que se utilizan en la Web. En resumen, el proyecto “Model” es el encargado de interactuar con la base de datos y gestionar la parte lógica de la aplicación mediante los servicios disponibles.



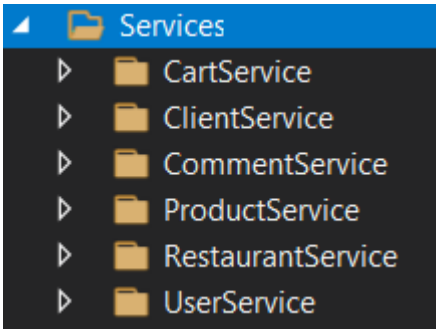
### 1.1.1. Daos

Este directorio está dividido en subdirectorios, uno para cada entidad de la base de datos. En cada subdirectorio se encuentra la implementación de una interfaz para el DAO y su implementación. Cabe destacar que todos los DAOs heredan de GenericDao y además existe un subdirectorio Util, el cual implementa las clases PagedResult y QueryableMethods, que son una extensión de LINQ para agregar paginación fácilmente a cualquier consulta que realicen los DAOs.



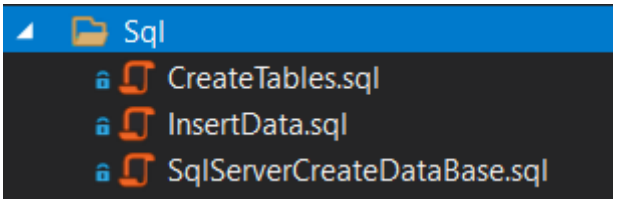

## 1.1.2. Services

Este directorio también se dividió en subdirectorios, esta vez con uno para cada entidad que estuviese relacionada con las funcionalidades. Para cada subdirectorio además se crearon más subdirectorios en caso de necesitarlos:
DTOs (contiene los DTOs necesarios para el servicio), Exceptions (contiene las excepciones exclusivas del servicio) y Util (contiene implementaciones que ayudan en las funcionalidades del servicio, en este caso para encriptar la contraseña).



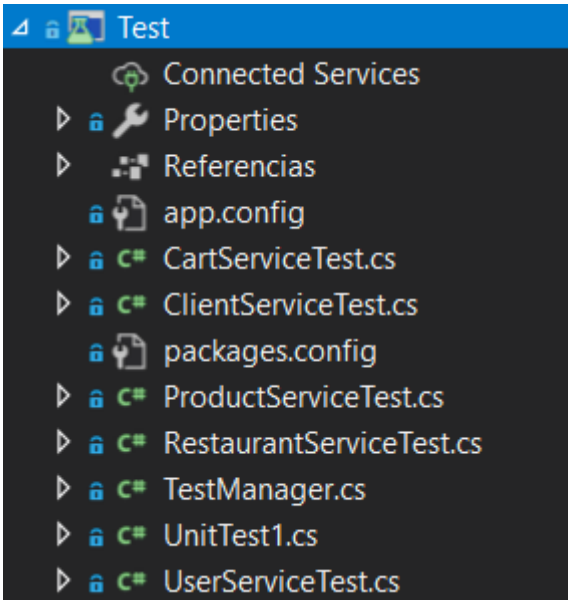
### 1.1.3. SQL

Este directorio solo contiene los scripts que son necesarios para la creación de la base de datos y de sus tablas.



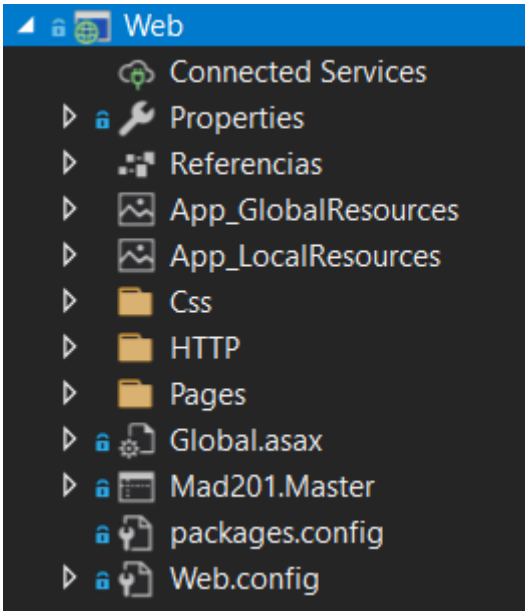

## 1.2. Test

En esta carpeta nos encontramos con las distintas clases que implementan los tests para cada funcionalidad del modelo.



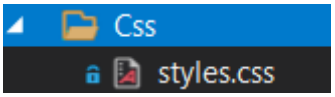
## 1.3. Web

Dentro de esta carpeta está la parte correspondiente con la vista del programa, en la cual nos encontramos con 3 subdirectorios principales: Css, para los estilos de la página; HTTP, para la implementación de la lógica de la web; y Pages, que almacena los archivos .aspx encargados de crear las páginas web de ASP.NET Web Forms.



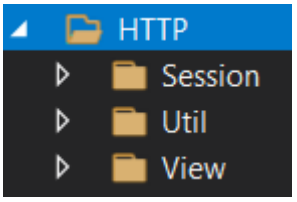

### 1.3.1. CSS

La carpeta CSS contiene el archivo global de hojas de estilo en cascada (CSS, por sus siglas en inglés) utilizados para definir la presentación visual del sitio web.



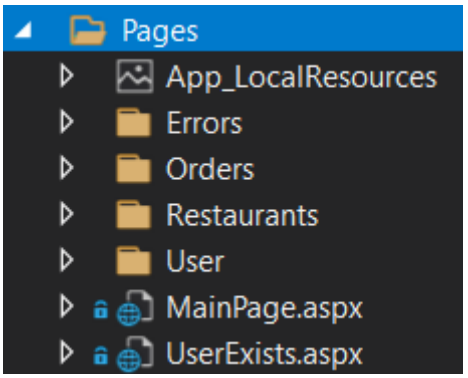
### 1.3.2. HTTP

Esta carpeta contiene la gestión relacionada con las peticiones y respuestas HTTP que se realizan en nuestra página web. Session contiene las clases encargadas del funcionamiento de cada sesión creada en la parte Web. Mantiene el idioma, contiene parámetros del usuario logueado, etc. Util contiene funcionalidades para poder controlar las cookies. View se usa para controlar la selección del idioma del usuario.



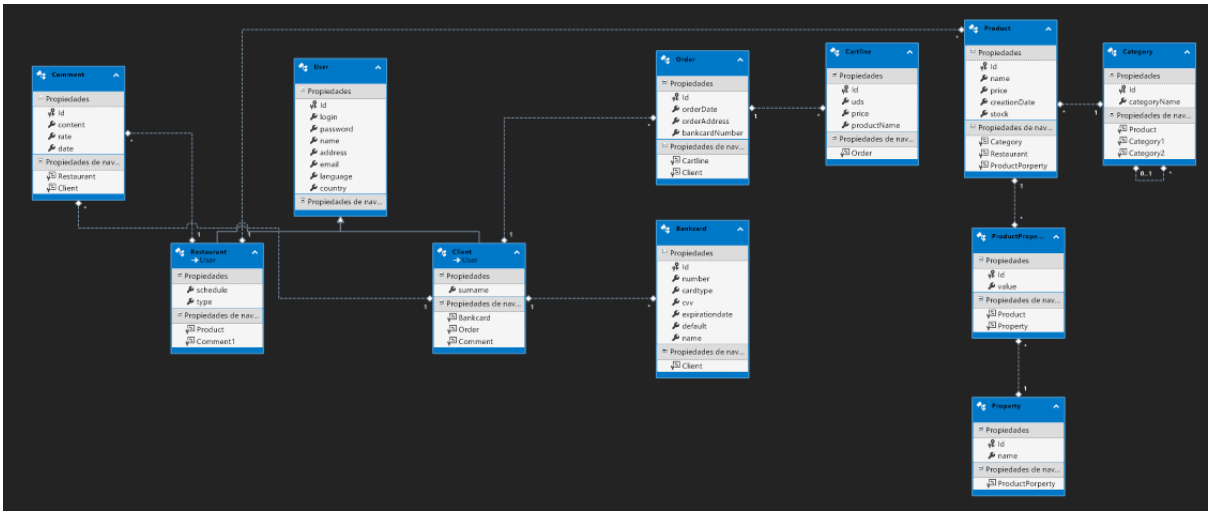

### 1.3.3. Pages

Este directorio contiene las páginas web del proyecto ASP.NET, además de subdirectorios para tener mejor organizadas los diferentes tipos de cliente a los que va dirigida la página. Cada subdirectorio contiene las distintas vistas con sus code-behind. El directorio Errors las contiene vistas para los distintos errores que se puedan recibir del modelo. En el App_LocalResources se encuentran los archivos de internacionalización.



# 2. Modelo

## 2.1. Clases Persistentes



El modelo está formado por 11 clases persistentes interconectadas, que trabajan en conjunto para gestionar los productos de los restaurantes, su clasificación en diferentes categorías, los diferentes usuarios del sistema, los comentarios de los clientes y los posibles pedidos de los clientes.

La clase **User** es una superclase de la que heredan las subclases **Restaurant** y **Client** los atributos que tienen en común y, además, las subclases añaden sus propios atributos y relaciones.

Los restaurantes, modelados con la clase **Restaurant**, están relacionados con los productos, modelados con la clase **Product**, con una relación de 1 a N, ya que un restaurante puede tener varios productos, pero un producto solo puede pertenecer a un restaurante. Además la clase **Product** está relacionada con las clases **PropertyProduct** y **Category**, con una relación de N a 1 con **Category** y de 1 a N con **PropertyProduct**; que es una clase intermedia que almacena las relaciones entre las propiedades existentes y los productos. Además **Category** tiene una relación de 1 a N consigo misma para representar las diferentes subcategorías. La clase **Property** tiene una relación de 1 a N con la clase intermedia **PropertyProduct**.

El cliente, modelado con la clase **Client**, interactúa con las tarjetas bancarias, modeladas mediante la clase **Bankcard**, en una relación de 1 a N, ya que una tarjeta solo pertenece a un cliente, pero un cliente puede tener varias tarjetas. También puede hacer pedidos, modelados por la clase **Order**, creando diferentes líneas de pedido, modeladas por la clase **Cartline**, la cual es la que interactúa con los productos a la hora de hacer el pedido.

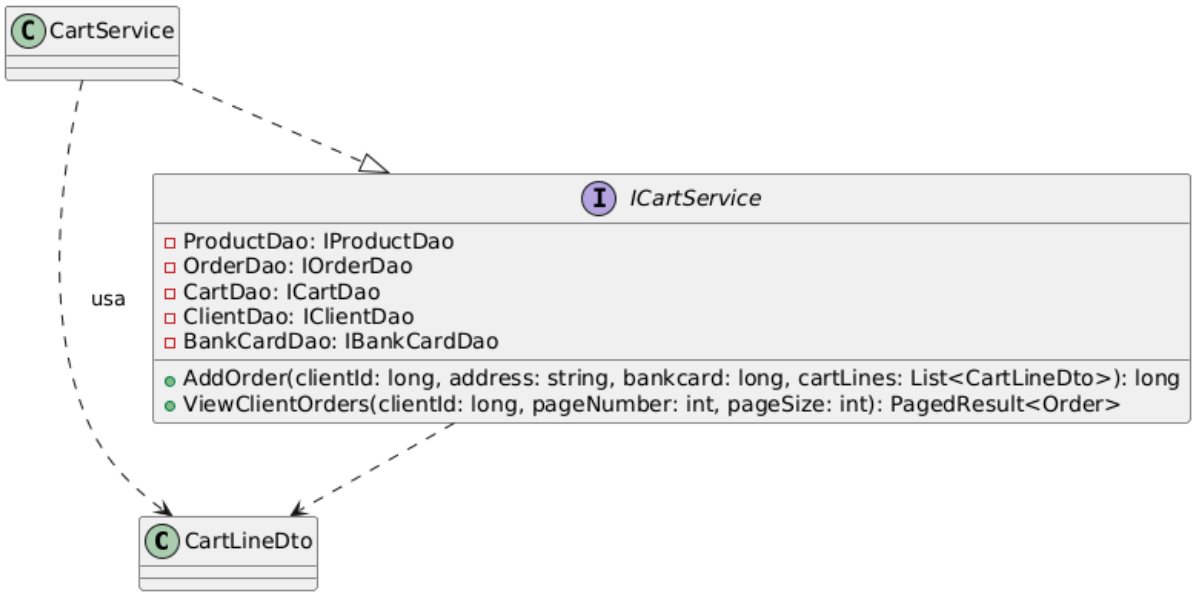
Además están los comentarios, modelados con la clase **Comment**, que interactúan tanto como con el cliente como con los restaurantes, con una relación N a 1 con cada clase correspondiente.

## 2.2. Interfaces de los Servicios Ofrecidos por el Modelo

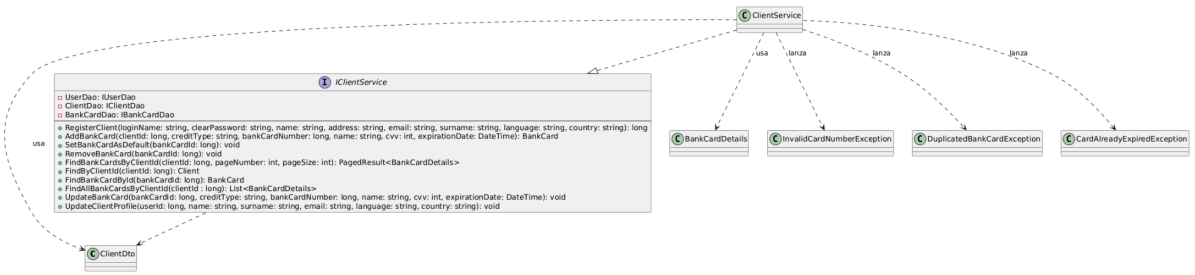
La organización de los servicios se ha realizado atendiendo a su funcionalidad y al ámbito de los procesos que controlan. Cada interfaz de servicio ha sido creada para ocuparse de tareas afines, con el objetivo de mantener una estructura modular y comprensible dentro del sistema. Todos los servicios disponen de la implementación de una interfaz y de otra clase que implementa la interfaz.

A continuación, se detalla brevemente cómo se ha llevado a cabo esta agrupación, mostrando todos los atributos y funciones de las interfaces, además de las clases que la implementen o use y excepciones que lance:

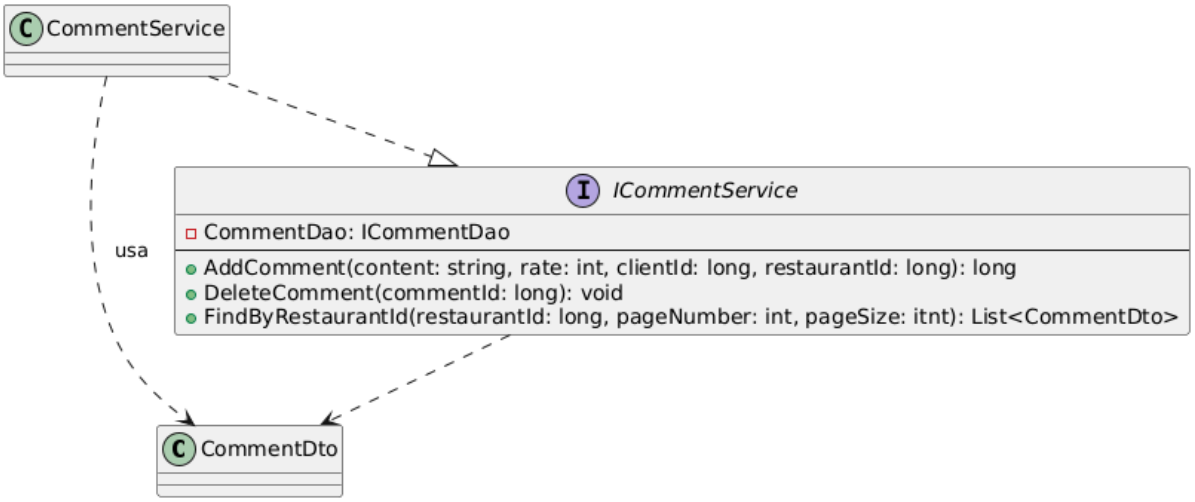
- **Cart Service:**
  Este servicio se encarga de gestionar todo lo relacionado con el carrito del cliente y sus pedidos dentro del sistema. Su propósito principal es la búsqueda y creación de pedidos por parte del cliente. Este servicio también incluye el DTO necesario para la clase **Cartline**.



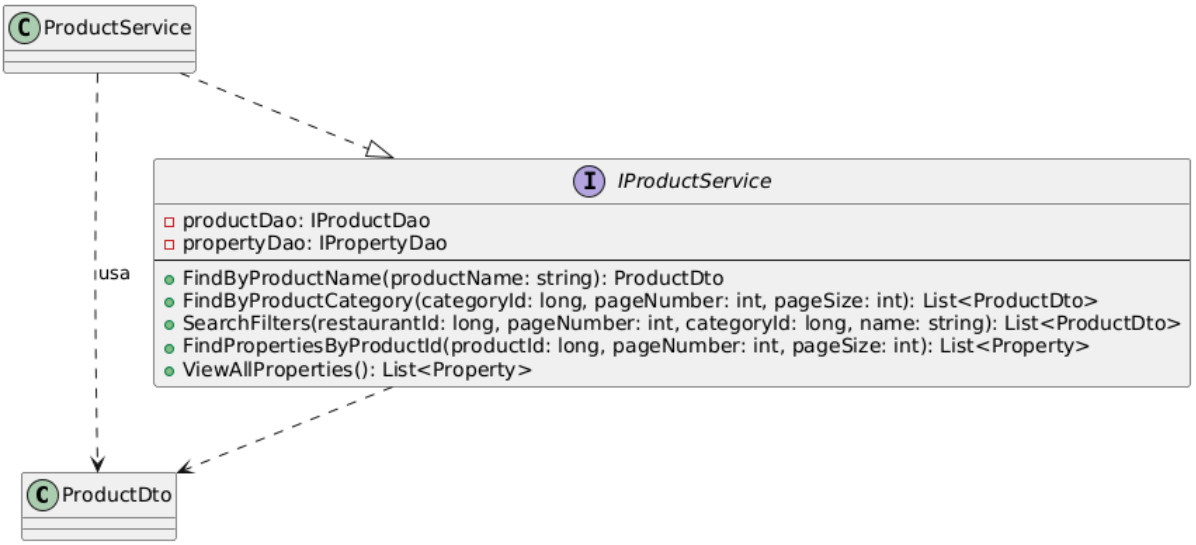
- **Client Service:**
  Este servicio es el encargado de manejar las peticiones realizadas por el cliente, desde la creación de un nuevo usuario, hasta buscar las diferentes tarjetas bancarias del cliente. También incluye una clase encargada de recuperar los detalles de las diferentes tarjetas bancarias que tenga el cliente registradas. Implementa tanto DTOs para **Client** y **Bankcard**, como excepciones que lanzan los métodos.



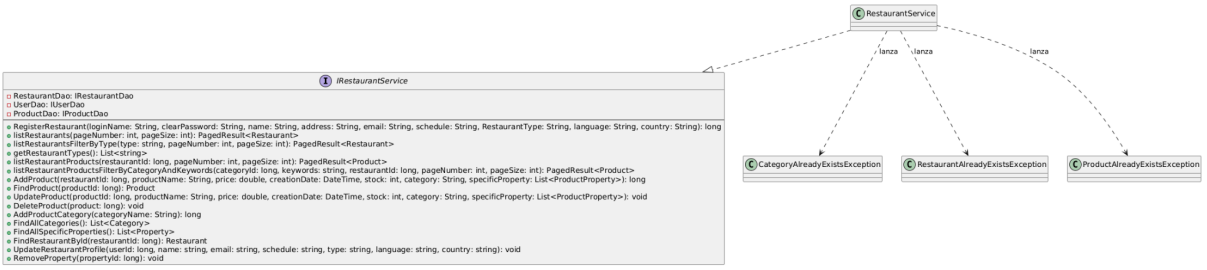

- **Comment Service:**
  El servicio se ocupa de la gestión de los comentarios realizados por parte de un cliente a un restaurante. Se puede añadir o eliminar un comentario por parte de un cliente y encontrar los comentarios que correspondan con un restaurante. También implementa el DTO de la clase **Comment**.



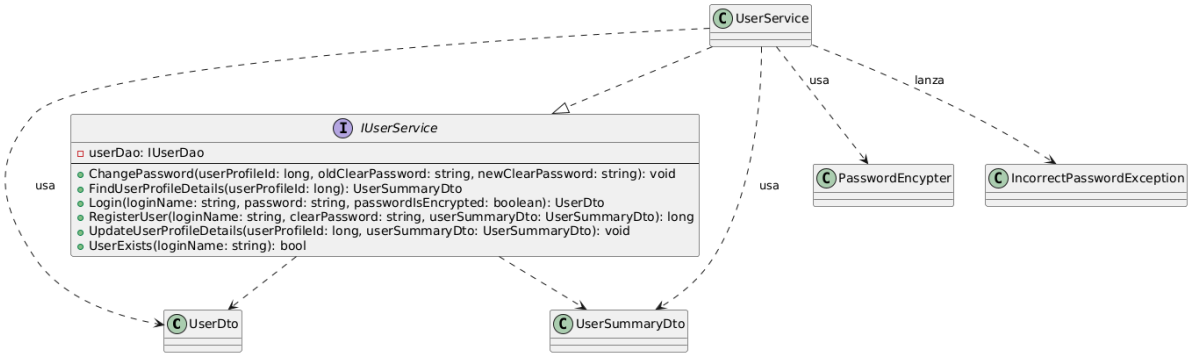
- **Product Service:**
  Este servicio está diseñado para manejar las operaciones relacionadas con buscar los productos de un restaurante y sus propiedades. Hay diferentes funciones para poder aplicar diferentes filtros o requisitos a la hora de buscar productos. Implementa el DTO de la clase **Product**.




- **Restaurant Service:**
  El servicio se encarga de todas las operaciones relacionadas con los restaurantes además de funciones que se encargan de modificar los productos de un restaurante: añadirlos, editarlos o eliminarlos. En este servicio solo se implementa aparte las excepciones que va a lanzar el servicio.



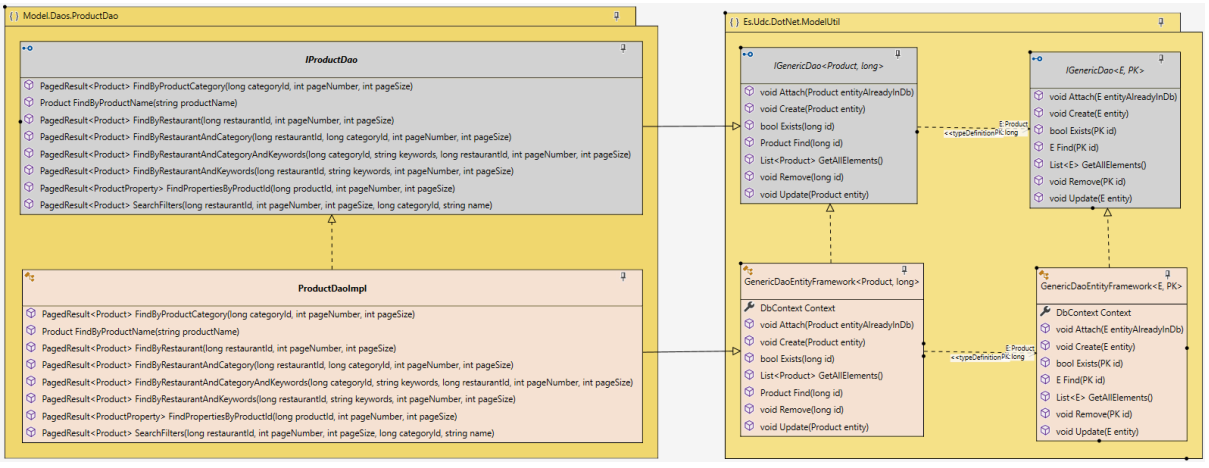
- **User Service:**
  El servicio está orientado a la gestión de los atributos que tienen en común tanto la clase **Restaurant** como la clase **Client**. También es el encargado de la lógica del login de los distintos usuarios del sistema. Implementa el DTO de la clase **User** además de las excepciones que lanza el servicio y la clase **PasswordEncrypter**, encargada de encriptar las contraseñas.



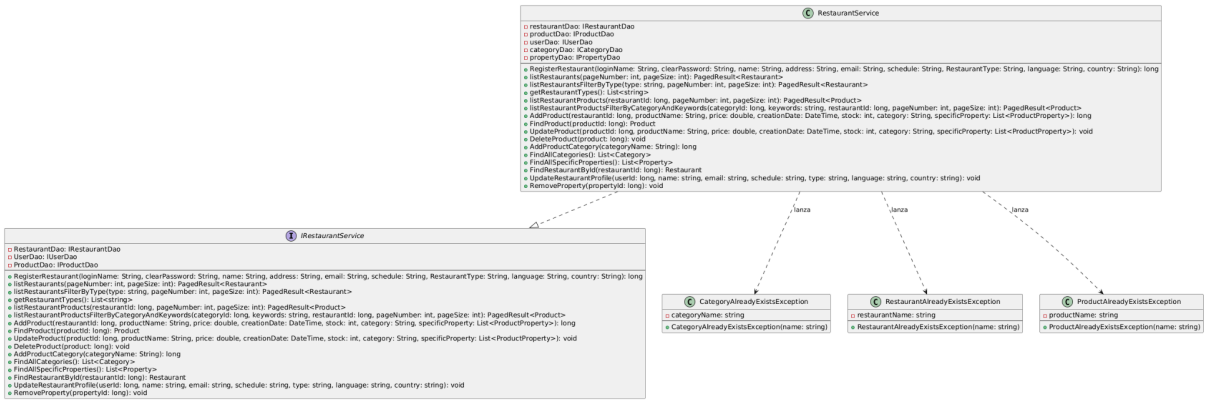

## 2.3. Diseño de un DAO

En el diseño, cada "Data Access Object" (DAO) está configurado para heredar de la interfaz **IGenericDao**, que define un conjunto estandarizado de funciones básicas. Cuando se requieren funcionalidades adicionales, adoptamos un enfoque más especializado. Estas funciones se implementarán en la clase **DaoEntityFramework** asociada. Cabe destacar que, antes de la implementación, dichas funciones deben ser declaradas en la interfaz del DAO correspondiente, asegurando una estructura clara y coherente.

Se muestra el Dao de la clase **Product**, ya que parecía el más representativo al ser el que cuenta con más operaciones.



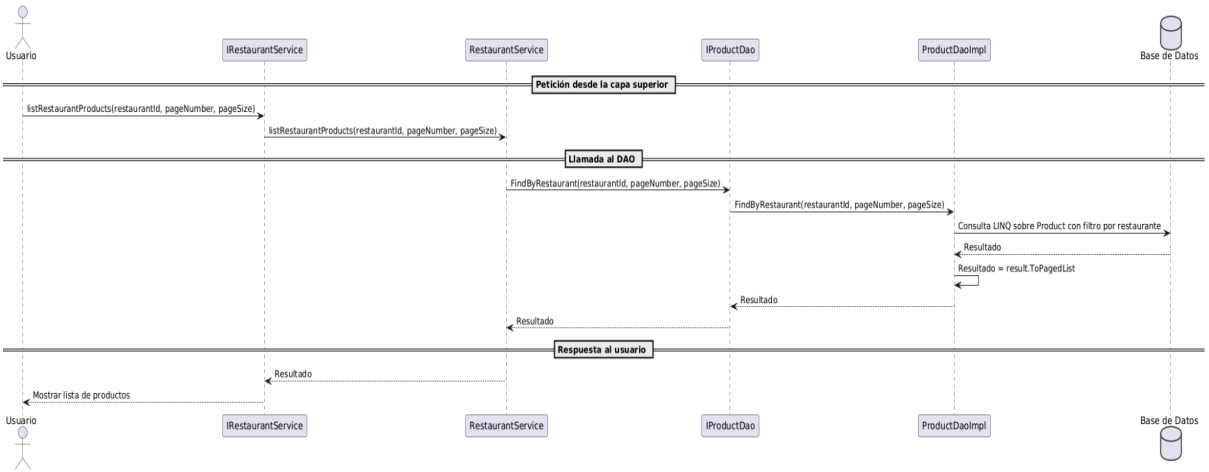
## 2.4. Diseño de un Servicio del Modelo




Se usará el caso de uso “Añadir Producto” para explicar este apartado. Para ello, se mostrará el servicio representado con la fachada **IRestaurantService**. No se mostrarán el resto de fachadas debido a su alta similitud en la arquitectura.

Como se puede ver en la imagen, para esta interfaz se usan los DAO’s mostrados en el diagrama (**IProductDao**, **IPropertyDao**, **IRestaurantDao**, **IUserDao** e **ICategoryDao**; que son las interfaces de los DAO’s correspondientes). La clase **RestaurantService** implementa esta interfaz.

En el diagrama de secuencia que se muestra más abajo se puede ver con claridad el funcionamiento del caso de uso seleccionado, desde su llamada a la función de *listRestaurantProducts* hasta el acceso a la base de datos para realizar la consulta select necesaria. Cada clase devuelve el resultado hasta que se muestra.

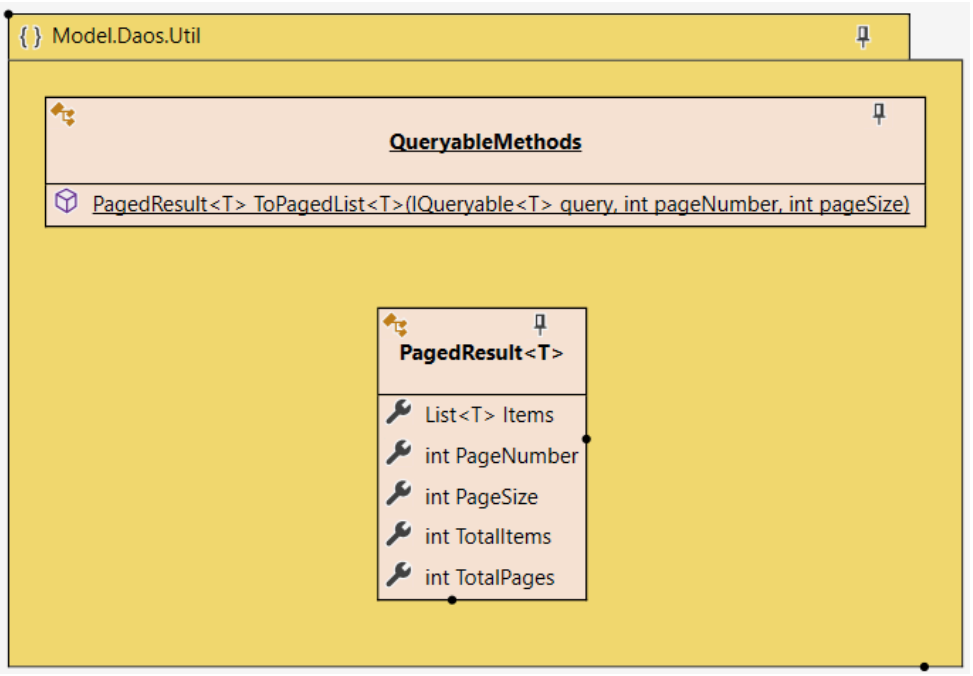


## 2.5. Otros Aspectos

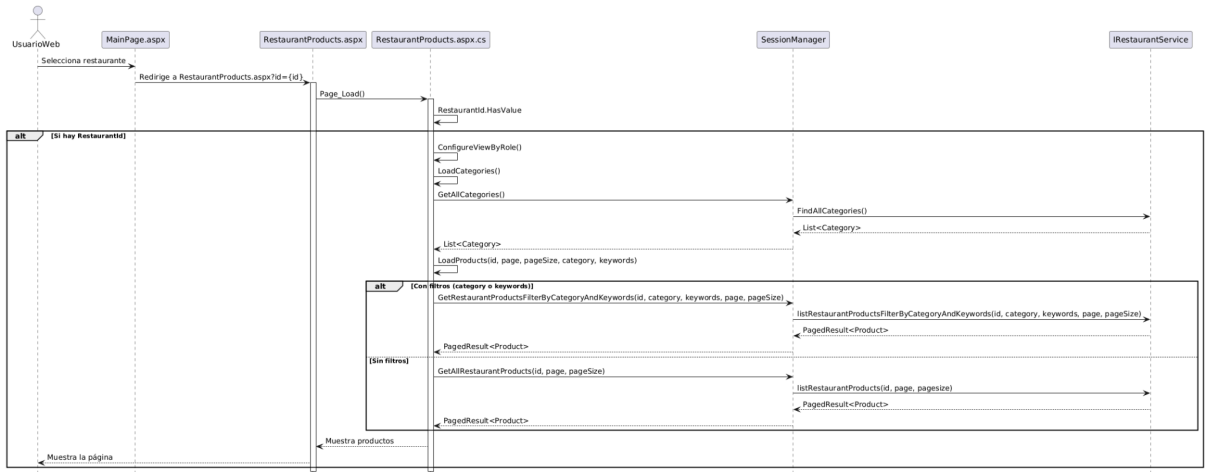
Para la internacionalización de la página web se emplearon los idiomas inglés, español y gallego.

También se empleó la paginación para los resultados mediante la clase **PagedResult** y la paginación para las consultas con la clase **QueryableMethods**.

Gracias a esta paginación, se almacenan los resultados en el atributo *Items* y mediante el resto de atributos podemos saber si quedan más resultados o no, dividiendo la lista de resultados en diferentes bloques manejables, mejorando así el rendimiento del sistema.



# 3. Interfaz Gráfica




Este diagrama de secuencia representa cómo funciona el caso de uso “Ver productos de un restaurante”, comenzando desde el click del usuario (ya sea cliente o el propio restaurante), hasta la interacción con el modelo, específicamente con la interfaz **IRestaurantService**.

El detalle del procesamiento de la llamada al servicio se ha simplificado en este caso, ya que se muestra de forma completa en el diagrama de secuencia del apartado 2.4.

# 4. Compilación e Instalación de la Aplicación

Para asegurar una ejecución exitosa de la aplicación y facilitar la documentación, siga estos pasos, asumiendo que el entorno está correctamente instalado:

Abra la solución del proyecto en Visual Studio.

1. Ejecute el script SqlServerCreateDatabase.sql ubicado en Model > Sql para crear la base de datos.

2. Ejecute el script CreateTables.sql en Model > Sql para crear las tablas.

3. Ejecute el script InsertData.sql en Model > Sql para insertar datos en las tablas.

4. Compile la solución del proyecto haciendo clic derecho sobre el proyecto y seleccionando “Compilar”.

5. Sobre el proyecto Web, click derecho establecer como predeterminado, esto habilitará en el botón superior de ejecución la ejecución de IIS.

Estos pasos aseguran una ejecución fluida de la aplicación.

# 5. Problemas Conocidos

1. No se traducen ni las fechas ni las concurrencias al usar la internacionalización.