DecorateYourHouse Memoria

Universidad Politécnica de Valencia. Escuela Técnica Superior de Ingeniería Informática.
Autor: Cristian Miguel Pérez Matturro. Director: Vicente Pelechano Farragud.
Ingeniería Técnica en Informática de Sistemas.
Código proyecto final de carrera DSIC-164.
Valencia, Julio 2012


DECORATE
YOUR HOUSE

Índice


DECORATE
YOUR HOUSE

1. Introducción

Este documento es la memoria del proyecto final de carrera correspondiente al curso 2011-12 de Ingeniería Técnica en Informática de Sistemas, cursada en la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Valencia.

1.1 Objetivo del proyecto

El proyecto tiene como meta el desarrollo de un portal web a modo de tienda virtual que permita la venta de vinilos decorativos y fotomurales a nivel nacional. La tienda, en adelante *Decorate Your House*, deberá estar basada en el mismo modelo que han seguido otros competidores del sector. Esto es, deberá estar preparada para albergar un amplio catálogo de productos que se organizará en categorías. Otras características típicas serán el carrito de compras, registro de usuarios, seguimiento de pedidos, integración con sistemas de pago (Paypal), promociones (envío de boletines informativos, banners), interfaces de administración para llevar una óptima gestión de la tienda, etc. La venta de vinilos decorativos por medio de la web tiene ya un tiempo en circulación. Algunas empresas ya cuentan con un simulador que permite apreciar como sería el resultado de la aplicación del vinilo sobre la pared. Por tanto, otro de nuestros objetivos será el de elaborar nuestro propio simulador que deberá ser lo más realista posible y estará integrado con el portal.

1.2 Contexto

Los vinilos decorativos y los fotomurales se han impuesto indiscutiblemente como dos de los productos que más interés despiertan entre interioristas, diseñadores, arquitectos y clientes. Sus diseños espectaculares, la facilidad de colocación, gran durabilidad y sus precios asequibles le han otorgado ventaja respecto a opciones más tradicionales como pueden ser los papeles pintados o la pintura sobre pared. [2] Y es que por eso que cada vez son más las empresas dedicadas a la fabricación y distribución de estos elementos decorativos.

1.3 ¿Qué son los vinilos?

El vinilo o técnicamente llamado PVC es un material plástico que lleva una cola adhesiva por la parte trasera que nos permite pegarlo en casi cualquier tipo de superficie lisa pero no en paredes de gotelé, por ejemplo. [2] Aunque en principio lo más normal era encontrarlos en locales de copas y los comercios más elegantes o a la moda, lo cierto es que los vinilos han

dado el salto a las viviendas de jóvenes y no tan jóvenes que buscan un estilo contemporáneo. Hoy en día es habitual colocarlos sobre paredes de dormitorios o salones, muebles, azulejos, espejos o incluso en la mampara de la ducha.
Sus orígenes se remontan justamente en el papel pintado pero al contrario de su antecesor, los vinilos son más sencillos de preparar, vienen preencolados y además cuentan con la ventaja de estar formados por una capa de plástico que permite su lavado con agua sin que se estropeen. [1]

Los vinilos vienen en acabado brillo o mate aunque es más frecuente el uso de acabado mate para evitar que las luces de casa no se reflejen en el adhesivo. La forma de compra preferida por los clientes a la hora de adquirir un vinilo es el comercio electrónico especializado por la gran oferta de motivos y colores. Además el propio cliente puede encargarse de su colocación. Se trata de un proceso proceso rápido, intuitivo y fácil.

1.4 A la vanguardia en decoración: fotomurales

Los fotomurales son vinilos decorativos que se diferencian de estos en su gran tamaño -suelen cubrir una pared o pasillo enteros- y en la forma de su elaboración: al contrario de los vinilos decorativos que son vinilos de rotulación y para los cuales se emplea un plotter, los fotomurales son vinilos de impresión digital. Llevan impreso una fotografía a gran escala que puede ser desde una panorámica de tu ciudad preferida hasta un plato de comida apetitoso. Los fotomurales se diseñan a medida. Aunque son fáciles de aplicar, requieren algo más de habilidad que en el caso de los vinilos decorativos.
Suelen presentarse en dos texturas: liso o lienzo. Es frecuente encontrarlos en centros comerciales o sitios públicos, así como en exposiciones y ferias. [3]

2. Especificación de requisitos

2.1 Introducción

2.1.1 Propósito

El objetivo de la especificación es el de definir de forma clara y precisa todas las funcionalidades y restricciones demandadas para construir el portal web.

2.1.2 Ámbito

Una empresa de publicidad y rotulación quiere expandir su portafolio de productos abarcando el área de la decoración interior y para ello quiere abrir una tienda online en donde ofrecer su catálogo de vinilos. La tienda se basará en un CMS ultra ligero de creación propia que carezca de instalación y/o configuración. El CMS hará principal hincapié en el apartado de presentación y la usabilidad.

2.1.3 Definiciones, acrónimos y abreviaturas

Vinilo decorativo: Pegatina autoadhesiva hecha de un material plástico que se adhiere por encima de las paredes, techos o pisos.

Fotomural: Fotografía gigante impresa en alta definición sobre vinilo. Son removibles y además pueden encontrarse en modelos traslúcidos.

Rotulación: Arte de escribir letras y números siguiendo unas normas preestablecidas.

Decoración interior/interiorismo: Es una actividad profesional de diseño orientada a procurar, como servicio a la sociedad, la más idónea resolución del entorno habitable del hombre, mediante la aplicación de determinados elementos y normas básicas de diseño, técnicas funcionales, estéticas, ambientales, psico-sociales, sensoriales, económicas y legales, con objeto de mejorar la calidad de vida de los usuarios. [4]

Anónimo: Cualquier usuario que ingresa al portal sin introducir sus credenciales.

Cliente: Usuario que ha completado el proceso de registro y se encuentra habilitado para realizar compras.

Administrador: Usuario con privilegios que puede acceder a las páginas de gestión del sitio web para realizar un seguimiento de pedidos, clientes, productos..etc.

Área de clientes: Conjunto de páginas de acceso restringido sólo para clientes. Estos pueden

acceder a su perfil personal y comprobar el estado de sus pedidos.

Panel de control: Conjunto de páginas de acceso restringido sólo para el usuario administrador. Este tendrá acceso a todas las pantallas de gestión de la tienda.

CMS: *Content Management System*. Software que provee una infraestructura para la creación y administración de contenidos.

Cookies: Datos enviados desde el servidor que se almacenan en el navegador del usuario. Cuando el usuario vuelve a ingresar en el sitio, la información guardada en la cookie puede ser recuperada por el sitio web de manera que el portal es capaz de "recordar" la actividad previa del usuario.

JavaScript: Lenguaje de programación que se ejecuta del lado del cliente y está destinado a formar parte de un navegador web de manera que este pueda visualizar los contenidos dinámicos de los sitios web y las interfaces de usuario avanzadas.

Plugin: Componente que añade capacidades específicas a una aplicación software.

CSS: *Cascading Style Sheets.* Es un lenguaje de hojas de estilo que permite describir la semántica de presentación (apariencia y formateo) de un documento. [5]

Silverlight: Framework de aplicaciones web que soporta multimedia, gráficos y animaciones.

IPN: *Instant Payment Notification*. Es un tecnología promovida por PayPal que permite automatizar el proceso de pago y evitar fraudes.

IIS: *Internet Information Services*. Es un servidor de aplicaciones web y conjunto de módulos de extensión creados por Microsoft. [6]

GUI: *Graphical User Interface*. Es una interfaz de usuario que permite interactuar con dispositivos electrónicos. Representa información y las acciones disponibles a través de iconos, indicadores, etc.

AJAX: *Asynchronous JavaScript And Xml*. Técnica web usada del lado del cliente que consiste en recuperar o enviar datos de forma asíncrona al servidor o hacia el servidor sin tener que realizar una petición de página entera.

JQuery: Es una librería de Javascript que facilita la programación del lado del cliente.

PayPal: Es una empresa del sector del comercio electrónico que permite la transferencia de dinero entre usuarios.

Fotolia: Es una agencia de fotografía dedicada a la distribución de imágenes "stock". En julio del 2010, se estimaba que Fotolia contenía un banco de imágenes, vectores y videos de alrededor de 9.8 millones. [7]

XML: *Extensible Markup language*. Es un lenguaje de datos textuales que permite codificar documentos de manera que sea entendible por humanos y máquinas. Su objetivo es el de representar estructuras de datos arbitrarias.

HTTP: *Hypertext Transfer Protocol*. Es un protocolo de aplicaciones para sistemas de información distribuidos, colaborativos y de hipermedia. Es la base para la comunicación de datos en la web. [8]
XML-RPC: Es un protocolo de llamada a un procedimiento remoto (*Remote Procedure Call)* que usa XML para codificar los mensajes y cuyo medio de transporte es HTTP.

Checkout: Etapa en el proceso de compra en la que el cliente "pasa por caja". Comprende la verificación de los datos de envío, la selección de una forma de pago y un último vistazo al carrito antes de realizar la compra propiamente dicha.

2.2 Descripción general

2.2.1 Perspectiva del producto

Aunque *Decorate Your House* no estará empotrado en ningún sistema mayor, existen dos dependencias. El proceso de compra dependerá del servicio PayPal si es seleccionada la opción de compra con tarjeta de crédito. Por otro lado, las imágenes de la galería de fotomurales se obtienen de un servicio web externo proporcionado por Fotolia.

2.2.2 Funciones del producto

Acción	Anónimo	Cliente	Administrador
Acceder a la páginas de información	✔	✔	✔
Navegar por el catálogo de productos	✔	✔	✔
Realizar búsquedas de imágenes	✔	✔	✔
Acceder a los detalles de un producto	✔	✔	✔

Personalizar un producto	✔	✔	✔
Ver el producto personalizado desde el simulador	✔	✔	✔
*Simulador	✔	✔	✔
Añadir ítems a su carrito de compras	✔	✔	✔
Consultar estado del carrito	✔	✔	✔
Registrar una cuenta	✔		
Contactar con el personal	✔	✔	
Identificarse como cliente de la tienda		✔	✔
Recuperar contraseña		✔	✔
Suscribirse a boletines informativos		✔	
Desapuntarse de los boletines informativos		✔	
Acceder al Área de clientes		✔	
Ver datos del perfil		✔	
Modificar datos del perfil		✔	
Ver mis pedidos		✔	
Hacer un pedido		✔	
Salir del sistema		✔	✔
Acceder al Panel de control			✔
Añadir vinilo			✔
Modificar vinilo			✔
Editar las propiedades de los fotomurales			✔

Acceder al listado de clientes			✔
Buscar cliente			✔
Acceder al listado de pedidos			✔
Buscar pedido			✔
Ver detalles del pedido			✔
Cambiar estado de los pedidos			✔
Borrar pedido			✔
Enviar boletines informativos			✔

* Nota: Esta acción incluye todas las funciones que involucran el uso del simulador.

2.2.3 Características de los usuarios

Nuestro portal está destinado a profesionales del sector del interiorismo, decoradores, arquitectos o bien particulares que deseen dotar de un toque de personalidad a sus espacios.

Decorate Your House sigue el modelo estándar de tienda online con lo cual será difícil que el usuario tenga dudas a la hora de completar el proceso de búsqueda de un producto y posterior compra. De todas maneras, no es un requisito imprescindible que el usuario cuente con experiencia previa en este tipo de portales. El sitio está organizado de manera que los productos sean fácilmente accesibles y la compra siempre se lleve a cabo en un número reducido de pasos.

En el caso del manejo del Panel de control que se llevará a cabo por el rol de administrador, no estaría de más une breve explicación sobre el funcionamiento de cada pantalla y como hacer un buen uso de las mismas.

2.2.4 Restricciones generales

El desarrollo no estuvo sujeto a ninguna limitación referente a políticas de empresa. El uso del portal no requiere de ningún hardware específico. En principio, se puede acceder desde cualquier tipo de dispositivo con conexión a Internet.

A nivel de software, el portal es compatible con la mayoría de navegadores actuales. En el caso de Internet Explorer se recomienda utilizar la versión 9 o posterior por temas de compatibilidad con CSS. Aunque en la mayoría de navegadores vienen activados por defecto, es conveniente asegurarse que tanto cookies como Javascript se encuentran habilitados. Para utilizar el simulador de vinilos, hace falta instalar el plugin de Silverlight.

En el apartado de seguridad, para verificar la veracidad de las compras con Paypal, se requiere el uso de IPN.

2.2.5 Supuestos y dependencias

Se da por supuesto que el portal será alojado en un hosting con sistema anti-downtime por problemas ocasionados por hardware (réplicas de bases de datos, restauración de base de datos anterior). El servidor de aplicaciones web deberá ser IIS 6.0 corriendo bajo el sistema operativo Windows. El gestor de base de datos recomendado es SQL Server 2008. En caso de variar los anteriores detalles, podría hacer falta revisar y cambiar los requisitos.

2.2.6 Requisitos futuros

Como esbozo de futuras mejoras a incorporar en el portal, cabría destacar la posibilidad de añadir dos nuevas categorías de productos: lienzos y espejos. Así como los vinilos decorativos tienen el azul como color identificativo y los fotomurales el magenta, los nuevos productos también estarían identificados por un color. Restaría por discutir si se trata de productos personalizables a gusto de consumidor o bien sólo estarían disponibles en ciertas medidas, colores, etc. Es probable que se requieran nuevas interfaces de gestión y que alguna de las páginas actuales sufra modificaciones (por ejemplo, listado de ítems del carrito de compras).

2.3 Requisitos específicos

2.3.1 Requisitos de interfaces externas

2.3.1.1 Interfaces de usuario

Se pretende una GUI de diseño atractivo que esté relacionada con la temática de los vinilos decorativos.

En primer lugar se busca evitar la saturación de elementos en una misma pantalla de manera que el usuario no caiga en confusiones. Queremos un diseño centrado que ofrezca confianza, balance y simpleza. El fondo deberá ser simple y relajado para que sirva de base a colores vibrantes.

Es prioritario guardar la consistencia entre páginas. Lo ideal es que todas nuestras páginas sigan la misma plantilla. Esta plantilla contará con un menú navegacional ubicado en la parte superior que ocupe todo el tamaño de la ventana. Desde este menú se debe poder acceder a todas las secciones de la web (Información sobre vinilos y fotomurales, simulador, detalles sobre la colocación, más información sobre cómo realizar compras, un enlace al área de clientes, y otro para contactar con nosotros). El logo de la empresa se situará encima del menú, a un lado de la página propiamente dicha. Inmediatamente debajo del menú, tiene que ser visible nuestro estatus (conectado o desconectado) y un enlace que nos lleve al carrito de compras. Cuando estemos conectados, se indicará el nombre del usuario en forma de ancla que al pincharla nos llevará a nuestro perfil. La página que ocupará el centro, y en donde se insertarán los contenidos no comunes entre páginas, tendrá del lado izquierdo una botonera que permitirá navegar entre productos y categorías en un sólo click. La plantilla también contendrá un pie de página en el que se indicarán los datos de contacto (teléfono, dirección, email, etc).

Sería útil disponer de una zona donde poder promocionar productos concretos o presentar nuevas promociones. Por ello se pide que encima del menú exista un banner que vaya cambiando de imágenes de forma automática.

Es importante destacar el tema de la coherencia de colores. Si estamos en una pantalla que tiene que ver con los vinilos, nos gustaría que el color predominante fuera el azul. En cambio, si estamos en la sección de fotomurales el color de botones, fondos, incluso logo debería cambiar al magenta.

No nos parece mala idea que el usuario comprenda rápidamente las acciones que pueden realizarse y sepa cuál será la visualización de un contenido en pantalla. Es decir, si un usuario entra a los detalles de un producto, que sepa que se va a encontrar con su referencia, su descripción, una paleta de colores, una serie de controles para ir variando las propiedades, y por último su precio.

Nuestra página web debe estar pensada para usuarios curiosos e impacientes. Las cargas completas de páginas dificultan la interfacción con el usuario. Por eso se pide hacer uso intensivo de AJAX en combinación con JQuery.

En cuanto a las interfaces de administración, se apostará por usar los elementos convencionales. Es preferible contar con interfaces más austeras pero robustas y que hagan lo que se supone que tienen que hacer.

Queremos que las páginas se presenten de forma clara. De esta manera, interesa que en la pantalla de inicio lo primero que se vea sea nuestra oferta de productos. Estos productos tienen que estar organizados según la temática.

2.3.1.2 Interfaces software

Decorate Your House interaccionará con el sistema de pago de PayPal para realizar compras y se comunicará con el servicio web de Fotolia para obtener las imágenes de la galería de fotomurales.

En el primer caso, para iniciar el proceso de pago hará falta construir un formulario "POST" con un conjunto de campos ocultos que tendrán una nomenclatura específica impuesta por PayPal y que contendrán información acerca de los productos: identificador, nombre, coste. También hará falta un botón para enviar el formulario. Como nota, comentar que los precios deben expresarse con dos dígitos después del punto. Es decir, si un producto cuesta 50 € su campo del formulario contendrá el valor "50.00". Al pulsar el botón, los datos del formulario deben enviarse a la URL https://www.paypal.com/cgi-bin/webscr.

Abajo se muestra la apariencia que debe tener el formulario para que PayPal reciba la información de forma correcta y efectúe el proceso de pago:

```
<form method="post" action= "https://www.paypal.com/cgi-bin/webscr">
    <input type="hidden" name="cmd" value="_xclick">
    <input type="hidden" name="business" value="info@vinilosyfotomurales.com">
    <input type="hidden" name="item_name" value="Vinilo decorativo">
    <input type="hidden" name="currency_code" value="EUR">
    <input type="hidden" name="item_number" value="1">
    <input type="hidden" name="amount" value="50.00">
    <input type="hidden" name="no_shipping" value="1">
    <input type="hidden" name="custom" value="OF005">
    <input type="hidden" name="return"
value="vinilosyfotomurales.com/ConfirmarPedido.aspx?Mensaje=true">
    <input type="submit" value="Comprar">
</form>
```

Debido a que siempre existe la posibilidad de que un usuario malintencionado intercepte y modifique los datos del formulario, es necesario implementar un método de verificación automática de pago. PayPal ya nos provee una solución a este problema a través de IPN. Esta tecnología depende de un script especial localizado en el servidor del vendedor. Cuando

ocurre un evento de transferencia de pago, PayPal envía una petición "POST" con los datos de la transacción a nuestro script. Este tendrá que comprobar que los datos son correctos y enviará una respuesta al servidor de PayPal confirmando o rechazando la transferencia.

Nuestro script IPN tiene que asegurarse de que se cumplan los siguientes puntos:

- Ha sido llamado desde el servidor de PayPal.
- El destinatario de pago es Decorate Your House.
- La cantidad a pagar y la moneda son correctas.

En lo referente a la obtención de imágenes, la comunicación con el API de Fotolia se hará por medio de XML-RPC. Los mensajes serán peticiones HTTP-POST con el cuerpo en XML. Un procedimiento se ejecutará en el servidor de Fotolia y devolverá un valor formateado también en XML.

Ejemplos:

Cabecera:

```
POST /Xmlrpc/rpc HTTP/1.0
Host: api.fotolia.com
Connection: close
Content-Type: text/xml
Content-Length: 569
```

Consulta:

```
<?xml version="1.0" encoding="utf-8"?>
<methodCall>
    <methodName>xmlrpc.test</methodName>
    <params>
        <param>
            <value>
                <string>API_KEY_HERE</string>
            </value>
        </param>
    </params>
</methodCall>
```

Respuesta:

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>test</name>
            <value>
```

```
                <string>Success</string>
              </value>
            </member>
          </struct>
        </value>
      </param>
    </params>
</methodResponse>
```

En nuestro caso nos interesa hacer uso del método *getSearchResults.* Visitar la documentación del método en http://es.fotolia.com/Services/API/Method/getSearchResults para obtener la lista de argumentos, valores devueltos, formato de respuesta y posibles códigos de error.

2.3.2 Requisitos funcionales

En esta sección se detallan todas aquellas acciones (funciones) que deberán estar contempladas en nuestro portal.

Acceder a las páginas de información	
Propósito	La tienda deberá tener un apartado dedicado a información general sobre los vinilos decorativos y otro para los fotomurales. Cada apartado contendrá una descripción y fotografías de muestra. Serán accesible para cualquier usuario.
Entrada	–
Proceso	El usuario se dirige al apartado de "Información". Una vez dentro, si quiere saber más sobre vinilos, hará click en la pestaña "Vinilos" y si está intersado en los fotomurales, hará click en la pestaña "Fotomurales".
Salida	En la pestaña "Vinilos" se detallará información sobre qué son los vinilos, sus ventajas y posibles aplicaciones. En la pestaña "Fotomurales" se enseñará al usuario fotos de ejemplo y se brindará una explicación de cómo sería el proceso de fabricación.

Navegar por el catálogo de productos	
Propósito	La tienda deberá tener un catálogo de productos navegable por cualquier usuario. Este catálogo se incluirá en la página de inicio. Sea cual fuere la página donde el usuario se encuentre, siempre tiene que ser posible dirigirse al catálogo en un sólo click. El catálogo estará dividido en categorías. Los resultados que se muestren estarán paginados.
Entrada	–
Proceso	Desde la página de inicio o bien haciendo click en la "botonera de categorías", el usuario podrá ir recorriendo el catálogo a su gusto.
Salida	Se muestra una "botonera de categorías" del lado izquierdo. La categoría actual aparece resaltada en otro color. Se muestran 6 productos por página. De cada producto será visible su imagen y su código de referencia. La imagen será un enlace a la página de detalles del producto.

Realizar búsquedas de imágenes	
Propósito	La tienda deberá incorporar un buscador de imágenes con un campo de texto para incluir los términos de búsqueda y un botón para llevar a cabo la misma.
Entrada	Cadena de texto a buscar.
Proceso	El usuario se dirige al apartado de "Fotomurales", ingresa la cadena deseada y pulsar el botón "Buscar".
Salida	Se muestra un listado de imágenes que coinciden con los términos de búsqueda organizadas en forma de rejilla. Los resultados estarán paginados en 18 elementos por página. Se indicará el número de página actual así como un botón para avanzar o retroceder.

Acceder a los detalles de un producto	
Propósito	La tienda deberá tener una página "perfil de producto" en la que se puedan consultar los detalles de un producto por cualquier usuario.
Entrada	–
Proceso	El usuario mientras navega por el catálogo, hará click sobre un producto concreto y será redirigido a la página de detalles de dicho producto.
Salida	La página renderizada tendrá como título el nombre de la categoría

	actual. En un primer bloque, se incluye una fotografía ampliable y un párrafo descriptivo. También debe de estar presente el botón "Recomendar" de Facebook. En una zona inferior, estarán ubicados los controles para personalizar el producto en cuestión. Para acabar, el usuario encontrará un enlace al simulador para comprobar el resultado de la personalización y un botón para añadir al carrito.

Personalizar un producto	
Propósito	La tienda deberá de proporcionar de controles que permitan cambiar las propiedades de un producto.
Entrada	En el caso de los vinilos, se encuentra a disposición del usuario: – Una paleta de colores por cada capa que contenga el vinilo. – Inputs de tipo *radio* para cambiar el acabado y la orientación. – Una tabla para seleccionar la medida deseada. – Input para cambiar la cantidad de productos. Para los fotomurales: – Un input de tipo *text* para seleccionar la altura y otro para la anchura. – Un input de tipo *radio* para elegir la orientación. – Un input de tipo *checkbox* para aplicar el efecto "Blanco y negro". – Input para cambiar la cantidad de productos.
Proceso	Para los vinilos decorativos, el usuario debe ingresar a la página de detalles de algún vinilo. Para cambiar sus propiedades, simplemente tiene que accionar cualquiera de los controles. Para los fotomurales, dirigirse a la sección "Fotomurales", realizar una búsqueda y seleccionar alguno de los resultados obtenidos. Al igual que los vinilos, para que los cambios surjan efecto deberá accionar cualquiera de los controles disponibles.
Salida	Al cambiar el color, debe aparecer una marca encima del nuevo color. Si cambiamos las preferencias de acabado u orientación, la nueva preferencia tiene que estar resaltada convenientemente. La nueva medida seleccionada quedará patente por el input *radio* que estará pulsado. Al cambiar la cantidad, se debe actualizar el contador. En el caso de los fotomurales: al cambiar las medidas se tiene que actualizar el área seleccionada dentro de la imagen de manera que sea acorde a las medidas ingresadas. Si aplicamos el efecto "Blanco y negro", la imagen debe mostrarse en escala de grises.

Ver el producto personalizado desde el simulador	
Propósito	La tienda debe mostrar en el simulador las modificaciones concretas que el usuario ha hecho a partir del diseño estándar.
Entrada	Controles descritos en la acción "Personalizar un producto". Botón probar en la pantalla de detalles de producto.
Proceso	El proceso es el mismo que se describe en "Personalizar un producto" para el caso de los vinilos. Después de realizar los pasos ahí descritos, el usuario pulsará sobre el botón "Probar".
Salida	Se debe abrir el simulador en una nueva ventana. El simulador habrá cargado el vinilo con los parámetros elegidos por el usuario.

Abrir un vinilo en el simulador	
Propósito	La tienda debe disponer de un simulador que cargue todos los modelos del catálogo.
Entrada	Miniatura clickable del vinilo.
Proceso	Una vez abierto el simulador, el usuario podrá ir navegando por las distintas categorías de vinilos y elegir el vinilo que más le guste. Para ello, bastará con pinchar sobre la miniatura del vinilo.
Salida	El vinilo elegido debe aparecer sobre la hoja de trabajo. Se tiene que apreciar el número de capas del mismo, sus nombres y su color actual. Al hacer "hover" sobre una capa, se muestra la previsualización de la capa.

Cambiar color de un vinilo	
Propósito	El simulador debe dar la posibilidad de cambiar el color de las capas del vinilo.
Entrada	Rectángulo de color clickable que se encuentra a la derecha del nombre de una capa.
Proceso	Abrir simulador y clickar sobre el modelo de vinilo deseado. Una vez cargado, seleccionar una capa concreta o todas pulsando el *checkbox* "Todas". Elegir un color de la paleta de colores circular.
Salida	Las capas afectadas cambian de color.

Cambiar color de fondo	
Propósito	El simulador debe permitir elegir un color de fondo que acompañe al vinilo.
Entrada	Paleta de colores situada en la esquina inferior izquierda.
Proceso	Abrir simulador y clickar sobre el modelo de vinilo deseado. Una vez cargado, hacer click sobre uno de los rectángulos de color de la paleta.
Salida	La hoja de trabajo cambia al color deseado.

Subir imagen al simulador	
Propósito	El simulador debe permitir la subida de imágenes para insertarlas como fondo del vinilo.
Entrada	Botón "Fondo" ubicado en la esquina inferior izquierda.
Proceso	Abrir simulador y clickar sobre el modelo de vinilo deseado. Una vez cargado, hacer click sobre el botón "Fondo". Seleccionar una fichero imagen desde nuestro ordenador y hacer click sobre "Aceptar".
Salida	La imagen del usuario se inserta sobre la hoja de trabajo.

Desplazar vinilo	
Propósito	El simulador debe permitir el desplazamiento horizontal y vertical del vinilo sobre la hoja de trabajo.
Entrada	*Sliders* "desplazamiento horizontal" y "desplazamiento vertical" del panel de propiedades.
Proceso	Abrir simulador y clickar sobre el modelo deseado. Una vez cargado, pulsar sobre el vinilo y arrastrar el puntero hacia otra ubicación. Otra posibilidad es la de utilizar el panel de propiedades: mover los *Sliders* "desplazamiento horizontal" y "desplazamiento vertical" según se desee.
Salida	El vinilo cambia de posición.

Cambiar tamaño de un vinilo	
Propósito	El simulador debe permitir la modificación del ancho y alto de un vinilo.
Entrada	*Sliders* "ancho" y "alto" del panel de propiedades.
Proceso	Abrir simulador y clickar sobre el modelo deseado. Una vez cargado, seleccionar vinilo, y dejar pulsado el ratón sobre una de las esquinas. Arrastrar hacia fuera para aumentar el tamaño o hacia dentro para reducirlo. Otra posibilidad es la de utilizar el panel de propiedades: mover los *Sliders* "ancho" y "alto" según se desee.
Salida	El vinilo cambia de tamaño.

Ajustar rotación de un vinilo	
Propósito	El simulador debe permitir cambiar el ángulo de rotación de un vinilo.
Entrada	*Slider* "rotación" del panel de propiedades.
Proceso	Abrir simulador y clickar sobre el modelo deseado. Una vez cargado, seleccionar vinilo, y dejar pulsado el ratón sobre una de las esquinas. Mover el ratón formando el ángulo deseado. Otra posibilidad es la de utilizar el panel de propiedades: mover el *slider* "rotación" hacia la derecha para girar el vinilo en sentido horario o hacia la izquierda para hacerlo en sentido anti-horario.
Salida	El vinilo cambia su ángulo de rotación.

Proyectar vinilo	
Propósito	El simulador debe permitir la aplicación del efecto de proyección sobre un vinilo.
Entrada	*Sliders* "Proyección X" y "Proyección Y" del panel de propiedades.
Proceso	Abrir simulador y clickar sobre el modelo deseado. Una vez cargado, seleccionar vinilo, y a continuación mover la rueda del ratón para aumentar o disminuir la proyección en el eje X. Otra posibilidad es la de utilizar el panel de propiedades: mover los *sliders* "Proyección X" y "Proyección Y" para aumentar o disminuir la proyección en el eje X o eje Y respectivamente.
Salida	El efecto proyección se aplica sobre el vinilo.

Centrar vinilo	
Propósito	El simulador debe permitir el centrado del vinilo.
Entrada	*Checkbox* "Centrado".
Proceso	Abrir simulador y clickar sobre el modelo deseado. Una vez cargado, seleccionar vinilo, y desde el panel de propiedades pulsar "Centrado".
Salida	El vinilo se mueve hacia el punto medio de la hoja de trabajo.

Restablecer valores por defecto	
Propósito	El simulador debe permitir deshacer todos los cambios hechos hasta el momento.
Entrada	Botón "Por defecto".
Proceso	Abrir simulador, clickar sobre el modelo deseado y aplicar las modificaciones pertinentes. Abrir el panel de propiedades y pulsar sobre el botón "Por defecto".
Salida	El vinilo vuelve a su estado original.

Imprimir resultado	
Propósito	El simulador debe permitir que se impriman los cambios realizados.
Entrada	Botón "Imprimir".
Proceso	Abrir simulador, clickar sobre el modelo deseado y aplicar las modificaciones pertinentes. Al finalizar, pulsar botón "Imprimir".
Salida	Se nos abre el asistente para configurar los parámetros de impresión.

Añadir ítems a su carrito de compras	
Propósito	La tienda debe permitir que un usuario añada ítems a su carrito de compras.
Entrada	Botón "Añadir al carrito".
Proceso	Para los vinilos: entrar a la página de detalles de un vinilo. Pulsar sobre "Añadir al carrito". Para los fotomurales: ir a la sección "Fotomurales". Clickar sobre la foto deseada. Pulsar "Añadir al carrito".
Salida	Una animación nos dice que el ítem ha sido añadido correctamente.

Consultar estado del carrito	
Propósito	La tienda debe permitir que el usuario pueda consultar el estado de su carrito de compras.
Entrada	Botón "Carrito".
Proceso	Desde cualquier sitio de la web, pulsar el botón "Carrito".
Salida	Se nos abre un *pop-up* que nos enseña un listado detallado de todos los ítems que han sido añadidos al carrito hasta el momento.

Registrar una cuenta	
Propósito	La tienda debe ser capaz de recoger los datos de clientes y almacenarlos en el sistema.
Entrada	Datos personales del usuario: – Nombre, apellidos, NIF/CIF, email, contraseña, empresa, dirección, código postal, población, provincia, teléfono, fax.
Proceso	Ingresar a la página de registro, completar los campos obligatorios y pulsar el botón "Registrar".
Salida	Se nos muestra un mensaje con el resultado de la operación. Si ha acabado con éxito, habremos recibido un email de bienvenida.

Contactar con el personal	
Propósito	La tienda debe dar la opción de que los usuarios puedan plantear sus dudas o sugerencias a través de una página de contacto.
Entrada	– Nombre, empresa, email, teléfono y consulta.
Proceso	Ingresar a la página de contacto, completar los campos obligatorios y pulsar el botón "Enviar consulta".
Salida	Se nos muestra un mensaje con el resultado de la operación.

Identificarse como cliente de la tienda	
Propósito	La tienda debe permitir la acreditación e ingreso en el portal de usuarios previamente registrados en el sistema.
Entrada	– Email y contraseña.
Proceso	Ingresar a la página de autentificación, completar los campos obligatorios y pulsar el botón "Enviar".
Salida	En caso de éxito, el usuario ingresa en el sistema y es redirigido a la página de inicio. En caso de error, se informa de lo sucedido mediante una alerta.

Recuperar contraseña	
Propósito	La tienda debe enviar los datos de la cuenta del cliente en caso de que este así lo solicite.
Entrada	Email
Proceso	Ingresar a la página de autentificación, pinchar sobre "He olvidado mi contraseña", ingresar el email en el campo de texto y pulsar el botón "Recordar".
Salida	Si la dirección de email proporcionada es correcta, se habrá enviado un email recordatorio de los datos de su cuenta. En caso contrario, se notificará que el email no ha sido reconocido a través de una alerta.

Suscribirse a boletines informativos	
Propósito	La tienda debe permitir a sus clientes apuntarse a los boletines informativos.
Entrada	–
Proceso	El cliente debe pinchar sobre el icono con forma de carta ubicado en el lateral derecho.
Salida	Un mensaje informa que el cliente se ha apuntado con éxito.

Desapuntarse de los boletines informativos	
Propósito	La tienda debe permitir a sus clientes anular su suscripción a los boletines informativos.
Entrada	*Checkbox* "Recibir boletines".
Proceso	El cliente debe dirigirse al "Área de clientes", pinchar sobre "Ver o modificar mi perfil", desmarcar el *checkbox* "Recibir boletines" y pulsar sobre "Guardar".
Salida	Si todo ha ido bien, el cliente es redirigido hacia la página anterior.

Acceder al Área de clientes	
Propósito	La tienda debe permitir el acceso al "Área de clientes" a todos aquellos usuarios que han sido autentificados.
Entrada	–
Proceso	Una vez ingresado, el cliente debe pulsar sobre el enlace "Clientes" del menú navegacional.
Salida	Página de "Área de clientes".

Ver datos del perfil	
Propósito	La tienda debe permitir a sus clientes la posibilidades de comprobar sus datos personales.
Entrada	–
Proceso	Una vez ingresado, el cliente debe pulsar sobre el enlace "Clientes" del menú navegacional. Posteriormente se dirigirá a "Ver o modificar mi perfil".
Salida	Página que contiene los datos del cliente ingresados cuando completó el registro.

Modificar datos del perfil	
Propósito	La tienda debe permitir a sus clientes la posibilidades de modificar sus datos personales.
Entrada	Campos: Nombre, Apellidos, NIF/CIF, Empresa, Teléfono, Fax, Dirección, Población, Provincia, Código postal, Contraseña actual, Nueva contraseña, Repetir contraseña, Email.

Proceso	Una vez ingresado, el cliente debe pulsar sobre el enlace "Clientes" del menú navegacional. Posteriormente se dirigirá a "Ver o modificar mi perfil". Modificar cualquiera de los campos citados en Entrada y pulsar "Guardar".
Salida	Si la nueva información no supera la validación, una notificación nos alertará de lo sucedido. Si todo ha ido bien, el usuario es redirigido a la página anterior.

Ver mis pedidos	
Propósito	La tienda debe permitir a sus clientes consultar el estado de los pedidos realizados.
Entrada	–
Proceso	Una vez ingresado, el cliente debe pulsar sobre el enlace "Clientes" del menú navegacional. Posteriormente se dirigirá a "Ver mis pedidos".
Salida	Tabla que contiene ID del pedido, fecha y estado.

Hacer un pedido	
Propósito	La tienda debe permitir a sus clientes adquirir productos del catálogo.
Entrada	Productos que se encuentren en el carrito de compras. Datos de envío, datos de pago.
Proceso	– Para comprar un vinilo, el cliente se dirigirá a la página de detalles del vinilo de su elección y hará click sobre el botón "Añadir al carrito". – Para comprar un fotomural, el cliente debe dirigirse a la sección de fotomurales, elegir una imagen y pulsar "Añadir al carrito". Una vez que el cliente haya acabado de conformar su carrito de compras, procederá al *checkout*. Para ello, debe pulsar sobre el icono de carrito de compras y luego seleccionar la opción "Realizar pedido". El cliente deberá indicar los datos de envío y forma de pago. Para finalizar, pulsará sobre "Confirmar".
Salida	Pantalla que nos informa que el proceso ha culminado con éxito. Se muestran los detalles del pedido y un mensaje de agradecimiento. Se habrá enviado un email con el resguardo de la compra.

Salir del sistema	
Propósito	La tienda debe permitir el cierre de sesión a aquellos usuarios autentificados.
Entrada	–
Proceso	Pulsar sobre el ancla "Salir".
Salida	Página de inicio. El usuario se muestra como desconectado.

Acceder al Panel de control	
Propósito	La tienda debe permitir el acceso del usuario administrador al Panel de control.
Entrada	–
Proceso	Ingresar al portal como administrador y pulsar el enlace "Admin".
Salida	Panel de control para el administrador.

Añadir vinilo	
Propósito	La tienda debe permitir que un usuario administrador pueda dar de alta nuevos vinilos que serán añadidos al catálogo.
Entrada	– Un Categoría. – Una o varias medidas. Cada medida incluye: ancho (cm), alto (cm) y precio. – Una imagen. – Una o varias capas. Cada capa lleva su nombre.
Proceso	El administrador accede al Panel de control; hace click sobre "Añadir" en la pestaña "Vinilos"; rellena los campos descritos en "Entrada" y pulsa el botón "Confirmar".
Salida	Mensaje de información sobre el resultado de la operación. De ser todo correcto, el nuevo vinilo debe mostrarse primero en la lista de la categoría a la que pertenece y debe destacarse con la etiqueta "Nuevo".

Modificar vinilo	
Propósito	La tienda debe permitir que un usuario administrador pueda editar las características de un vinilo existente.
Entrada	– Identificador del vinilo (código de referencia). – Botones "Añadir medida", "Modificar" o "Eliminar" para editar las existentes. – Botones "Añadir capa", "Modificar" o "Eliminar" para editar las capas.
Proceso	El administrador accede al Panel de control; hace click sobre "Modificar" en la pestaña "Vinilos"; introduce la ID del vinilo a editar; usar los campos descritos en "Entrada" para variar su información y pulsa el botón "Confirmar".
Salida	Mensaje de información sobre el resultado de la operación.

Editar las propiedades de los fotomurales	
Propósito	La tienda debe permitir que un usuario administrador pueda editar los costes de los fotomurales. Los costes vienen agrupados en franjas de medidas. Cada franja tiene su correspondiente coste.
Entrada	Medida (m2): – Desde – Hasta Precio
Proceso	El administrador accede al Panel de control; se dirige a la pestaña "Fotomurales". Para añadir un rango de medidas utiliza los controles descritos en "Entrada" y pulsa "Agregar".
Salida	Tabla de precios con una nueva fila representando el rango de medidas agregado.

Acceder al listado de clientes	
Propósito	La tienda debe permitir que un usuario administrador pueda consultar el listado de los clientes que se encuentran registrados. Esto incluye: nombre, apellidos, email, teléfono, fax, dirección, población, provincia, código postal, empresa, y CIF/NIF.
Entrada	–
Proceso	El administrador accede al Panel de control y se dirige a la pestaña "Clientes".
Salida	Listado paginado de usuarios con la información expuesta en "Propósito". El

	listado es ordenable por columnas.

Buscar cliente	
Propósito	La tienda debe permitir que un usuario administrador pueda buscar un cliente siguiendo como criterio uno de sus datos personales.
Entrada	Cualquiera de los siguientes datos: – Nombre, apellidos, email, teléfono, fax, dirección, población, provincia, código postal, empresa o CIF/NIF.
Proceso	El administrador accede al Panel de control y se dirige a la pestaña "Clientes". A continuación, hace click sobre "Buscar". Selecciona un criterio de búsqueda de los anteriormente mencionados; introduce el término de búsqueda y pulsa "Buscar"
Salida	Se muestra el mismo listado que en el caso de "Acceder al listado de clientes" con la salvedad de que sólo se muestran aquellos clientes que cumplen los criterios de búsqueda.

Acceder al listado de pedidos	
Propósito	La tienda debe permitir que un usuario administrador pueda consultar el listado de los pedidos registrados hasta el momento.
Entrada	–
Proceso	El administrador accede al Panel de control y se dirige a la pestaña "Pedidos".
Salida	Listado paginado de pedidos. Por cara pedido se muestra: Un identificador, el nombre del cliente que ha realizado el pedido, la fecha, un *select* que muestra el estado actual, unos datos de entrega que aparecerán rellenados sólo si el usuario ha especificado una dirección de envío distinta a la suya a la hora de realizar la compra y unos datos de facturación que también aparecerán rellenados sólo en el caso de que el usuario haya ingresado alguna dirección de facturación durante la compra; tipo de pago.

Buscar pedido	
Propósito	La tienda debe permitir que un usuario administrador pueda buscar un pedido.
Entrada	Cualquiera de los siguientes datos: – Identificador, cliente, fecha, estado, datos de entrega, datos de facturación, tipo de pago.

Proceso	El administrador accede al Panel de control y se dirige a la pestaña "Pedidos". A continuación, hace click sobre "Buscar". Selecciona un criterio de búsqueda de los anteriormente mencionados; introduce el término de búsqueda y pulsa "Buscar".
Salida	Se muestra el mismo listado que en el caso de "Acceder al listado de pedidos" con la salvedad de que sólo se muestran aquellos pedidos que cumplen los criterios de búsqueda.

Ver detalles del pedido	
Propósito	La tienda debe permitir que un usuario administrador pueda consultar los detalles de un pedido.
Entrada	–
Proceso	El administrador accede al Panel de control y se dirige a la pestaña "Pedidos". Pulsa sobre el botón "+" que ocupa la primera celda de la fila deseada.
Salida	Aparecen tantas filas "hijas" de la fila seleccionada como productos ha adquirido el cliente. Cada fila "hija" contiene los siguientes campos: referencia, medida, código del color o colores, orientación, acabado, efecto blanco y negro (si/no), cantidad y precio. De tratase de un fotomural, además se incluirá un enlace que lleva a la página de fotomurales para ver con exactitud cuales son las opciones que ha escogido el cliente en su pedido.

Cambiar estado de los pedidos	
Propósito	La tienda debe permitir que un usuario administrador pueda alterar el estado de un pedido.
Entrada	Nuevo estado.
Proceso	El administrador accede al Panel de control y se dirige a la pestaña "Pedidos". A continuación debe accionar el *select* del pedido correspondiente y cambiarlo al nuevo estado.
Salida	El *select* debe reflejar el nuevo estado. El cambio debe aplicarse también al listado de pedidos del Área de clientes. Estos deben ver reflejado el nuevo estado del pedido.

Borrar pedido	
Propósito	La tienda debe permitir que un usuario administrador pueda deshacer un pedido.
Entrada	Pedido.
Proceso	El administrador accede al Panel de control y se dirige a la pestaña "Pedidos". A continuación debe seleccionar una de las filas del listado de pedidos y accionar el botón "Borrar".
Salida	Listado de pedidos que excluye el pedido borrado. También debe desaparecer del listado de pedidos localizado en el Área de clientes del usuario en cuestión.

Enviar boletines informativos	
Propósito	La tienda debe permitir que un usuario administrador pueda enviar boletines, notificaciones, avisos de promociones, emails generales a todos los usuarios de la web o simplemente a aquellos que estén subscritos.
Entrada	– Asunto. – Destinatario: usuarios de la web / usuarios subscritos. – Cuerpo del mensaje.
Proceso	El administrador accede al Panel de control y se dirige a la pestaña "Boletines". Ingresa los campos detallados en "Entrada" y pulsa "Enviar boletín".
Salida	Una mensaje nos avisa del resultado de la operación.

2.3.3 Atributos

2.3.3.1 Seguridad

El atributo de calidad de mayor importancia en nuestro sistema es la seguridad. El modelo de negocio está sustentado en las ventas por tanto es crítico evitar la posibilidad de que un usuario malintencionado incurra en un fraude. Para conseguir este objetivo se hará uso de la tecnología IPN proporcionada por PayPal. Esta tecnología nos garantiza que en el momento en el que el usuario efectivamente ingresa dinero en nuestra cuenta, PayPal se encargará de mandarnos una notificación. Es entonces que estaremos en disposición de proceder a la fabricación de los artículos solicitados.

También referente a la seguridad, se pondrá especial énfasis en dejar clara la separación entre las áreas de gestión de la tienda -sólo accesibles para el administrador- y el resto de la misma; o bien el Área de clientes -para usuarios autentificados- y el área común. Esto se puede conseguir implantando una estructura de directorios en la que aquellas páginas de acceso restringido se guarden bajo directorios aparte. Estos directorios contarán con un fichero de configuración de acceso propio que dictará las normas de acceso; por ejemplo, permitir el acceso a un usuario específico y denegarlo al resto.

3. Análisis

3.1 Introducción

3.1.1 La necesidad de un ORM

Decorate Your House es una aplicación web orientada a objetos que hace uso de una base de datos relacional para la persistencia de datos. Tras dedicar un tiempo de estudio, se ha detectado que existe una correspondencia directa entre las tablas de la base de datos y las clases que requiere nuestra aplicación. Es frecuente que en este tipo de aplicaciones se requiera escribir una gran cantidad de código para conectar y lanzar consultas a la base de datos, código que difiere muy poco de clase a clase y es tedioso de escribir. Se trata de un proceso mecánico: diseñar la base de datos para permitir la persistencia de clases, crear la base de datos, escribir código que haga de interface con la base de datos.

Además, en nuestra aplicación se da el caso de tener clases y tablas paralelas. Por ejemplo, la clase Cliente (con sus atributos: Nombre, Apellidos, NIF, etc.) y la tabla Clientes (con sus columnas: nombre, apellidos, nif, etc). Esta duplicidad de información requerirá una duplicidad de trabajo aumentando así las probabilidades de introducir errores. Si en algún momento quisiéramos añadir un nuevo atributo al objeto Cliente, los cambios deberían ser replicados en la base de datos, de lo contrario nuestra aplicación dejaría de funcionar. Lo ideal sería automatizar esta duplicación de modo que sea trasparente para el desarrollador.

Por fortuna, para solventar los problemas de la duplicidad de información y la pérdida de tiempo debido a tareas repetitivas expuesto en el primer párrafo, existe una tecnología conocida como mapeo objeto-relacional (*Object-Relational mapping*, O/RM, ORM, O/R) que busca automatizar el puente entre el mundo de la programación orientada a objetos (*Object-oriented programming*, OOP).

Otra motivación es el denominado "Desajuste de impedancia objeto-relacional" (*object-relational impedance mismatch*). El *Desajuste de impedancia objeto-relacional* ocurre porque el paradigma orientado a objetos y el de bases de datos relacionales tienen diferentes concepciones de datos. Para el paradigma orientado a objetos, los objetos son importantes no sólo por los datos que contienen sino también por su habilidad de llevar a cabo tareas sobre los datos e intercambiar información con otros objetos. El paradigma relacional está enfocado en los datos. La importancia recae en los datos en sí mismos y en sus relaciones estructurales (no comportamentales) con otros datos. Para aclararlo de una forma menos abstracta, en la programación orientada a objetos las tareas se ejecutan manipulando objetos que son valores no escalares. Por ejemplo, considerar un pedido. Un pedido puede ser modelado en un objeto "Pedido" que contiene los atributos: cliente, fecha y una lista de productos. El cliente contendría en si mismo un objeto "Cliente"; la lista de productos contendría una colección de objetos de tipo "Producto". Varios métodos pueden estar asociados al objeto Pedido.
Sin embargo, los sistemas gestores de bases de datos solamente pueden almacenar y

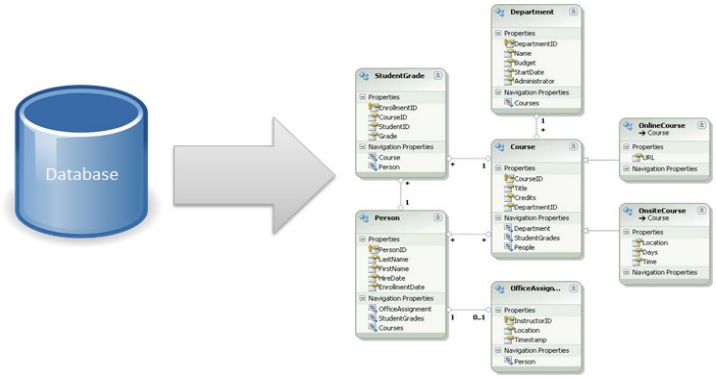
manipular datos escalares como, por ejemplo, enteros o cadenas que se organizan en tablas. El programador está obligado a convertir los datos del objeto en datos primitivos para su almacenamiento. En caso de necesitar ese objeto en alguna parte de nuestra aplicación, tendremos que reconstruir el objeto a partir de los datos primitivos.

El mapeo objeto-relacional nos ayudará a olvidarnos completamente de como convertir los objetos en datos primitivos y viceversa.

3.1.2 Ventajas de un ORM

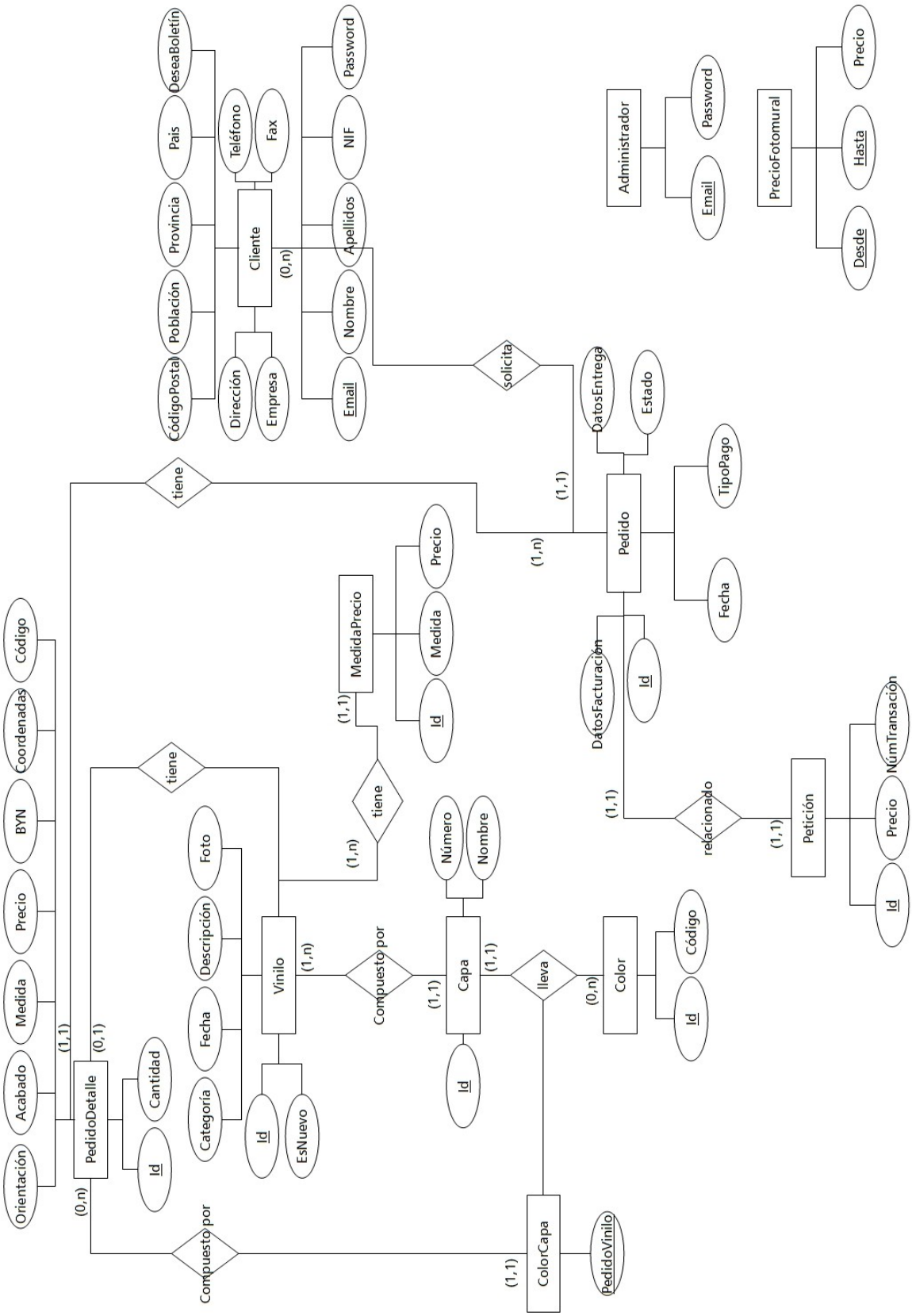
- Gran reducción de código. Las herramientas ORM generan la lógica CRUD (*Create Read Update Delete*) permitiendo que los desarrolladores se centren en la construcción de la lógica de negocio.
- Los cambios se hacen en un sólo lugar. Si necesitamos actualizar nuestras definiciones de objetos, lo hacemos sobre el modelo. El ORM se encargará de actualizar automáticamente las estructuras para insertar, actualizar o recuperar datos. Nos ahorramos la faena de tener que modificar instrucciones SQL en ninguna de las capas de la aplicación.
- Gran capacidad de consultas. Los ORM aportan un lenguaje propio orientado a objetos para atacar la base de datos. Esto permite que los desarrolladores se centren en el modelo de objetos y no tener que preocuparse por la estructura de la base de datos o por la semántica de las consultas SQL. Las herramientas ORM ya se encargan por si solas de traducir el lenguaje de consultas en la sintaxis apropiada para la base de datos.
- Navegación. Es posible navegar a través de relaciones de objetos de forma transparente. Los objetos relacionados se cargan automáticamente cuando se necesiten. Por ejemplo, si tenemos un objeto "Pedido" y queremos acceder al cliente que ha realizado ese pedido, simplemente con ejecutar Pedido.Cliente el ORM se hará cargo de cargar los datos sin ningún esfuerzo por parte del programador.
- Carga de datos configurable. Podemos elegir entre cargar un objeto junto con todos los objetos que estén relacionados a él, o bien, cargar el objeto sin ninguno de sus hijos / objetos relacionados.
- Soporte para concurrencia. Los ORM ofrecen soporte para los casos en que múltiples usuarios actualicen los mismos datos en simultáneo.
- Gestión de caché. Las entidades se cachean en memoria permitiendo reducir la carga sobre la base de datos.
- Gestión de transacciones y aislamiento. Todos los cambios sobre los objetos se dan en el ámbito de una transacción. La transacción puede ser llevada a cabo o revertida. Varias transacciones pueden estar activas en memoria al mismo tiempo y cada cambio de una transacción es aislado de otro.
- Gestión de claves. Las claves primarias y las claves ajenas se propagan y gestionan de forma automática.

3.1.3 Aproximación de base de datos



Para la elaboración de este proyecto vamos a seguir una aproximación orientada a la base de datos. En primer lugar, nos ocuparemos del modelado de datos; esto quiere decir que vamos a hacer uso de un modelo entidad-relación que esclarezca las entidades relevantes así como sus interrelaciones y sus propiedades. Luego nos ocuparemos de construir la base de datos con nuestro gestor.
A partir de la base de datos, el ORM inferirá una estructura de clases y generará por nosotros todo el código de los objetos.

3.2 Modelo entidad-relación



Resumen:

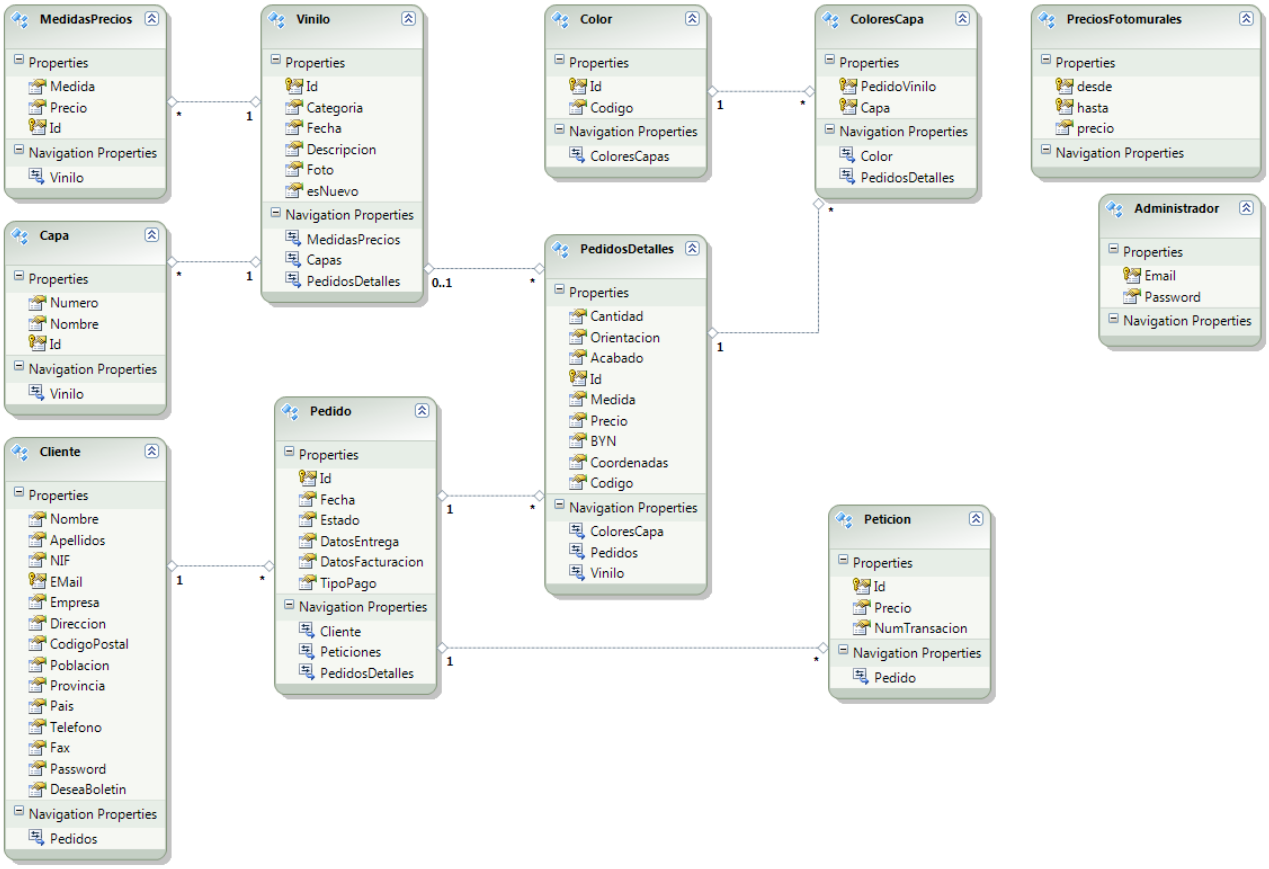
– Entidades detectadas:

- **Administrador**: Usuario con privilegios encargado del mantenimiento de la tienda. Sólo nos interesa registrar su email y contraseña para permitir su ingreso.
- **PrecioFotomural**: Representan un rango de medidas en metros cuadrados y su correspondiente precio en euros. Sólo para fotomurales. Así, el atributo "Desde" especifica la medida inicial y "Hasta" la medida final.
- **Cliente**: La entidad cliente representa los usuarios que han completado el proceso de registro. Los clientes se diferencian entre si inequívocamente por su email. De cada uno nos interesa almacenar sus datos personales: nombre, apellidos, NIF/CIF, contraseña, dirección, empresa, código postal, población, provincia, país, teléfono, y si desea recibir boletines o no.
- **Pedido**: Representan los encargos de productos. Cada pedido tendrá un identificador y una serie de información básica: fecha, forma de pago, estado, dirección de entrega, dirección de facturación.
- **PedidoDetalle**: Esta entidad representa un producto personalizado. Cada producto personalizado se diferencia por un identificador. Algunos de sus campos son comunes tanto para vinilos como fotomurales, por ejemplo: orientación, cantidad, precio. Los vinilos tendrán además los atributos: acabado y medida, mientras que los fotomurales contendrán cierta información adicional como es la opción de activar el efecto "Blanco y negro" si/no, el código de la imagen del fotomural y las coordenadas del área de selección de la imagen. Es decir, esta entidad equivale a una "instancia" de un producto.
- **Petición**: Esta entidad no tiene ninguna utilidad a nivel de lógica de negocios. Su única justificación es por cuestiones de implementación del proceso de compra e integración con PayPal.
- **Vinilo**: Entidad que representa cada uno de los vinilos que serán visibles por los usuarios desde la tienda. Aparte de tener un identificador o código de referencia, un vinilo pertenece a una categoría, tiene una fecha de alta, un párrafo descriptivo, y una foto. El campo "EsNuevo" nos vale para indicar si este vinilo tiene que ser destacado con la etiqueta "Nuevo" en la galería.
- **MedidaPrecio**: Su propósito es el de registrar las medidas posibles para un vinilo y su precio.
- **Capa**: Cada capa tiene un número que es el lugar que ocuparía dentro del vinilo y un nombre que le permitirá al usuario indentificarla con mayor facilidad.
- **Color**: Nuestra paleta de colores se compone de una multitud de entidades de tipo "Color". Cada una, tiene un indentificador y su código RGB.
- **ColorCapa**: Nos permite conocer, para cada capa, el color seleccionado por el cliente.

– Relaciones:

- Los clientes (entidad “Cliente”) pueden solicitar pedidos (entidad “Pedido”).
- Por cada pedido, se genera una petición (entidad “Petición”).
- Un pedido debe tener al menos un producto asociado, es decir, un pedido involucra uno o varios productos personalizados (entidad “PedidoDetalle”).
- Las entidades “PedidoDetalle” guardan la configuración personalizada que un cliente ha hecho sobre un fotomural o sobre un vinilo. Si se trata de un vinilo, un “PedidoDetalle” tendrá asociada varias entidades “ColorCapa” que representan el color que el cliente ha elegido para cada capa del vinilo.
- Un “Vinilo” tiene asociado al menos una entidad “MedidaPrecio” que representa una medida posible de fabricación y su coste.

3.3 Diagrama de clases



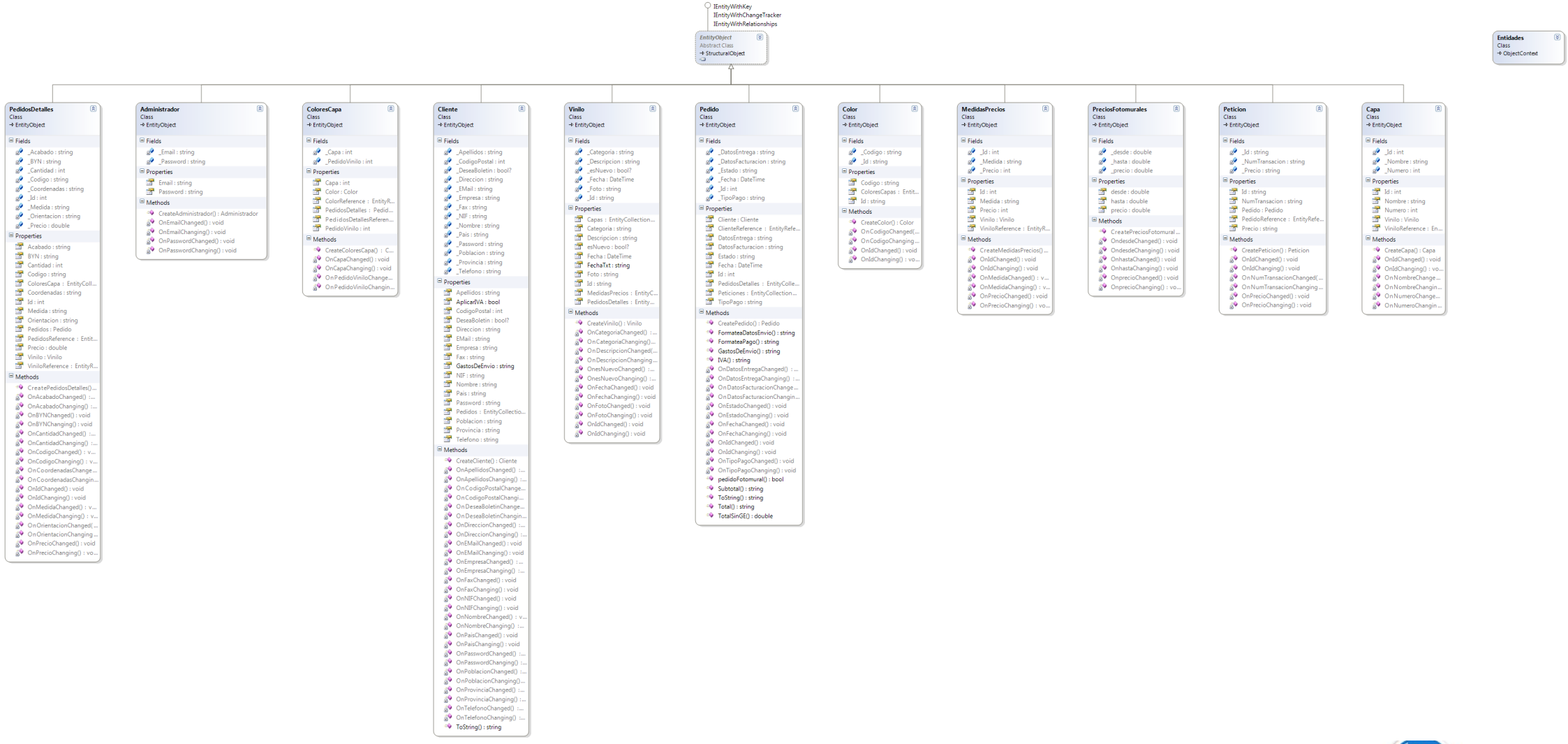
En la imagen superior podemos observar como *Entity Framework*, nuestro ORM, ha generado un modelo conceptual de entidades a partir de la base de datos existente.

Algunas puntualizaciones:

- Se ha creado una clase por cada tabla de la base de datos.
- Las columnas de las tablas en la base de datos han sido mapeadas en propiedades de objetos.
- Las relaciones entre las tablas son representadas mediante propiedades de navegación.

En la página siguiente se muestra el diagrama de clases obtenido en un sencillo paso a partir del modelo de entidades. Notar que todas las clases heredan de **EntityObject** y tienen varios campos, propiedades y métodos en gris lo cual significa que han sido generados de forma automática por *Entity Framework*. Las clases generadas son clases parciales, lo cual significa que pueden ser extendidas o bien usar herencia para añadir nueva funcionalidad. Este es el caso de aquellas propiedades y métodos que se encuentran en negrita. Además de las clases de entidades, se genera también una clase "Entidades" que hereda de **ObjectContext** y

representa el contexto de la base de datos. A través de esta clase tendremos acceso a las entidades y colecciones del modelo y podremos realizar todas las operaciones de lectura y escritura de datos desde y hacia la base de datos subyacente.


IEntityWithKey
IEntityWithChangeTracker
IEntityWithRelationships
EntityObject
Abstract Class
StructuralObject
Entidades
Class
ObjectContext
PedidosDetalles
Class
EntityObject
Fields
_Acabado : string
_BYN : string
_Cantidad : int
_Codigo : string
_Coordenadas : string
_Id : int
_Medida : string
_Orientacion : string
_Precio : double
Properties
Acabado : string
BYN : string
Cantidad : int
Codigo : string
ColoresCapa : EntityColl...
Coordenadas : string
Id : int
Medida : string
Orientacion : string
Pedidos : Pedido
PedidosReference : Entit...
Precio : double
Vinilo : Vinilo
ViniloReference : EntityR...
Methods
CreatePedidosDetalles()...
OnAcabadoChanged() :...
OnAcabadoChanging() :...
OnBYNChanged() : void
OnBYNChanging() : void
OnCantidadChanged() :...
OnCantidadChanging() :...
OnCodigoChanged() : v...
OnCodigoChanging() : v...
OnCoordenadasChange...
OnCoordenadasChangin...
OnIdChanged() : void
OnIdChanging() : void
OnMedidaChanged() : v...
OnMedidaChanging() : v...
OnOrientacionChanged(...
OnOrientacionChanging...
OnPrecioChanged() : void
OnPrecioChanging() : vo...
Administrador
Class
EntityObject
Fields
_Email : string
_Password : string
Properties
Email : string
Password : string
Methods
CreateAdministrador() : Administrador
OnEmailChanged() : void
OnEmailChanging() : void
OnPasswordChanged() : void
OnPasswordChanging() : void
ColoresCapa
Class
EntityObject
Fields
_Capa : int
_PedidoVinilo : int
Properties
Capa : int
Color : Color
ColorReference : EntityR...
PedidosDetalles : Pedid...
PedidosDetallesReferen...
PedidoVinilo : int
Methods
CreateColoresCapa() : C...
OnCapaChanged() : void
OnCapaChanging() : void
OnPedidoViniloChange...
OnPedidoViniloChangin...
Cliente
Class
EntityObject
Fields
_Apellidos : string
_CodigoPostal : int
_DeseaBoletin : bool?
_Direccion : string
_EMail : string
_Empresa : string
_Fax : string
_NIF : string
_Nombre : string
_Pais : string
_Password : string
_Poblacion : string
_Provincia : string
_Telefono : string
Properties
Apellidos : string
AplicarIVA : bool
CodigoPostal : int
DeseaBoletin : bool?
Direccion : string
EMail : string
Empresa : string
Fax : string
GastosDeEnvio : string
NIF : string
Nombre : string
Pais : string
Password : string
Pedidos : EntityCollectio...
Poblacion : string
Provincia : string
Telefono : string
Methods
CreateCliente() : Cliente
OnApellidosChanged() :...
OnApellidosChanging() :...
OnCodigoPostalChange...
OnCodigoPostalChangi...
OnDeseaBoletinChange...
OnDeseaBoletinChangin...
OnDireccionChanged() :...
OnDireccionChanging() :...
OnEMailChanged() : void
OnEMailChanging() : void
OnEmpresaChanged() :...
OnEmpresaChanging() :...
OnFaxChanged() : void
OnFaxChanging() : void
OnNIFChanged() : void
OnNIFChanging() : void
OnNombreChanged() : v...
OnNombreChanging() :...
OnPaisChanged() : void
OnPaisChanging() : void
OnPasswordChanged() :...
OnPasswordChanging() :...
OnPoblacionChanged() :...
OnPoblacionChanging()...
OnProvinciaChanged() :...
OnProvinciaChanging() :...
OnTelefonoChanged() :...
OnTelefonoChanging() :...
ToString() : string
Vinilo
Class
EntityObject
Fields
_Categoria : string
_Descripcion : string
_esNuevo : bool?
_Fecha : DateTime
_Foto : string
_Id : string
Properties
Capas : EntityCollection...
Categoria : string
Descripcion : string
esNuevo : bool?
Fecha : DateTime
FechaTxt : string
Foto : string
Id : string
MedidasPrecios : EntityC...
PedidosDetalles : Entity...
Methods
CreateVinilo() : Vinilo
OnCategoriaChanged() :...
OnCategoriaChanging()...
OnDescripcionChanged(...
OnDescripcionChanging...
OnesNuevoChanged() :...
OnesNuevoChanging() :...
OnFechaChanged() : void
OnFechaChanging() : void
OnFotoChanged() : void
OnFotoChanging() : void
OnIdChanged() : void
OnIdChanging() : void
Pedido
Class
EntityObject
Fields
_DatosEntrega : string
_DatosFacturacion : string
_Estado : string
_Fecha : DateTime
_Id : int
_TipoPago : string
Properties
Cliente : Cliente
ClienteReference : EntityRefe...
DatosEntrega : string
DatosFacturacion : string
Estado : string
Fecha : DateTime
Id : int
PedidosDetalles : EntityColle...
Peticiones : EntityCollection...
TipoPago : string
Methods
CreatePedido() : Pedido
FormateaDatosEnvio() : string
FormateaPago() : string
GastosDeEnvio() : string
IVA() : string
OnDatosEntregaChanged() :...
OnDatosEntregaChanging() :...
OnDatosFacturacionChange...
OnDatosFacturacionChangin...
OnEstadoChanged() : void
OnEstadoChanging() : void
OnFechaChanged() : void
OnFechaChanging() : void
OnIdChanged() : void
OnIdChanging() : void
OnTipoPagoChanged() : void
OnTipoPagoChanging() : void
pedidoFotomural() : bool
Subtotal() : string
ToString() : string
Total() : string
TotalSinGE() : double
Color
Class
EntityObject
Fields
_Codigo : string
_Id : string
Properties
Codigo : string
ColoresCapas : Entit...
Id : string
Methods
CreateColor() : Color
OnCodigoChanged(...
OnCodigoChanging...
OnIdChanged() : void
OnIdChanging() : vo...
MedidasPrecios
Class
EntityObject
Fields
_Id : int
_Medida : string
_Precio : int
Properties
Id : int
Medida : string
Precio : int
Vinilo : Vinilo
ViniloReference : EntityR...
Methods
CreateMedidasPrecios() ...
OnIdChanged() : void
OnIdChanging() : void
OnMedidaChanged() : v...
OnMedidaChanging() : v...
OnPrecioChanged() : void
OnPrecioChanging() : vo...
PreciosFotomurales
Class
EntityObject
Fields
_desde : double
_hasta : double
_precio : double
Properties
desde : double
hasta : double
precio : double
Methods
CreatePreciosFotomural ...
OndesdeChanged() : void
OndesdeChanging() : void
OnhastaChanged() : void
OnhastaChanging() : void
OnprecioChanged() : void
OnprecioChanging() : vo...
Peticion
Class
EntityObject
Fields
_Id : string
_NumTransacion : string
_Precio : string
Properties
Id : string
NumTransacion : string
Pedido : Pedido
PedidoReference : EntityRefe...
Precio : string
Methods
CreatePeticion() : Peticion
OnIdChanged() : void
OnIdChanging() : void
OnNumTransacionChanged(...
OnNumTransacionChanging...
OnPrecioChanged() : void
OnPrecioChanging() : void
Capa
Class
EntityObject
Fields
_Id : int
_Nombre : string
_Numero : int
Properties
Id : int
Nombre : string
Numero : int
Vinilo : Vinilo
ViniloReference : En...
Methods
CreateCapa() : Capa
OnIdChanged() : void
OnIdChanging() : vo...
OnNombreChange...
OnNombreChangin...
OnNumeroChange...
OnNumeroChangin...

4. Diseño

4.1 Introducción

En este apartado comentaremos el uso de una arquitectura basada en n-capas y el porqué hemos llegado a este diseño arquitectónico. Primero que nada, n-capas se refiere a la arquitectura de una aplicación que al menos cuenta con tres capas lógicas o partes separadas. Cada capa interactúa únicamente con la capa que se encuentra inmediatamente por debajo y tiene una función específica de la que es responsable.

La ventaja principal que aporta una arquitectura de n-capas es la posibilidad de que cada capa pueda alojarse físicamente en servidores diferentes. Esto tiene un impacto directo sobre la escalabilidad de la aplicación ya que al contar con una aplicación distribuida en varios servidores estamos aumentando la capacidad de carga. En nuestro caso, esta no sería la razón fundamental para implementar este diseño ya que no prevemos tener que soportar un volumen tal de carga que haga faltar distribuir la información en servidores separados.

De lo que sí podríamos sacar beneficio es el hecho de que cada capa es independiente de las otras, es decir que lo que hace cada capa internamente es invisible al resto lo cual nos permite introducir cambios en una capa sin tener que modificar o siquiera recompilar las otras. Por ejemplo, en caso de que hubiera un cambio a nivel de servidor de bases de datos, es probable que haya que actualizar el código perteneciente a la capa de acceso a datos o DAL (*data access layer*). Sin embargo, como la lógica de negocio y el acceso a datos se encuentra separado en distintos niveles lógicos, en principio no hará falta ni recompilar ni alterar una línea de código de la lógica de negocio.

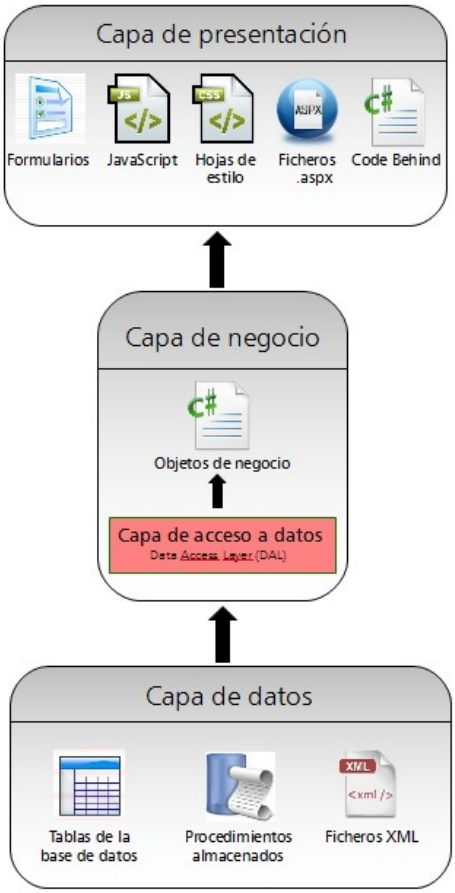
En general, una arquitectura de n-capas cuenta con las ventajas (ordenadas de mayor a menor trascendencia para nuestro proyecto):

- Mayor mantenibilidad gracias a la escasa dependencia entre capas.
- Es más fácil añadir nueva funcionalidad.
- Las capas hacen que las aplicaciones sean más testeables.
- Si tenemos las capas replicadas en localizaciones físicas separadas, aumenta el rendimiento y la tolerancia a fallos.
- Otras aplicaciones pueden reusar la funcionalidad expuesta por alguna de las capas.

Siguiendo el diseño clásico de tres capas, nuestra aplicación quedaría fraccionada en tres áreas importantes de funcionalidad:

- La **capa de presentación** que hospeda la interfaz gráfica de usuario y todo el código relacionado con la presentación.
- La **capa de negocio** que mantiene las reglas de negocio y lógica.
- La **capa de datos** que gestiona el almacenamiento físico y la recuperación de datos.

Dentro de estas capas, puede que exista una serie de sub-capas que proveen una división incluso más granular de las áreas de funcionalidad de la aplicación. Este es el caso de la **capa de acceso a datos** que se encuentra enmarcada dentro de la capa de negocio.



4.2 Arquitectura

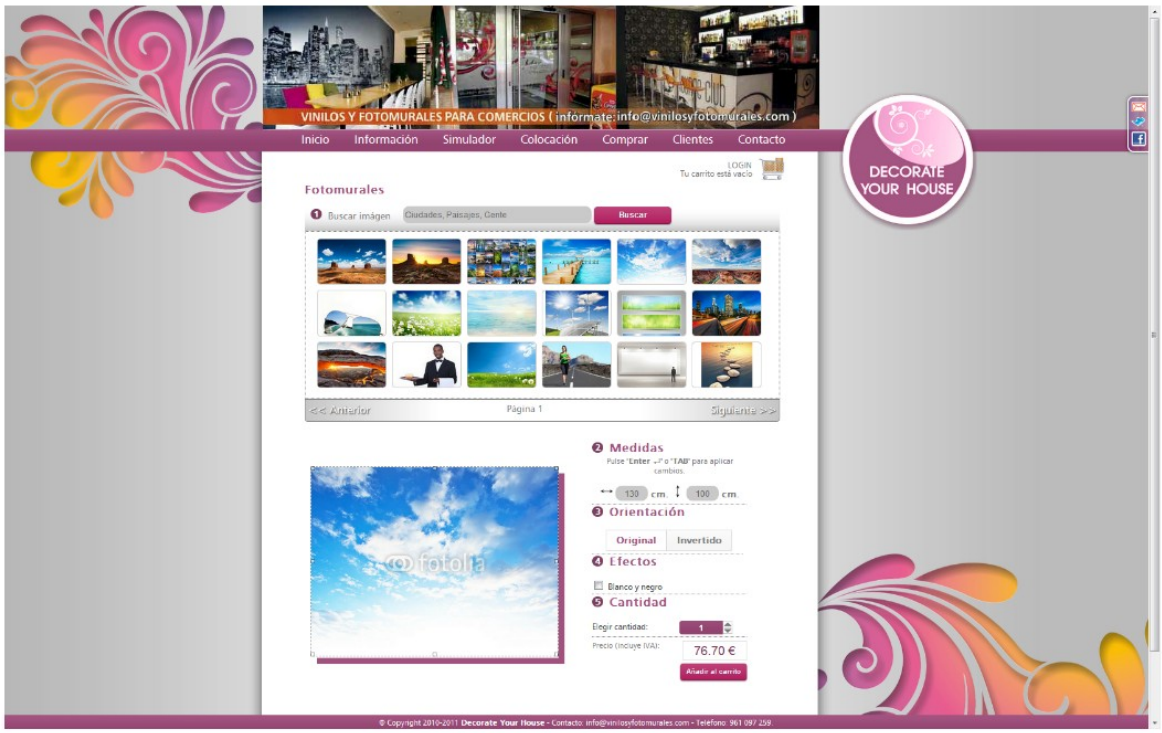
4.2.1 Capa de presentación

La capa de presentación de lo único que se preocupa es de mostrar datos al usuario y permitirle interaccionar con los mismos. Cada página de nuestra web consiste en un fichero .aspx que contiene un lenguaje de marcas para definir la apariencia y el *layout* de la página. Por cada fichero .aspx hay un fichero .cs que contiene lo que se conoce como *code behind* o lógica de presentación. Este mecanismo es un claro ejemplo del diseño en capas. A la hora de introducir cambios, un diseñador no tendrá que preocuparse por no "fastidiar" el código, mientras que un desarrollador no se preocupará por aspectos de la interfaz.

La imagen de la siguiente página ilustra la pantalla de inicio del portal. En ella distinguimos una serie de zonas a todas las páginas:

1. **Header**: Encabezado de la página que contiene un *banner* publicitario.
2. **Navbar**: Menú de navegación horizontal desde el cual se puede acceder a todos los subapartados de la web.
3. **Logo**: Logotipo identificativo de la tienda. Su diseño está inspirando en un vinilo decorativo floral acompañado del título de la empresa.
4. **ShoppingCart**: Es el área donde se sitúa el estatus del usuario: conectado / desconectado. En caso de estar desconectado, aparece un enlace que nos lleva a la página de ingreso/registro. Por el contrario, si el usuario ya está conectado, nos muestra un mensaje de bienvenida y un enlace hacia nuestro perfil. También podemos desconectarnos pulsando el *link* "Salir". Por supuesto que esta zona también incluye el estado de nuestro carrito; se indicará el número actual de ítems que se encuentran en el mismo.
5. **LeftMenu**: Botonera que nos permite recorrer las distintas categorías de vinilos o bien cambiar a la sección de fotomurales.
6. **MainContent**: Zona de la página donde se ubican los contenidos propios de la página que estamos visualizando o dicho de otra forma, aquellos contenidos no comunes.
7. **Footer**: Pie de página que contiene los datos de contacto de la empresa y un aviso legal.
8. **SocialSideBar**: Barra lateral de integración con las redes sociales. Desde ella se puede acceder a la cuenta de Twitter o el perfil de Facebook de nuestra empresa. También cuenta con un acceso directo para apuntarse a los boletines informativos.

Todas las páginas de la web siguen un mismo *layout* o plantilla. En ASP .NET esto se consigue mediante el uso de *master pages*. Una master page es un tipo especial de página en ASP .NET que define el marcado común entre todas las páginas de contenido así como también las regiones que son personalizables. En la foto inferior, vemos como la sección fotomurales lleva una *master page* ligeramente modificada a la que se le ha aplicado un estilo diferente.

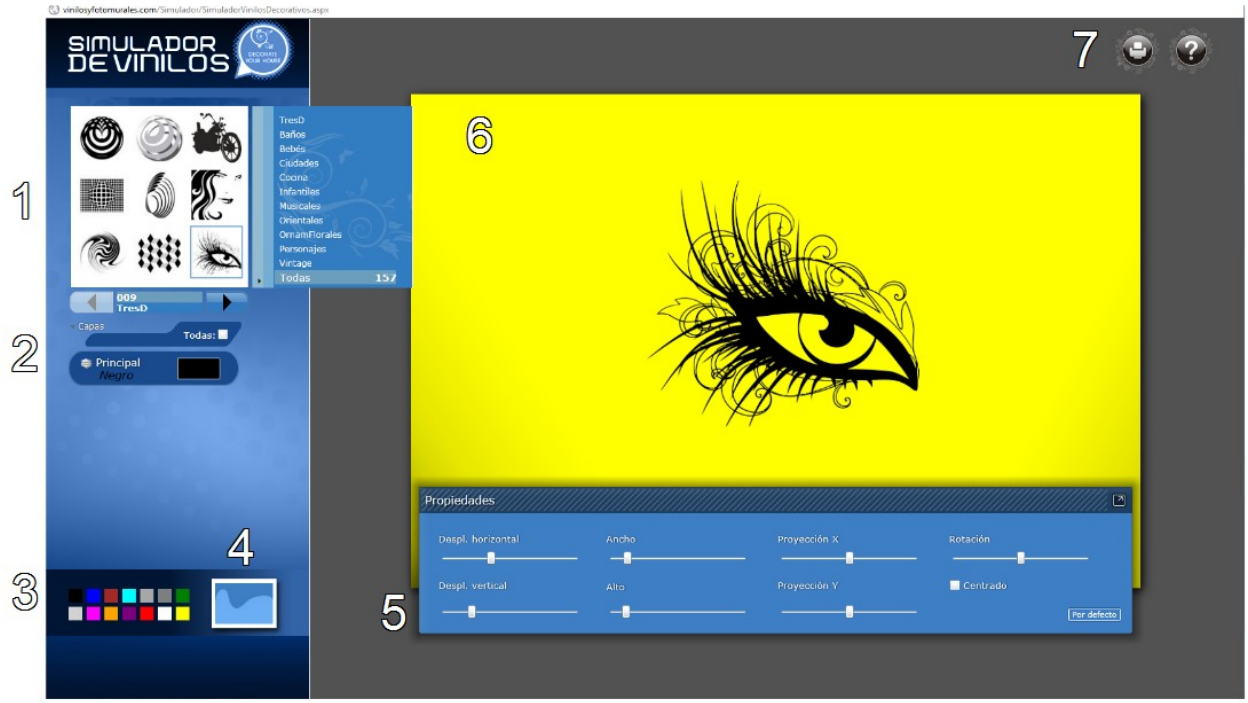


En el caso del simulador de vinilos, la idea es la misma que la comentada al inicio de este apartado exceptuando que en lugar de tener ficheros .aspx, usamos ficheros .xaml cuyo principal objetivo es el de definir elementos de la interfaz gráfica incluyendo el estilo aplicado a ellos, el enlazado de datos, manejadores de eventos y otras características. Es decir, un fichero .xaml define la apariencia visual de la interfaz gráfica pero aún seguimos necesitando un fichero de *code behind* asociado que defina la lógica.

La interfaz del simulador puede reducirse en:

1. **Carousel**: Listado de miniaturas desplazable lateralmente que muestra los vinilos por categorías. Es posible cambiar de categoría o ver todos los vinilos a la vez.
2. **Capas**: Cuando un vinilo ha sido marcado, se nos habilita una lista que muestra las capas pertenecientes al vinilo con sus nombres y color por defecto. Haciendo *click* sobre el rectángulo de color tenemos la posibilidad de cambiar el color para el vinilo seleccionado o bien podemos presionar sobre "Todos" y la acción anterior se aplicaría a todas las capas. Haciendo *hover* sobre el nombre de la capa obtendríamos una previsualización de la misma.
3. **Selector de color**: Paleta de colores para cambiar el color sólido de fondo.
4. **Botón cambiar fondo**: Botón que nos permite subir una imagen de nuestro ordenador y aplicarla como fondo.

5. **Panel de propiedades**: Contiene los controles que nos permiten desplazar el vinilo horizontal o verticalmente, cambiar su tamaño, aplicar proyecciones, rotar o centrar.
6. **Hoja**: Área de trabajo donde se sitúa el vinilo. Puede llevar un color sólido o bien una imagen.
7. **Botón de imprimir / Ayuda**.



4.2.2 Capa de negocio

Nuestra capa de negocios en realidad se compone de dos sub-capas. Por un lado tenemos la capa conocida como *Business Logic Layer* o **BLL** que son los objetos de negocio, y por otro la capa *Data Access Layer* o **DAL**.
Los objetos de negocio son componentes que encapsulan los datos y la lógica de proceso para una entidad concreta del negocio. Sin embargo no se tratan de mecanismos de almacenamiento permanentes. Ya que los objetos de negocio no pueden almacenar datos de forma indefinida, la capa BLL debe apoyarse en la capa de datos para almacenar información y recuperarla.
Debido a esto, la capa BLL contendrá lógica para recuperar datos persistentes desde la capa de datos e insertarlos en los objetos de negocios y al revés, lógica que guarda datos obtenidos a partir de los objetos. Esto se conoce como la capa DAL.

Si bien la capa BLL depende como hemos visto de la capa DAL, sigue teniendo una gran importancia al hacer de intermediario entre la capa de presentación y la capa DAL. Habrá situaciones en las que necesitemos forzar la aplicación de las reglas de negocio. Por ejemplo,

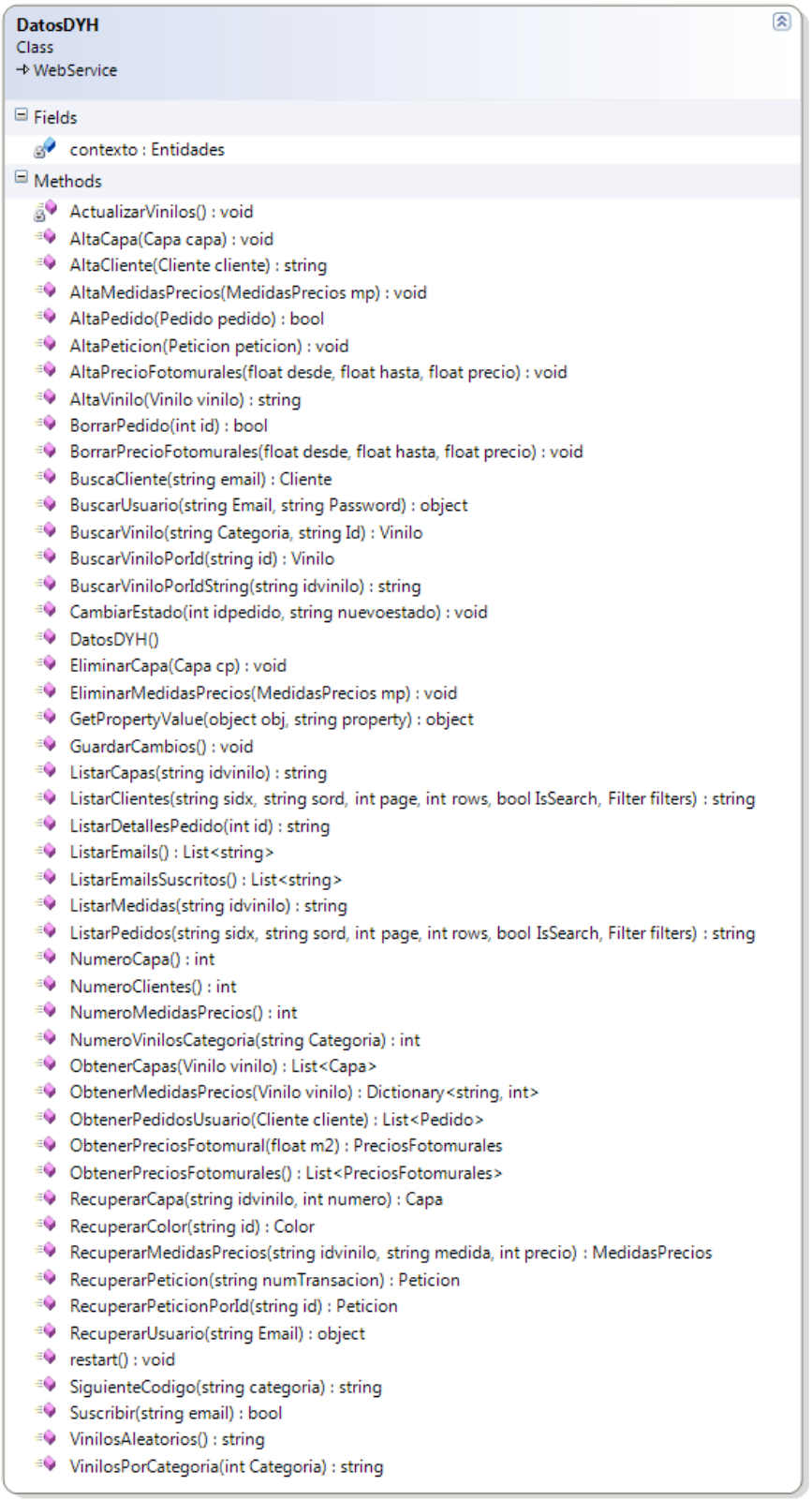
en el caso de *Decorate Your House*, los vinilos de tipo "Baños" no pueden añadirse al carrito ya que su compra se realiza por metro cuadrado y no podemos establecer 2 o 3 medidas de referencia como en los otros vinilos. Otra regla más trivial es el hecho de que no podemos adquirir dos veces el mismo producto. O que los administradores no puedan realizar compras. Para estos escenarios, necesitaremos aplicar unas reglas de negocio que estarán centralizadas en la capa BLL.
Aunque en una aplicación de mayor envergadura hubiéramos creado una librería aparte con todo el código de las clases, en nuestro caso, la capa BLL la hemos situado bajo el directorio /App_Code. En concreto las clases son Cliente.cs, Pedido.cs y Vinilo.cs. Recordemos que se trata de clases parciales porque en realidad estamos extendiendo la funcionalidad de las clases creadas automáticamente por Entity Framework.

En cuanto a cómo se relacionan las capas BLL y DAL, una opción era la de poner directamente la lógica de acceso a datos dentro de los objetos de negocio. En un principio parece una buena opción ya que permite tener todo bastante bien empaquetado. El problema es que si sufrimos algún cambio importante a nivel de base de datos, por ejemplo, una migración desde SQL Server a Oracle, y todo esto teniendo en cuenta que el código de los objetos puede ir variando, nuestra única opción es realizar una copia entera del código de los objetos para poder actualizar la capa DAL y así dar soporte a Oracle. Si efectivamente ocurre un cambio en el código de los objetos, con la estructura elegida, tendremos que actualizar tanto el código para SQL Server como para Oracle, lo cual puede llegar a ser tortuoso.

Una aproximación más flexible involucraría quitar todo lo que tiene que ver con el acceso a datos de dentro de los objetos de negocio y situarlo en una librería aparte. Esto nos da una clara separación entre los objetos de negocio y la capa de acceso a datos usada para rellenar esos objetos de negocio. Si volvemos al caso que discutíamos anteriormente, con la nueva estructura, simplemente tendríamos que hacer una copia de la capa DAL para SQL Server y convertirla para que funcione con Oracle. Cuando cambie la lógica de negocio, ya no hará falta introducir cambios en varios lugares porque no tendremos varias copias del mismo objeto que sólo se diferencian por el DAL que llevan dentro. Además cuando acabemos de convertir el DAL, nuestra aplicación será capaz de soportar bases de datos de dos proveedores distintos.

Cada vez que un objeto necesite acceder a la capa de datos, lo hará a través de una API de llamadas proporcionada por la DAL en vez de llamar directamente a la capa de datos. Es decir, todo el código relativo a la base de datos, estará contenido dentro del DAL haciendo que los objetos del negocio sean independientes de la base de datos.

Nuestra DAL se encuentra en el fichero DatosDYH.cs también del directorio /App_Code.
En la página siguiente se muestra la API de llamadas de nuestra DAL que como se puede observar es independiente de la base de datos, es decir que si quisiéramos dar soporte a otro gestor de bases de datos, bastaría con implementar una librería para con el mismo nombre y las mismas firmas de métodos, intercambiar una por la otra y todo seguiría funcionando exactamente igual. Tanto los parámetros como los valores devueltos son clases y tipos .NET.


DatosDYH
Class
WebService
Fields
contexto : Entidades
Methods
ActualizarVinilos() : void
AltaCapa(Capa capa) : void
AltaCliente(Cliente cliente) : string
AltaMedidasPrecios(MedidasPrecios mp) : void
AltaPedido(Pedido pedido) : bool
AltaPeticion(Peticion peticion) : void
AltaPrecioFotomurales(float desde, float hasta, float precio) : void
AltaVinilo(Vinilo vinilo) : string
BorrarPedido(int id) : bool
BorrarPrecioFotomurales(float desde, float hasta, float precio) : void
BuscaCliente(string email) : Cliente
BuscarUsuario(string Email, string Password) : object
BuscarVinilo(string Categoria, string Id) : Vinilo
BuscarViniloPorId(string id) : Vinilo
BuscarViniloPorIdString(string idvinilo) : string
CambiarEstado(int idpedido, string nuevoestado) : void
DatosDYH()
EliminarCapa(Capa cp) : void
EliminarMedidasPrecios(MedidasPrecios mp) : void
GetPropertyValue(object obj, string property) : object
GuardarCambios() : void
ListarCapas(string idvinilo) : string
ListarClientes(string sidx, string sord, int page, int rows, bool IsSearch, Filter filters) : string
ListarDetallesPedido(int id) : string
ListarEmails() : List<string>
ListarEmailsSuscritos() : List<string>
ListarMedidas(string idvinilo) : string
ListarPedidos(string sidx, string sord, int page, int rows, bool IsSearch, Filter filters) : string
NumeroCapa() : int
NumeroClientes() : int
NumeroMedidasPrecios() : int
NumeroVinilosCategoria(string Categoria) : int
ObtenerCapas(Vinilo vinilo) : List<Capa>
ObtenerMedidasPrecios(Vinilo vinilo) : Dictionary<string, int>
ObtenerPedidosUsuario(Cliente cliente) : List<Pedido>
ObtenerPreciosFotomural(float m2) : PreciosFotomurales
ObtenerPreciosFotomurales() : List<PreciosFotomurales>
RecuperarCapa(string idvinilo, int numero) : Capa
RecuperarColor(string id) : Color
RecuperarMedidasPrecios(string idvinilo, string medida, int precio) : MedidasPrecios
RecuperarPeticion(string numTransacion) : Peticion
RecuperarPeticionPorId(string id) : Peticion
RecuperarUsuario(string Email) : object
restart() : void
SiguienteCodigo(string categoria) : string
Suscribir(string email) : bool
VinilosAleatorios() : string
VinilosPorCategoria(int Categoria) : string

4.2.3 Capa de persistencia

En nuestra capa de persistencia también hay sub-capas con los sistemas de bases de datos. Las **tablas** definen el almacenamiento físico de la información en una base de datos pero los **procedimientos almacenados** permite manipular los datos en el momento en el que se insertan en esas tablas o son leídos desde ellas. Por ejemplo, supongamos que tenemos que de-normalizar una tabla y por tanto cambiar su estructura física, si accediéramos a los datos directamente desde la capa de negocios, nos vernos forzados a actualizar la capa de negocio para adaptarla a los cambios que se han hecho sobre la tabla. Si usamos una capa de procedimientos almacenados para acceder a los datos entonces podremos exponer la misma estructura lógica simplemente actualizando el procedimiento almacenado sin tener que tocar nada de código de la capa de negocios. El diseño en capas también dentro de la capa de persistencia lo que hace es reducir el impacto global de los cambios que puedan eventualmente surgir en la aplicación.

Volviendo al tema de los ORM, en nuestro caso a *Entity Framework*, comentar que cuando hemos generado un modelo de entidades -que lo hemos hecho a partir del esquema de base de datos existente- se nos ha creado un fichero .edmx que contiene una representación XML de:

- El modelo conceptual (CSDL o *Conceptual Schema Definition Language*).
- El modelo de datos (SSDL o *Store Schema Definition Language*).
- El mapeo entre ambos modelos (MSL o *Mapping Specification Language*).

La siguiente imagen muestra el contenido del fichero .edmx:

```
<edmx:Runtime>
  <!-- SSDL content -->
  <edmx:StorageModels>...</edmx:StorageModels>
  <!-- CSDL content -->
  <edmx:ConceptualModels>...</edmx:ConceptualModels>
  <!-- C-S mapping content -->
  <edmx:Mappings>...</edmx:Mappings>
</edmx:Runtime>
```

Es importante entender que el modelo conceptual o CSDL no debe ser necesariamente igual al modelo de datos o SSDL y por ello existe el modelo de mapeo (MSL). A partir de este archivo .edmx se ha generado de forma automática otro archivo Modelo.designer.cs que contiene las clases que representan el modelo conceptual. Para ver la implementación de la clase Cliente, ver el anexo 1.

El simulador es un caso aparte ya que la información necesaria estará almacenada físicamente en un fichero catalogos.xml en vez de contar con tablas en la base de datos. De esta manera, para obtener las categorías, vinilos, capas, etc. el simulador irá parseando el fichero XML línea a línea y habrá de construir un objeto manipulable. En la siguiente página se muestra la

estructura escogida para el fichero XML.

```
<ArbolCatalogos>
  <catalogo id="ID_CATALOGO">
    <categoria id="ID_CATEGORIA">
   <vinilo id="ID_VINILO" miniatura="DIRECTORIO_MINIATURA">
        <capa imagen="DIRECTORIO_IMAGEN" nombre="NOMBRE_CAPA" colorA="0" colorR="0"
colorG="0" colorB="0" codigoColor="CODIGO_COLOR" />
      </vinilo>
    </categoria>
  </catalogo>
</ArbolCatalogos>
```

5. Implementación

Este capítulo estará dedicado a hacer una breve reseña de las tecnologías y herramientas sobre las que nos hemos apoyado. Posteriormente, comentaremos algunos de los problemas críticos durante la implementación y la solución abordada en cada caso.

5.1 Tecnologías

- **HTML/XHTML/DHTML**: siglas de *HyperText Markup Language*, es un lenguage de marcado que se usa para describir la estructura y el contenido de una página web. Mientras el HTML es una aplicación del estándar SGML (*Standard Generalized Markup Language*), el XHTML o *eXtensible HyperText Markup Language* se trata de HTML que cumple las especificaciones más estrictas de XML. Entre las numerosas ventajas que ofrece el XHTML encontramos su gran extensibilidad a través del uso de nombres de espacio o *namespaces*. Las páginas XHTML permiten incluir fragmentos de otros lenguajes basados en XML.
  DHTML o *Dynamic HTML* se refiere a una colección de tecnologías usadas para crear aplicaciones interactivas. Se trata de una combinación de HTML estático, un lenguaje de programación del lado del cliente como puede ser JavaScript, un lenguaje de definición de la presentación del estilo de CSS, y el DOM. Una página DHTML tiene la capacidad de ir variando una vez cargada gracias a los *scripts* que se ejecutan y que pueden alterar características de la presentación o de los contenidos.

- **JavaScript/JQuery**: JavaScript es un lenguaje de *scripting* diseñado para añadir interactividad a las páginas web creando aplicaciones web. Fue implementado por Netscape en 1995. Es un lenguaje que se ejecuta en la parte cliente y está orientado a objetos. Se dice que es un lenguaje basado en prototipos; es decir, las clases no están presentes y la reutilización de comportamiento o herencia se lleva a cabo clonando objetos que sirven a modo de prototipos. Los objetos pueden heredar propiedades directamente de los unos a los otros, formando una cadena de objetos prototipo. Es un lenguaje interpretado, en otras palabras, los *scripts* pueden ejecutarse sin necesidad de una previa compilación. En las implementaciones más modernas, el código JavaScript admite el compilado *just-in-time* o JIT. Es decisión del navegador que partes del código deben ser compiladas para mejorar el rendimiento.
  JQuery es una librería JavaScript que simplifica muchas de las acciones complejas, por ejemplo: la manipulación de objetos del DOM, llamadas a servicios web, manejo de eventos, realizar animaciones, etc. Permite minimizar el número de líneas de código necesarias, reducir los ciclos de desarrollo y mejorar la mantenibilidad de la aplicación.

- **CSS**: Las hojas de estilo en cascada o *Cascading Style Sheets* fueron inventadas en 1997. Su objetivo es el de dar un formato al contenido de los documentos HTML, esto es, definir un layout, colores, fuentes, etc. Esto se consigue definiendo un sistema de reglas que influyen directamente sobre las propiedades de la página web. Las reglas se pueden insertar en el documento HTML o en un fichero aparte de manera que éstas

pueden afectar a muchos documentos a la vez. Están diseñadas para permitir la separación entre el contenido del documento y el formato del mismo.

- **AJAX**: También conocido como *Asynchronous JavaScript And XML*, es un método de construcción de aplicaciones para la web que procesa las peticiones del usuario de forma inmediata. No se trata de una tecnología nueva en sí misma, sino que combina varias herramientas que ya estabas disponibles incluyendo JavaScript, DHTML, XML, CSS, el modelo de objetos del documento o DOM, y el objeto XMLHttpRequest de Microsoft.
  AJAX permite que el contenido de las páginas web se actualice de forma inmediata cuando un usuario lleva a cabo una acción, al contrario de lo que ocurre con las peticiones HTTP, durante las cuales el usuario debe esperar que se complete la carga entera de una nueva página. Las aplicaciones creadas con AJAX usan un motor que actúa de intermediario entre el navegador y el servidor del cual se está solicitando la información. En vez de cargar una página de la forma tradicional, el navegador carga el motor AJAX, el cual muestra la página que el usuario acaba viendo. El motor continúa corriendo de fondo usando JavaScript para comunicarse con el servidor web. Cualquier entrada por parte del usuario provoca una llamada JavaScript al motor AJAX que responde de forma inmediata. Si el motor necesita datos extra, se los pide al servidor, usualmente usando XML o JSON mientras actualiza la página en simultáneo.

- **JSON**: *JavaScript Object Notation*. Se trata de un formato ligero usado para intercambiar datos. Un ejemplo de uso son las respuestas de los servicios web. En el pasado, los servicios web usaban XML como el formato de datos principal para devolver información, pero desde que se ha implantado JSON su uso ha ido desplazando al XML fundamentalmente por su ligereza y simplicidad. JSON está construido en dos estructuras: Una colección de parejas nombre/valor, lo que viene a ser un objeto, registro, estructura, diccionario, tabla *hash* o simplemente un array asociativo. Por otro lado, una lista ordenada de valores. En la mayoría de lenguajes, viene a ser un array, vector, lista o secuencia.

- **XML**: El XML o *eXtensible Markup Language* fue introducido en 1996 por la *World Wide Web Consortium* (W3C). Es un lenguaje de marcado tal como el HTML pero sin un formato fijo. Es decir que cuenta con palabras y *tags* que describen un documento e identifican sus partes. Si HTML estaba orientado a la presentación, el XML funciona como un medio para almacenar y transportar datos estructurados. Todos los archivos XML siguen unas reglas básicas de sintaxis y forma. XML no tiene como fin mostrar datos, sino que hacerlos portables y accesibles.

- **Servicios web**: Un servicio web es una interfaz externa provista por un sitio web que se puede llamar desde otro sitio web. Los servicios web dan lugar a una técnica para acceder a un método de un objeto de forma local o remota. Un servicio web puede ser utilizado por una aplicación escrita en cualquier lenguaje y corriendo sobre cualquier sistema operativo. Utilizan HTTP como el medio de transporte por debajo. Soportan la

tanto la comunicación síncrona como asíncrona entre cliente y servidor web donde se hospeda el servicio. Cuando la conexión es síncrona, el cliente envía una petición al servidor y espera su respuesta quedando incapacitado para realizar otras acciones mientras espera. Por el otro lado, en una conexión asíncrona, el cliente puede seguir procesando otras tareas mientras sigue esperando. El cliente responde al resultado de la petición del servicio web cuando esté disponible. La comunicación entre cliente y servidor se hace intercambiando mensajes siguiendo el protocolo *Simple Object Access Protocol* o SOAP.

- **XML-RPC**: XML-RPC o *eXtensible Markup Language Remote Procedure Call* es un protocolo para compartir mensajes entre cliente y servidor. Fue desarrollado para lidiar con las limitaciones de XML/HTTP. Se trata de un protocolo de llamadas a procedimientos remotos ligero, simple y portable. Está pensado para intercambiar estructuras de información en un escenario distribuido y descentralizado. Ha sido desarrollado para ser independiente de cualquier modelo de programación.

- **Silverlight**: Es una implementación del *framework* .NET multi-plataforma y multi-navegador para desarrollar aplicaciones interactivas o más conocidas como *Rich Interactive Applications* (RIA). Se trata de una combinación de diferentes tecnologías dentro de una misma plataforma de desarrollo. Permite construir interfaces de usuario avanzadas, muy semejantes a las aplicaciones de escritorio. Posibilita conexiones asíncronas con el servidor de manera que la interfaz de usuario sigue funcionando mientras se espera una respuesta. Otras de los beneficios de las aplicaciones Silverlight es su integración nativa con servicios web, el uso de SSL o *Secure Socket Layer* para comunicarse con el servidor de forma segura, o la posibilidad de implementar el *code behind* en cualquiera de los lenguajes de programación ofrecidos por la plataforma .NET.

- **ASP .NET/C#**: Es una tecnología del lado del servidor para el desarrollo de sitios web dinámicos, aplicaciones web y servicios web. Esta construida sobre el *Common Language Runtime* o CLR que es la máquina virtual responsable de la ejecución de los programas .NET. Al tratarse de una plataforma de ejecución común, ASP.NET soporta múltiples lenguajes de programación incluyendo C#, VB.NET, C++, J#, etc. ASP .NET soporta el compilado automático. La primera vez que un usuario realiza una solicitud, ASP.NET compila el código en una librería .dll. Como se sabe, el código compilado se ejecuta mucho más velozmente que los lenguajes de *scripting* al requerir estos un parseo adicional. Los *Web Forms,* son el bloque principal para el desarrollo de aplicaciones. ASP.NET viene con una serie de controles *Web Form* incorporados que son responsables de generar la interfaz de usuario. Son equivalentes a los controles HTML como, por ejemplo, los campos de texto o botones sólo que están preparados para disparar eventos y configurar manejadores que den respuesta a los mismos. Además permiten la separación entre el código que se ejecuta en el servidor y el fichero HTML.

- **IIS**: Se le llama IIS o *Internet Information Server* a los servidores de Internet (web, FTP, SMTP, etc.) de Microsoft. Se trata de una colección integrada de servicios para publicar contenidos en Internet. IIS se administra a través del *Internet Services Manager.* IIS es capaz de soportar varios servidores virtuales funcionando como servidores web independientes. Esta capacidad hace de IIS un servidor aconsejable para la publicación de contenidos a larga escala pero también para necesidades más pequeñas como puede ser una intranet corporativa. IIS soporta aplicaciones CGI o *Common Gateway Interace* además de las *Active Server Pages* (ASP) y ASP .NET.

- **Entity Framework**: Es un mapeador relacional de objetos, o en inglés *Object Relational Mapper* (ORM). Básicamente, se encarga de generar los objetos de negocio y entidades en concordancia con la base de datos y las tablas. Además realiza un seguimiento de esos objetos, de manera que si cambia la fuente de datos, los objetos se actualizan de forma automática y viceversa. EF provee de mecanismos para llevar a cabo de forma automática operaciones CRUD (*Create*, *Read*, *Update*, *Delete*), gestionar relaciones "1 a 1", "1 a muchos", "muchos a muchos", y soporta la relaciones de herencia entre entidades. Otra gran ventaja de usar un ORM, en este caso *Entity Framework*, es el hecho de poder contar con un lenguaje de *queries* que sea independiente del proveedor de base de datos. Las *queries* se escriben en LINQ o Entity SQL y luego son traducidas en tiempo de ejecución por los proveedores para cada caso particular de sintaxis de base de datos.

5.2 Herramientas

- **Visual Studio 2010**: Es un entorno de desarrollo integrado o *Integrated Development Environment* (IDE). Visual Studio se puede usar para escribir aplicaciones de consola, aplicaciones para Windows, servicios web, aplicaciones para Windows Mobile, aplicaciones web en cualquiera de los lenguajes de la plataforma .NET. Visual Studio incluye algunas características orientadas a hacernos la vida más fácil a la hora de programar. Por ello, cabe destacar la característica de *IntelliSense* que lo que hace es mostrarnos una lista de las clases, propiedades y métodos disponibles sobre la clase en la que estamos trabajando. Visual Studio incluye diseñadores WYSIWYG (*What You See Is What You Get* - "Lo que ves es lo que obtienes") para aplicaciones de Windows, teléfonos móviles, y aplicaciones ASP.NET. Las interfaces gráficas pueden diseñarse de forma visual sabiendo que el resultado de nuestro diseño será exactamente igual a la apariencia final que tenga la aplicación. Otra característica muy útil para tareas de mantenimiento y resolución de *bugs* es la posibilidad de depurar el código o *debbuging*. Podemos detener la ejecución en cualquier línea. Dispondremos de una consola para ejecutar instrucciones, un panel con la lista de variables y sus valores en el instante preciso de la detención, un detalle de la pila de llamadas, etc. En esta versión, Visual Studio ha incorporado herramientas para ayudar a la gestión del equipo, organizar el trabajo, dividir las tareas y realizar un seguimiento de las mismas.

- **Expression Blend 4**: Es una herramienta de diseño de interfaces gráficas desarrollada por Microsoft. Contiene un editor WYSIWYG que permite diseñar visualmente una interfaz y que nos genera todo el código XAML automáticamente por nosotros. Recordemos que las interfaces de las aplicaciones Silverlight o WPF (*Windows Presentation Foundation*) se escriben en XAML. Blend 4 trae varias herramientas para mejorar la interactividad. Por ejemplo podemos escribir "disparadores" y "acciones". Los disparadores son objetos que contienen una o más acciones y las invocan en respuesta a un estímulo. Por otro lado tenemos los "comportamientos"; a primera vista se parecen a las acciones ya que también se tratan de unidades contenidas de funcionalidad. La diferencia radica en que las acciones esperan ser invocadas y cuando lo son realizan una operación. Un comportamiento no mantiene el concepto de invocación; actúa más bien como una añadidura opcional a un objeto. Un ejemplo de comportamiento es la capacidad de arrastrar y soltar que podemos añadirla a cualquier objeto.

- **SQL Server 2008**: Se trata de un gestor de bases de datos o RDMSD (*relational database management system*). La versión que hemos empleado viene con algunas mejoras destinadas a mejorar el mantenimiento de la base de datos, disponibilidad, seguridad y rendimiento. SQL Server incluye un monitor gráfico de actividades/procesos. Los administradores de bases de datos pueden ver a alto nivel los procesos ejecutándose y las estadísticas que ayuden a entender y resolver problemas. SQL Server trae un servicio de auditoria que permite llevar un registro exhaustivo de los eventos más importantes que ocurren: quién está accediendo, qué cambios ha hecho, la hora de los cambios...etc. A medida que las base de datos crecen, aumenta el espacio en disco necesario, la cantidad de memoria, y el número de operaciones I/O. Para resolver estos problemas SQL Server ofrece la compresión de datos ya sea a tablas, particiones de tablas, índices. Otras características avanzadas: TDE o *Transparent Data Encryption*. Permite encriptar los datos mientras se encuentran en el disco de forma transparente al resto de aplicaciones. *Database snapshots* son copias de la base de datos original de sólo lectura. Se usan como herramientas de reporte aunque también en caso de error puede reemplazarse la base de datos primaria por su *snapshot*. Destacar la integración CLR, es decir, la posibilidad de crear objetos .NET con el motor de la base de datos. Podremos crear procedimientos almacenados, disparadores, y funciones usando la sintaxis de lenguajes conocidos como C# o VB.NET.

- **Gimp**: Es el editor gráfico de imágenes que hemos usado para adaptar las imágenes que nos ha pasado el cliente. Algunas de ellas las hemos tenido que convertir en formato PNG para aplicar transparencias. Más allá de alguna transformación de medida o colores, no se le ha dado gran uso al editor fundamentalmente porque CSS3 nos ahorra la tarea de tener que estar creando imágenes para cada elemento gráfico: botones, fondo, menús, etc.

- **OpenOffice Writer/Draw**: OpenOffice Writer es el editor de texto que hemos echado en mano para redactar la memoria y escribir nuestras notas a lo largo del proyecto.

Trae todas las características de un editor de textos moderno: diccionario autocorrector multilenguaje, formateo y estilo con CSS, generación de tablas, indexado de términos, conversor a formato PDF o HTML. Para realizar diagramas también hemos optado por otro producto de la familia OpenOffice en este caso Draw. Con Draw es posible crear dibujos simples o complejos y exportarlos en casi cualquier formato. Está permitido insertar tablas, gráficos o cualquier ítem creado con programas de OpenOffice.

- **FileZilla**: Es el cliente de FTP empleado para subir los archivos al entorno de producción. Soporta FTP, FTP Secure o FTP-SSL y SSH FTP. Puede correr en cualquier plataforma: Windows, Linux, BSD, Mac. Otras características a destacar son el soporte para IPv6; resumen de transferencias, transferencias de archivos de más de 4GB, permite configurar límites de velocidad, permite el filtrado por nombre de archivos.

- **Administrador de IIS 6.0**: Es la interfaz gráfica desde la que hemos configurado nuestro portal en el entorno de desarrollo. Permite configurar la piscina de aplicaciones, sitios FTP, SMTP o NNTP. Desde esta interfaz se puede añadir sitios, borrarlos, detener o reanudar su ejecución; restablecer una configuración de servidor, crear directorios virtuales, etc.

5.3 Detalles de implementación

5.3.1 Capa de presentación

En este subapartado centraremos nuestra atención en la GUI o interfaz gráfica de usuario. Hablaremos de los pasos necesarios para construir un formato unificado de página o sistema de plantillas. Haremos referencia sobre las reglas que nos aporta CSS3 para ajustar la apariencia de los elementos. También haremos mención de algunas de las librerías JavaScript que nos han proporcionado controles listos para ser usados. Por supuesto comentaremos cómo se realiza la comunicación por medio de AJAX a la capa de negocios o a servicios web.

5.3.1.1 Sistema de plantillas

ASP .NET, nuestro framework de desarrollo, permite la creación de *layouts* o estructuras de documentos consistentes y unificados a través de lo que se conoce por el nombre de *Master Pages*. Una *Master Page*, define el *look and feel* y el comportamiento común para todas las páginas del portal o bien para un grupo de páginas. Cuando el usuario realiza una petición, nuestro portal genera los contenidos de la página que pueden ser dinámicos o no, y el código HTML resultante se mergea con los contenidos de la *Master Page* para producir la salida final, es decir, se está combinando el *layout* proporcionado por la *Master Page* y los contenidos de la página pedida por el usuario.

Una *Master Page* es un archivo ASP.NET con la extensión .master. Este archivo puede incluir texto estático, etiquetas HTML, controles del servidor, hojas de estilo, código JavaScript entre otros elementos. En nuestro proyecto hemos creado las siguientes plantillas:

- **Maestra.master**: Usada para las páginas normales del sitio excluyendo la sección "Fotomurales".
- **MasterFotomurales.master**: Usada exclusivamente para la sección "Fotomurales".
- **MasterAdmin.master**: Usada tanto para el "Área de clientes" como para el "Panel de control" del administrador.

En la siguiente página podemos apreciar el detalle de la plantilla "Maestra.master".

En amarillo se muestran las zonas de importancia que vamos a comentar. En la primera zona encontramos la directiva **@ Master** que indica que el presente documento es una página maestra. Más abajo, dentro del *header* encontramos todas las importaciones de *scripts* que al ser añadidas en la página maestra, serán heredadas por todas las páginas que hagan uso de ella. Es de notar que nuestra librería propia de JavaScript se encuentra empaquetada dentro del fichero **decorateyourhouse.js**. Nuestro estilo viene enmarcado en el fichero **pordefecto.css**. A continuación del título, nos encontramos con el tag *ContentPlaceHolder* que no es más que un punto de inserción para que las páginas de contenidos que vayan a usar esta plantilla introduzcan el código HTML, CSS, JS, etiquetas, meta tags, etc. que fueren necesarios. Ya casi al final del documento, dentro del *body*, encontramos otra etiqueta *ContentPlaceHolder* que será el sitio donde se vuelquen los contenidos de la página hija.

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="Maestra.master.cs"
Inherits="Maestra" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <link rel="shortcut icon" href="/images/favicon.ico" />
    <meta name="keywords" content="vinilos decorativos fotomurales valencia españa"/>
    <meta name="description" content="Vinilos decorativos - Fotomurales - Amplio catálogo -
Vinilos personalizables. La mejor tienda online en rapidez y calidad. Envíos a toda España
en 48 horas."/>
    <meta name="google-site-verification" content="Z-
2gyh7NcUSJSdR32F8oBqrDQmUBayHA3t6NHUEzF6M" />
    <meta http-equiv="Content-Language" content="es-es" />
    <meta name="DC.title" content="Vinilos decorativos" />
    <meta name="geo.region" content="ES-VC" />
    <meta name="geo.placename" content="Torrent" />
    <meta name="geo.position" content="39.436574;-0.473299" />
    <meta name="ICBM" content="39.436574, -0.473299" />

    <script src="/scripts/jquery-1.4.1.min.js" type="text/javascript"></script>
    <script src="/scripts/jquery-ui-1.8.6.custom.min.js" type="text/javascript"></script>
    <script src="/scripts/jquery.cookie.js" type="text/javascript"></script>
```

```
    <script src="/scripts/jquery.json-2.2.js" type="text/javascript"></script>
    <script src="nivo-slider/jquery.nivo.slider.pack.js" type="text/javascript"></script>
    <script src="/scripts/decorateyourhouse.js" type="text/javascript"></script>

    <link href="/css/pordefecto.css" rel="stylesheet" type="text/css" />
    <link href="/css/jquery-ui-1.8.6.custom.css" rel="stylesheet" type="text/css" />
    <link href="nivo-slider/nivo-slider.css" rel="stylesheet" type="text/css" />
    <link href="nivo-slider/themes/default/default.css" rel="stylesheet" type="text/css" />

    <title>Vinilos Decorativos Fotomurales Valencia España </title>

   <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
 <div class="slider-wrapper theme-default">
    <div id="slider" class="nivoSlider">
    <a href=""><img src="../nivo-slider/demo/images/1.jpg" alt=""  /></a>
    <a href=""><img src="../nivo-slider/demo/images/2.jpg" alt=""  /></a>
    </div>
</div>
<div id="vinilos-cristales">
    <div id="cent">
        <a href="/VinilosDecorativos.aspx">Inicio</a>
        <a href="/DecoracionParedes.aspx">Información</a>
        <a style="cursor:pointer;"
onclick="popup('/Simulador/SimuladorVinilosDecorativos.aspx')">Simulador</a>
        <a href="/ColocarVinilosDecorativos.aspx">Colocación</a>
        <a href="/ComprarVinilosDecorativos.aspx">Comprar</a>
        <a id="btnAreaClientes" href="/miembros/AreaClientes.aspx">Clientes</a>
        <a href="/Contacto.aspx">Contacto</a>
        </div>
    </div>
<div id="vinilos-adhesivos-murales-fotomurales">
<div id="nuevo-vinilo-decorativo"></div>
    <div id="compras-vinilos">
    <table id='tabla-vinilos-pared' class='horzebra2'>
    <thead>
    <tr>
     <th scope='col'>Eliminar</th>
    <th scope='col'>Ref.</th>
    <th scope='col'>Medida</th>
    <th scope='col'>Color</th>
    <th scope='col'>Orientación</th>
    <th scope='col'>Acabado</th>
    <th scope='col'>Blanco y negro</th>
    <th scope='col'>Cantidad</th>
    <th scope='col'>Precio</th>
    </tr>
    </thead>
    <tbody></tbody>
    </table>
    <span>Los envíos se realizan por medio de agencia y tienen un plazo de 24 / 48 horas.
    Su costo es de 8 € para toda la Península y de 12 € para Baleares (Islas Mayores). Para
otros lugares consultar importe.
    <strong> Envío GRATIS para compras a partir de 60 €</strong>
    </span>
```

```
    </div>
    <img id="abanicoizq" alt="" src="/images/abanico_izq.png" width="483" height="345" />


    <div id="tienda-de-vinilos-decorativos">

    <div id="tipos-de-vinilos-decorativos">
    <ul>
    <li id="cabFotomurales" class=""><a href="../Fotomurales.aspx"
onclick="">Fotomurales</a></li>
<!--<li id="cabCuadros" class=""><a href="Cuadros.aspx" onclick="">Cuadros</a></li>
<li id="cabEspejos" class=""><a href="Espejos.aspx" onclick="">Espejos</a></li> -->
<li id="cabVinilos" class="notRotated"><a onclick="">Vinilos</a>
<ul>
<li class="childLi"><a name="vinilos-ornamentales"
onclick="javascript:jQuery.fn.MandarPeticion(this,'0')">Ornamentales</a></li>
<li class="childLi"><a name="vinilos-de-ciudades"
onclick="javascript:jQuery.fn.MandarPeticion(this,'1')">Ciudades</a></li>
<li class="childLi"><a name="vinilos-orientales"
onclick="javascript:jQuery.fn.MandarPeticion(this,'2')">Orientales</a></li>
<li class="childLi"><a name="vinilos-infantiles"
onclick="javascript:jQuery.fn.MandarPeticion(this,'3')">Infantiles</a></li>
<li class="childLi"><a name="vinilos-musicales"
onclick="javascript:jQuery.fn.MandarPeticion(this,'4')">Musicales</a></li>
<li class="childLi"><a name="vinilos-de-personajes"
onclick="javascript:jQuery.fn.MandarPeticion(this,'5')">Personajes</a></li>
<li class="childLi"><a name="vinilos-vintage"
onclick="javascript:jQuery.fn.MandarPeticion(this,'6')">Vintage</a></li>
<li class="childLi"><a name="vinilos-cocina"
onclick="javascript:jQuery.fn.MandarPeticion(this,'7')">Cocina</a></li>
<li class="childLi"><a name="vinilos-bebes"
onclick="javascript:jQuery.fn.MandarPeticion(this,'10')">Bebés</a></li>
<li class="childLi"><a name="vinilos-para-baños"
onclick="javascript:jQuery.fn.MandarPeticion(this,'8')">Baños</a></li>
<li class="childLi"><a name="vinilos-varios"
onclick="javascript:jQuery.fn.MandarPeticion(this,'11')">Varios</a></li>
</ul>
</li>
</ul>

    </div>
    <div id="vinilos-personalizados">

    <div id="pegatinas-vinilo">

    <img name="murales-fotomurales" id="carrito" alt="Carrito" src="/images/carrito.png" />
    <div id="vinilos-decoracion">


    <form id="formLogOut" action="/Salir.aspx" method="post"><span id="btnLogin"><asp:Label
runat="server" CssClass="labelNombreUsuario" ID="nombreUsuario"
Text=""></asp:Label></span><label id="btnSalir" runat="server"></label></form>
        <span id="vinilos-decorativos-infantiles">Tu carrito está vacío</span>

    </div>
      <img id="ajax-loader" src="/images/ajax-loader.gif" alt="" />
  </div>
```

```
        <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">

        </asp:ContentPlaceHolder>

        </div>
        <img id="logo" alt="" src="/images/logo.png" width="223" height="223" />

    </div>

       <div id="vinilos-decorativos-en-pared">

       <div>© Copyright 2010-2011 <b>Decorate Your House</b> - Contacto:
info@vinilosyfotomurales.com - Teléfono: 961 097 259. <br /> Vinilos decorativos - Vinilos
adhesivos - Vinilos para cristales - Vinilos en pared - Vinilos personalizados - Pegatinas
vinilos - Decoración interior y exterior en vinilo - Murales - Fotomurales.</div>
     </div>

     <img id="abanicoder" alt="" src="/images/abanico_der.png" width="595" height="401" />

     </div>
     <ul id="followTab">
  <li><a class="newsletter" href="" title="Suscribirse a nuestros boletines
informativos"><span>Suscribirse a nuestros boletines informativos</span></a></li>
  <li><a class="twitter" href="http://twitter.com/vinilosvlc" title="Sigue DecorateYourHouse
en Twitter"><span>Sigue DecorateYourHouse en Twitter</span></a></li>
  <li><a class="facebook" href="http://www.facebook.com/pages/VINILOS-DECORATIVOS-
FOTOMURALES-VALENCIA/229330393779654" title="Sigue DecorateYourHouse en
Facebook"><span>Sigue DecorateYourHouse en Facebook</span></a></li>
</ul>

</body>
</html>
```

En las páginas de contenido, que no son más que archivos .aspx que pueden o no tener un fichero .cs con su *Code Behind*, nos encontraremos con una directiva @ Page que establece a qué *Master page* la página se encuentra "atada". En la siguiente página de contenidos, que se trata de la página de inicio, vemos que el HTML se inserta dentro de un tag *asp:Content* lo cual establece que ese contenido debe ubicarse dentro de la región indicada por el atributo *ContentPlaceHolderID* de la plantilla a la que está vinculada.

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Maestra.master" AutoEventWireup="true"
CodeFile="VinilosDecorativos.aspx.cs" Inherits="Inicio" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">

    <div id="inicio">

 <ul class="gallery">
       <li>
             <a href=""  class="link"><figure><span><img class="tb" src="" alt=""
/><figcaption><h3>Titulo</h3><p>Contenido</p></figcaption></span></figure></a>
```

```
        </li>
    <li>
                <a href=""  class="link"><figure><span><img class="tb" src="" alt=""
/><figcaption><h3>Titulo</h3><p>Contenido</p></figcaption></span></figure></a>
        </li>
    <li>
                <a href=""  class="link"><figure><span><img class="tb" src="" alt=""
/><figcaption><h3>Titulo</h3><p>Contenido</p></figcaption></span></figure></a>
        </li>
       <li>
                <a href=""  class="link"><figure><span><img class="tb" src="" alt=""
/><figcaption><h3>Titulo</h3><p>Contenido</p></figcaption></span></figure></a>
        </li>
       <li>
                <a href=""  class="link"><figure><span><img class="tb" src="" alt=""
/><figcaption><h3>Titulo</h3><p>Contenido</p></figcaption></span></figure></a>
        </li>
       <li>
                <a href=""  class="link"><figure><span><img class="tb" src="" alt=""
/><figcaption><h3>Titulo</h3><p>Contenido</p></figcaption></span></figure></a>
        </li>
</ul>
<div id="areaPaginas"></div>
    </div>
</asp:Content>
```

5.3.1.2 Estilo con CSS3

Para dar formato a nuestros documentos HTML hemos recurrido a las hojas de estilo en cascada en su última versión. Las hojas de estilo nos permiten tener separada la presentación de la estructura. Esto es posible aislando las características de la presentación en un documento aparte en lugar de tenerlas incrustadas en el propio código HTML. Las ventajas saltan por sí solas; un mantenimiento más fácil y eficiente y un sistema más flexible porque estamos construyendo nuestra aplicación en módulos. Las hojas de estilo nos permiten mejorar la usabilidad de nuestra web; teniendo las propiedades de estilo contenidas en un sólo fichero y quitando todas las referencias a estilo en cada uno de los documentos HTML, conseguimos reducir su tamaño y además la hoja de estilo sólo se descargaría una vez (las siguientes veces estará cacheada).

Usar CSS3 nos ha permitido diseñar el portal directamente desde el navegador en vez de tener que construir un *mockup* con un programa del estilo de Photoshop. Además, nos ha sido posible probar distintos tamaños de pantalla y ver la funcionalidad desde el primer momento algo que no es posible con un *mockup*. Si hubiéramos optado por hacer un diseño en Photoshop, tendríamos la duda de cómo reaccionaría ese diseño ante un cambio de tamaño de pantalla. Además no hubiésemos sabido como se vería el diseño en los distintos navegadores, o cómo reaccionaría ante interacciones básicas como :hover.

Antes de CSS3 era común recurrir a Photoshop o similares para conseguir efectos de esquinas redondeadas, sombras difuminadas, resplandores, gradientes, etc. Sin embargo, con CSS3 y

la capacidad de renderizado de los navegadores se puede conseguir los mismos resultados que si hubiésemos tratados las imágenes con un editor. A continuación vamos a comentar el uso en nuestro proyecto de algunas de las novedades de CSS3.

RGBA

Introducir colores con RGBA es fácil. Podemos especificar la cantidad de rojo, verde y azul que tiene un color. La A se refiere a "alfa", tiene que ver con el nivel de opacidad del color o la cantidad de transparencia. No sólo podemos cambiar los valores de color sino también controlar cuánto debe verse de aquello que esté detrás. Algo así como las capas en Photoshop. En la imagen de la derecha podemos ver el resultado de aplicar la regla **`background-color: rgba(0,0,0,.7)`** sobre el elemento **`figcaption`**. El color se ha establecido a negro con una transparencia del 70%.

Fuentes web con @font-face

No es una característica nueva de CSS3, de hecho ya se había propuesto para CSS2 y se ha había implementado en Internet Explorer desde la versión 5, eso sí, siguiendo un formato que muchos navegadores no decidieron adoptar. Con la salida de Safari 3.1, los diseñadores empezaron a usar fuentes OpenType (.otf) o TrueType (.ttf). Para usar fuentes web, cada variante de la familia de fuentes debe declararse usando la regla @font-face. De esta manera el navegador podrá saber la fuente y descargarla.

```
@font-face {
    font-family: 'Humanist777 BT';
    src: url('humanist/font-webfont.eot');
    src: url('humanist/font-webfont.eot?#iefix') format('embedded-opentype'),
         url('humanist/font-webfont.woff') format('woff'),
         url('humanist/font-webfont.ttf') format('truetype'),
         url('humanist/font-webfont.svg#Humanist777Regular') format('svg');
    font-weight: normal;
    font-style: normal;
}
```

Ya para aplicarla sobre un elemento concreto, no tenemos más que hacer:

```
body {
  ...
  font-family:Humanist777 BT;
  ...
}
```

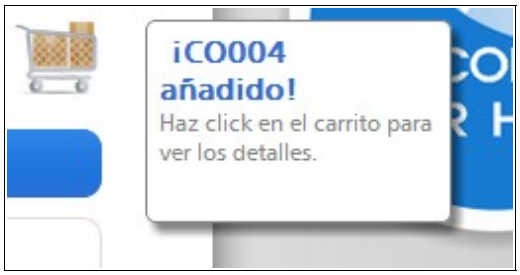
Text-shadow

Para aplicar un efecto simple de sombras a textos hemos usado la regla text-shadow. La siguiente imagen muestra el resultado para el texto del botón de compra.


```
.awesome {
text-shadow: 0 -1px -4px rgba(0, 0, 0, 1.0);
}
```

Box-shadow

La propiedad box-shadow permite implementar de forma fácil sombras difuminadas (exteriores o interiores) en elementos de tipo caja especificando un valor de color, tamaño, difuminado y desplazamiento.




```
#nuevo-vinilo-decorativo {
  ...
  box-shadow: 4px 5px 5px rgba(0,0,0,0.5), inset 4px 5px 5px rgba(255,255,255,0.75);
  ...
}
```

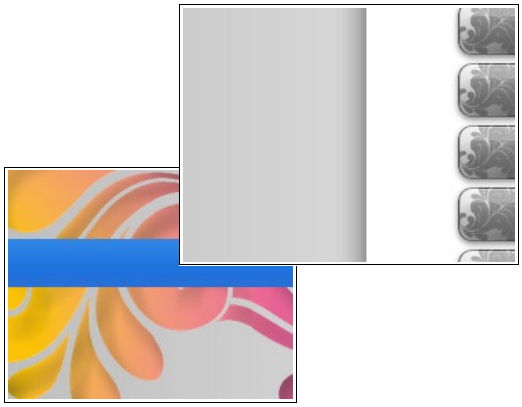
Border-radius

Pasar a bordes redondeados puede ser una pesadilla si pensamos hacerlo desde un programa de edición de imágenes ya que implica tener que recrear cada imagen y elemento de la interfaz. Aplicando la regla border-radius a nuestras clases, conseguimos formas redondeadas en un momento.


```
#header-vinilo {
  ...
  border-radius: 10px;
}
```

Uso de gradientes

Para contrarrestar el diseño plano, hemos combinado imágenes transparentes PNG con gradientes. Para lograr el gradiente blanco a azul, ponemos un fondo plano de color #3366cc y por encima insertamos una imagen con un 50% de margen que se repite horizontalmente. El borde de la página de contenidos es una imagen con un gradiente de transparente a gris y luego blanco que hemos repetido en vertical.



```
#vinilos-cristales {
background: #3366cc url(../images/ui-bg_highlight-soft_25_0073ea_1x100.png) 50% 50% repeat-x;
}

#tienda-de-vinilos-decorativos {
background-image:url('../images/tb_bg.png');
background-repeat:repeat-y;
background-position:left top;
...
}
```

5.3.1.3 Librerías JavaScript

Usar librerías JavaScript frente a escribir el código uno mismo tiene como principal beneficio poder reusar el código en diferentes partes del sitio. Los ficheros JavaScript van a ser más pequeños en tamaño, más fáciles de manipular y depurar. Cuando necesitamos cambiar una parte del código, los cambios se hacen en un sitio específico y no hay que estar replicando el mismo cambio en distintos documentos lo cual reduce el riesgo de introducir errores humanos. Ya que en general las librerías vienen optimizadas para hacer las cosas de la mejor manera posible, importando librerías nos aseguramos una reducción del número de líneas innecesarias. Las librerías suelen contar con documentación o al menos estar comentadas en código. Si más adelante un compañero se sumara al proyecto, podría comenzar de inmediato sin tener que aprender los conocimientos del programador que estuvo a cargo previamente. Las librerías nos ahorran mucha tarea a la hora de hacer que nuestra aplicación sea compatible con todos los navegadores. Estas suelen venir preparadas para soportar todos los navegadores y en caso de que salga una nueva versión del navegador, basta con actualizar la librería ahorrándonos horas de trabajo para adaptarlas nosotros mismos.

Las librerías JavaScript ofrecen una vía rápida para incorporar funcionalidades que son comunes en muchos sitios web. Desde la validación de datos de entrada, ventanas emergentes, efectos y animaciones, a casi cualquier código encapsulado que puede extender la funcionalidad y que otros programadores pueden usar. Además, como las librerías son de dominio público y muchos sitios webs ya las están empleando, contamos con la ventaja que ya han sido probadas y depuradas por la comunidad. El no tener que implementar un listado, o un botón, que parecen cosas triviales pero pueden llegar a consumir un tiempo de desarrollo considerable, ha sido positivo para centrarnos en lo que realmente importa que es la construcción de la lógica de negocio.
Las librerías JavaScript son un paso más para mejorar la experiencia de usuario y acercarla a lo que sería una aplicación de escritorio.

Algunas de las librerías que hemos usado:

JQuery
Ha sido básica para facilitarnos la tarea de programar en JavaScript. Nos permite encontrar elementos del DOM usando selectores de la misma manera que en CSS además de poder manipularlos de una manera muy intuitiva. Se ha hecho un uso intensivo de JQuery sobre todo para cambiar el estilo de los elementos *on the fly* o dinámicamente. Según lo que el usuario accione en el interfaz, podemos hacer que aparezca un elemento, se desplace, cambie de color, de tamaño, realice algún tipo de animación, etc.

```
$("#nuevo-vinilo-decorativo").animate({ opacity: 1, width: '141'}, 500, function () {
 $("#nuevo-vinilo-decorativo").animate({ height: '90'}, 1000, function () {
   $("#nuevo-vinilo-decorativo").animate({height: '1'}, 500, function () {
    $("#nuevo-vinilo-decorativo").animate({opacity: '1', width: '1'}, 500, function () {
      $("#nuevo-vinilo-decorativo").hide();
      });
    });
   }).delay(2000);
 });
```

Arriba vemos el código que implementa una serie de animaciones concadenadas que cambian propiedades de estilo del nuevo-vinilo-decorativo. Este div aumenta de tamaño hasta alcanzar una medida, permanece estático dos segundos para luego reducir su tamaño hasta desaparecer por completo.

Otra operación que se repite frecuentemente es la de cambiar la estructura HTML de la página; encontrar un elemento e insertar HTML en su interior, o remplazar un elemento entero por otro, o bien cambiar el orden de los nodos HTML, etc. Podemos verlo ilustrado en el siguiente ejemplo en el que se inserta un dinámicamente dentro del :

```
$("#nuevo-vinilo-decorativo").append("<span class='val'>Haz click en el carrito para ver los
detalles.</span>");
```

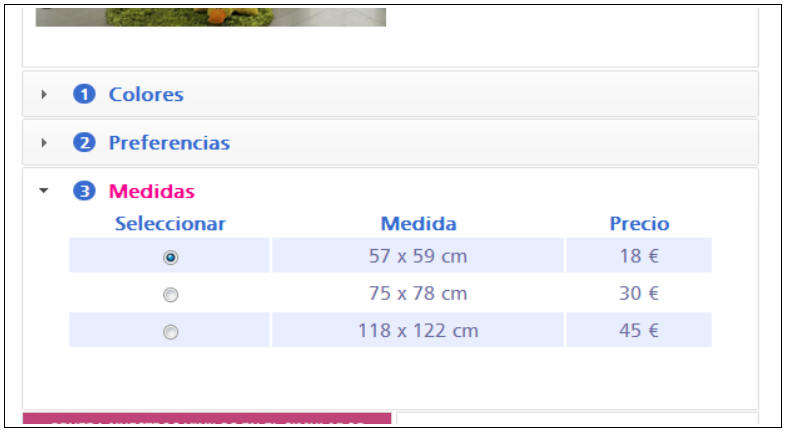
Por supuesto que de nada nos serviría tener unos controles muy atractivos estéticamente pero que al accionarlos no hubiera respuesta. Para ello, necesitamos asignar manejadores a los eventos. En este caso vemos parte del código que se ejecuta al pulsar el botón de "Añadir al carrito":

```
$("#areaComprarVinilosDecorativos button").click(function () {
 if (!userIsAdmin()) {
   var ref = vinilos.getReferencia();
   var col = ""
   for (var i = 0; i < coloresElegidos.length - 1; i++) {
     col += hex2cod(coloresElegidos[i]) + "/"
   }
   col += hex2cod(coloresElegidos[coloresElegidos.length - 1])
   var ori = $('input[id=original]').is(':checked') ? "Original" : "Invertido"
   var aca = $('input[id=mate]').is(':checked')) ? "Mate" : "Brillo"
   var can = $("input#cantidad").val()
   var pre = parseFloat(precioSeleccionado.substring(0, precioSeleccionado.length-1))*can
   ...
});
```

JQuery UI

Construida por encima de JQuery, es una librería que contiene interacciones, *widgets*, efectos, y temas para crear aplicaciones RIA o *Rich Internet Applications*. La librería sigue el mismo estilo de JQuery, una arquitectura basada en eventos y centrada en los estándares de la web, accesibilidad y estilos flexibles. Hemos usado la librería para importar los elementos de interfaz *Accordion*, *Dialog* y *Tabs*.

La página de detalles de un vinilo nos permite acceder a información avanzada sobre el vinilo y por medio de unos pasos iremos realizando una personalización a medida del mismo. Teniendo en cuenta que contamos con poco espacio y que queremos mantener la idea de que hace falta completar unos pasos antes de realizar la compra para que le usuario no se desoriente, parece adecuado usar el control "Acordeón" que se trata de unos paneles replegables en vertical.



Por medio de este componente queda muy clara la separación de secciones y que existe un orden entre ellas. Además, el tener uno de los expandidos a la vez nos permite ahorrar suficiente espacio. Para utilizar el acordeón primero hemos tenido que montar una estructura HTML:

```
<div id="accordion">
        <h3>
            <span class="num">1</span><a href="#">Colores</a></h3>
        <div id="colores">
        </div>
        <h3>
            <span class="num">2</span><a href="#">Preferencias</a></h3>
        <div id="preferencias">
            <form action="">
            <div id="brillomate">
                <input type="radio" id="mate" name="radio" checked="checked" /><label
for="mate">Mate</label>
                <input type="radio" id="brillo" name="radio" /><label
for="brillo">Brillo</label>
            </div>
            </form>
            <form action="">
```

```
            <div id="invertidorignal">
                <input type="radio" id="original" name="radio" checked="checked" /><label
for="original">Original</label>
                <input type="radio" id="invertido" name="radio" /><label
for="invertido">Invertido</label>
            </div>
            </form>
        </div>
        <h3>
            <span class="num">3</span><a href="#">Medidas</a></h3>
        <div id="medidas">
            <table runat="server" id="tablaMedidas" class="horzebra">
                <thead>
                    <tr>
                        <th scope="col">
                            Seleccionar
                        </th>
                        <th scope="col">
                            Medida
                        </th>
                        <th scope="col">
                            Precio
                        </th>
                    </tr>
                </thead>
                <tbody>
                </tbody>
            </table>
        </div>
```

Al cargar la página, nos aseguraremos de que se ejecuta la siguiente instrucción:

```
$("#accordion").accordion();
```

Todos los popups o ventanas emergentes son controles que también nos aparto jQuery UI. En concreto, el popup que se nos abre al hacer clic sobre el carrito de compras se obtiene de la siguiente manera:

```
$("#carrito").click(function () {
  creaTabla();
  $dialog.dialog('open');
  $("#compras-vinilos").css('visibility', 'visible');
});
```

Siendo *$dialog* una variable global que mantiene el código HTML que se insertará dentro de la ventana.
Otra manera de organizar la información es mediante el uso de pestañas. JQuery UI trae el control *Tabs* que al igual que en los casos anteriores requiere previamente que tengamos cierta estructura en nuestra página:

```
<div id="tabs">
  <ul>
    <li><a href="#decora-tus-paredes">Vinilos</a></li>
    <li><a href="#decoracion-fotomurales">Fotomurales</a></li>
  </ul>
  <div id="decora-tus-paredes">
    <!-- CONTENIDOS TAB 1 -->
  </div>
  <div id="decoracion-fotomurales">
    <!-- CONTENIDOS TAB 2 -->
  </div>
</div>
```

Código JS (en respuesta al evento document.ready):

```
$("#tabs").tabs();
```

JQuery Spin

Nos permite añadir un control Spinner que en nuestro caso ha sido útil para seleccionar la cantidad de productos a comprar.

Código HTML:

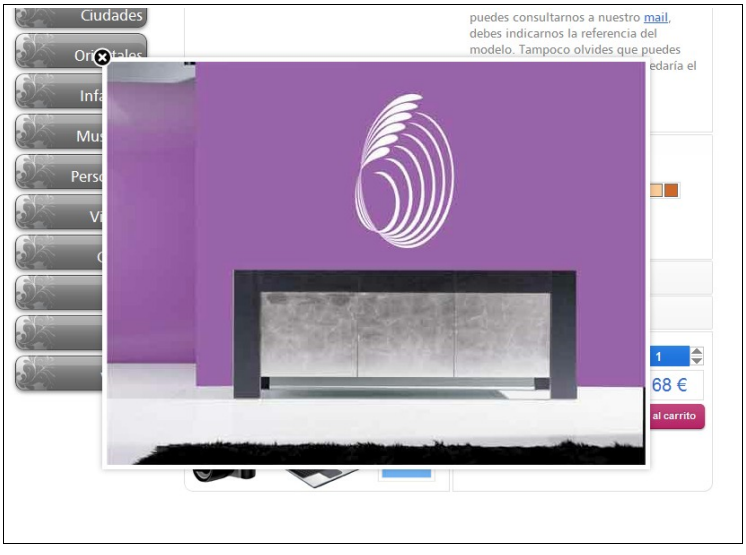
```
<input type="text" id="cantidad" name="cantidad"  class="disabled" value="1"  />
```

Código JS:

```
  $.spin.imageBasePath = './images/';
  $('#cantidad').spin({ max: 20, min: 1 });
```

Image Zoom

Nos aplica un efecto de zoom al hacer clic sobre las imágenes. Útil en la página de detalles del vinilo donde interesa que el usuario pueda apreciar con mayor nivel de detalle la imagen



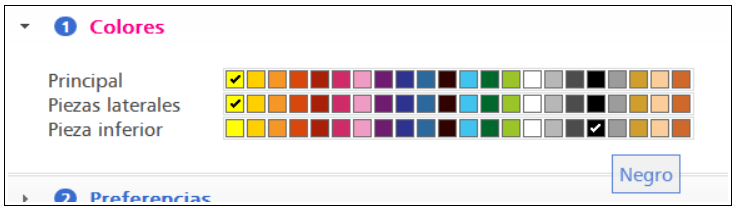
del producto.

Código JS:

```
$(document.body).imageZoom();
```

JQuery colorSelect + JQuery Tooltip

Una de las propiedades personalizables para los vinilos es la capacidad de elegir el color que llevarán las capas. Esto lo hemos conseguido implementando una paleta de colores. Esta paleta de colores tiene por entrada los colores de nuestro catálogo con sus respectivos códigos. El código se muestra en forma de *"tooltip"* al hacer *hover* sobre el recuadro del color.



Código JS:

```
$(div).colorPicker({
  defaultColor: 0, // index of the default color
  columns: 94,     // number of columns
  color: ['#ffff00', '#ffcc00', '#f39222', '#d6420b', '#a41c06', '#cc2662', '#ef96c0',
'#691769', '#282e88', '#266297', '#2c0200', '#3abfee', '#006227', '#95c123', '#ffffff',
'#b3b3b3', '#464646', '#000000', '#979797', '#cd9927', '#fbca96', '#cc6226'],
  click: function (color) {ultimoColor = color;}
});
```

Para cada "cuadrado" de color, se ejecuta la función tooltip:

```
   $(div).children().each(function () {
     $(this).tooltip({delay: 0, showURL: false, bodyHandler: function () {
                                var color = $(this).css('background-color');
                                return hex2cod(rgb2hex(color));}
                        });
                   });
```

ImgAreaSelect

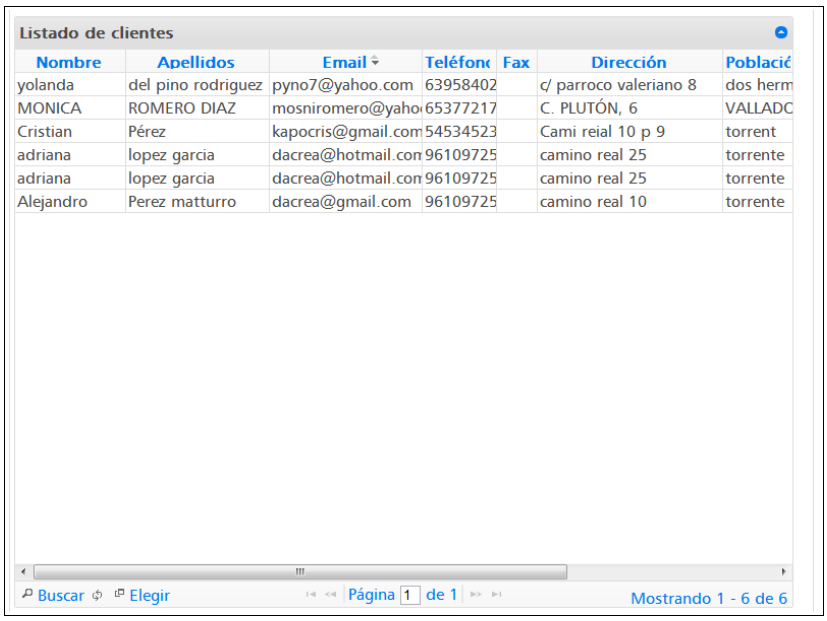
Se trata de un *plugin* jQuery que nos permite seleccionar un área por encima de un objeto como puede ser una imagen y nos genera una representación gráfica por medio de coordenadas del área que ha sido marcada. Es ideal para el caso de los Fotomurales donde necesitamos que el usuario seleccione un ancho y una altura para la fotografía.

Código para seleccionar un área (siendo #fotoliaGrande una imagen y urlVars un array que mantiene las coordenadas del área de selección):

```
 $('#fotoliaGrande').imgAreaSelect({ x1: urlVars['x1'], y1: urlVars['y1'], x2:
urlVars['x2'], y2: urlVars['y2'],
                    handles: true,
                    keys: true,
                    parent: "#containerGrande",
                    persistent: true,
                    aspectRatio: aspectRatio + ":1",
                    instance: true });
```

jqGrid

Se trata de un plugin con funciones avanzadas de ordenación, filtrado y búsqueda que toma como entrada unos datos que pueden estar en formato JSON y los representa en forma de listado o *grid*. La información se muestra en páginas. Las filas se pueden agrupan dentro de otras filas siguiendo algún criterio.



Para obtener el listado de la ilusión se ejecutaría un código similar a:

```
$("#clientes").jqGrid({url: '../ListarClientes.aspx', datatype: 'json', mtype: 'GET',
colNames: ['Nombre', 'Apellidos', 'Email', 'Teléfono', 'Fax', 'Dirección', 'Población',
'Provincia', 'CP', 'Empresa', 'NIF'], colModel: [
          { name: 'Nombre', index: 'Nombre', searchoptions: { sopt: ['eq']} },
          { name: 'Apellidos', index: 'Apellidos', searchoptions: { sopt: ['eq']} },
          { name: 'EMail', index: 'EMail', searchoptions: { sopt: ['eq']} },
          { name: 'Telefono', index: 'Telefono', searchoptions: { sopt: ['eq']} },
          { name: 'Fax', index: 'Fax', searchoptions: { sopt: ['eq']} },
          { name: 'Direccion', index: 'Direccion', searchoptions: { sopt: ['eq']} },
          { name: 'Poblacion', index: 'Poblacion', searchoptions: { sopt: ['eq']} },
          { name: 'Provincia', index: 'Provincia', searchoptions: { sopt: ['eq']} },
          { name: 'CodigoPostal', index: 'CodigoPostal', searchoptions: { sopt: ['eq']} },
          { name: 'Empresa', index: 'Empresa', searchoptions: { sopt: ['eq']} },
          { name: 'NIF', index: 'NIF', searchoptions: { sopt: ['eq']} }
          ],
            rowNum: 15,
            caption: 'Actualizando',
            sortname: 'EMail',
            sortorder: "desc",
            viewrecords: true,
            width: 763,
            height: 500,
            caption: 'Listado de clientes'
        });
```

Vemos que se puede configurar el tipo de datos de entrada, en este caso json. Otros parámetros configurables son el nombre de las columnas, la correspondencia entre cada columna y el campo del objeto de entrada, el número de registros por página, la altura del listado así como el ancho, etc.

5.3.1.4 Interacción con AJAX

Como hemos comentado en capítulos previos, AJAX elimina los ciclos de carga completa de página, ciclos en los que el usuario permanece bloqueado esperando hasta que llegue la respuesta del servidor con la página entera solicitada, lo que hace que las aplicaciones web se parezcan y se comporten más a lo que son las aplicaciones de escritorio. Previamente a las aplicaciones RIA, las aplicaciones web se centraban en el modelo cliente-servidor donde el cliente el único papel que desempeñaba era el de mostrar contenidos estáticos mientras que por el servidor debía pasar toda la interacción lo cual requiere que cada vez los datos se manden al servidor, el servidor envíe una respuesta, y la página se recargue en el cliente. La diferencia entre las aplicaciones web tradicionales y las aplicaciones RIA está en la calidad y cantidad de interacción posible entre el usuario y el interfaz. Una aplicación RIA tiene un mayor rango de controles que mejoran la interacción con el usuario y dan lugar a una interacción más eficiente, mejor gestión de errores, *feedback* y mayor sensación de satisfacción en general. Además, las aplicaciones RIA pueden cachear los datos en el cliente de lo cual se deduce una interfaz con mayor capacidad de respuesta y menos viajes ida-vuelta al servidor.

Nuestra aplicación se puede clasificar dentro del grupo de aplicaciones conocidas como "completamente AJAX" (*fully AJAXed*). AJAX es necesario tanto para funcionalidades como para las interfaces. Este tipo de aplicaciones se alejan del concepto tradicional en el que navegar por una web consistía en hacer clic sobre enlaces y cargar páginas nuevas. En nuestro caso, si pulsamos un botón o un enlace podemos esperar que cambie cierta parte de la vista actual. La cantidad de datos transferida es mínima; ya no estamos enviando un formulario de datos lo cual hace que la utilización de la red baje y pueden darse más operaciones en simultáneo.

Vamos a plantear un caso concreto de interfaz AJAX, la "interfaz de fotomurales". La página Fotomurales.aspx es la típica página AJAX que se trae desde el servidor en partes. En una primera parte se incluyen los contenidos estáticos y ya cuando la página ha acabado cargarse en el navegador, por medio de una llamada AJAX se hace una petición de los *thumbnails* o miniaturas que se insertarán en el panel de la galería. Lo bueno de este método es que el cliente no tiene que esperar a que los contenidos de la galería hayan sido recuperados para empezar a interactuar con la página.

```
$(document).ready(function() {
  cargarCaptions();
});
```

CargarCaptions es una función intermedia de nuestra librería que lo que hace es invocar por debajo a *getData* pasándole como parámetros:

- *offset*: indica el índice de partida para recuperar las miniaturas. Si queremos ver la primera página, el indice deberá ser 0.
- *words*: cadena de texto a buscar.
- Nombre del servicio web. En este caso "cargarCaptions".

```
var offset = 0;
var textoBuscado = 'paisajes';
cargarCaptions = function() {
  return getData("{offset: '" + 0 + "', words: '" + textoBuscado + "'}", 'cargarCaptions');
}
```

La función getData es la encargada de realizar la llamada AJAX propiamente dicha. Para ello invocará a el método JQuery.ajax con los siguientes settings:

- type: Es el tipo de petición HTTP, puede ser "GET" o "POST".
- data: Parámetros que se van a enviar al servidor. En nuestro caso le pasaremos un array de valores que será serializado.
- datatype: Es el tipo de datos que esperamos recibir por parte del servidor, en nuestro caso "json". La respuesta será evaluada como JSON y se devolverá un objeto JavaScript.
- contentType: Tipo de datos que se le envía al servidor.
- url: Dirección donde se envía la petición.

El objeto de clase "ajax-loader" simplemente es el GIF animado que se muestra mientras se realiza la petición y se oculta al mostrar las miniaturas. Una vez obtenida la respuesta, el objeto res contiene un campo "html" con la estructura de la galería que será insertada dentro del "fotoliaGallery" y otro campo "numResults" con el número de imágenes recuperadas.

```
getData = function (params, functionName) {
            $('#ajax-loader').css('visibility', 'visible');
            $.ajax({
                type: 'POST',
                data: params,
                dataType: "json",
                contentType: "application/json; charset=utf-8",
                url: 'Fotomurales.aspx/' + functionName,
                success: function (data) {
                    $('#ajax-loader').css('visibility', 'hidden');
                    res = $.evalJSON(data.d)
                    ...
                    $("#fotoliaGallery").html(res.html);
                    numResultados = res.numResults;
                    ...
                },
                error: function () {
                    return false;
                }
            });
            return true;
        }
```

Aunque se escape de los contenidos de este subapartado, continuando con el ejemplo vamos a analizar lo que pasa cuando la llamada AJAX alcanza al servidor. En primer lugar, las imágenes se obtienen de un servicio proporcionado por Fotolia. Para realizar la comunicación con Fotolia necesitamos una librería de XML-RPC que nos facilite la implementación de un cliente en este protocolo. Todo lo que necesitamos es implementar una interface representando el "punto de fin" de XML-RPC y posteriormente usar la clase *XmlRpcProxyGen* que automáticamente generará el código de un proxy para realizar la comunicación.

La interfaz se encuentra en el fichero FotoliaAPI.cs:

```
[XmlRpcUrl("http://api.fotolia.com/Xmlrpc/rpc")]
public interface IFotoliaAPI : IXmlRpcProxy
{

    [XmlRpcMethod("xmlrpc.getSearchResults")]
    SearchResults GetResults(
        string appKey,
        SearchRequest parametros
        );
    ...

}
```

Como se puede observar, en la interfaz hemos declarado el método *GetResults* que recibe como parámetros la clave de usuario de Fotolia que ya nos hemos encargado de solicitar, y una estructura *SearchRequest* que tiene la siguiente forma:

```
[XmlRpcMissingMapping(MappingAction.Ignore)] public struct
SearchRequest
  {
        [XmlRpcMember("language_id")] public int LanguageID;
        [XmlRpcMember("limit")] public int Limit;
        [XmlRpcMember("thumbnail_size")] public int ThumbnailSize;
        [XmlRpcMember("words")] public string Words;
        [XmlRpcMember("offset")] public int Offset;
        [XmlRpcMember("thumbnail_size")] public int Size;
        [XmlRpcMember("media_id")] public int Id;
}
```

GetResults también devuelve un tipo de datos complejos expresado en forma de estructura:

```
[XmlRpcMissingMapping(MappingAction.Ignore)] public struct SearchResults {
    [XmlRpcMember("0")] public Result member0;
    [XmlRpcMember("1")] public Result member1;
    [XmlRpcMember("2")] public Result member2;
    [XmlRpcMember("3")] public Result member3;
    [XmlRpcMember("4")] public Result member4;
    [XmlRpcMember("5")] public Result member5;
    [XmlRpcMember("6")] public Result member6;
    [XmlRpcMember("7")] public Result member7;
    [XmlRpcMember("8")] public Result member8;
```


```
    [XmlRpcMember("9")] public Result member9;
    [XmlRpcMember("10")] public Result member10;
    [XmlRpcMember("11")] public Result member11;
    [XmlRpcMember("12")] public Result member12;
    [XmlRpcMember("13")] public Result member13;
    [XmlRpcMember("14")] public Result member14;
    [XmlRpcMember("15")] public Result member15;
    [XmlRpcMember("16")] public Result member16;
    [XmlRpcMember("17")] public Result member17
}
```

SearchResults es una estructura que agrupa un conjunto de estructuras de tipo *Result*. El número de miembros de esta estructura no es arbitrario, hemos establecido que se muestren 18 miniaturas por página y por ello también es ese el número de miembros que están contenidos en *SearchResults*.

Por último, nos falta precisar como se compone la estructura *Result*:

```
[XmlRpcMissingMapping(MappingAction.Ignore)]
public struct Result
{
    [XmlRpcMember("id")]
    public int Id;
    [XmlRpcMember("title")]
    public string Title;
    [XmlRpcMember("creator_id")]
    public int CreatorId;
    [XmlRpcMember("creator_name")]
    public string CreatorName;
    [XmlRpcMember("thumbnail_url")]
    public string ThumbnailUrl;
    [XmlRpcMember("thumbnail_html_tag")]
    public string ThumbnailHtmlTag;
    [XmlRpcMember("thumbnail_width")]
    public int Width;
    [XmlRpcMember("thumbnail_height")]
    public int Height;
    [XmlRpcMember("name")]
    public string LicenceName;
    [XmlRpcMember("price")]
    public int Price;
    [XmlRpcMember("nb_views")]
    public int Views;
    [XmlRpcMember("nb_downloads")]
    public int Downloads;
    [XmlRpcMember("keywords")]
    public string Keywords;
}
```

Como vemos, el objetivo de FotoliaAPI.cs es representar respuestas y peticiones XML-RPC en tipos .NET. El siguiente paso será el de crear un proxy que derive de esta interfaz.

En el *code-behind* de Fotomurales.aspx contamos con el atributo:

```
private static IFotoliaAPI proxy = XmlRpcProxyGen.Create<IFotoliaAPI>();
```

Tenemos varios métodos que hacen de servicio web en la página de fotomurales. Para indicar que un método debe quedar expuesto como servicio web, debemos hacer que el método sea de tipo **public** y asociarle la etiqueta **[System.Web.Services.WebMethod()].** Si recordamos, el método que se llamaba desde el cliente era "cargarCaptions". *cargarCaptions* lo primero que hará es crear una estructura *SearchRequest*, rellenará sus parámetros obligatorios. Creará otra estructura *SearchResults* e invocará al método *GetResults* de la API de Fotolia a través del *proxy*. El *proxy* se ocupará de poblar la estructura *SearchResults*. A partir de ahí y si no ha habido ningún error, dentro de un bucle iremos obteniendo una a uno los miembros de *SearchResults* y construiremos el código HTML. Finalmente construimos un objeto genérico "json" con dos campos: *html* que es un *string* que contiene la concatenación del código HTML para cada miniatura, y *numResults* que es un entero. El método devuelve el objeto convertido a formato JSON.

```
[System.Web.Services.WebMethod()]
public static object cargarCaptions(int offset, string words) {
SearchRequest defaultRequest = new SearchRequest();
     defaultRequest.Words = words;
     JavaScriptSerializer serializer = new JavaScriptSerializer();
     defaultRequest.Limit = 18;
     defaultRequest.Offset = offset;
     SearchResults resultados = new SearchResults();
     try
     {
         resultados = proxy.GetResults(appKey, defaultRequest);
     }
     catch (Exception ex)
     {
         var jsonError = new
         {
             error = ex.Message
         };
         return serializer.Serialize((object)jsonError);
     }
     string res = "";
     var cant = 0;
         Type type = resultados.GetType();
         List<FieldInfo> miembrosFieldInfo = type.GetFields().Where(f =>
f.Name.StartsWith("member")).ToList();
         foreach (FieldInfo fieldInfo in miembrosFieldInfo)
         {
             Result unmiembro = (Result)fieldInfo.GetValue(resultados);

             if (unmiembro.Id != 0)
             {
                 res += "<div id='img-" + unmiembro.Id + "' class='fotoliaTb'><a
style='cursor:pointer;' onclick='return clickAncla(this);'><img height='" +
unmiembro.Height +"' width='" + unmiembro.Width + "' border='0' src='" +
unmiembro.ThumbnailUrl + "' title='" + unmiembro.Title + "' alt='" + unmiembro.Title
```

```
+"'/></a></div>";
                cant++;
            }
        }


     var json = new
    {
        html = res,
        numResults = cant,
    };

       return serializer.Serialize((object)json);

   }
```

5.3.2 Capa de negocio

En este apartado vamos a profundizar en algunos de los detalles de implementación que consideramos que pueden ser de aportación para dejar más en claro si cabe la idea de las dos subcapas contenidas en este nivel: la subcapa de lógica de negocio o BLL y la subcapa de acceso a datos o DAL.

5.3.2.1 Subcapa BLL

Cuando empezamos a plantar los cimientos de nuestra aplicación, al realizar la división en capas, en un principio vimos que la capa BLL simplemente hacía de pasarela hacia la capa DAL. Esto es normal que ocurra en aplicaciones de pequeño tamaño. A medida que el proyecto fue avanzando, empezamos a descubrir "reglas de negocio" cuyo sitio más acertado es la capa BLL. Es decir, la capa BLL tiende a crecer con el tiempo. A pesar de que nuestra capa BLL no es especialmente "gruesa", sigue siendo necesario contar con ella porque en primer lugar la aplicación debe estar concebida para añadir más complejidad, o sea debe ser extensible y en segundo lugar debemos mantener el aislamiento entre capas. No podríamos meter lógica en la capa DAL; esta sólo se ocupa de las operaciones CRUD. Tener la lógica de negocio localizada en un sólo lugar nos permite aplicar cambios con mayor facilidad. Si el código de la interfaz de usuario se encargara de hacer cosas que no tuvieran que ver con la GUI, como la lógica de negocio; o si la capa DAL se ocupara de más tareas además de establecer conexiones o recuperar/guardar datos incurriríamos en lo que se conoce como falta de "separación de atribuciones" (Véase *Separation of Concerns* - http://en.wikipedia.org/wiki/Separation_of_concerns).

Uno de los objetivos de la capa BLL es definir las reglas de negocio. Vamos a ejemplificar esto por medio del siguiente código:

```
public string Total() {
  double total = 0;
  double st = double.Parse(this.Subtotal());
  double ge = this.pedidoFotomural() || st > 60 ? 0 :
    double.Parse(this.Cliente.GastosDeEnvio);
  if (this.Cliente.AplicarIVA) {
    double iva = double.Parse(this.IVA());
    total = st + ge + iva;
  } else {
      total = st + ge;
  }
  return String.Format("{0:0.00}", total);
}
```

El método Total() de la clase Pedido devuelve una cadena formateada con el valor total de la compra hecha por el cliente. Este método sigue la regla: "Si el cliente ha comprado un fotomural o el valor de la compra sin impuestos supera los 60 euros, entonces no se cargarán los gastos de envíos".

¿Cómo se determinan los gatos de envío? En nuestro caso, hemos implementado una propiedad pública "GastosDeEnvio" de la clase Cliente que devuelve una cadena formateada con el valor del gasto dependiendo de la provincia en la que esté localizado el cliente:

```
public string GastosDeEnvio {
  get {
    switch (this.Provincia) {
      case "Ceuta": return String.Format("{0:0.00}", 0);
      case "Melilla": return String.Format("{0:0.00}", 0);
      case "Palmas (Las)": return String.Format("{0:0.00}", 0);
      case "Santa Cruz de Tenerife": return String.Format("{0:0.00}", 0);
      case "Balears (Illes)": return String.Format("{0:0.00}", 12);
      default: return String.Format("{0:0.00}",8);
    }
  }
}
```

A parte de aplicar las reglas de negocio otra función de la capa BLL es la de validar los datos de entrada provenientes de la capa de presentación o también depurarlos de cara a su utilización (prevención de datos maliciosos). Siempre es recomendable hacer una doble validación tanto en la parte del cliente y luego en el servidor, de esta manera nos evitamos el procesamiento innecesario en el servidor. Lo siguiente es el código que se ejecuta cuando el usuario envía el formulario de realizar consulta. Es un sencillo ejemplo de validación de datos de entrada en la capa BLL.

```
protected void Page_Load(object sender, EventArgs e) {
  string nombre = Request["Nombre"];
  string empresa = Request["Empresa"];
  string telefono = Request["Telefono"];
  string mensaje = Request["Mensaje"];
  string origen = Request["Email"];
  if((nombre != null && nombre.Length > 0) &&
     (mensaje != null && mensaje.Length > 0) &&
     (origen != null && Email.IsValid(origen))) {
    string asunto = "Consulta desde la web - Decorate Your House";
    string destino = ConfigurationManager.AppSettings.Get("Email");
    ...
    try {
      Email.EnviarEmail(destino, asunto, cuerpo);
      Response.Write("Gracias por ponerse en contacto con nosotros. En breve tendrá su
respuesta.");
    }catch(SmtpException smpt){
       Response.Write("Error: El proceso no pudo ser completado. Vuelva a intentarlo");
    }
   Response.End();
  }
  Response.Write("Los datos de entrada no son correctos.");
  Response.End();
}
```

El método comprueba que el usuario ha ingresado los campos obligatorios, comprueba el formato del email y si todo va bien realiza el envío de la consulta al email de la empresa que está guardado como un *setting*. El código también está preparado para alertar al usuario en caso de que ocurra una excepción de SMTP.

Como es sabido, mientras que en la capa BTT se mantiene la lógica, la recuperación de datos o modificación se hace mediante llamadas a la capa DAL. Los esquemas de bases de datos son algo que debería estar encapsulado y escondido del resto de la aplicación; ya sea un ORM o cualquier solución es mejor tenerla "escondida" dentro del DAL. La capa BLL y superiores simplemente debe dedicarse a usar los tipos de datos (clases). En el ejemplo a continuación vemos como la capa BTT manipula las entidades del negocio y se apoya en la capa DAL para la persistencia de datos. Se trata del proceso de confirmación de pedido. El *Page_Load* de "ConfirmarPedido.aspx.cs" comprueba el paso actual de la compra, si estamos en el último paso es momento de guardar el pedido:

```
protected void Page_Load(object sender, EventArgs e) {
  if (Request["DatosModificados"] != null && Request["tipo"] != null) {
  ...
  Pedido p = Session["pedido"] as Pedido;
  ...
  if (Request["paso"] == null && Request.QueryString["Mensaje"] == null) {
    CrearPedido(); // → Crea un pedido y lo guarda como variable de sesión.
  ...
  } else {
    if (Request["paso"] == "2") {
    ...
    }
```

```
    else if (Request["paso"] == "3") {
       ...
    }
    else if (Request["paso"] == "4") {
       ...
    // Último paso. El cliente ha confirmado el pedido. Guardarlo en BD.
    Pedido p = Session["pedido"] as Pedido;
    p.Estado = "En espera de pago";
    try {
      new DatosDYH().AltaPedido(p);
    }catch (Exception ex){
      Response.Write("Hubo un error al guardar el pedido");
    }
    string id = System.Guid.NewGuid().ToString().Substring(0, 7);
    Peticion peticion = new Peticion { Id = id, Precio = (Session["resultadoFinal"] as
string).Replace(',', '.'), Pedido = p };
    new DatosDYH().AltaPeticion(peticion);
    Email.EnviarEmail(...);
    ...
     Response.End();
     return;
        }
  }
 ...
}
```

Las líneas en amarillo representan el código en el que la capa BLL se comunica con DAL. Para dar de alta un pedido se llama al método "AltaPedido" del objeto "DatosDYH". El mismo procedimiento para las peticiones. La capa BLL simplemente se preocupa de construir los objetos de negocio y ya la capa inferior se ocupará de almacenarlos.

```
private void CrearPedido() {
  Pedido p = new Pedido();
  foreach (string key in Request.Cookies.AllKeys) {
    int clave;
    if (int.TryParse(key, out clave)) {
    PedidosDetalles pv = CrearDetallesPedido(Request.Cookies[key].Value.ToString());
    p.PedidosDetalles.Add(pv);
    }
  }
  p.Cliente = Request.Cookies["cookieUsuario"] != null ? new
DatosDYH().BuscaCliente(Request.Cookies["cookieUsuario"].Value.ToString()) :
Session["usuario"] as Cliente;
  p.Fecha = DateTime.Now;
  Session.Timeout = 60;
  Session["Pedido"] = p;
}
```

El método de arriba se centra en la creación de pedidos. Vemos que un pedido se puede componer de varios "PedidosDetalles". Por tanto el objeto Pedido tiene un atributo que será una colección de objetos PedidoDetalles. Notar que para rellenar el campo "Cliente" del Pedido, se verifica que existe la *cookie* que contiene el ID del usuario y con ella se invoca a un

método de la capa DAL "BuscarCliente" que realiza una búsqueda de clientes y devuelve el objeto Cliente cuyo ID es el proporcionado.

Adentrándonos aún más en nivel de detalle, vamos a analizar como se crea un objeto de tipo "PedidoDetalles":

```
private PedidosDetalles CrearDetallesPedido(string datos) {
  datos = Server.UrlDecode(datos);
  JavaScriptSerializer js = new JavaScriptSerializer();
  Dictionary<string, object> objetoDatos = (Dictionary<string,
object>)js.DeserializeObject(datos);
  var referencia = objetoDatos["referencia"];
  var medida = objetoDatos["medida"];
  var color = objetoDatos["color"];
  var orientacion = objetoDatos["orientacion"];
  var acabado = objetoDatos["acabado"] != null ? objetoDatos["acabado"].ToString() : null;
  var cantidad = objetoDatos["cantidad"];
  var precio = objetoDatos["precio"];
  var num = objetoDatos["num"];
  var byn = (objetoDatos["byn"] != null) ? objetoDatos["byn"].ToString() : null;
  var coord = objetoDatos["coordenadas"] != null ? objetoDatos["coordenadas"].ToString() :
null;

  PedidosDetalles pv = new PedidosDetalles();
  pv.Cantidad = int.Parse((string)cantidad);
  pv.Medida = medida.ToString();
  pv.Orientacion = orientacion.ToString();
  pv.BYN = byn;
  pv.Coordenadas = coord;
  pv.Acabado = acabado;
  DatosDYH d = new DatosDYH();
  pv.Vinilo = d.BuscarViniloPorId(referencia.ToString());
  if (pv.Vinilo != null) {
    var listaPrecios = d.ObtenerMedidasPrecios(pv.Vinilo);
    pv.Precio = (listaPrecios[(pv.Medida).Substring(0, pv.Medida.Length - 3)]) *pv.Cantidad;
  } else {
    pv.Codigo = referencia.ToString();
    pv.Precio = double.Parse(((string)precio).Substring(0, ((string)precio).Length - 1));
  }

  if (color != null) {
    var colores = color.ToString().Split('/');
    foreach (string c in colores) {
      ColoresCapa cp = new ColoresCapa();
      cp.Capa = pv.ColoresCapa.Count + 1;
      cp.Color = d.RecuperarColor(c);
      pv.ColoresCapa.Add(cp);
    }
  }
  return pv;
}
```

"CrearDetallesPedido" lo primero que hace es obtener un diccionario de palabras-objeto con los valores de la personificación que ha hecho el usuario sobre el producto. Luego crea un objeto "PedidoDetalles" cuyos campos irá rellenando con los valores obtenidos previamente. El campo vinilo lo obtiene llamando a la capa DAL, concretamente al método "BuscarViniloPorId" al que le pasa la referencia del vinilo. Este método devuelve un objeto de tipo "Vinilo". Para conocer el precio, hace falta saber las medidas-precios disponibles. Para ello se llama al método "ObtenerMedidasPrecio" que recibe como argumento un objeto Vinilo. Teniendo esta lista y sabiendo cual es la medida que ha elegido el cliente es fácil obtener el precio.

Ya hemos visto como la capa BLL manipula objetos de negocio y se comunica con la capa DAL. Otras tareas que se desempeñan en esta capa son: la autentificación de usuarios, la autorización de usuarios (mediante una estructura de directorios y archivo Web.config) y el cacheo de datos (mediante *Entity Framework*).

5.3.2.2 Subcapa DAL

La capa DAL contiene métodos que ayudan a la capa BLL a conectarse con los datos y realizar las tareas requeridas ya sea devolver o manipular información. El propósito de esta capa es el de servir como una interfaz reusable hacia la base de datos. Obviamente, esta capa no debe contener reglas de negocio o manipulación de datos / transformación de la lógica.

Hemos usado *Entity Framework* como nuestra capa DAL. El primer paso llevado a cabo, ha sido crear un modelo de definición de entidades (fichero .edmx) a partir del esquema de base de datos. El asistente creará de forma automática una clase que hereda de *ObjectContext* y una clase entidad que hereda de *EntityObject* para cada tabla de la base de datos. La clase con el nombre "Entidades" que es la que hereda de *ObjectContext* es la forma principal que tenemos para interactuar con la base de datos. Esta expone propiedades que representan las tablas modeladas. Podremos construir sentencias *LINQ To Entities* que ataquen esas propiedades y así recuperar o manipular objetos. Además este objeto lleva un seguimiento de los cambios que hacemos sobre los objetos. Por ejemplo, al cargar un objeto, y modificar alguna de sus propiedades, "Entidades" lo que hará es recordar los cambios realizados y en el momento que se ejecuta el método "SaveChanges()" entonces generará todas las sentencias SQL de actualización necesarias para llevar esos cambios a la base de datos.

Dejando de lado el objeto *ObjectContext*, al compilar, la herramienta "EdmGen.exe" nos generará los archivos CSDL, SSDL, y MSL que ya hemos comentado en capítulos anteriores y además el código de las entidades. Habiendo hecho estos pasos podemos decir que tenemos nuestra capa DAL completada.
Ahora bien, tal cual está organizada nuestra aplicación en este momento, tendríamos que la capa BLL trabaja directamente sobre el objeto *ObjectContext* "Entidades". Esto no sería un problema en aplicaciones pequeñas o en aquellas donde se poco previsible que ocurran cambios. Sin embargo, esta aproximación puede ser problemática a la hora de mantener el código en aplicaciones propensas a sufrir cambios o en aquellas donde el *testing* sea un

requisito.

Para solucionar este problema hemos optado por aplicar el patrón "Repositorio" (Véase http://msdn.microsoft.com/en-us/library/ff649690.aspx). Una clase repositorio nos ayuda a encapsular las consultas a los datos, la lógica de persistencia y permite abstraer los detalles de implementación de la persistencia de datos al resto de la aplicación. Además de que el código será más limpio, usar el patrón repositorio facilita que en el futuro se pueda cambiar la implementación de base de datos (cambiar el gestor) y por otro lado la labor de *unit testing* será más fácil porque dejaremos de necesitar una base de datos real para realizar pruebas.
La firma de nuestra clase es algo del estilo:

```
public class DatosDYH {
  Entidades contexto;

  public DatosDYH() {
    contexto = new Entidades();
  }
  ...
}
```

Ejemplo de consulta:

El código de abajo devuelve una lista de objetos Pedido a partir del Cliente.

```
public List<Pedido> ObtenerPedidosUsuario(Cliente cliente) {
  var consultaPedidos = from p in contexto.Pedidos
                        where p.Cliente.EMail == cliente.EMail
                        select p;
  return consultaPedidos.ToList();
}
```

Ejemplos de inserción/actualización:

El siguiente ejemplo demuestra cómo se cambia el estado de un pedido. Cualquier modificación que se haga en el repositorio no se lleva a la base de datos hasta que se llama al método SaveChanges(). *Entity Framework* automáticamente incluye todos los cambios dentro de una transacción, es decir que se aplican todos los cambios o ninguno cuando se llama a SaveChanges().

```
public void CambiarEstado(int idpedido, string nuevoestado) {
  contexto.Pedidos.Where(p => p.Id == idpedido).First().Estado = nuevoestado;
  this.GuardarCambios();
}
public void GuardarCambios() {
  this.contexto.SaveChanges();
}
```

Este método da de alta una "medida-precio" para los fotomurales.

```
public void AltaPrecioFotomurales(float desde, float hasta, float precio) {
  PreciosFotomurales ft = new PreciosFotomurales();
  ft.precio = precio;
  ft.desde = desde;
  ft.hasta = hasta;
  contexto.AddToPreciosFotomurales(ft);
  contexto.SaveChanges();
}
```

Ejemplo de borrado:

El código recupera un objeto "PreciosFotomurales" existente y luego lo marca para ser borrado. Cuando se llama a SaveChanges() el borrado se pasa a la base de datos.

```
public void BorrarPrecioFotomurales(float desde, float hasta, float precio) {
  var consulta = from pf in contexto.PreciosFotomurales
                 where pf.hasta == hasta && pf.desde == pf.desde && pf.precio == precio
                 select pf;
  PreciosFotomurales opf = consulta.FirstOrDefault();
  if (opf != null) {
    contexto.DeleteObject(opf);
    contexto.SaveChanges();
  }
}
```

5.3.3 Capa de persistencia

Se podría decir que la capa de persistencia es básicamente el DBMS o *Database Management System*, es decir, el gestor de base de datos-- en nuestro caso, SQL Server para la página web, y un fichero XML para el simulador. Esta capa se supone que debe lidiar con el almacenamiento de los datos en el medio físico y su recuperación. No se preocupa en absoluto de ninguna manipulación que se vaya a hacer con esos datos. Dentro de la capa de persistencia se incluyen los procedimientos almacenados. A continuación se muestra un extracto del *log* de creación de la base de datos en SQL Server. Para ver la versión completa, ir al apartado de Anexo 2.

```
IF NOT EXISTS (SELECT * FROM sys.schemas WHERE name = N'DecorateYourHouse')
EXEC sys.sp_executesql N'CREATE SCHEMA [DecorateYourHouse] AUTHORIZATION [dacrea]'

GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[establecernuevos]') AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'-- ============================================
-- Author:         Cristian Pérez
```

```
-- Create date:
-- Description:
-- =============================================
CREATE PROCEDURE [DecorateYourHouse].[establecernuevos]

AS
BEGIN
	SET NOCOUNT ON;

    UPDATE Vinilos
	  SET esNuevo=0 WHERE Id IN (SELECT Id
                                                              FROM Vinilos
                                                              WHERE
(DATEDIFF(minute,Fecha,getutcdate()) >= 86400 ))
RETURN 0
END

'
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[PreciosFotomurales]') AND type in (N'U'))
BEGIN
CREATE TABLE [DecorateYourHouse].[PreciosFotomurales](
	[desde] [float] NOT NULL,
	[hasta] [float] NOT NULL,
	[precio] [float] NOT NULL,
 CONSTRAINT [PK_PreciosFotomurales] PRIMARY KEY CLUSTERED
(
	[desde] ASC,
	[hasta] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[Vinilos]') AND type in (N'U'))
BEGIN
CREATE TABLE [DecorateYourHouse].[Vinilos](
	[Id] [varchar](20) NOT NULL,
	[Categoria] [varchar](40) NULL,
	[Fecha] [datetime] NULL CONSTRAINT [DF__Vinilos__Fecha__0BC6C43E]  DEFAULT
(getdate()),
	[Descripcion] [varchar](120) NULL,
	[Foto] [varchar](120) NULL,
	[esNuevo] [bit] NULL CONSTRAINT [DF_Vinilos_esNuevo]  DEFAULT ((1)),
```

```
 CONSTRAINT [PK_Vinilos] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
```

6. Pruebas

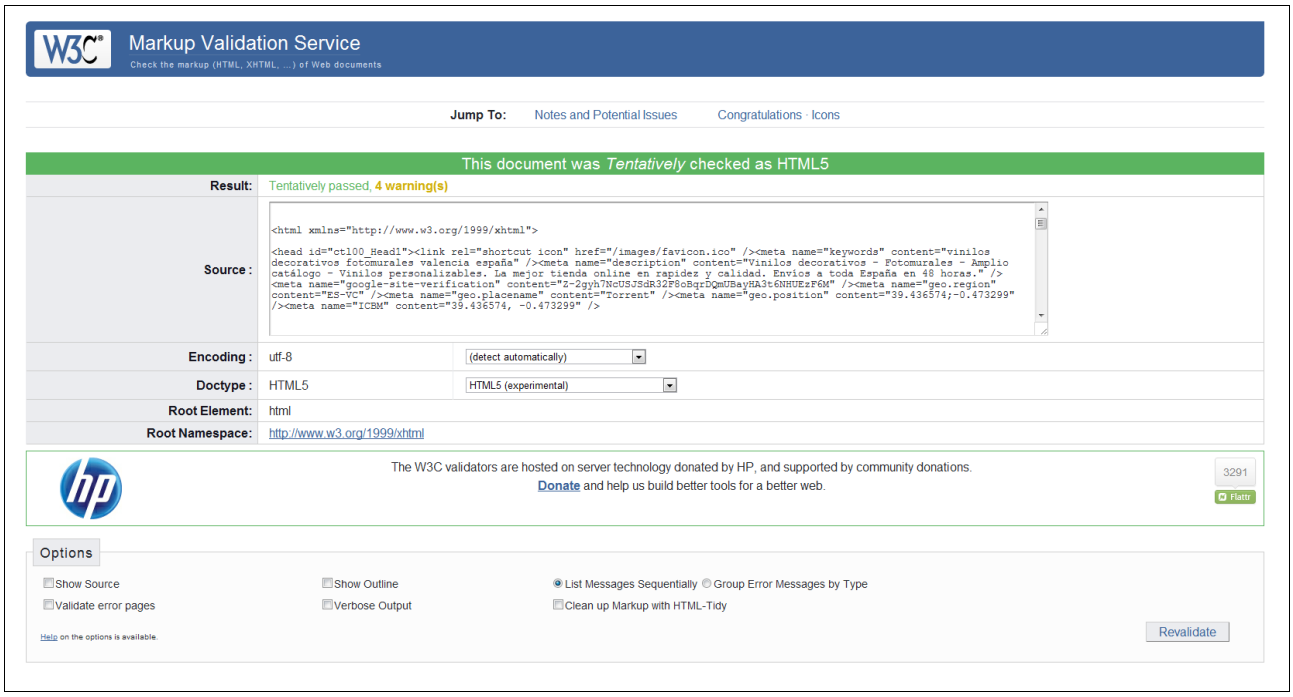
6.1 Introducción

En este apartado se detallan algunos de los procedimientos básicos que hemos seguido durante el desarrollo para comprobar que nuestra página web se adecua a los estándares oficiales. En concreto, vamos a hablar de las herramientas QA (*Quality Assurance*) de la web del W3C. Más adelante hablaremos de cómo hemos usado el *Sandbox* de Paypal para verificar que el proceso de compra se lleva a cabo de forma correcta.

6.2 Herramientas QA de W3C

6.2.1 MarkUp Validator

El test es accesible desde http://validator.w3.org, hemos seleccionado la pestaña "Validate Direct Input", el *doctype* lo hemos puesto a "HTML5" porque en la portada que es el HTML que vamos a validar tenemos etiquetas *figure* y *figcaption*. La imagen abajo muestra el resultado de la prueba.

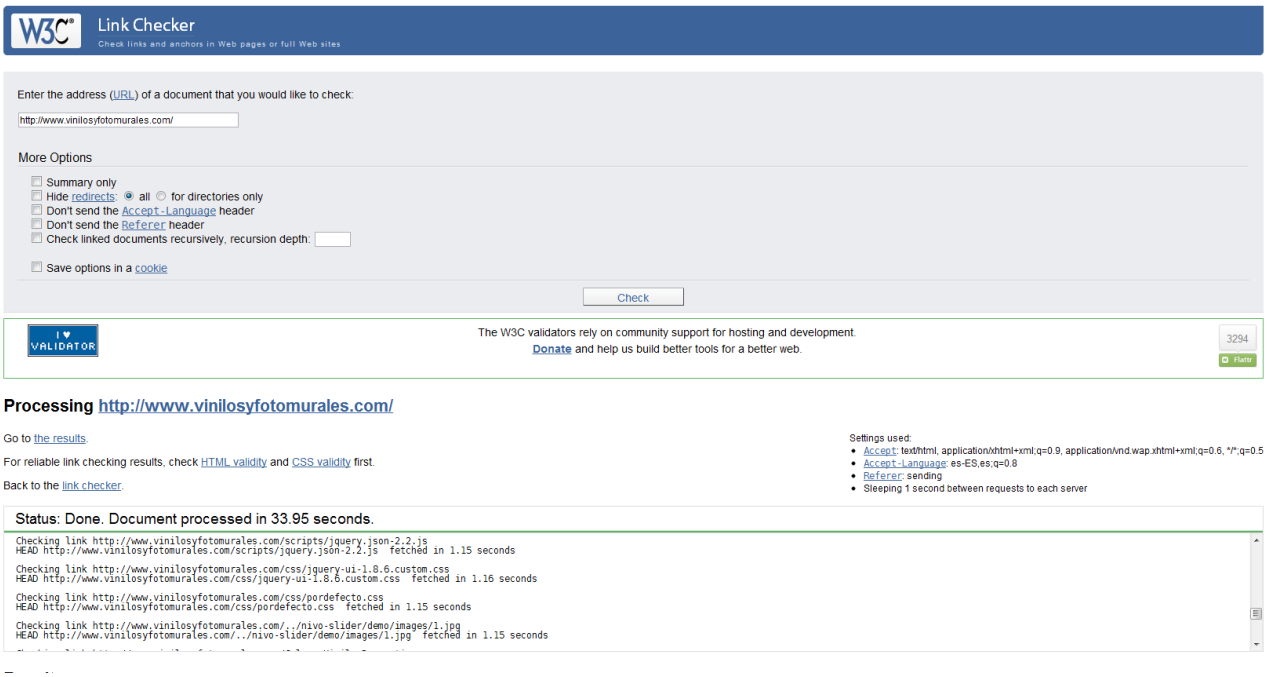


6.2.2 CSS Validator

Para comprobar que nuestra hoja de estilo no contiene errores nos dirigimos al test CSS Validator a través de la URL http://jigsaw.w3.org/css-validator. Seleccionamos la pestaña "mediante Entrada directa" y elegimos como perfil "CSS versión 3".

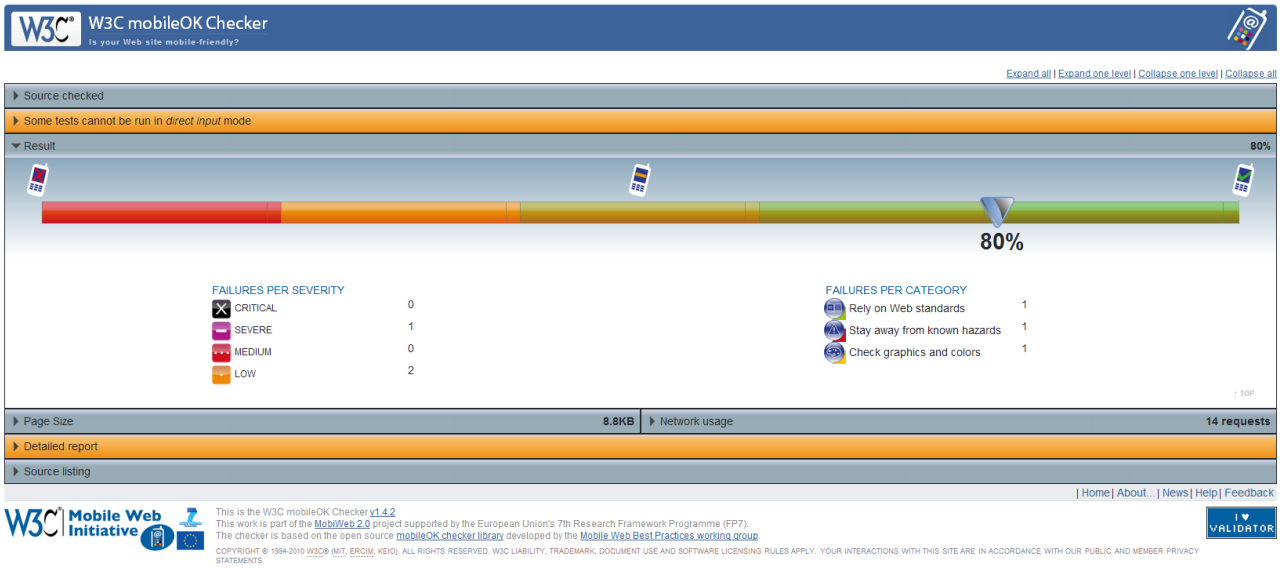
6.2.3 Link Checker

Accesible desde http://validator.w3.org/checklink, este *test* lo que hace es comprobar el código de las respuestas HTTP después de cada petición al servidor. Puede navegar por todo los *links* de una página o incluso hacer un análisis recursivo.



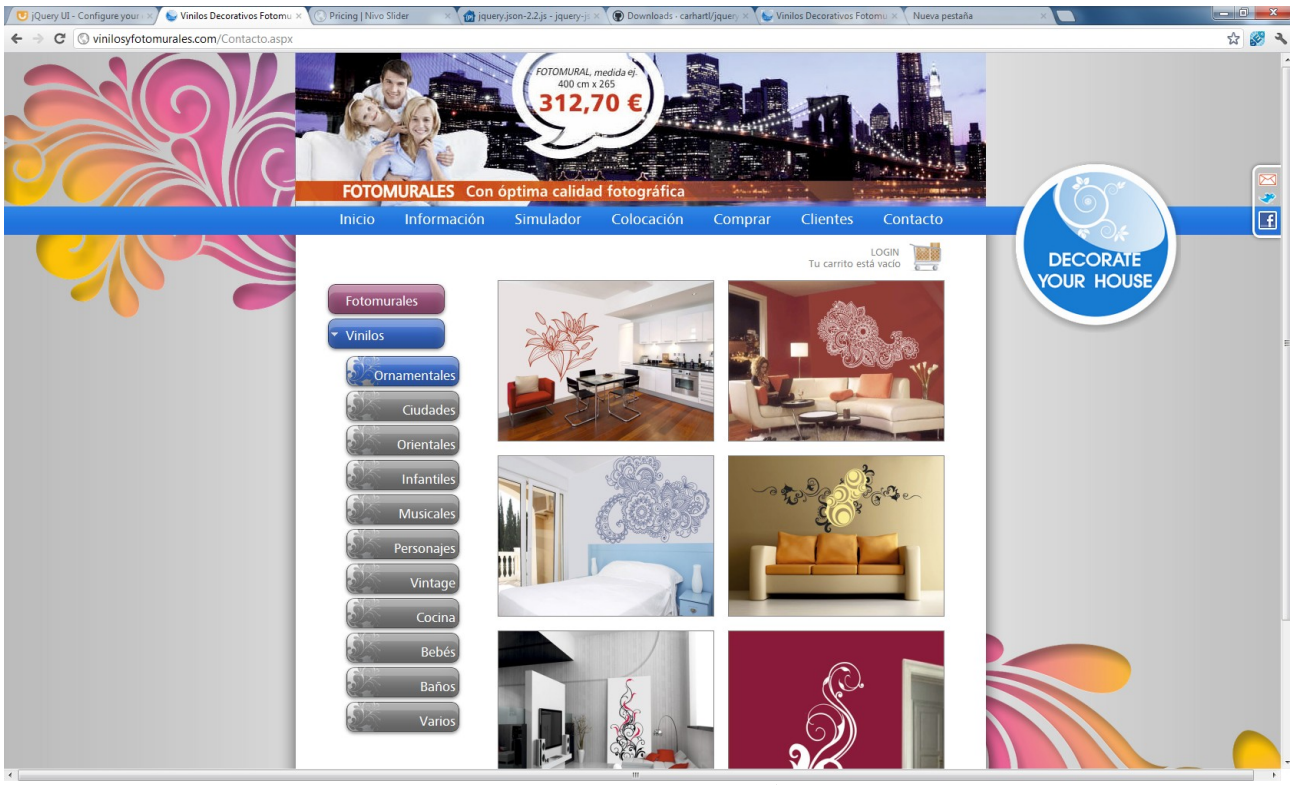
6.2.4 mobileOK Checker

Este test se encuentra en http://validator.w3.org/mobile. Realiza varias comprobaciones para determinar el grado de compatibilidad del portal con dispositivos móviles. Algunas de esas comprobaciones se refieren a los gráficos y colores, tamaño de la página, estándares web para móviles, etc. Los errores encontrados se clasifican según severidad en: críticos, severos, medios o bajos. Después de correr el *test* hemos obtenido un porcentaje de compatibilidad del 80% lo cual indica que la web se visualizará adecuadamente en cualquier dispositivo móvil moderno.

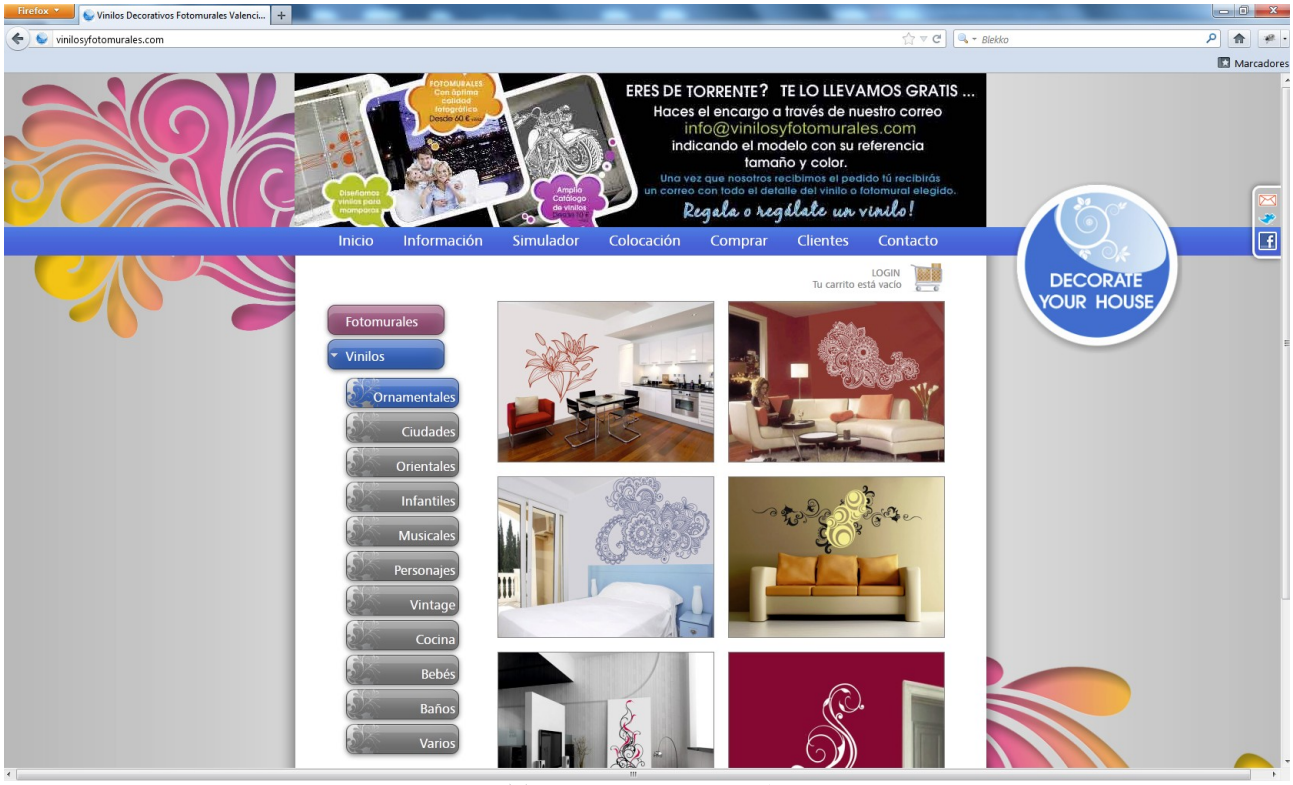


6.3 Test de compatibilidad para navegadores

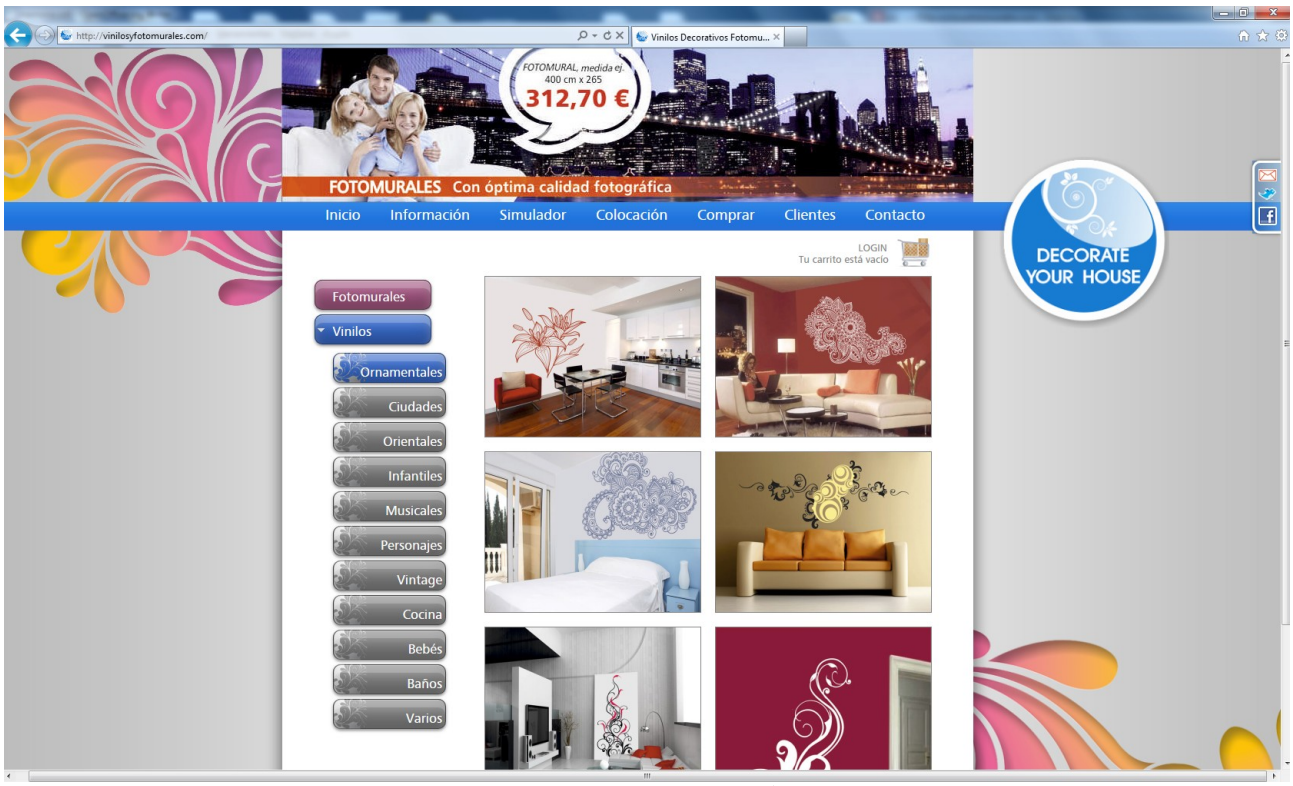
La web http://browsershots.org nos ofrece una herramienta que permite simular como se vería en la ventana de nuestro navegador el portal probado desde distintos entornos y para diferentes navegadores/versiones. Para realizar este test, hemos establecido como configuración la opción de JavaScript habilitado. Los navegadores participantes son los más populares: Google Chrome, Internet Explorer y Mozilla Firefox en sus últimas versiones. Las pruebas se ejecutan tanto en entorno Windows como Linux. El resultado de los *tests* nos arroja que la experiencia de usuario es prácticamente la misma en todos los navegadores salvo en Internet Explorer en cuyo caso algunos efectos de transiciones están desactivados.



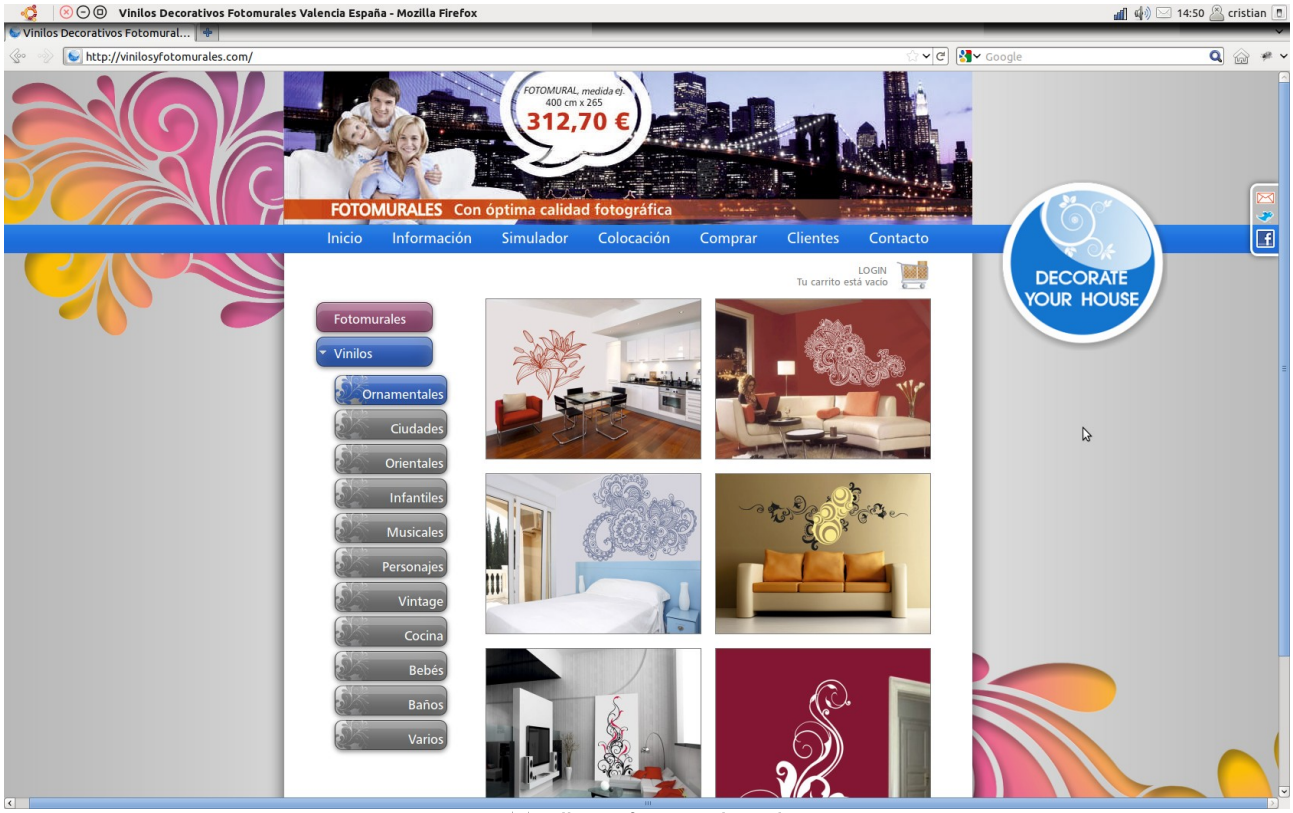
Vista en Google Chrome 19 sobre Windows 7



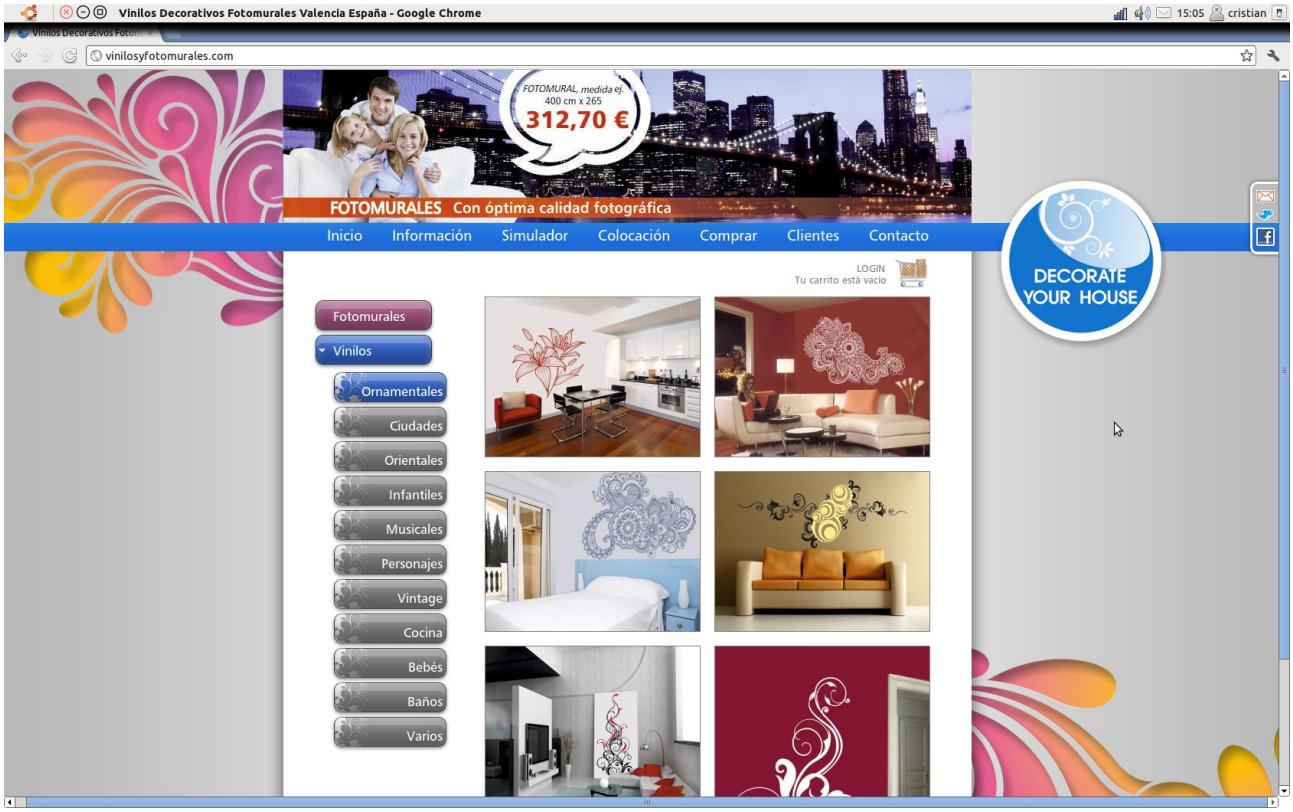
Vista en Mozilla Firefox 12 sobre Windows 7



Vista en Internet Explorer 9 sobre Windows 7



Vista en Mozilla Firefox 12 sobre Ubuntu 12



Vista en Google Chrome 19 sobre Ubuntu 12

6.4 Pruebas en el sandbox de Paypal

Para nuestra web lo más importante es asegurarnos de que no ocurran errores en el momento más crucial que es cuando el cliente se dispone a adquirir nuestros productos. Nuestro sistema está integrado con el proceso de comrpa de Paypal. Por suerte, Paypal nos ofrece un entorno de pruebas o *sandbox* para probar compras con transacciones de pago que no son reales. Estas pruebas permitirán probar nuestro algoritmo IPN que como hemos ya dicho en otros capítulos se trata de una tecnología que nos asegura que los pagos se han hecho de forma segura. Se trata de un algoritmo algo complejo. En nuestro proyecto, lo hemos implementado en el fichero "IPNHandler.aspx.cs". Para consultar los detalles de implementación, ver el anexo 3.
Para hacer pruebas en el sandbox, hemos seguido los siguientes pasos:

- Crear una cuenta de desarrollador en http://developer.paypal.com
- Crear una cuenta de persona particular en el *sandbox* y añadir una tarjeta de crédito.

- Crear una cuenta de empresa en el *sandbox* y añadir una tarjeta de crédito. Confirmar una cuenta bancaria.
- Cambiar la URL a la que se postean los datos en nuestro formulario de compra que es enviado a Paypal. La URL debe ser https://www.sandbox.paypal.com/cgi-bin/webscr en lugar de la habitual, https://www.paypal.com/cgi-bin/webscr.
- Realizar todo el proceso de pago utilizando el correo electrónico y la contraseña de la cuenta particular.

Imagen: Pantalla principal del sandbox.

7. Conclusiones

Llegados a este punto nos disponemos a realizar un comentario de cómo ha transcurrido el desarrollo del proyecto, las dificultades enfrentadas, así como realizar también un comentario sobre la solución final obtenida y posibles mejoras a aplicar.

La primer decisión tomada antes del abordaje de este proyecto fue establecer un conjunto de tecnologías a utilizar. Dada la experiencia durante la carrera, he optado por escoger el *framework* de Microsoft ASP .NET ya que cuenta con un IDE muy poderoso que trae incorporado numerosas herramientas las cuales nos ahorran el tener que implementar las bases de nuestro portal y poder dedicarnos a nuestra lógica que es lo verdaderamente importante. Lo cierto es que a la hora de la práctica lo que en un principio parecía beneficioso por momentos llegó a ser contraproducente. A medida que el proyecto iba creciendo en volumen y cada vez íbamos incorporando nuevas librerías, Visual Studio se hacía más pesado de mover, acusando una lentitud excesiva a la hora de manipular los documentos en operaciones triviales como puede ser búsqueda de términos. Bien es cierto que no contábamos con un ordenador de última generación pero de todas maneras nuestro ordenador de desarrollo cumplía sobradamente los requisitos establecidos. A parte de que el rendimiento de Visual Studio dificultó nuestro desarrollo, comentar también los numerosos problemas que no ha dado el ORM *Entity Framework*. Es sabido de la existencia de múltiples *bugs* que hasta la versión 4 no han sido solventados. Esto implicaba que a la hora de hacer pruebas, se provocaban excepciones muchas veces al azar sin poder nosotros en encontrar una causa justificada de las mismas, ya que podía fallar como no. Hemos invertido un buen número de horas en "cazar" fallos que no eran propiamente nuestros. De esta manera si bien *Entity Framework* nos ha ahorrado un montón de tiempo gracias a la generación automática de código, por otro lado hemos tenido que invertir ese tiempo en depurar código, muchas veces, sumamente farragoso para encontrar el porqué de las excepciones y para finalmente llegar a la conclusión de que se trata de *bugs* del propio *framework*.

Dejando al margen estos inconvenientes, comentar la ausencia de un *framework* de *testing* automático. En todo proyecto, hace falta realizar *testing* tanto unitario como de integración. En el momento en que se dio partida al proyecto nuestro desconocimiento sobre estas tecnologías era total y para cuando quisimos incorporar *testing* era ya demasiado tarde. Lo ideal hubiese

sido seguir una metodología conducida por *tests*, o como es mayormente conocida: "*Test driven development*" (TDD). Es decir, primero se abordan los *tests* construidos siguiendo las especificaciones y teniendo muy en claro cuál es el resultado final de cada *feature* o función a ser implementada y en segundo paso se elabora el código que hace que ese test sea pasado con éxito. Para darle una vuelta más de tuerca, incluso podríamos haber implementado unos *tests* más generales, no tanto a nivel de método sino de interfaz de usuario con herramientas como *Selenium* lo cual nos permite tener garantía de que cualquier cambio en el código no repercutirá en ningún otro sitio de nuestra web y todo seguirá funcionando como es esperado. Ante la ausencia de estas medidas, evidentemente el desarrollo de nuestro proyecto fue por momento caótico; las cosas funcionaban y dejaban de funcionar aleatoriamente y lo peor es que muchas veces tardábamos un buen tiempo en detectar los errores. Lo que está claro es que en el próximo proyecto a abordar, nos preocuparemos por encontrar un *framework* de *testing* que podamos integrar y nos ofrezca la tranquilidad de saber que en el momento en el que se sube el código a el entorno de producción se trata de código estable.

Es que en realidad, no es fortuito que hayamos dejado de lado el *testing* en nuestro proyecto. Nuestro tiempo era limitado y a él hay que sumarle un número de horas importantes dedicadas al *coaching* personal para poder desenvolvernos en las tecnologías que íbamos a tocar. Este aprendizaje nos ayudó a entender como funcionaban las hojas de estilo para maquetar la interfaz gráfica; nos ayudó a entender como funcionan las aplicaciones escritas en *Silverlight*; nos permitió profundizar en el tema de cómo organizar una aplicación siguiendo una estructura en capas; cómo construir aplicaciones gráficas con alta interactividad gracias a JavaScript; cómo funciona la comunicación con servicios web y un largo etcétera. Y es que cada problema enfrentado, implicó un trabajo de indagación muchas veces consultando en páginas web o foros. Tal es así que, por ejemplo, cuando intentábamos lograr la comunicación con el servicio de Fotolia a través de XML-RPC, fue necesario dirigirse al propio creador de la librería escrita en C# que íbamos estar empleando para aclararnos de ciertas dudas. Lo que intentamos transmitir es que el número de horas empleadas en averiguar cómo funcionan las cosas para recién ahí ponerse manos en obra es absolutamente incalculable y desde luego no lo hemos incluido en la planificación porque el proyecto a priori se nos hubiera ido de las manos y quizá hasta hubiese sido desestimado por el alto riesgo que conllevaba.

Sin embargo, todo este esfuerzo no ha sido en vano. Comprobando el resultado final vemos que hemos alcanzado los objetivos. Una tienda sencilla, de diseño llamativo, que permita realizar compras personalizadas en un corto número de pasos, incluso poder previsualizar como quedarían nuestros productos antes de colocarlos en la pared. Un CMS para gestionar productos, pedidos y clientes. Incluso hemos dejado sentadas las bases para que en un futuro se puedan hacer cambios como añadir nuevos productos y categorías sin que ello implique cambios profundos del código. Si el cliente quiere darle un lavado de cara al portal, tampoco sería necesario una gran transformación.

En definitiva, tanto nosotros como el cliente estamos satisfechos con el resultado obtenido y podemos dar por concluido con éxito el proyecto.

8. Referencias

1. La moda de los papeles pintados vuelve con los vinilos decorativos
   http://hogar.mapfre.com/noticia/181/la-moda-de-los-papeles-pintados-vuelve-con-los-vinilos-decorativos
2. Información general sobre los vinilos
   http://blog.difasa.com/?tag=adhesivos
3. Fotomurales, la nueva tendencia en decoración e interiorismo.
   http://www.opendeco.es/fotomurales-nueva-tendencia-decoracion-interiorismo/
4. Decoración de interiores
   http://decoraciondeinterioresmx.blogspot.com.es/2010/09/que-es-la-decoracion-de-interiores.html
5. Hojas de estilo (CSS)
   http://en.wikipedia.org/wiki/CSS
6. Internet Information Services
   http://en.wikipedia.org/wiki/Internet_Information_Services
7. Fotolia
   http://en.wikipedia.org/wiki/Fotolia
8. HTTP
   http://en.wikipedia.org/wiki/HTTP

9. Anexos

9.1 Anexo 1: Implementación clase Cliente

```
public partial class Cliente : global::System.Data.Objects.DataClasses.EntityObject
    {
        /// <summary>
        /// Create a new Cliente object.
        /// </summary>
        /// <param name="nombre">Initial value of Nombre.</param>
        /// <param name="apellidos">Initial value of Apellidos.</param>
        /// <param name="nIF">Initial value of NIF.</param>
        /// <param name="eMail">Initial value of EMail.</param>
        /// <param name="direccion">Initial value of Direccion.</param>
        /// <param name="codigoPostal">Initial value of CodigoPostal.</param>
        /// <param name="poblacion">Initial value of Poblacion.</param>
        /// <param name="provincia">Initial value of Provincia.</param>
        /// <param name="pais">Initial value of Pais.</param>
        /// <param name="telefono">Initial value of Telefono.</param>
        /// <param name="password">Initial value of Password.</param>
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        public static Cliente CreateCliente(string nombre, string apellidos, string
nIF, string eMail, string direccion, int codigoPostal, string poblacion, string
provincia, string pais, string telefono, string password)
        {
            Cliente cliente = new Cliente();
            cliente.Nombre = nombre;
            cliente.Apellidos = apellidos;
            cliente.NIF = nIF;
            cliente.EMail = eMail;
            cliente.Direccion = direccion;
            cliente.CodigoPostal = codigoPostal;
            cliente.Poblacion = poblacion;
            cliente.Provincia = provincia;
            cliente.Pais = pais;
            cliente.Telefono = telefono;
            cliente.Password = password;
            return cliente;
        }
        /// <summary>
        /// There are no comments for property Nombre in the schema.
        /// </summary>
        [global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(IsNullable=
false)]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        public string Nombre
        {
            get
            {
```

```
                return this._Nombre;
            }
            set
            {
                this.OnNombreChanging(value);
                this.ReportPropertyChanging("Nombre");
                this._Nombre =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, false);
                this.ReportPropertyChanged("Nombre");
                this.OnNombreChanged();
            }
        }
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        private string _Nombre;
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnNombreChanging(string value);
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnNombreChanged();
        /// <summary>
        /// There are no comments for property Apellidos in the schema.
        /// </summary>
        [global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(IsNullable=
false)]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        public string Apellidos
        {
            get
            {
                return this._Apellidos;
            }
            set
            {
                this.OnApellidosChanging(value);
                this.ReportPropertyChanging("Apellidos");
                this._Apellidos =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, false);
                this.ReportPropertyChanged("Apellidos");
                this.OnApellidosChanged();
            }
        }
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        private string _Apellidos;
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnApellidosChanging(string value);
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnApellidosChanged();
```

```
        /// <summary>
        /// There are no comments for property NIF in the schema.
        /// </summary>
        [global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(IsNullable=
false)]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        public string NIF
        {
            get
            {
                return this._NIF;
            }
            set
            {
                this.OnNIFChanging(value);
                this.ReportPropertyChanging("NIF");
                this._NIF =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, false);
                this.ReportPropertyChanged("NIF");
                this.OnNIFChanged();
            }
        }
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        private string _NIF;
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnNIFChanging(string value);
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnNIFChanged();
        /// <summary>
        /// There are no comments for property EMail in the schema.
        /// </summary>
        [global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(EntityKeyPr
operty=true, IsNullable=false)]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        public string EMail
        {
            get
            {
                return this._EMail;
            }
            set
            {
                this.OnEMailChanging(value);
                this.ReportPropertyChanging("EMail");
                this._EMail =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, false);
                this.ReportPropertyChanged("EMail");
```

```
                this.OnEMailChanged();
            }
        }
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        private string _EMail;
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnEMailChanging(string value);
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnEMailChanged();
        /// <summary>
        /// There are no comments for property Empresa in the schema.
        /// </summary>
        [global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        public string Empresa
        {
            get
            {
                return this._Empresa;
            }
            set
            {
                this.OnEmpresaChanging(value);
                this.ReportPropertyChanging("Empresa");
                this._Empresa =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, true);
                this.ReportPropertyChanged("Empresa");
                this.OnEmpresaChanged();
            }
        }
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        private string _Empresa;
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnEmpresaChanging(string value);
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnEmpresaChanged();
        /// <summary>
        /// There are no comments for property Direccion in the schema.
        /// </summary>
        [global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(IsNullable=
false)]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        public string Direccion
        {
```

```
            get
            {
                return this._Direccion;
            }
            set
            {
                this.OnDireccionChanging(value);
                this.ReportPropertyChanging("Direccion");
                this._Direccion =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, false);
                this.ReportPropertyChanged("Direccion");
                this.OnDireccionChanged();
            }
        }
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        private string _Direccion;
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnDireccionChanging(string value);
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnDireccionChanged();
        /// <summary>
        /// There are no comments for property CodigoPostal in the schema.
        /// </summary>
        [global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(IsNullable=
false)]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        public int CodigoPostal
        {
            get
            {
                return this._CodigoPostal;
            }
            set
            {
                this.OnCodigoPostalChanging(value);
                this.ReportPropertyChanging("CodigoPostal");
                this._CodigoPostal =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value);
                this.ReportPropertyChanged("CodigoPostal");
                this.OnCodigoPostalChanged();
            }
        }
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        private int _CodigoPostal;
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnCodigoPostalChanging(int value);
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
```

```
yClassGenerator", "4.0.0.0")]
        partial void OnCodigoPostalChanged();
        /// <summary>
        /// There are no comments for property Poblacion in the schema.
        /// </summary>
        [global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(IsNullable=
false)]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        public string Poblacion
        {
            get
            {
                return this._Poblacion;
            }
            set
            {
                this.OnPoblacionChanging(value);
                this.ReportPropertyChanging("Poblacion");
                this._Poblacion =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, false);
                this.ReportPropertyChanged("Poblacion");
                this.OnPoblacionChanged();
            }
        }
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        private string _Poblacion;
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnPoblacionChanging(string value);
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnPoblacionChanged();
        /// <summary>
        /// There are no comments for property Provincia in the schema.
        /// </summary>
        [global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(IsNullable=
false)]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        public string Provincia
        {
            get
            {
                return this._Provincia;
            }
            set
            {
                this.OnProvinciaChanging(value);
                this.ReportPropertyChanging("Provincia");
                this._Provincia =
```

```
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, false);
                this.ReportPropertyChanged("Provincia");
                this.OnProvinciaChanged();
            }
        }
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        private string _Provincia;
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnProvinciaChanging(string value);
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnProvinciaChanged();
        /// <summary>
        /// There are no comments for property Pais in the schema.
        /// </summary>
        [global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(IsNullable=
false)]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        public string Pais
        {
            get
            {
                return this._Pais;
            }
            set
            {
                this.OnPaisChanging(value);
                this.ReportPropertyChanging("Pais");
                this._Pais =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, false);
                this.ReportPropertyChanged("Pais");
                this.OnPaisChanged();
            }
        }
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        private string _Pais;
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnPaisChanging(string value);
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnPaisChanged();
        /// <summary>
        /// There are no comments for property Telefono in the schema.
        /// </summary>
        [global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(IsNullable=
false)]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
```

```
yClassGenerator", "4.0.0.0")]
        public string Telefono
        {
            get
            {
                return this._Telefono;
            }
            set
            {
                this.OnTelefonoChanging(value);
                this.ReportPropertyChanging("Telefono");
                this._Telefono =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, false);
                this.ReportPropertyChanged("Telefono");
                this.OnTelefonoChanged();
            }
        }
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        private string _Telefono;
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnTelefonoChanging(string value);
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnTelefonoChanged();
        /// <summary>
        /// There are no comments for property Fax in the schema.
        /// </summary>
        [global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        public string Fax
        {
            get
            {
                return this._Fax;
            }
            set
            {
                this.OnFaxChanging(value);
                this.ReportPropertyChanging("Fax");
                this._Fax =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, true);
                this.ReportPropertyChanged("Fax");
                this.OnFaxChanged();
            }
        }
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        private string _Fax;
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
```

```
        partial void OnFaxChanging(string value);
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnFaxChanged();
        /// <summary>
        /// There are no comments for property Password in the schema.
        /// </summary>
        [global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(IsNullable=
false)]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        public string Password
        {
            get
            {
                return this._Password;
            }
            set
            {
                this.OnPasswordChanging(value);
                this.ReportPropertyChanging("Password");
                this._Password =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value, false);
                this.ReportPropertyChanged("Password");
                this.OnPasswordChanged();
            }
        }
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        private string _Password;
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnPasswordChanging(string value);
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnPasswordChanged();
        /// <summary>
        /// There are no comments for property DeseaBoletin in the schema.
        /// </summary>
        [global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        public global::System.Nullable<bool> DeseaBoletin
        {
            get
            {
                return this._DeseaBoletin;
            }
            set
            {
                this.OnDeseaBoletinChanging(value);
                this.ReportPropertyChanging("DeseaBoletin");
```

```
                this._DeseaBoletin =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value);
                this.ReportPropertyChanged("DeseaBoletin");
                this.OnDeseaBoletinChanged();
            }
        }
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        private global::System.Nullable<bool> _DeseaBoletin;
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnDeseaBoletinChanging(global::System.Nullable<bool> value);
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        partial void OnDeseaBoletinChanged();
        /// <summary>
        /// There are no comments for Pedidos in the schema.
        /// </summary>
        [global::System.Data.Objects.DataClasses.EdmRelationshipNavigationPropertyAttri
bute("DecorateYourHouseModel", "FK_Pedidos", "Pedidos")]
        [global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.Entit
yClassGenerator", "4.0.0.0")]
        [global::System.Xml.Serialization.XmlIgnoreAttribute()]
        [global::System.Xml.Serialization.SoapIgnoreAttribute()]
        [global::System.Runtime.Serialization.DataMemberAttribute()]
        public global::System.Data.Objects.DataClasses.EntityCollection<Pedido> Pedidos
        {
            get
            {
                return
((global::System.Data.Objects.DataClasses.IEntityWithRelationships)
(this)).RelationshipManager.GetRelatedCollection<Pedido>("DecorateYourHouseModel.FK_Ped
idos", "Pedidos");
            }
            set
            {
                if ((value != null))
                {
                    ((global::System.Data.Objects.DataClasses.IEntityWithRelationships)
(this)).RelationshipManager.InitializeRelatedCollection<Pedido>("DecorateYourHouseModel
.FK_Pedidos", "Pedidos", value);
                }
            }
        }
    }
```

9.2 Anexo 2: Log creación base de datos en SQL Server

```
IF NOT EXISTS (SELECT * FROM sys.schemas WHERE name = N'DecorateYourHouse')
EXEC sys.sp_executesql N'CREATE SCHEMA [DecorateYourHouse] AUTHORIZATION [dacrea]'

GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[establecernuevos]') AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'-- ============================================
-- Author:          Cristian Pérez
-- Create date:
-- Description:
-- ============================================
CREATE PROCEDURE [DecorateYourHouse].[establecernuevos]

AS
BEGIN
       SET NOCOUNT ON;

    UPDATE Vinilos
       SET esNuevo=0 WHERE Id IN (SELECT Id
                                                        FROM Vinilos
                                                        WHERE
(DATEDIFF(minute,Fecha,getutcdate()) >= 86400 ))
RETURN 0
END

'
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[PreciosFotomurales]') AND type in (N'U'))
BEGIN
CREATE TABLE [DecorateYourHouse].[PreciosFotomurales](
       [desde] [float] NOT NULL,
       [hasta] [float] NOT NULL,
       [precio] [float] NOT NULL,
 CONSTRAINT [PK_PreciosFotomurales] PRIMARY KEY CLUSTERED
(
       [desde] ASC,
       [hasta] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[Vinilos]') AND type in (N'U'))
BEGIN
CREATE TABLE [DecorateYourHouse].[Vinilos](
	[Id] [varchar](20) NOT NULL,
	[Categoria] [varchar](40) NULL,
	[Fecha] [datetime] NULL CONSTRAINT [DF__Vinilos__Fecha__0BC6C43E]  DEFAULT
(getdate()),
	[Descripcion] [varchar](120) NULL,
	[Foto] [varchar](120) NULL,
	[esNuevo] [bit] NULL CONSTRAINT [DF_Vinilos_esNuevo]  DEFAULT ((1)),
 CONSTRAINT [PK_Vinilos] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[Colores]') AND type in (N'U'))
BEGIN
CREATE TABLE [DecorateYourHouse].[Colores](
	[Id] [varchar](20) NOT NULL,
	[Codigo] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Colores] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[Clientes]') AND type in (N'U'))
BEGIN
CREATE TABLE [DecorateYourHouse].[Clientes](
	[Nombre] [varchar](120) NOT NULL,
	[Apellidos] [varchar](120) NOT NULL,
	[NIF] [varchar](20) NOT NULL,
	[EMail] [varchar](120) NOT NULL,
	[Empresa] [varchar](120) NULL,
	[Direccion] [varchar](120) NOT NULL,
	[CodigoPostal] [int] NOT NULL,
```

```
	[Poblacion] [varchar](120) NOT NULL,
	[Provincia] [varchar](60) NOT NULL,
	[Pais] [varchar](60) NOT NULL,
	[Telefono] [varchar](60) NOT NULL,
	[Fax] [varchar](60) NULL,
	[Password] [varchar](60) NOT NULL,
	[DeseaBoletin] [bit] NULL,
 CONSTRAINT [PK_Clientes] PRIMARY KEY CLUSTERED
(
	[EMail] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[Administradores]') AND type in (N'U'))
BEGIN
CREATE TABLE [DecorateYourHouse].[Administradores](
	[Email] [varchar](60) NOT NULL,
	[Password] [varchar](20) NOT NULL,
 CONSTRAINT [PK_Administradores] PRIMARY KEY CLUSTERED
(
	[Email] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[Peticiones]') AND type in (N'U'))
BEGIN
CREATE TABLE [DecorateYourHouse].[Peticiones](
	[Id] [varchar](550) NOT NULL,
	[Precio] [varchar](50) NULL,
	[NumTransacion] [varchar](500) NULL,
	[Pedido] [int] NULL,
 CONSTRAINT [PK_Peticiones] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[PedidosDetalles]') AND type in (N'U'))
BEGIN
CREATE TABLE [DecorateYourHouse].[PedidosDetalles](
	[Pedido] [int] NOT NULL,
	[Referencia] [varchar](20) NULL,
	[Cantidad] [int] NOT NULL,
	[Orientacion] [varchar](20) NOT NULL,
	[Acabado] [varchar](20) NULL,
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[Medida] [varchar](50) NOT NULL,
	[Precio] [float] NOT NULL,
	[BYN] [char](2) NULL CONSTRAINT [DF_PedidosDetalles_Blanco y negro]  DEFAULT
(NULL),
	[Coordenadas] [varchar](50) NULL CONSTRAINT [DF_PedidosDetalles_Coordenadas]
DEFAULT (NULL),
	[Codigo] [varchar](20) NULL,
 CONSTRAINT [PK_PedidosVinilos] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[ColoresCapa]') AND type in (N'U'))
BEGIN
CREATE TABLE [DecorateYourHouse].[ColoresCapa](
	[PedidoVinilo] [int] NOT NULL,
	[Capa] [int] NOT NULL,
	[Color] [varchar](20) NOT NULL,
 CONSTRAINT [PK_ColoresCapa] PRIMARY KEY CLUSTERED
(
	[PedidoVinilo] ASC,
	[Capa] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[MedidasPrecios]') AND type in (N'U'))
BEGIN
CREATE TABLE [DecorateYourHouse].[MedidasPrecios](
	[Vinilo] [varchar](20) NOT NULL,
	[Medida] [varchar](20) NOT NULL,
	[Precio] [int] NOT NULL,
```

```
	[Id] [int] IDENTITY(1,1) NOT NULL,
 CONSTRAINT [PK_MedidasPrecios] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [UK_MedidasPrecios] UNIQUE NONCLUSTERED
(
	[Vinilo] ASC,
	[Medida] ASC,
	[Precio] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[Capas]') AND type in (N'U'))
BEGIN
CREATE TABLE [DecorateYourHouse].[Capas](
	[Vinilo] [varchar](20) NOT NULL,
	[Numero] [int] NOT NULL,
	[Nombre] [varchar](20) NULL,
	[Id] [int] IDENTITY(1,1) NOT NULL,
 CONSTRAINT [PK_Capas] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[Pedidos]') AND type in (N'U'))
BEGIN
CREATE TABLE [DecorateYourHouse].[Pedidos](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[Cliente] [varchar](120) NULL,
	[Fecha] [datetime] NULL CONSTRAINT [DF__Pedidos__Fecha__09DE7BCC]  DEFAULT
(getdate()),
	[Estado] [varchar](20) NULL,
	[DatosEntrega] [varchar](500) NULL,
	[DatosFacturacion] [varchar](500) NULL,
	[TipoPago] [varchar](2) NULL,
 CONSTRAINT [PK_Pedidos] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

```
END
GO
IF NOT EXISTS (SELECT * FROM sys.check_constraints WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[CK_LongitudMinima]') AND parent_object_id =
OBJECT_ID(N'[DecorateYourHouse].[Clientes]'))
ALTER TABLE [DecorateYourHouse].[Clientes]  WITH CHECK ADD  CONSTRAINT
[CK_LongitudMinima] CHECK  ((len(ltrim(rtrim([Password])))>=(5)))
GO
ALTER TABLE [DecorateYourHouse].[Clientes] CHECK CONSTRAINT [CK_LongitudMinima]
GO
IF NOT EXISTS (SELECT * FROM sys.check_constraints WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[CK_LongMinima]') AND parent_object_id =
OBJECT_ID(N'[DecorateYourHouse].[Administradores]'))
ALTER TABLE [DecorateYourHouse].[Administradores]  WITH CHECK ADD  CONSTRAINT
[CK_LongMinima] CHECK  ((len(ltrim(rtrim([Password])))>=(5)))
GO
ALTER TABLE [DecorateYourHouse].[Administradores] CHECK CONSTRAINT [CK_LongMinima]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[FK_Peticiones_Pedidos]') AND parent_object_id =
OBJECT_ID(N'[DecorateYourHouse].[Peticiones]'))
ALTER TABLE [DecorateYourHouse].[Peticiones]  WITH CHECK ADD  CONSTRAINT
[FK_Peticiones_Pedidos] FOREIGN KEY([Pedido])
REFERENCES [DecorateYourHouse].[Pedidos] ([Id])
GO
ALTER TABLE [DecorateYourHouse].[Peticiones] CHECK CONSTRAINT [FK_Peticiones_Pedidos]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[FK_PedidosVinilos_Pedidos]') AND parent_object_id =
OBJECT_ID(N'[DecorateYourHouse].[PedidosDetalles]'))
ALTER TABLE [DecorateYourHouse].[PedidosDetalles]  WITH CHECK ADD  CONSTRAINT
[FK_PedidosVinilos_Pedidos] FOREIGN KEY([Pedido])
REFERENCES [DecorateYourHouse].[Pedidos] ([Id])
GO
ALTER TABLE [DecorateYourHouse].[PedidosDetalles] CHECK CONSTRAINT
[FK_PedidosVinilos_Pedidos]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[FK_PedidosVinilos_Vinilos]') AND parent_object_id =
OBJECT_ID(N'[DecorateYourHouse].[PedidosDetalles]'))
ALTER TABLE [DecorateYourHouse].[PedidosDetalles]  WITH CHECK ADD  CONSTRAINT
[FK_PedidosVinilos_Vinilos] FOREIGN KEY([Referencia])
REFERENCES [DecorateYourHouse].[Vinilos] ([Id])
GO
ALTER TABLE [DecorateYourHouse].[PedidosDetalles] CHECK CONSTRAINT
[FK_PedidosVinilos_Vinilos]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[FK_ColoresCapa_Colores]') AND parent_object_id =
OBJECT_ID(N'[DecorateYourHouse].[ColoresCapa]'))
ALTER TABLE [DecorateYourHouse].[ColoresCapa]  WITH CHECK ADD  CONSTRAINT
[FK_ColoresCapa_Colores] FOREIGN KEY([Color])
REFERENCES [DecorateYourHouse].[Colores] ([Id])
GO
```

```
ALTER TABLE [DecorateYourHouse].[ColoresCapa] CHECK CONSTRAINT [FK_ColoresCapa_Colores]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[FK_ColoresCapa_Pedidos]') AND parent_object_id =
OBJECT_ID(N'[DecorateYourHouse].[ColoresCapa]'))
ALTER TABLE [DecorateYourHouse].[ColoresCapa]  WITH CHECK ADD  CONSTRAINT
[FK_ColoresCapa_Pedidos] FOREIGN KEY([PedidoVinilo])
REFERENCES [DecorateYourHouse].[PedidosDetalles] ([Id])
GO
ALTER TABLE [DecorateYourHouse].[ColoresCapa] CHECK CONSTRAINT [FK_ColoresCapa_Pedidos]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[FK_MedidasPrecios]') AND parent_object_id =
OBJECT_ID(N'[DecorateYourHouse].[MedidasPrecios]'))
ALTER TABLE [DecorateYourHouse].[MedidasPrecios]  WITH CHECK ADD  CONSTRAINT
[FK_MedidasPrecios] FOREIGN KEY([Vinilo])
REFERENCES [DecorateYourHouse].[Vinilos] ([Id])
GO
ALTER TABLE [DecorateYourHouse].[MedidasPrecios] CHECK CONSTRAINT [FK_MedidasPrecios]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[FK_Capas]') AND parent_object_id =
OBJECT_ID(N'[DecorateYourHouse].[Capas]'))
ALTER TABLE [DecorateYourHouse].[Capas]  WITH CHECK ADD  CONSTRAINT [FK_Capas] FOREIGN
KEY([Vinilo])
REFERENCES [DecorateYourHouse].[Vinilos] ([Id])
GO
ALTER TABLE [DecorateYourHouse].[Capas] CHECK CONSTRAINT [FK_Capas]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[DecorateYourHouse].[FK_Pedidos]') AND parent_object_id =
OBJECT_ID(N'[DecorateYourHouse].[Pedidos]'))
ALTER TABLE [DecorateYourHouse].[Pedidos]  WITH CHECK ADD  CONSTRAINT [FK_Pedidos]
FOREIGN KEY([Cliente])
REFERENCES [DecorateYourHouse].[Clientes] ([EMail])
GO
ALTER TABLE [DecorateYourHouse].[Pedidos] CHECK CONSTRAINT [FK_Pedidos]
```

9.3 Anexo 3: Manejador IPN para compras con Paypal

```
public partial class IPNHandler : System.Web.UI.Page {
private string business = ConfigurationManager.AppSettings["BusinessEmail"].ToString();
private string currency_code =
ConfigurationManager.AppSettings["CurrencyCode"].ToString();

protected void Page_Load(object sender, EventArgs e) {
  string requestUriString;
  CultureInfo provider = new CultureInfo("en-us");
  string strFormValues =
Encoding.ASCII.GetString(this.Request.BinaryRead(this.Request.ContentLength));

  // getting the URL to work with
  if (String.Compare(ConfigurationManager.AppSettings["UseSandbox"].ToString(), "true",
false) == 0) {
    requestUriString = "https://www.sandbox.paypal.com/cgi-bin/webscr"; }
  else {
    requestUriString = "https://www.paypal.com/cgi-bin/webscr";
  }
  // Create the request back
  HttpWebRequest request = (HttpWebRequest)WebRequest.Create(requestUriString);
  // Set values for the request back request.Method = "POST";
  request.ContentType = "application/x-www-form-urlencoded";
  string obj2 = strFormValues + "&cmd=_notify-validate";
  request.ContentLength = obj2.Length;
  // Write the request back IPN strings
  StreamWriter writer = new StreamWriter(request.GetRequestStream(), Encoding.ASCII);
  writer.Write(RuntimeHelpers.GetObjectValue(obj2));
  writer.Close();
  //send the request, read the response
 HttpWebResponse response = (HttpWebResponse)request.GetResponse();
 Stream responseStream = response.GetResponseStream();
 Encoding encoding = Encoding.GetEncoding("utf-8");
 StreamReader reader = new StreamReader(responseStream, encoding);

 // Reads 256 characters at a time.

 char[] buffer = new char[0x101];
 int length = reader.Read(buffer, 0, 0x100);
 while (length > 0) {
   // Dumps the 256 characters to a string
   string requestPrice;
   string IPNResponse = new string(buffer, 0, length);
   length = reader.Read(buffer, 0, 0x100);

   // getting the total cost of the goods in
   // cart for an identifier
   // of the request stored in the "custom" variable
   requestPrice = GetRequestPrice(this.Request["custom"].ToString());
   if (String.Compare(requestPrice, "", false) == 0) {
     try {
       Peticion peti = new DatosDYH().RecuperarPeticionPorId(this.Request["custom"]);
```

```
      // La transacción no se ha finalizado correctamente.
      // Enviar email.
      string asunto = "Fallo en la solicitud de pedido";
      string cuerpo = "";
      ...
      EMail.EnviarEmail(peti.Pedido.Cliente.EMail, asunto, cuerpo);
     }catch (Exception ex){
        reader.Close();
        response.Close();
        return;
     }
  reader.Close();
  response.Close();
  return;
 }


NumberFormatInfo info2 = new NumberFormatInfo();
info2.NumberDecimalSeparator = ".";
info2.NumberGroupSeparator = ",";
info2.NumberGroupSizes = new int[] { 3 };

// if the request is verified
if (String.Compare(IPNResponse, "VERIFIED", false) == 0) {
 EMail.EnviarEmail("kapocris@gmail.com", "esto no va", "si");
 // check the receiver's e-mail (login is user's
 // identifier in PayPal)
 // and the transaction type
 if ((String.Compare(this.Request["receiver_email"],
     this.business, false) != 0) ||
     (String.Compare(this.Request["txn_type"],
     "web_accept", false) != 0)) {
      //parameters are not correct
       DatosDYH datos = new DatosDYH();
       try {
        Peticion p = datos.RecuperarPeticionPorId(this.Request["custom"].ToString());
        p.NumTransacion = this.Request["txn_id"].ToString();
       datos.GuardarCambios();
        }catch (Exception ex) {
          reader.Close();
          response.Close();
       }

  // La transacción no se ha finalizado correctamente.
  // Enviar email.
  Try {
    Peticion peti = new DatosDYH().RecuperarPeticionPorId(this.Request["custom"]);
    string asunto = "Fallo en la solicitud de pedido";
    string cuerpo = "...";
    EMail.EnviarEmail(peti.Pedido.Cliente.EMail, asunto, cuerpo);
   }catch (Exception ex) {
      reader.Close();
      response.Close();
```

```
      return;
    }
    reader.Close();
    response.Close();
    return;

   }

  // check whether this request was performed
  // earlier for its identifier
  if (this.IsDuplicateID(this.Request["txn_id"])) {
    // the current request is processed. Write
    // a response from PayPal
    // and create a record in the Log file.
    reader.Close();
    response.Close();
    return;
   }

   // the amount of payment, the status of the
   // payment, and a possible reason of delay
   // The fact that Getting txn_type=web_accept or
   // txn_type=subscr_payment are got odes not mean that
   // seller will receive the payment.
   // That's why we check payment_status=completed. The
   // single exception is when the seller's account in
   // not American and pending_reason=intl
   if (((String.Compare(
     this.Request["mc_gross"].ToString(provider),
     requestPrice, false) != 0) ||
     (String.Compare(this.Request["mc_currency"],
     this.currency_code, false) != 0)) ||
     ((String.Compare(this.Request["payment_status"],
     "Completed", false) != 0) &&
     (String.Compare(this.Request["pending_reason"],
     "intl", false) != 0))) {
   /* parameters are incorrect or the payment was delayed. A response from
      PayPal  should not be written to DB of an XML file because it may lead to a
      failure of uniqueness check of the request identifier. Create a record in the Log
      file with information about the request. */

      reader.Close();
      response.Close();
      return;
}

    // Exito. Registrar pedido en la BD.
    DatosDYH dat = new DatosDYH();
    Peticion peticion = null;
    try {
      peticion = dat.RecuperarPeticionPorId(Request["custom"]);
      peticion.Pedido.Estado = "Verificado";
      dat.GuardarCambios();
    }
```

```
     catch (Exception ex) {}
     // La transacción ha finalizado correctamente.
     // Enviar email.
     if (peticion != null) {

     string a = "Solicitud de pedido procesada con éxito";
     string b = "...";
     EMail.EnviarEmail(peticion.Pedido.Cliente.EMail, a, b);
      }
  }
   else  {
     EMail.EnviarEmail("kapocris@gmail.com", "esto no va", "no");

    try {
      Peticion peti = new DatosDYH().RecuperarPeticionPorId(this.Request["custom"]);
      string asunto = "Fallo en la solicitud de pedido";
      string cuerpo = "...";
      EMail.EnviarEmail(peti.Pedido.Cliente.EMail, asunto, cuerpo);
    }catch (Exception ex){}
   }
}
reader.Close();
response.Close();

}

public string GetRequestPrice(string request_id) {
       Peticion p = new DatosDYH().RecuperarPeticionPorId(request_id);
        if (p == null) return ""; else return p.Precio;
}

public bool IsDuplicateID(string txn_id) {
  DatosDYH datos = new DatosDYH();
  Peticion p = datos.RecuperarPeticion(txn_id);
  return p != null;
}

}
```