PracticaMaD

Joaquín Solla Vázquez

José Luis Pardo López

Lucas Campos Camiña

Índice

Arquitectura global

La solución del proyecto se divide en 3 paquetes principales:

- Model: La parte modelo con los métodos de la aplicación.

  (Es.Udc.DotNet.PracticaMaD.Model)

- Test: Proyecto de pruebas del modelo.

  (Es.Udc.DotNet.PracticaMaD.Test)

- Web: La parte de interfaz usuario, que define las páginas que se verán en el navegador y su funcionamiento.

  (Es.Udc.DotNet.PracticaMaD.Web)

A continuación se detalla la estructura de cada paquete:

Estructura de Model: Contiene 3 subpaquetes:

- Es.Udc.DotNet.PracticaMaD.Model.DAOs: Están contenidos todos los DAOs y sus interfaces. Cabe mencionar que para cada DAO se añade otro subpaquete (ej. Es.Udc.DotNet.PracticaMaD.Model.DAOs.ProductDao).
- Es.Udc.DotNet.PracticaMaD.Model.Exceptions: Contiene todas las excepciones del modelo que han sido definidas (a mayores de las de Model.Util).
- Es.Udc.DotNet.PracticaMaD.Model.Services: Contiene los servicios y las clases necesarias (ej. ShoppingCart), que a su vez tiene la siguiente estructura:
  - Es.Udc.DotNet.PracticaMaD.Model.Services.CatalogService: Contiene la funcionalidad relacionada con la gestión y búsqueda de productos además de los comentarios y tags.
  - Es.Udc.DotNet.PracticaMaD.Model.Services.ShoppingService: Contiene las clases ShoppingCart y ShoppingCartLine además de los métodos que gestionan el carrito y la compra de productos.
  - Es.Udc.DotNet.PracticaMaD.Model.Services.UserService: Contiene los métodos de gestión de usuarios y las clases relacionadas al usuario (ej. BankCardDetails).

Estructura de Test: Contiene 2 subpaquetes:

- Es.Udc.DotNet.PracticaMaD.Test.DAOs: Contiene todos los tests de los DAOs.
- Es.Udc.DotNet.PracticaMaD.Test.Services: Con los tests de los servicios.

Estructura de Web: Contiene 2 subpaquetes:

- Es.Udc.DotNet.PracticaMaD.Web.HTTP: Contiene a su vez otros paquetes (no es relevante listarlos) con archivos necesarios para el correcto funcionamiento del proyecto web (ej. SessionManager.cs).
- Es.Udc.DotNet.PracticaMaD.Web.Pages: Paquete con todas las páginas del proyecto (salvo la página master). Las páginas están organizadas en 4 subpaquetes:
  - Es.Udc.DotNet.PracticaMaD.Web.Pages.Catalog: Todas las páginas de la parte correspondiente a Catalog en Model.
  - Es.Udc.DotNet.PracticaMaD.Web.Pages.Errors: Contiene una página para mostrar ante posibles errores.
  - Es.Udc.DotNet.PracticaMaD.Web.Pages.Shopping: Todas las páginas de la parte correspondiente a Shopping en Model.
  - Es.Udc.DotNet.PracticaMaD.Web.Pages.User: Todas las páginas de la parte correspondiente a User en Model.

Modelo

Clases persistentes

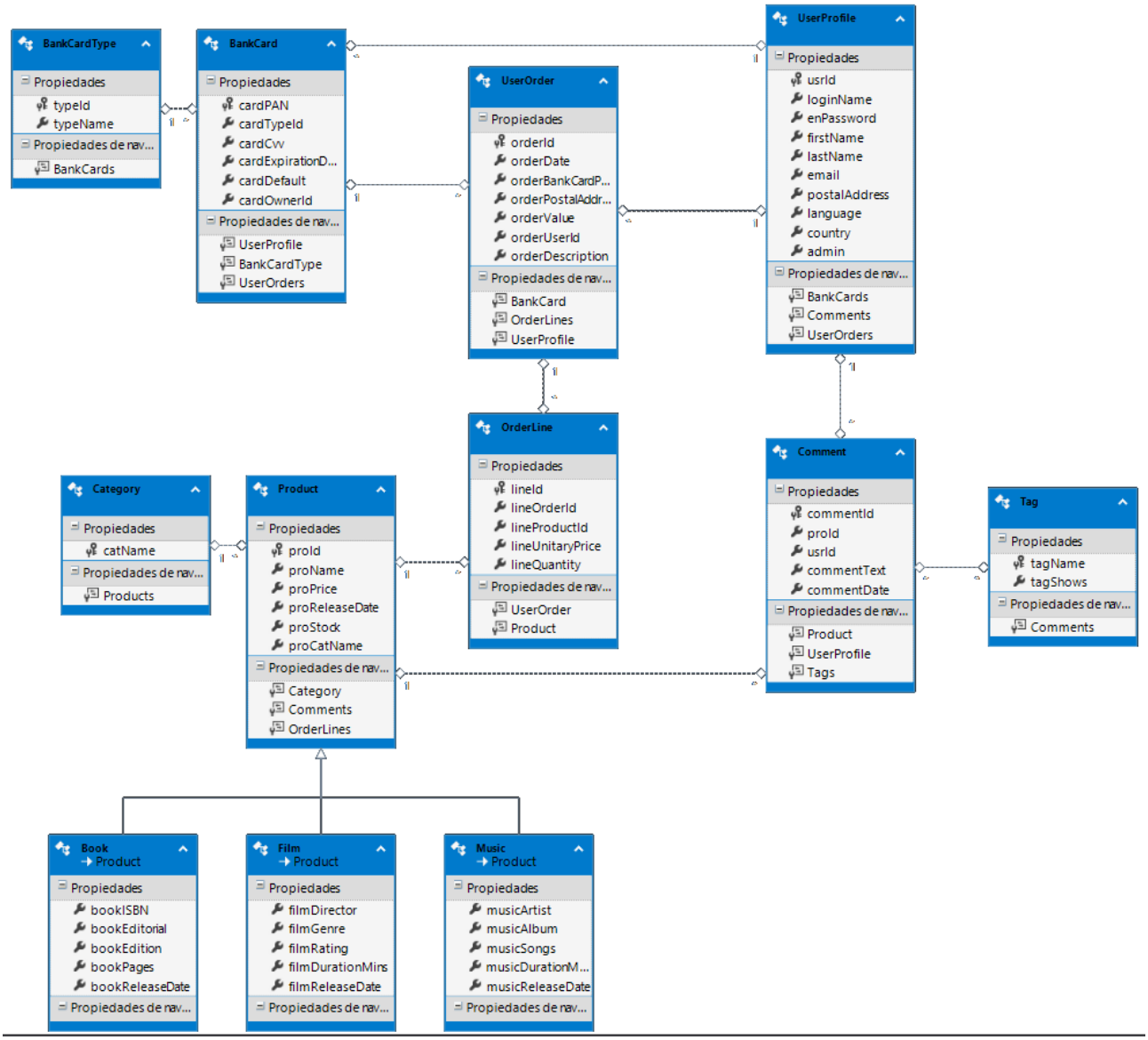


Diagrama de clases persistentes (PracticaMaD.Model.edmx).

Interfaces de los servicios ofrecidos por el modelo

CatalogService
+ UpdateProductDetails(ProductDetails productDetails, long usrId): void
+ FindProducts(string keyword, string proCatName, int startIndex, int size): ProductBlock
+ FindProducts(string keyword, int startIndex, int size): ProductBlock
+ FindAllCategories(): List<Category>
+ FindProductDetails(long proId): ProductDetails
+ ProductExists(long proId): bool
+ AddCommentToProduct(long proId, long usrId, string text, List<string> tags): long
+ UpdateComment(long idComment, long usrId, string newComment, List<string> newTags): void
+ DeleteComment(long idComment, long usrId): void
+ CommentExists(long idComment): bool
+ CommentBelongsToUser(long usrId, long idComment): bool
+ FindCommentsByProduct(long proId, int startIndex, int size): CommentBlock
+ TagExists(string tagName): bool
+ FindAllTags(): List<Tag>
+ FindCommentsByTagName(string tagName, int startIndex, int size): CommentBlock
+ FindCommentById(long commentId): Comment

ShoppingService
+ CreateShoppingCart(): ShoppingCart
+ AddProductToShoppingCart(long proId, ShoppingCart shoppingCart): void
+ DeleteProductFromShoppingCart(long proId, ShoppingCart shoppingCart): void
+ AddOneToShoppingCart(long proId, ShoppingCart shoppingCart): void
+ RemoveOneFromShoppingCart(long proId, ShoppingCart shoppingCart): void
+ ToggleGiftAtShoppingCart(long proId, ShoppingCart shoppingCart): void
+ ShoppingCartIsEmpty(ShoppingCart shoppingCart): bool
+ EmptyShoppingCart(ShoppingCart shoppingCart): void
+ BuyOrder(long usrId, long cardPAN, string postalAddress, string orderDescription, ShoppingCart shoppingCart): long

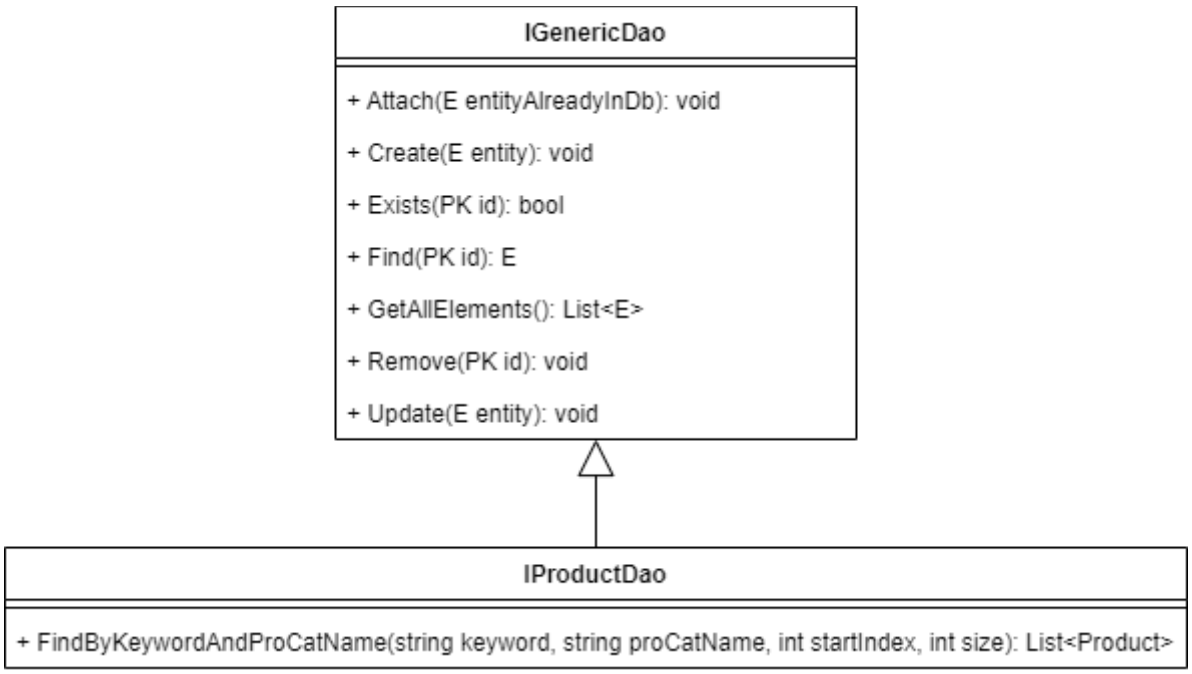
UserService
+ ChangePassword(long userProfileId, String oldClearPassword, String newClearPassword): void
+ FindUserProfileDetails(long userProfileId): UserProfileDetails
+ Login(String loginName, String password, Boolean passwordIsEncrypted): LoginResult
+ RegisterUser(String loginName, String clearPassword, UserProfileDetails userProfileDetails): long
+ UpdateUserProfileDetails(long userProfileId, UserProfileDetails userProfileDetails): void
+ UserExists(string loginName): bool
+ FindAllBankCardTypes(): List<BankCardType>
+ FindUserBankCards(long cardOwnerId): List<BankCard>
+ FindBankCardDetails(long cardPAN): BankCardDetails
+ AddBankCard(long cardOwnerId, long cardPAN, BankCardDetails bankCardDetails): long
+ UpdateBankCardDetails(long cardPAN, BankCardDetails bankCardDetails): void
+ BankCardExists(long cardPAN): bool
+ FindUserOrders(long orderUserId, int startIndex, int size): OrderBlock
+ FindUserOrderDetails(long orderId): UserOrderDetails

Interfaces de los servicios del modelo.

Los métodos se han repartido en los 3 servicios mostrados. Hemos elegido esta distribución por las siguientes razones:

- *CatalogService* contiene los métodos relacionados con el “cataálogo de productos”: buscar productos, ver sus detalles además de gestionar los comentarios de los productos y sus tags.
- Los métodos de *ShoppingService* son todos aquellos relacionados con la compra de productos: gestionar los artículos del carrito y comprar.
- En *UserService* están todos los métodos de gestión de usuarios además de los métodos para gestionar sus atributos como las tarjetas de crédito y sus pedidos.

Diseño de un DAO



Diseño de la estructura de un DAO.

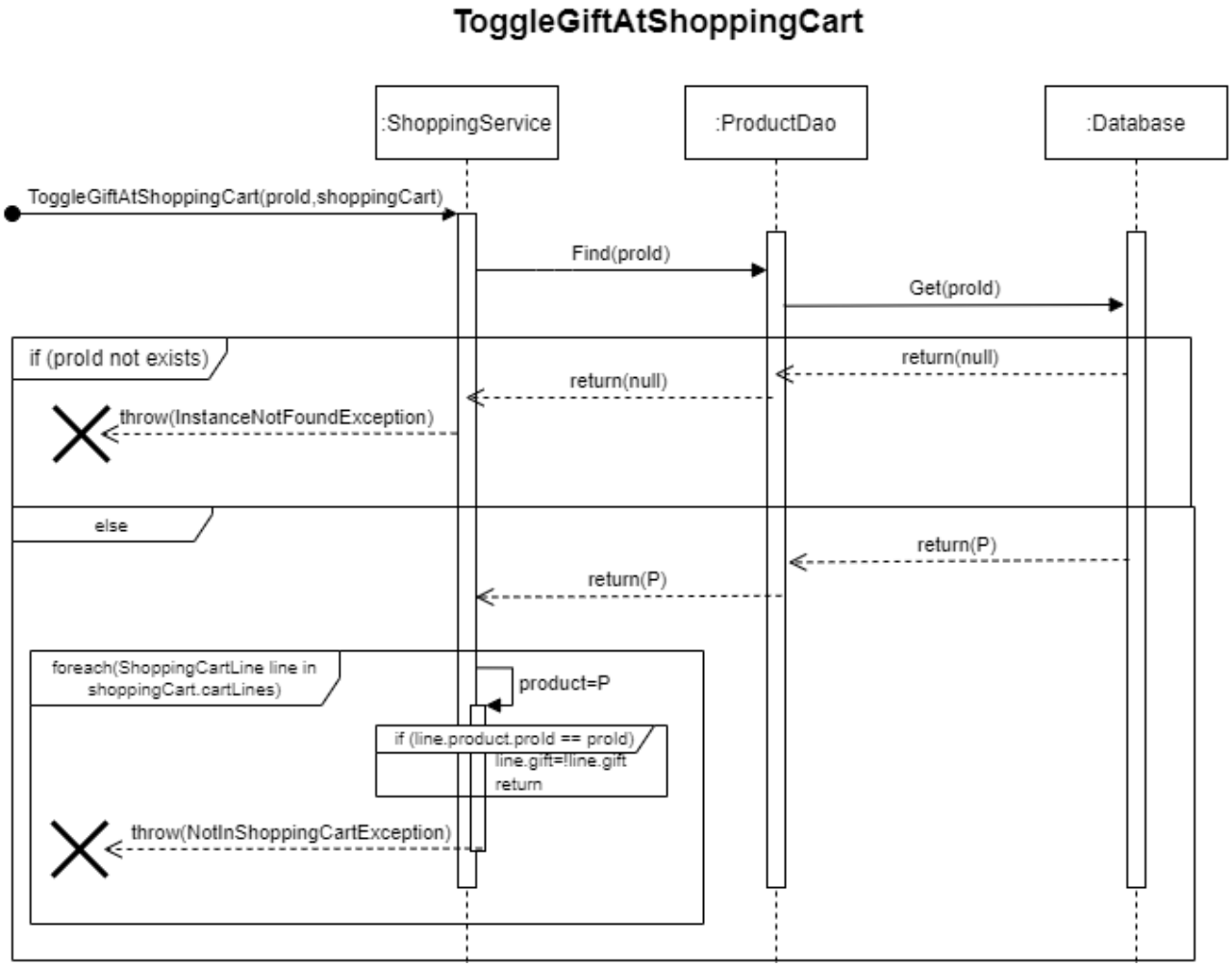
Se emplea la clase *GenericDao* de *ModelUtil*, que contiene métodos generales para todos los DAOs creados. A partir de ahí se han creado los DAOs correspondientes a cada entidad de la BD que heredan del *GenericDao*.

Si un DAO en concreto necesita algún método a mayores (como el representado en la imagen), se define en su interfaz y se implementa.

Diseño de un servicio del modelo

ShoppingService
+ CreateShoppingCart(): ShoppingCart
+ AddProductToShoppingCart(long prold, ShoppingCart shoppingCart): void
+ DeleteProductFromShoppingCart(long prold, ShoppingCart shoppingCart): void
+ AddOneToShoppingCart(long prold, ShoppingCart shoppingCart): void
+ RemoveOneFromShoppingCart(long prold, ShoppingCart shoppingCart): void
+ ToggleGiftAtShoppingCart(long prold, ShoppingCart shoppingCart): void
+ ShoppingCartIsEmpty(ShoppingCart shoppingCart): bool
+ EmptyShoppingCart(ShoppingCart shoppingCart): void
+ BuyOrder(long usrId, long cardPAN, string postalAddress, string orderDescription, ShoppingCart shoppingCart): long

Diseño de la estructura de un servicio del modelo.



Ejemplo de ejecución de un método del modelo.

Otros aspectos

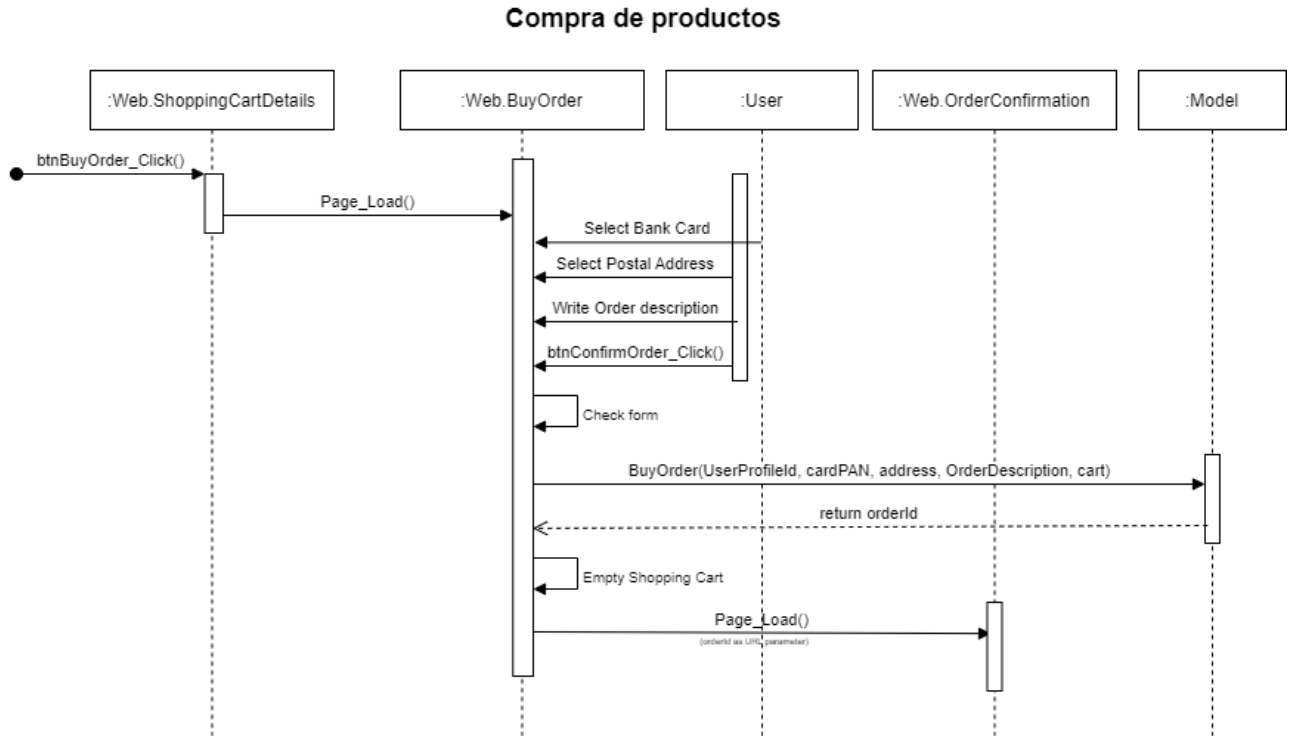
Scripts SQL: El paquete *Model* contiene la carpeta *Sql*, que a su vez contiene 2 scripts SQL:

- SqlServerCreateDatabase.sql: Genera la base de datos.
- SqlServerCreateTables.sql: Crea las tablas de la base de datos y añade filas por defecto.

La base de datos se llama *shopping*.

En el caso del paquete *Test*, la carpeta se llama *Scripts* y tan solo contiene un script SQL con todo el contenido de los dos anteriores pero para la base de datos *shopping_test*.

Interfaz gráfica



Ejemplo de compra de productos a través de la interfaz gráfica.

Funcionalidades adicionales

Comentario de productos

Para el diseño de esta parte, se ha decidido modelar de la siguiente manera la gestión de los comenatarios:

- Un producto puede tener de 0 a N comentarios.
- Un usuario puede publicar 0 a N comentarios.
- Un usuario puede comentar varias veces en un mismo producto.
- Un usuario puede editar un comentario (que le pertenezca).

En la interfaz web, para ver los comentarios de un producto se deberá entrar en la página de los detalles de producto y ahí acceder a la página de comentarios a través del botón con dicho texto para ver los comentarios acerca del producto.

Para añadir un comentario, se deberá clicar en el botón indicado dentro de la página de comentarios del producto.

Para editar un comentario, se debe clicar el botón indicado del comentario dentro de la lista de comentarios.

Etiquetado de comentarios

El diseño de esta parte se ha hecho de la siguiente forma:

- Un comentario puede tener de 0 a N tags.
- Un tag puede tener de 0 a N comentarios.
- No existen varios tags con el mismo nombre (no hay duplicados).
- No se pueden eliminar tags.

En la interfaz web existe un botón para acceder a la nube de tags, donde se muestran los tags existentes.

Se pueden añadir tags a un comentario cuando este se crea o edita.

Se pueden quitar tags de un comentario cuando se edita.

Compilación e instalación de la aplicación

Para una correcta ejecución de la aplicación, deben seguirse estos pasos en orden (se supone que el entorno está bien instalado):

1. Abrir la solución del proyecto en Visual Studio.
2. Ejecutar el script *SqlServerCreateDatabase.sql* dentro de *Model>Sql* (creación de la base de datos).
3. Ejecutar el script *SqlServerCreateTables.sql* dentro de *Model>Sql* (creación de las tablas e inserción de datos).
4. Compilar la solución del proyecto.
5. Ejecutar la página *MainPage.aspx* que se encuentra en *Web>Pages* (clic derecho > “Ver en explorador”).
6. * Otra forma de ejecutar la web es (a partir del paso 4) acceder a la URL: https://localhost:44386/Pages/MainPage.aspx

Problemas conocidos

Ejecución de los test: Los test funcionan todos correctamente tras ejecutarse el script .sql. Tras una ejecución de todo el proyecto *Test*, deberá ejecutarse el script .sql de nuevo.

Nube de tags: En la nube de tags, estos no tienen un tamaño de fuente en función del número de veces que aparezcan (todos tienen el mismo tamaño).

Comentarios: Por un despiste no se ha implementado en el proyecto Web la eliminación de comentarios (en el Model sí que está implementada).