Integración de Sistemas

Práctica .NET

Francisco Abel Cedrón Santaeufemia
francisco.cedron@udc.es

Nuria López Rivas
nuria.lrivas@udc.es

Abstract

En el presente documento se expone la práctica desarrollada por Francisco Abel y Nuria que se corresponde con la aplicación Web propuesta en el enunciado de la práctica 2 de Integración de Sistemas, así como algunos aspectos a tener en cuenta sobre la implementación realizada.

1. Arquitectura Global

En este apartado se comenta la estructura global de paquetes de la aplicación. A continuación se detallan los principales paquetes del proyecto, explicando brevemente lo que contienen:

- **Model:** Proyecto correspondiente a la capa modelo del proyecto. Incluye las clases persistentes (autogeneradas a partir de las tablas de base de datos diseñada), además de los siguientes paquetes:
  - **CommentDao**: Contiene la interfaz e implementación del DAO comentarios (clase persistente *Comment*, consúltese el aparato 2.1 para ver las clases persistentes).
  - **CommentService**: Contiene la interfaz e implementación del servicio dedicado a operaciones relacionadas con comentarios y sus etiquetas.
  - **Exceptions**: Contiene algunas excepciones que se podrían dar en cualquier servicio o DAO (más concretamente contiene la clase *EmptyStringException*).
  - **FavouriteDao**: Contiene la interfaz e implemetación del Dao de productos favoritos (clase persistente *Favourite*).
  - **FavouriteService**: Contiene la interfaz e implementación del servicio dedicado a operaciones relacionadas con la gestión de los productos favoritos de cada usuario).
  - **LabelDao**: Contiene la interfaz e implementación del DAO de etiquetas (clase persistente *Label*).
  - **ProductService**: Contiene la interfaz e implementación del servicio dedicado a las consultas sobre los productos [1].
  - **UserProfileDao**: Contiene la interfaz e implementación del DAO de usuarios (clase persistente *UserProfile*).
  - **UserService**: Contiene la interfaz e implementación del servicio dedicado a las operaciones sobre usuarios. También incluye un subpaquete con excepciones lanzadas por métodos de este servicio y un subpaquete con clases de utilidad para este servicio como por ejemplo la clase *PasswordEncrypter*.
  - **ValuationDao**: Contiene la interfaz e implementación del DAO de valoración (clase persistente *Valuation*).
  - **ValuationService**: Contiene la interfaz e implementación del servicio dedicado a las valoraciones que realizan los usuarios.
- **Test**: Proyecto de Test dedicado a las pruebas de integración de la capa modelo. Incluye una clase de test por cada servicio implementado en la capa modelo:
  - **ICommentServiceTest.cs**: Clase con los métodos de pruebas para el servicio de comentarios.
  - **IFavouriteServiceTest.cs**: Clase con los métodos de pruebas para el servicio de favoritos.
  - **IUserServiceTest.cs**: Clase con los métodos de pruebas para el servicio de usuarios.
  - **IValuationServiceTest.cs**: Clase con los métodos de pruebas para el servicio de valoración.
- **Web**: Proyecto correspondiente a la capa vista.
  - **App_GlobalResources**: Contiene recursos globales del proyecto web.
  - **App_LocalResources**: Contiene recursos locales de las páginas raíz del proyecto.
  - **Css**: Contiene hojas de estilo *css* utilizadas en la aplicación web.
  - **HTTP**: Incluye clases con funcionalidades para la aplicación (como la gestión de usuarios).
  - **ico**: Contiene el favicon de la applicación web.
  - **img**: Contiene las imágenes.
  - **js**: Incluye archivos javascript utilizados en la práctica.
  - **Pages**: Incluye las páginas de contenido de la aplicación, así como los paquetes de recursos locales para cada sección.

[1] En esta primera iteración la implementación está vacía cuya una función es lanzar una excepción de clase no implementada.

- **App_LocalResources**: Paquete con los recursos locales para las páginas del proyecto web incluidas en el paquete *Pages*.
- **Comment**: Contiene las páginas y archivos locales para los casos de uso orientados al servicio de comentatios.
- **Product**: Contiene las páginas y archivos locales para los casos de uso orientados al servicio de productos.
- **Valuation**: Contiene las páginas y archivos locales para los casos de uso orientados a las valoraciones.
- **User**: Contiene las páginas con los casos de uso orientados al servicio de usuarios y grupos de usuario (como, por ejemplo, el registro o la autenticación de un usuario).
- **Errors**: Páginas de error personalizadas.

- **XSLT**: Incluye los ficheros *xslt* para convertir los datos de los productos.

En la figura 1 se muestra la arquitectura global de paquetes de la aplicación mediante un diagrama de paquetes de UML.

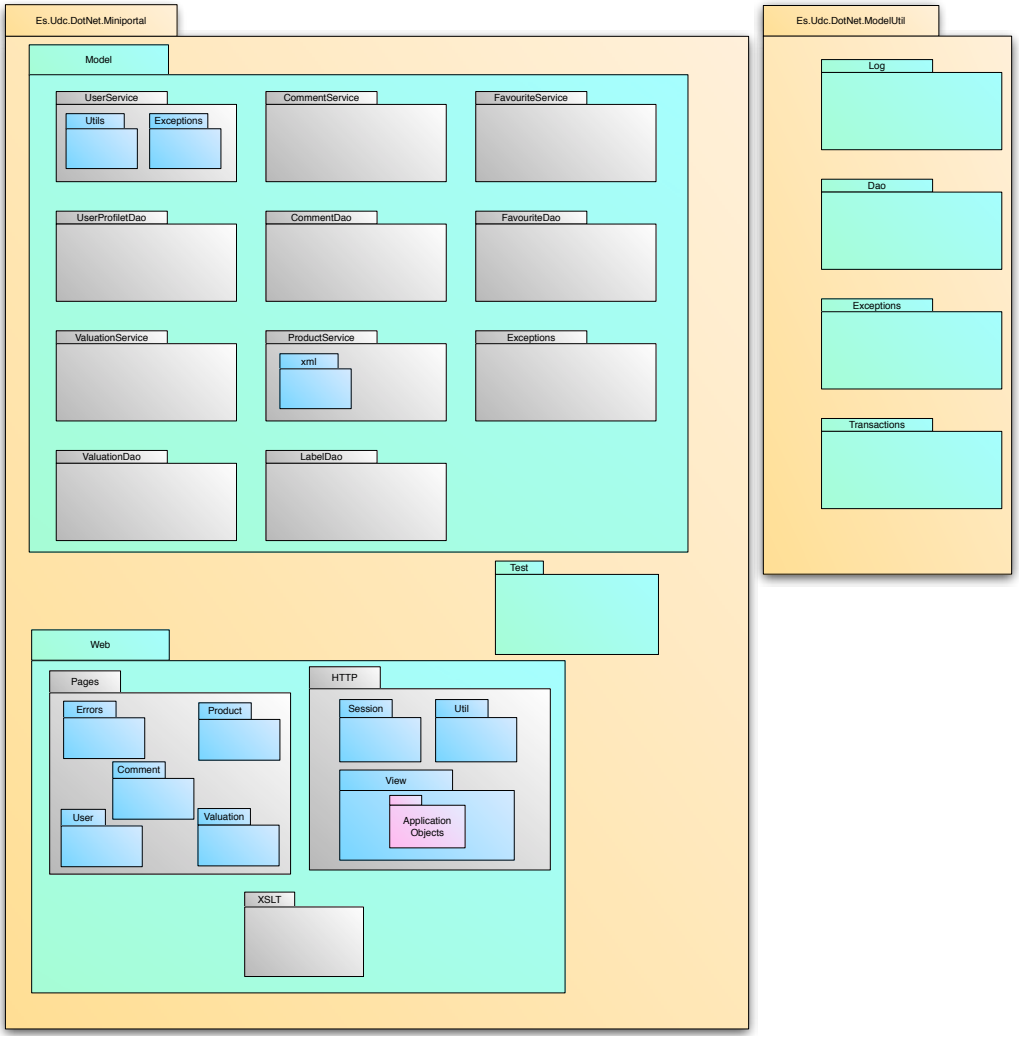

Figura 1: Arquitectura global de paquetes de la aplicación.

2. Modelo

En esta sección se muestran los detalles de diseño e implementación de la capa modelo de la aplicación.

2.1. Clases persistentes

En la figura 2, se muestran todas las clases persistentes modeladas, así como las relaciones relevantes entre ellas.

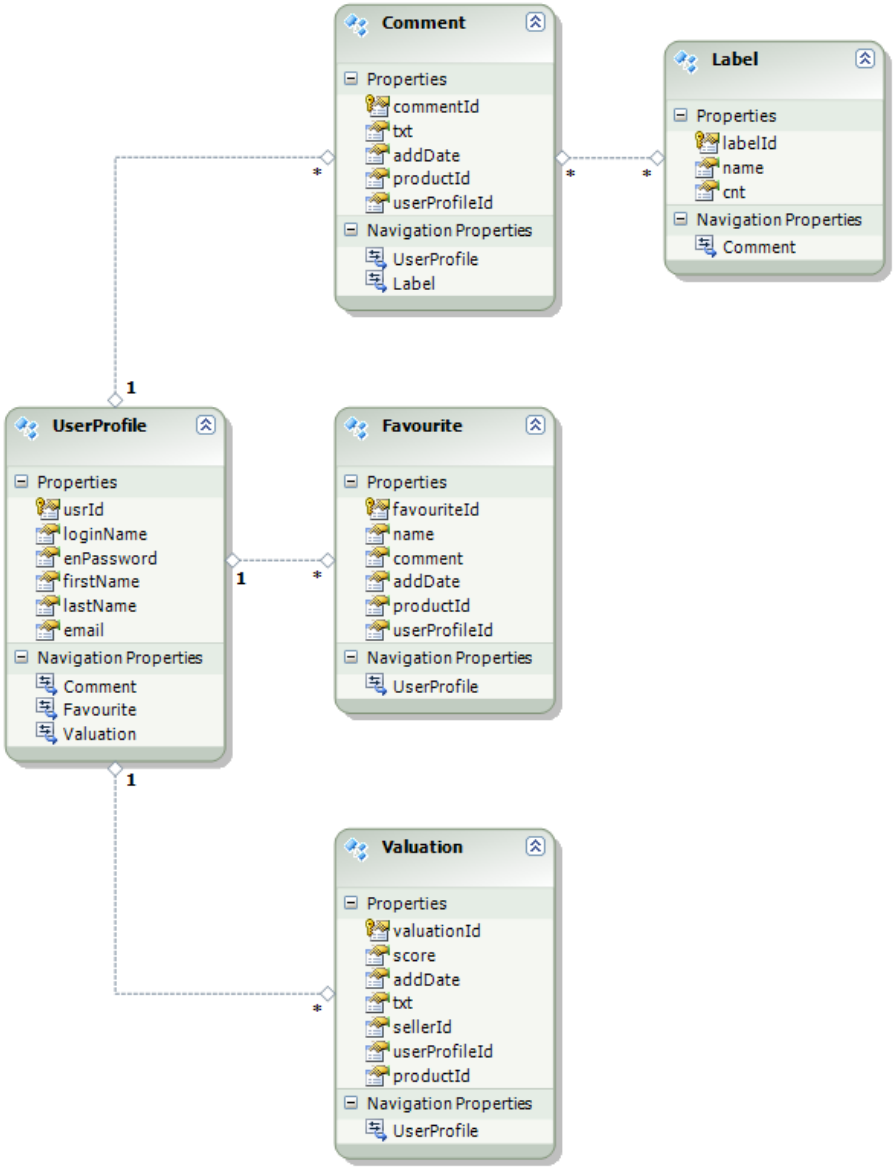

Figura 2: Diagrama de clases persistentes.

2.2. Interfaces de los servicios ofrecidos por el modelo

El criterio que se ha seguido a la hora de diseñar las interfaces de los servicios ha sido el de agrupar por funcionalidades relacionadas. Los criterios para agrupar los casos de uso son los siguientes: aquellos relacionados con la búsqueda de productos *ProductService*, la gestión los productos favoritos *FavouriteService*, los referentes a operaciones propias de un usuario en *UserService* y finalmente el resto de operaciones ofrecidas para tratar los comentarios así como las etiquetas de los mismos se agrupan en *CommentService*.

En la figura 3 se muestran las fachadas (interfaces de los servicios) ofrecidos por el modelo con la declaración completa de sus operaciones y los tipos de dato que utilizan.

Figura 3: Interfaces de los servicios ofrecidos por el modelo.

2.3. Diseño de un DAO

A continuación, se presenta un diagrama de clases que ilustra la implementación de uno de los DAOs de la aplicación (concretamente, del DAO de la entidad *Comment*) [2].

2.4. Diseño de un servicio del modelo

En este apartado se presenta un diagrama de clases con la implementación de un servicio del modelo, mostrando las dependencias con los DAOs que utiliza. Se ha elegido el servicio CommentService, puesto que se considera el más representativo y de mayor relevancia para la aplicación (en el sentido de que se trata de un sitio web de comentarios) [3].

[2] No se muestra el diseño del resto de DAOs, dado que su arquitectura es similar.

[3] Al igual que en el apartado anterior (correspondiente a los DAOs) tampoco se muestra el diseño del resto de servicios, dado que su arquitectura es similar.

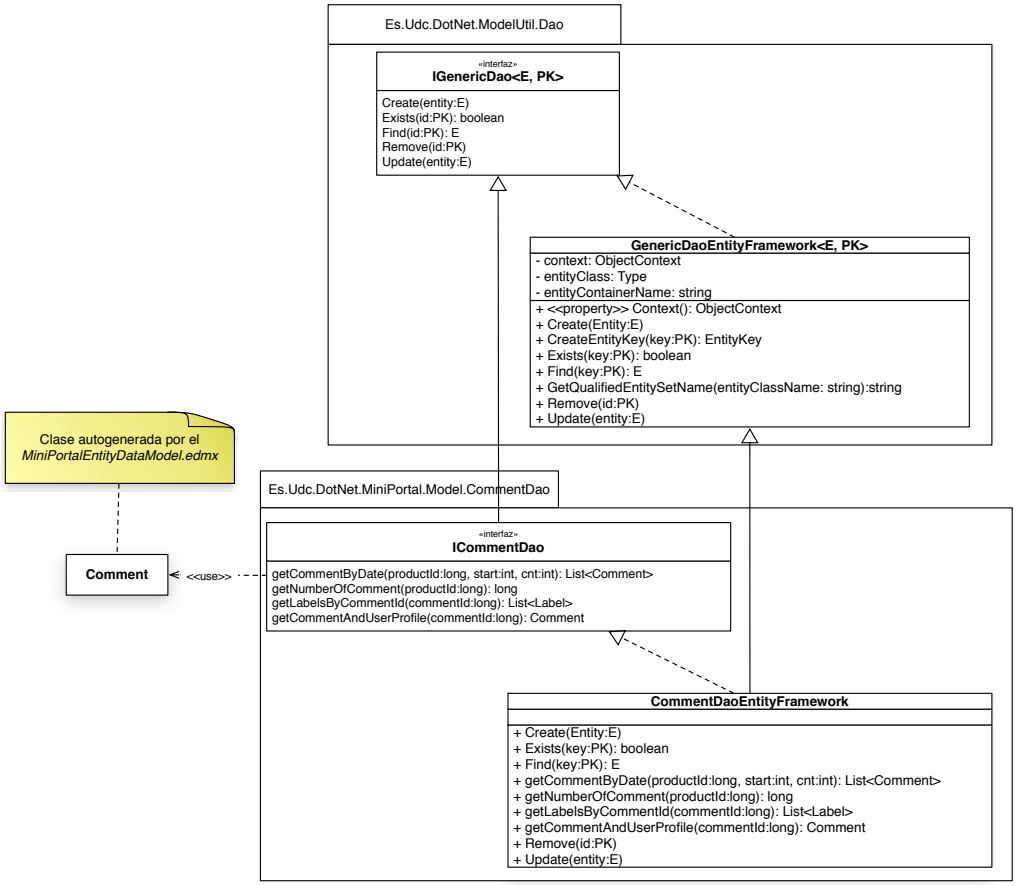

Figura 4: Diagrama de clases del DAO de la entidad *Comment*.

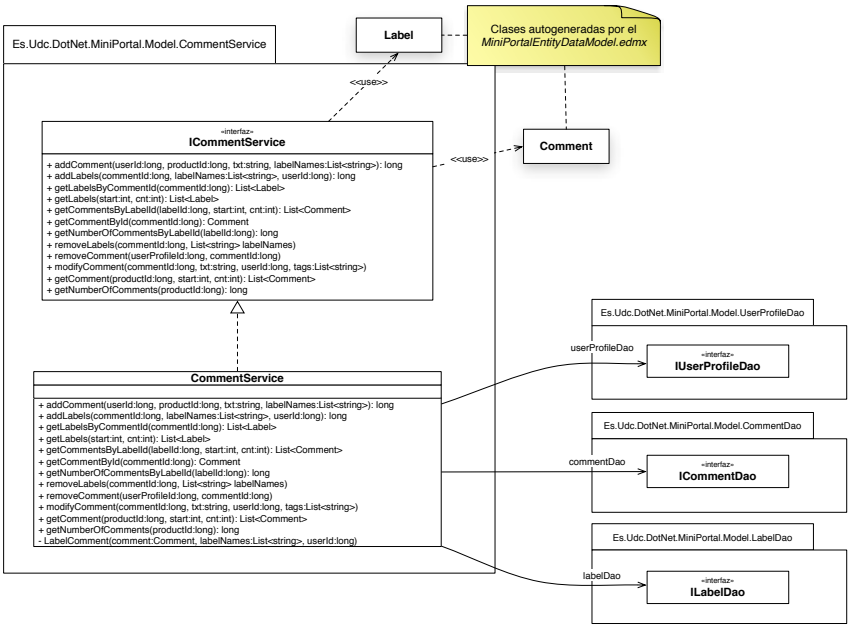

Figura 5: Diagrama de clases del servicio ofrecido por *CommentService*.

2.4.1. Diagrama de secuencia: Añadir comentario a un producto

En este apartado se expone un ejemplo de caso de uso que muestra la ejecución en la capa modelo. El caso de uso elegido ha sido el de *Añadir un comentario a un producto*, pues se considera representativo para la aplicación diseñada. A continuación se detallan los pasos realizados en este caso de uso (véase el diagrama de secuencia presentado a continuación para seguir los pasos en ejecución de este caso de uso con mayor facilidad).

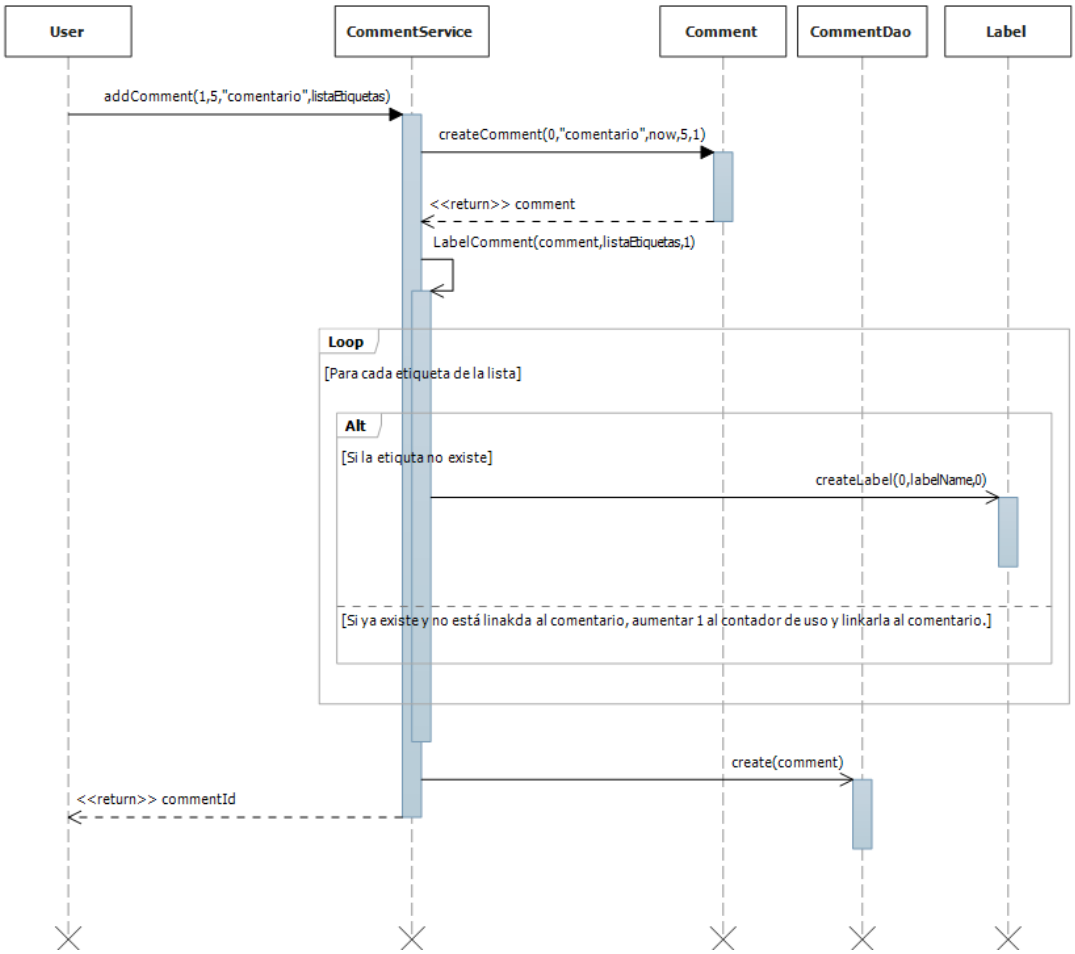

Figura 6: Diagrama de secuenta. Añadir un comentario.

2.5. Otros aspectos

En este apartado se documentan algunos aspectos particulares que se consideran relevantes para la correcta comprensión de la capa modelo diseñada.

2.5.1. Clases a medida

En algunos casos, en la capa web se requiere mostrar datos de más de una entidad. Devolviendo los objetos recuperados con *Entity Framework*, se recuperan los objetos correspondientes al resto de asociaciones realizando consultas adicionales contra la base de datos para cada vez que se requiere datos que estén asociados a la relación.

Para evitar esto, se pueden crear clases a medida que permiten obtener en una única consulta contra la base de datos la información necesaria de varias entidades.

Como apunte, cabe indicar que, como las entidades no poseen datos muy voluminosos se incluyen los datos de las entidades asociadas mediante *Include("EntidadAsociada")* para todos aquellos servicios que necesitan conocer algún valor de la relación de un objeto.

La única "clase" a medida es *AverageAndNumberOfValuations* que es una estructura para almacenar la puntuación de un vendedor y el número total de valoraciones que recibió hasta la fecha.

2.5.2. Caché de contadores

Se ha implementado contadores caché para evitar el uso de la sentencia *Count* contra la base de datos. De esta forma, cuando se necesite conocer el número total de datos asociados a una entidad, se recupera a través de un atributo que hace de contador incluido en la propia entidad.

El uso de esta caché se ha empleado en la entidad *Label* para poder conocer de antemano el número total de comentarios por cada etiqueta. Esto implica tener que realizar los controles necesarios desde *CommentService* en los métodos que añaden y eliminan etiquetas asociadas a comentarios.

2.5.3. Servicios XML

Para obtener los productos del servicio REST ofrecido por la aplicación web de subastas desarrollado en la primera parte de la asignatura, se añaden varios métodos en el servicio de productos dentro de la capa modelo. Concretamente dos métodos para obtener los ficheros XML con los datos de los productos, uno para la búsqueda de productos por palabras clave y otro para conocer los detalles de un producto concreto.

A continuación se detallan los métodos del servicio *ProductService*:

```
1 List<Product> searchProductByKeywords(string keywords, int start, int size);
```

Recupera el listado de los productos *XML* y lo parsea utilizando *XmlDocument*.

```
1 XPathNavigator searchProductByKeywordsXML(string keywords, int start, int size);
```

Devuelve un *XML* de tipo *XmlDocument* con el listado de los productos seleccionados a partir de una lista de palabras clave.

```
1 int getNumberOfProducts(string keywords);
```

Recupera a partir del *XML* el número total de productos.

```
1 XPathNavigator searchProductByIdXML(long productId);
```

Devuelve un *XML* de tipo *XmlDocument* con los detalles de un producto concreto.

```
1 string getUrl2Bid(int productId);
```

Devuelve la url con la que se puede pujar por un producto.

3. Interfaz gráfica

En esta sección se muestran los detalles de diseño e implementación de la capa vista de la aplicación.

3.1. Diagrama de secuencia: Añadir comentario a un producto

En este apartado se expone un ejemplo de caso de uso que muestra su ejecución en la capa vista. El caso de uso elegido ha sido el de *Añadir comentario a producto* [4] , pues se considera representativo para la aplicación diseñada. A continuación se detallan los pasos realizados en este caso de uso (véase el diagrama de secuencia presentado a en la figura 7 para seguir los pasos en la ejecución de este caso de uso con mayor facilidad).

3.2. Otros aspectos.

En esta sección se documentan los aspectos particulares que se consideran relevantes para la correcta comprensión de la capa web.

[4] Recuérdese que en un apartado previo 2.4.1 se presentó el diagrama de secuencia en la capa modelo de este mismo caso de uso, lo cual permitirá obtener una visión completa de la ejecución de dicho caso de uso

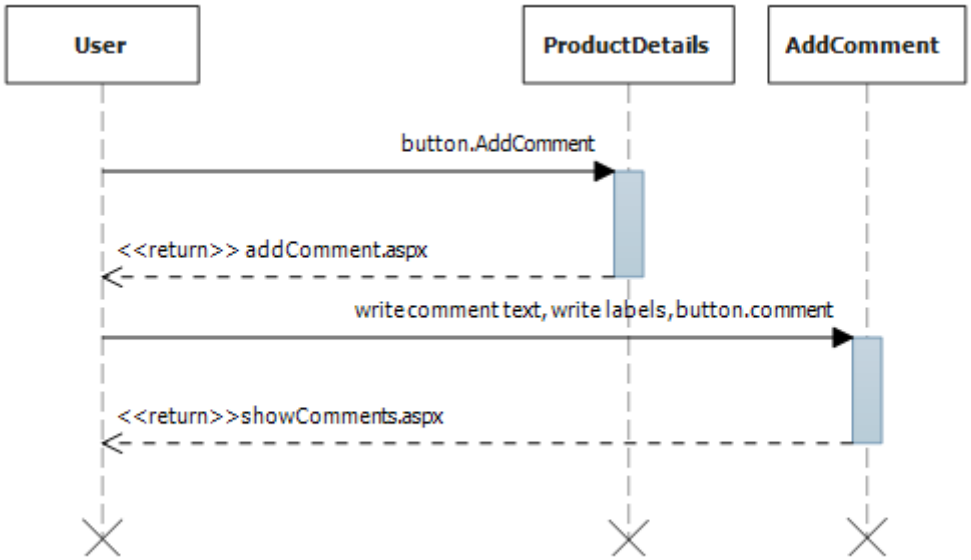

Figura 7: Diagrama de secuenta. Añadir un comentario.

3.2.1. Control *GridView*

El control *GridView* ofrece la posibilidad de listar los datos en formato tabla y soportando múltiples opciones como la posibilidad de control de eventos por fila y edición de los datos.

Para la paginación en los *GridView* se ha implementado un método que devuelve los enlaces anterior y siguiente y un par de páginas próximas a la actual. Aunque el *PagedDataSource* es implementado por los *GridView* se ha decidido no emplearlo por su complejidad y la opción de los enlaces permite seleccionar sólo los datos necesarios en cada petición de una forma muy simple.

Para realizar la personalización del listado se añade la llamada a la función en el *GridView* OnRowDataBound="DataBoundRecord". La función en el *Code-Begind* que realiza las siguientes comprobaciones se muestra a continuación.

```
1  protected void DataBoundRecord(object sender, GridViewRowEventArgs e)
2  {
3      if (e.Row.RowType == DataControlRowType.DataRow)
4      {
5          HyperLink linkMod = (HyperLink)e.Row.FindControl("lnkButtonModify");
6          LinkButton linkDel = (LinkButton)e.Row.FindControl("lnkButtonDelete");
7          Comment comment = (Comment)e.Row.DataItem;
8
9          linkMod.NavigateUrl = String.Format("~/Pages/Product/AddComment.aspx?comment={0}&id
               ={1}", comment.commentId,id);
10
11         if (userId != comment.userProfileId)
12         {
13             linkDel.Visible = false;
14             linkMod.Visible = false;
15         }
16     }
17
18 }
```

También es posible crear *GridView* que soporten las operaciones *CRUD*. Un ejemplo donde se puede ver el uso de este tipo de *GridView* se puede ver en el manejo de favoritos, donde un usuario puede eliminar un producto de su lista de deseos. En este caso de eliminación se se puede personalizar una columna con *TemplateField*. Así es posible por ejemplo definir la acción de un control. A continuación se muestra este ejemplo de eliminar un producto de la lista de favoritos del *GridView*:

```
1  <asp:TemplateField>
```

```
2       <ItemTemplate>
3           <asp:LinkButton ID="lnkButton" runat="server"
4               commandName="Delete" CssClass="btn btn-danger"
5               OnClientClick=<%# String.Format("return confirm('{0}');",
                    GetLocalResourcesObject("Comfirm_Delete")) %>
6
7              <span class="icon-trash icon-white">
8               </span>
9                
10              <asp:Localize ID="lclDelete" runat="server"
11                    Text="<%$ Resources:Common, Delete %>" />
12          </asp:LinkButton>
13      </ItemTemplate>
14  </asp:TemplateField>
```

La opción *OnClientClick* realiza una petición de javascript sobre el navegador del cliente solicitando una confirmación con el texto del recurso global *PONER AQUI LA CLAVE DEL ARCHIVO DE I18N*.

La opción *CommandName* define que tipo de acción implica el enlace definido.

Finalmente, el método del evento en el dato servidor se define en un atributo del elemento *GridView*: OnRowDeleting=”DeleteRecord” junto con el identificador del producto que va a eliminar DataKeyNames=”favouriteId”.

En el método *DeleteRecord* se obtiene el identificador a partir del objeto GridViewDeleteEventArgs

Con el identificador devuelto se realiza la llamada al método del servicio y finalmente se recargan los nuevos datos en el *GridView*.

3.2.2. XSLT

Los servicios web de productos a partir de ficheros *XML* remoto son procesados por el traductor *XSLT* que ayuda a la transfomación de *XML* facilitando su presentación en las páginas *ASP.NET*.

Tranformación *XSLT* del listado de productos.

El *XML* del listado de productos se procesa y se transforma en una tabla *HTML*.

Para obtener los productos se obtiene una única regla que recupera los nodos *product* a partir del nodo raíz *products*. Cada nodo *product* contiene lso atributos de nombre, categoría, precio y tiempo que resta para que finalice la subasta, el subastador entre otros.

Transformación *XSLT* de los detalles de un producto.

El *XML* de los detalles de producto es procesado y transformado en una tabla de *HTML*. Al igual que antes existe un elemento raíz *products* que en este caso solo contendrá un único elemento [5] *product* que contiene los detalles del producto en sus atributos.

Internacionalización en el fichero *XSLT*.

Para añadir *i18n* se añaden los valores de los recursos globales como parámetros al *XSLT* desde el *Code-Behind* de las páginas con *XsltArgumentList*.

Paginado.

También se contempla la paginación en el listado de los productos. Para ello se hace uso del atributo *total* que está en el nodo raíz (*products*) del fichero *XML* que indica el número de productos que cumplen el criterio de búsqueda especificado.

Gracias a ese atributo, podemos tener una paginación igual que la que se obtiene mediante el *GridView*.

4. Funcionalidad adicional

En este apartado se explica cómo se ha realizado cada parte adicional.

4.1. Parsing de XML

Se incorpora al formulario de búsqueda de productos la opción de especificar si la búsqueda debe incluir la información sobre los votos que recibió o no. Cuando un usuario especifica que se incluya información de los votos en el resultado de la búsqueda, por cada usuario que aparezca se mostrará adicionalmente el número de votos realizados sobre él.

El número de valoraciones sólo se mostrará si existen valoraciones para ese vendedor en concreto. Su implementación se explica en los siguientes apartados.

[5] En el caso de que se busque un producto que no exista se devolverá el nodo raíz *products* sin ningún nodo hijo.

4.1.1. Capa modelo

Para la implementación del parsing *XML* se incluye el método *searchProductByKeywords*. Se realiza la búsqueda de productos con la clase *XmlDocument* y el método *GetElementsByTagName* con el que se obtiene una lista de nodos *XmlNodeList*. Por cada nodo se crea un objeto de tipo Product, el cual contiene nombre, vendedor, categoría, precio y minutos para finalizar la subasta. Durante la creación se realizan las siguientes operaciones:

- El identificador de cada subastador se obtiene a partir del fichero *XML*.
- El número total de valoraciones se obtiene de la propia aplicación haciendo uso del servicio *ValuationService* con el método *getAverageAndNumberOfValuations* que recibe por parámetro el identificador de cada vendedor.

Para poder realizar el paginado se añade al servicio el método *getNumberOfProducts* que comprueba en el archivo *XML* (usando *XmlDocument* y *XPath*) si existe el atributo *total* dentro de la etiqueta *products*.

4.1.2. Capa web

En la página *aspx* se muestra la información de los productos a través de un *Repeater* en lugar de usar el XSLT. Los enlaces para añadir comentario, añadir favorito y valorar incluyen un parámetro en la URL con el identificador del producto mientras que el enlace de ver valoraciones incluye el usuario del vendedor en la URL.

4.2. Etiquetado de comentarios

En este apartado se especifica la funcionalidad de etiquetado de comentarios. En el momento de añadir un comentario el usuario podrá añadir una serie de etiquetas. Se podrán utilizar etiquetas ya existentes o incluir etiquetas nuevas. La aplicación es capaz de controlar, a través de los servicios de la capa modelo, si una etiqueta existe o debe crearse. Además, para cada comentario se realiza la comprobación de que la etiqueta no esté ya añadida al mismo. Para los comentarios ya etiquetados, se podrá "desetiquetar" una o varias etiquetas ya asociadas y también se permitirá añadir nuevas etiquetas.

Las etiquetas se pueden añadir en el momento de crear un comentario. Deben ir separadas por comas y pueden incluír espacios en blanco. Es posible añadir un comentario sin asignarle etiquetas, pudiendo asignárselas más tarde a través del botón "modificar" en la lista de comentarios.
Cuando se pulsa este botón, en la página a la que redirige, se cargan los datos que había previamente, tanto el texto del comentario como el de las etiquetas, en caso de que exista. Si se desean añadir más etiquetas, basta con seguir añadiéndolas, separadas por comas, a las ya existentes. Si se quieren eliminar de ese comentario, lo único que hay que hacer es borrar el texto de las etiquetas que ya no se quieran. Al pulsar el botón "comentar" se actualizarán las etiquetas ligadas a ese comentario.

4.2.1. Contador caché en etiquetas

Para evitar peticiones pesadas contra el SGBD, se implementa un contador caché sobre la entidad *Label*, el cual contabiliza el número de veces que aparece una etiqueta en los comentarios. Es en la propia implementación del servicio (*CommentService*) donde se tiene en cuenta el incremento o decremento del contador cuando se agrega o elimina una etiqueta concreta de un grupo de usuarios.

5. Compilación e instalación de la aplicación

En este apartado se explica cómo compilar e instalar la aplicación, asumiendo un entorno correctamente configurado.

En primer lugar, se debe configurar y crear la base de datos desde *Visual Studio 2010* con la solución del proyecto abierta. A continuación se condiguran los ficheros necesarios para el acceso y los valores de los datos de acceso al servidor de aplicaciones *Tomcat*. Estos pasos se detallan en la siguiente lista:

1. Crear la base de datos desde *Server Explorer* → *Data conections* → *Create New SQL Server Database*.... En la ventana seleccionar el nombre del servidor *"localhost\SQLExpress"*.

2. Ir al proyecto *Model* y en la carpeta *Sql* abrir *SqlServerCreateTables.sql* y sobre el dicho archivo pulsar con el botón derecho del ratón y seleccionar *Execute SQL*. Lo que hará sera ejecutar el script sobre la base de datos PrácticaIS.

3. Modificar el nombre del esquema de usuario de la base de datos en el proyecto *Model* → *MiniPortalEntityDataModel.edmx*. En el ejemplo, el esquema de usuario de BBDD es *dbo*.

```
1 <EntitySet Name="Comment" EntityType="Es.Udc.DotNet.MiniPortal.Model.Store.Comment"
      store:Type="Tables" Schema="dbo" />
2 <EntitySet Name="Favourite" EntityType="Es.Udc.DotNet.MiniPortal.Model.Store.
      Favourite" store:Type="Tables" Schema="dbo" />
3 <EntitySet Name="Label" EntityType="Es.Udc.DotNet.MiniPortal.Model.Store.Label"
      store:Type="Tables" Schema="dbo" />
4 <EntitySet Name="LabelComment" EntityType="Es.Udc.DotNet.MiniPortal.Model.Store.
      LabelComment" store:Type="Tables" Schema="dbo" />
5 <EntitySet Name="UserProfile" EntityType="Es.Udc.DotNet.MiniPortal.Model.Store.
      UserProfile" store:Type="Tables" Schema="dbo" />
6 <EntitySet Name="Valuation" EntityType="Es.Udc.DotNet.MiniPortal.Model.Store.
      Valuation" store:Type="Tables" Schema="dbo" />
```

4. Modificar el nombre de la conexión a la base de datos en el proyecto *Model* → *App.config*, el proyecto *Test* → *App.config* y en el proyecto *Web* → *Web.config*.

```
1 <connectionStrings>
2     <add name="MiniPortalEntitiesContainer"
3          connectionString="metadata=res://*/MiniPortalEntityDataModel.csdl|res://*/
               MiniPortalEntityDataModel.ssdl|res://*/MiniPortalEntityDataModel.msl;
               provider=System.Data.SqlClient;provider connection string=&amp;quot;data
               source=localhost\SQLExpress;initial catalog=PracticaIS;integrated
               security=True;multipleactiveresultsets=True;App=EntityFramework&amp;quot;"
4          providerName="System.Data.EntityClient" />
5
6 </connectionStrings>
```

5. Añadir los parámetros para el acceso al servicio web de subastas. Dentro de las propiedades del proyecto *Model* (botón derecho sobre el proyecto y la opción *Settings*) se debe configurar los parámetros de los servicios y páginas *XML*, además de los parámetros de búsqueda:

   - **XmlRemoteHost**: Especifica el servidor web remoto.
   - **XmlProductList**: Especifica la ruta hacia el servicio *REST* de búsqueda de productos.
   - **XmlProduct**: Especifica la ruta hacia el servicio *REST* para conocer los detalles de un producto.
   - **BidRemoteHost**: Especifica la ruta hacia la puja de un producto.
   - **DefaultPageSize**: Valor de elementos que se deben de recuperar en caso de que no se reciba un valor específico.

6. Seleccionar el proyecto *Web* como proyecto de inicio (seleccionamos el proyecto y con el botón derecho del ratón se selecciona la opción *Set as StartUp Project*).

7. Seleccionar la página principal en el proyecto *Web* → *Pages* → *MainPage.aspx* (click con el botón derecho sobre el archivo y seleccionar *Set as Start Page*).

8. Finalmente, para su ejecución en desarrollo pulsar *Control + F5* lo que iniciará el servidor por defecto y accederá a la página de inicio del proyecto web.

Índice de figuras