MEMORIA DE TRABAJO. APPESTADIO.

Diseño de sistemas multimedia. 2013-2014.

Autores:
Rubén Martínez Vilar
Pablo Marzal Garrigós
Fco. Javier Martínez Ruiz

DESCRIPCIÓN

Empecemos describiendo en que consiste nuestra aplicación, **AppEstadio**.

Es una aplicación capaz de gestionar la venta de productos de la tienda de un estadio de fútbol al igual que sus entradas y sus abonos.

La aplicación es accesible para aquellos clientes que no estén abonados también, pudiendo éstos realizar compras o abonarse.

La finalidad es la de ayudar en la gestión del campo y reducir el número de empleados de este, permitiendo de este modo la propia gestión de los usuarios en terminales por el mismo estadio o incluso por internet.

IMPLEMENTACIÓN

Vamos a ver un resumen de lo realizado en las distintas fases del desarrollo del proyecto junto a los problemas que hemos encontrado y como los hemos solucionado.

FASE 1. DISEÑO.

Mockups

Empezamos realizando los mockups (bocetos) de nuestra aplicación utilizando la herramienta "Balsamiq Mockups", este paso es necesario en todas aquellas aplicaciones que van a ser iniciadas. La realización de mockups nos proporciona una visión general de los campos que nuestra aplicación va ser capaz de cubrir así como el propósito de la propia aplicación.

En nuestro caso llegamos a la conclusión de que los usuarios iban a poder administrar la compra de entradas y de abonos, tratandolos como productos de la tienda.

Estos dos productos esenciales, son tratados de igual forma que productos en la estructura interna de la aplicación, no obstante para el público decidimos que había que separarlo de la tienda, ya que estos productos necesitan una mayor especificación, resaltando de este modo su importancia.

Gracias a los mockups también pudimos determinar que acciones iban a desempeñar los administradores en nuestra aplicación. En nuestr aplicación los administradores pueden agregar productos a la tienda, entradas o abonos así como también pueden modificar sus precios y características.

FASE 2. LÓGICA DE NEGOCIO

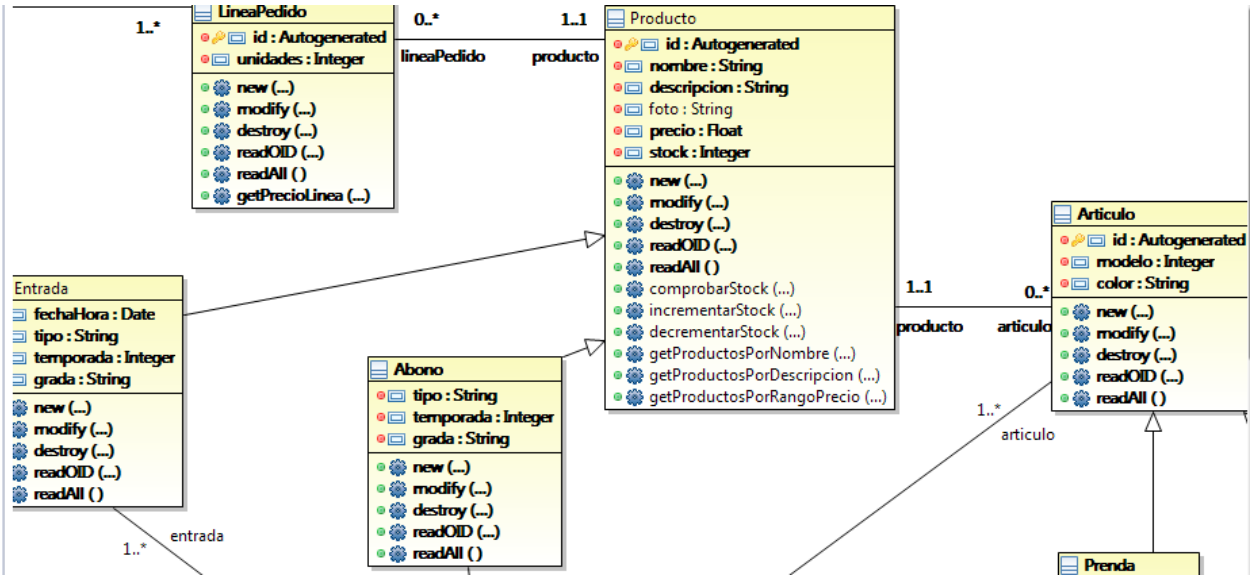
Modelo gráfico del modelo de dominio con OOH4RIA

El modelo de dominio de la aplicación AppStadio fue realizado a mediante la herramienta de modelado gráfico que nos proporcionaba la extensión para eclipse, OOH4RIA.

El modelado fue realizado con éxito pero debemos destacar algunos problemas que tuvimos durante este proceso.

En primer lugar tuvimos que dividir la tabla de "tallas" asociada a los Artículos a la tienda, para que cada producto tuviera su propia talla, ya que no todos los productos comparten la misma tipología a la hora de clasificar sus tallas.

Otro problema fue la implementación de las clases "Entrada" y "Abono", las cuales decidimos sacar fuera de los artículos de la tienda, ya que era necesario por los atributos distintos que estas clases presentan. Finalmente decidimos crear una clase "Producto" del cual heredan las clases "Entrada" y "Abono" y a su vez los artículos de la tienda tienen una relación de existencia hacía esta tabla.



Problemas encontrados:

- No pensamos que haría falta una factura, pero al final fue necesaria
- Nos costó encontrar la manera de dividir los tipos de productos entrada, abono y tienda.

IMPLEMENTACIÓN DE LA BASE DE DATOS

Esta parte del desarrollo de la aplicación fue creada mediante la herramienta Visual Studio, importamos el proyecto generado en OOH4RIA, y empezamos a trabajar en esta plataforma para esta nueva fase del proyecto.

Tuvimos algunas complicaciones en la obtención de las id de las clases, ya que no almacenamos de manera correcta las id creadas lo que nos dificultaba la comprobación de los distintos métodos ya que la mayoría de ellos necesitaban de una id.

Este problema fue solucionado sin mucha complicación, y pudimos realizar las comprobaciones adecuadas para determinar si las clases de nuestro modelo realizaban los métodos de manera correcta.
En esta parte del desarrollo también determinamos cuáles de estos métodos iban a ser consideradas CP, las cuales desarrollamos en la siguiente fase del proyecto.

FASE 3. IMPLEMENTACIÓN FINAL

De esta fase vamos a destacar lo primero el uso de patrones de programación.

PATRÓN SINGLETON

La clase **SessionManager** implementa un **patrón Singleton** para mantener objetos de sesión como son el **usuario** identificado en el sistema y el **carro de la compra**, que puede tener items aunque no haya usuario identificado (pero no permite finalizar la compra en ese caso).

Para la implementación de este patrón hemos seguido el segundo ejemplo del extracto del libro "C# in depth":
http://csharpindepth.com/Articles/General/Singleton.aspx#lock

MODEL VIEW PRESENTER

Como requisito exigido para la práctica hemos implementado el patrón model view presenter. Este patrón toma más sentido en algunas vistas que utilizan el mismo presenter:

- Las pantallas de login y configuración de perfil hacen uso del mismo PresenterUsuario.
- PresenterPedido agrupa las vistas en las que se listan pedidos, como MisCompras y Pedido Pendientes de administrador.

Podemos encontrar algún ejemplo más en la aplicación, aunque en muchos casos no pudimos agrupar vistas en un mismo presenter.

PROBLEMAS FINALES

Como punto negativo hemos de decir que en nuestro caso el aspecto visual de la aplicación ha quedado un poco discreto, hemos utilizado algunos estilos globales, que podemos encontrar en app.xaml. Estos estilos se centran básicamente en establecer el tipo de tipografía para los distintos controles.

Por otro lado tenemos que decir que somos un grupo de solo 3 personas, por lo que hemos hecho lo que hemos podido en las últimas semanas del desarrollo, además la parte de la tienda nos llevó mucho más tiempo del que pensábamos, en gran parte por la gestión de los pedidos, el carro, stock, etc.