GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO FINAL DE GRADO

Proyecto Smart Shopping

Autor:
Daniel CUESTA VALERO

Supervisor:
Adriana VENETE
Tutor académico:
Cristina CAMPOS SANCHO

Fecha de lectura: 24 de julio de 2015
Curso académico 2014/ 2015

Índice

Resumen

El documento que se expone a continuación es el resultado del trabajo desarrollado durante la estancia en prácticas del autor, correspondiente al Grado en Ingeniería Informática de la Universitat Jaume I. Dicha estancia se produjo durante los meses de abril a junio de 2015 en la empresa Opentix de Castellón, consultora especializada en la implantación del ERP de código abierto Openbravo.

Este documento explica el proceso que se ha llevado a cabo para el desarrollo del proyecto, desde su planteamiento y planificación, pasando por la definición de requisitos y el diseño de la aplicación, hasta los detalles de implementación.

Capítulo 1

Introducción

1.1. Descripción del proyecto y contexto

1.2. Objetivos del proyecto

1.1. Descripción del proyecto y contexto

El proyecto que se ha desarrollado consiste en un sistema software de asistencia al cliente en tienda, en principio orientado al sector textil. Este sistema pretende ser una ayuda para cliente, que tendrá fácil acceso a información sobre tallas, modelos, etc. También será una potente herramienta de venta, que sugerirá productos relacionados, y generará información que permitirá analizar el comportamiento del consumidor, así como medir el éxito de los productos.

El proyecto, conocido como Smart Shopping, se ha desarrollado para la empresa Opentix, que es una consultoría especializada en el ERP Openbravo, y está ubicada en Castellón. La juventud de la empresa, nacida hace sólo tres años, se compensa con la experiencia en implantación de ERPs de los socios fundadores, y la búsqueda constante de talento en una joven plantilla que casi alcanza las 20 personas y que recibe una exhaustiva formación continua.

El software Openbravo es un sistema de gestión integral para empresas, lo que se conoce habitualmente como *Enterprise Resource Planning* (ERP). Tiene la característica de ser *Open Source*, es decir, que su código está disponible para ser consultado y modificado libremente. También es modular, permitiendo desarrollos a medida que pueden ser combinados según las necesidades del cliente. Algunos de estos módulos son comerciales, requiriendo una licencia de pago para descargarlos y usarlos. El papel de los partners de Openbravo como Opentix es implantar el ERP en las empresas, configurar el software, desarrollar los módulos a medida que sean necesarios y prestar servicios de formación y soporte.

La idea de Smart Shopping surgió en Opentix como resultado de conversaciones con clientes del sector textil, que manifestaban la necesidad de conocer mejor los gustos de sus clientes y su comportamiento en la tienda. Por ejemplo, se considera de vital importancia conocer qué prendas se probaban y cuales de ellas finalmente adquirían. Para ello se desarrolló un dispositivo compuesto por una pantalla táctil y un escáner de códigos de barras que se instalaría en los probadores. De esta forma, el cliente podría consultar la disponibilidad de otras tallas, ver artículos relacionados y pedir ayuda al dependiente si fuera necesario. También se considera la posibilidad de pagar los productos de forma similar a cómo se haría en una tienda online, agilizando el trabajo de los empleados. Quedaría un registro de los artículos por los que se ha interesado el cliente, viendo cuales se han transformado en ventas, y cuales no. Esta información podría usarse, por ejemplo, para la redistribución de los artículos en la tienda. Los artículos que los clientes se prueban pero que finalmente no compran, podrían sustituirse por otros de más éxito, y los que no se están probando, quizá deberían estar en una ubicación más visible.

Hasta el momento de comenzar la estancia en prácticas, se habían hecho prototipos del dispositivo físico pero no se había comenzado el desarrollo de la aplicación. Se partía de una extensión de Openbravo denominada Openbravo Mobile que proporciona un entorno para la creación de aplicaciones web conectadas con el ERP. Algunas de las funcionalidades que proporciona Openbravo Mobile son:

- Acceso de usuarios (*login*)
- Envío de datos entre el ERP y la aplicación web
- Almacenamiento local en la aplicación web
- Herramientas de programación basadas en eventos mediante la librería *Backbone*
- Sistema de construcción de aplicaciones basado en componentes mediante la librería *Enyo*
- Soporte para lector de código de barras

1.2. Objetivos del proyecto

El objetivo esencial de este proyecto es desarrollar un sistema de asistencia en tienda que mejore la experiencia de compra y proporcione información valiosa al vendedor que contribuya al éxito del negocio. Este objetivo puede desglosarse en los siguiente subobjetivos:

- Proporcionar información de utilidad al cliente para incrementar las ventas.
- Fidelizar al cliente mediante una experiencia más adaptada a sus necesidades.
- Acercar la experiencia de la compra online a la tienda física.
- Agilizar el trabajo de los dependientes.
- Mejorar la gestión de la tienda mediante el análisis del comportamiento del cliente.

Alcance

Al finalizar este proyecto se desea tener una aplicación funcional con la que se puedan hacer demostraciones a clientes potenciales, pero teniendo en cuenta que cada cliente tendrá una versión personalizada en cuanto a diseño, tipos de productos, informes, etc. Por este motivo, no se pretende llegar a un producto terminado, sino a un prototipo funcional con las principales características, que luego se adaptará a las necesidades de cada cliente. En este sentido el diseño será sencillo y neutro, se creará un lote de productos de ejemplo de textil para hombre y mujer, y se elaborarán sólo informes básicos para demostrar las posibilidades de la aplicación.

Capítulo 2

Planificación del proyecto

2.1. Metodología y definición de tareas

2.2. Planificación temporal de las tareas

2.3. Estimación de recursos del proyecto

2.1. Metodología y definición de tareas

Para el desarrollo de este proyecto se ha contado con 450 horas de trabajo, 300 de la cuales se realizaron en la empresa, donde el alumno en prácticas contó con la ayuda del personal de Opentix. Se estableció que el alumno estaría en la empresa cinco horas diarias hasta cumplir el tiempo establecido, mientras que el resto de horas se realizarán en casa, y se dedicarán principalmente a la redacción de la memoria y otros informes relativos a la estancia en prácticas.

La metodología empleada se inspira en la metodología tradicional secuencial de proyectos, pero se ha simplificado y adaptado a las características de este proyecto y a la forma de trabajo de la empresa. También se han empleado distintos diagramas del estándar UML para representar casos de uso, secuencias de actividades o modelos entidad-relación.

A continuación se detallan las principales tareas del proyecto:

Inicialmente se realizó una entrevista con el personal de la empresa relacionado con el proyecto, donde se explicaron los antecedentes del proyecto y los requisitos que se deseaba que la aplicación cumpliera. Posteriormente estos requisitos se plasmaron en un diagrama de casos de uso y en una serie de especificaciones.

Al tratarse de un entorno nuevo para el alumno, se planificó una etapa de estudio del entorno destinada a familiarizarse con el ERP Openbravo, con Openbravo Mobile, y las tecnologías necesarias para el desarrollo. Al terminar esta etapa se evaluaría el tiempo necesario para implementar todos los requisitos y en caso de no ser posible el desarrollo de todos, se priorizaría cuales deberían desarrollarse y cuales quedarían fuera del alcance del proyecto.

En la fase de diseño se determinó la manera en que la aplicación final interactuará con el usuario y con el ERP asociado. También se diseñó una interfaz acorde con las necesidades de uso. El prototipo gráfico fue aprobado por la empresa.

En la primera etapa de la fase de desarrollo se instaló y configuró Openbravo en un equipo de desarrollo, junto con el IDE Eclipse y otras aplicaciones necesarias: administrador de base de datos, control de versiones (Mercurial), Tomcat, etc.

El lenguaje utilizado para la programación de la parte del cliente es principalmente Javascript, utilizando las librerías Enyo y Backbone. En la parte del servidor, Openbravo utiliza Java, y la base de datos es PostgreSQL. La comunicación entre ambas partes se realiza mediante servicios web, con JSON como formato de intercambio de datos.

2.2. Planificación temporal de las tareas

En la figura 1 se presenta el desglose de tareas del proyecto y la estimación de tiempo que se hizo inicialmente.

Id	Nombre de tarea	Duración	Comienzo	Fin	Predeces
1	**Inicio del proyecto**	**1 día**	**lun 16/03/15**	**lun 16/03/15**	
2	Reunión inicial con la empresa	0 días	lun 16/03/15	lun 16/03/15	
3	Elaboración de la propuesta técnica	1 día	lun 16/03/15	lun 16/03/15	2
4	**Análisi del entorno**	**26 días**	**mar 17/03/15**	**lun 27/04/15**	**3**
5	Introducción a Openbravo	12 días	mar 17/03/15	mar 07/04/15	
6	Estudio de Openbravo Mobile	5 días	mié 08/04/15	mar 14/04/15	5
7	Estudio de tecnologías asociadas	5 días	mié 15/04/15	mar 21/04/15	6
8	Definición de requisitos	4 días	mié 22/04/15	lun 27/04/15	7
9	**Diseño**	**10 días**	**mar 28/04/15**	**mar 12/05/15**	**8**
10	Diseño de los procesos	5 días	mar 28/04/15	mar 05/05/15	
11	Diseño de la interfaz	5 días	mié 06/05/15	mar 12/05/15	10
12	**Implementación**	**23 días**	**mié 13/05/15**	**vie 12/06/15**	
13	Preparación del entorno	4 días	mié 13/05/15	lun 18/05/15	11
14	Introducción datos de ejemplo	1 día	mar 19/05/15	mar 19/05/15	13
15	Programación	18 días	mié 20/05/15	vie 12/06/15	14
16	Fin de la estancia en prácticas	0 días	vie 12/06/15	vie 12/06/15	15

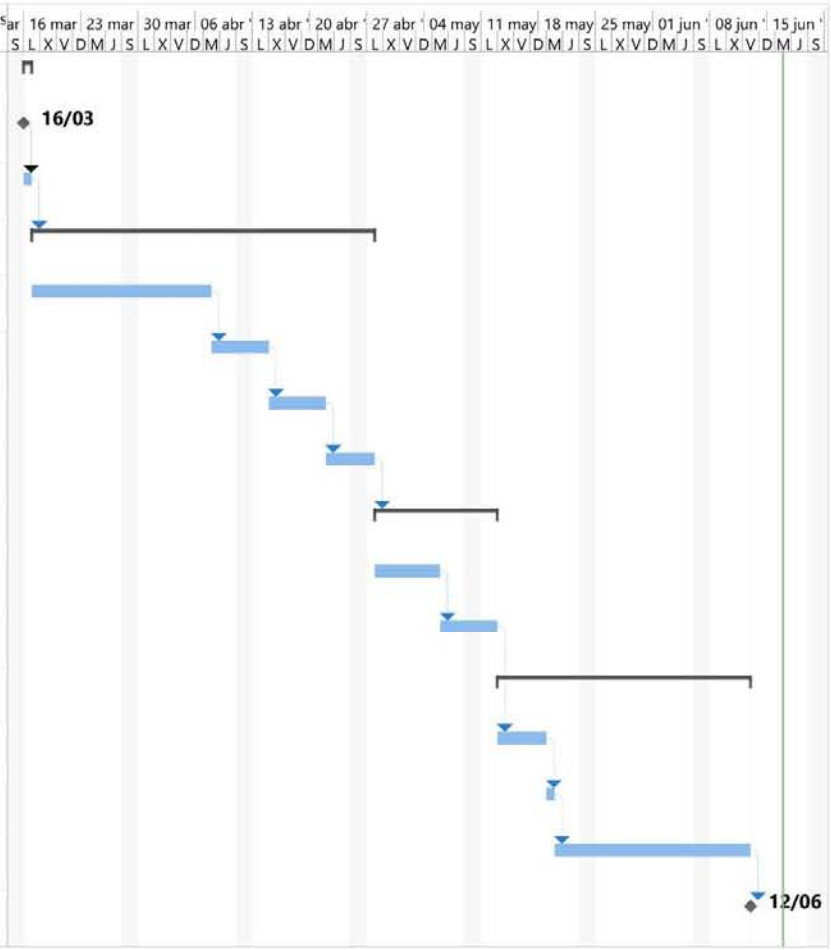

Figura 1. Planificación inicial

Con el paso de los días se comprobó que no era necesario destinar tanto tiempo al aprendizaje de las tecnologías y se adelantó la fecha del diseño y la implementación pudiendo dedicar más tiempo a desarrollar las funcionalidades del producto.

Cabe destacar que las etapas no son completamente secuenciales, sino que se desarrollan de forma iterativa. A medida que se va avanzando en la programación es necesario revisar requisitos y diseños. Teniendo en cuenta esto, la planificación real aproximada fue la que se muestra en la figura 2:

Id	Nombre de tarea	Duración	Comienzo	Fin	Predeces
1	**Inicio del proyecto**	**1 día**	**lun 16/03/15**	**lun 16/03/15**	
2	Reunión inicial con la empresa	0 días	lun 16/03/15	lun 16/03/15	
3	Elaboración de la propuesta técnica	1 día	lun 16/03/15	lun 16/03/15	2
4	**Análisi del entorno**	**9 días**	**mar 17/03/15**	**mar 31/03/15**	**3**
5	Introducción a Openbravo	4 días	mar 17/03/15	mar 24/03/15	
6	Estudio de Openbravo Mobile	2 días	mié 25/03/15	jue 26/03/15	5
7	Estudio de tecnologías asociadas	2 días	vie 27/03/15	lun 30/03/15	6
8	Definición de requisitos	1 día	mar 31/03/15	mar 31/03/15	7
9	**Diseño**	**3 días**	**mié 01/04/15**	**mar 07/04/15**	**8**
10	Diseño de los procesos	2 días	mié 01/04/15	jue 02/04/15	
11	Diseño de la interfaz	1 día	mar 07/04/15	mar 07/04/15	10
12	**Implementación**	**47 días**	**mié 08/04/15**	**vie 12/06/15**	
13	Preparación del entorno	2 días	mié 08/04/15	jue 09/04/15	11
14	Introducción datos de ejemplo	2 días	vie 10/04/15	lun 13/04/15	13
15	Programación	43 días	mar 14/04/15	vie 12/06/15	14
16	Fin de la estancia en prácticas	0 días	vie 12/06/15	vie 12/06/15	15

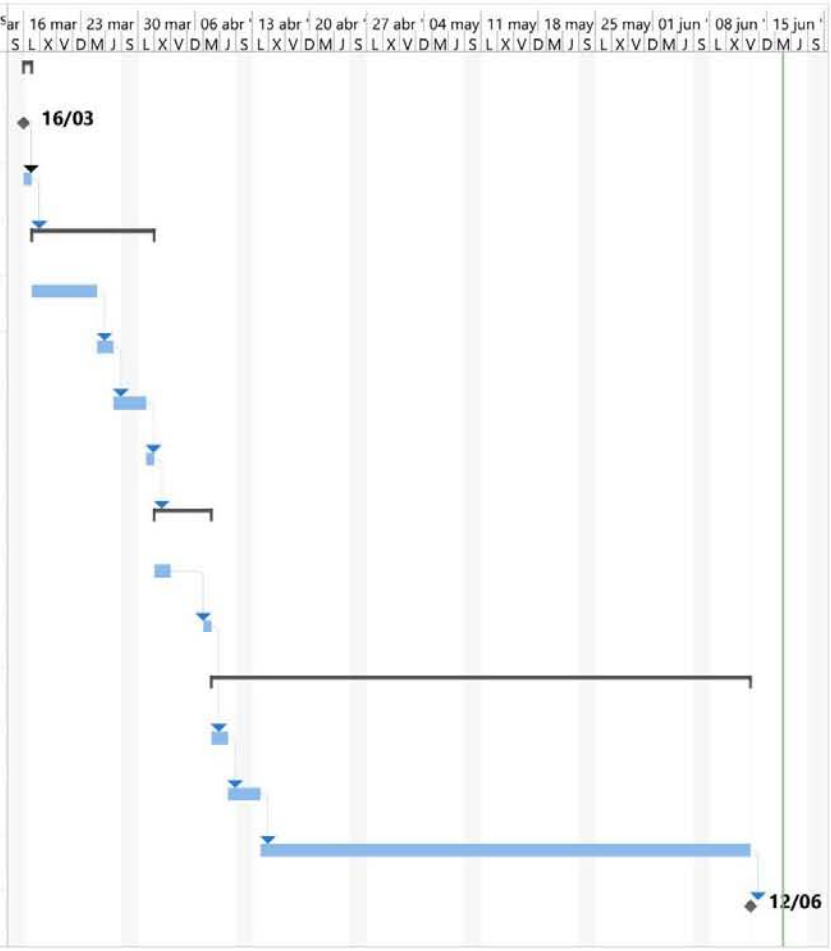

Figura 2. Planificación real

2.3. Estimación de recursos del proyecto

Para el desarrollo de este software se han empleado los siguientes recursos:

- 300 horas de trabajo de un programador junior
- 1 PC valorado en 1.000€
- Sistema operativo Linux y otro software libre

Para calcular el coste del proyecto tenemos que tener en cuenta las siguientes suposiciones:

- Se considera que para finalizar el proyecto serían necesarias 300 horas adicionales de trabajo.
- El coste mensual a jornada completa de un programador junior es de 1.200€, seguros sociales incluidos.
- El PC tiene una vida útil de 4 años. Según el horario de trabajo aplicado, estaría dedicado al proyecto aproximadamente 6 meses.
- No se tiene en cuenta el tiempo que el personal más experimentado ha dedicado a la supervisión del proyecto.
- No se tienen en cuenta para el cálculo los costes fijos de la empresa como alquileres, consumos, sueldos de personal administrativo, etc.

En la figura 3 se muestra el cálculo del coste del proyecto:

	Unidades	Precio unitario	Total
Programador junior	600 horas	9.375 €/hora	5,625.00 €
PC	0.5 años	250 €/año	125.00 €
		Coste total	**5,750.00 €**

Figura 3. Estimación del coste del proyecto

Capítulo 3

Análisis de requisitos

3.1. Definición de requisitos

3.2. Casos de uso

3.3. Especificación de casos de uso

3.1. Definición de requisitos

En este capítulo se estudian los requisitos que se especificaron para aplicación.

En el inicio del proyecto se definieron los objetivos globales que debería cumplir la aplicación, que se han descrito en el capítulo anterior y que básicamente son mejorar la experiencia de compra y recopilar información sobre el comportamiento de los clientes que pueda ser usada en la mejora del negocio. En reuniones posteriores se fueron concretando cuales sería los requisitos funcionales y no funcionales que debería cumplir la aplicación, que se detallan a continuación.

Requisitos funcionales

Los requisitos funcionales que se han desarrollado en este proyecto son:

- El sistema debe permitir al cliente identificarse mediante el escaneo de su tarjeta de cliente.
- El sistema debe permitir al cliente escanear los códigos de producto de los cuales se desea obtener información.
- El sistema debe permitir al cliente navegar por un catálogo visual de productos organizado en categorías y basado en características (talla, color, sexo, etc.).
- El sistema debe ofrecer al cliente información acerca del producto como precio, descuentos, características y disponibilidad.
- El sistema debe ofrecer al cliente productos alternativos o complementarios, basándose en información proporcionada por el vendedor y/o en las compras de otros clientes.
- El sistema debe integrarse con el ERP Openbravo para intercambiar información.
- El sistema debe permitir al cliente solicitar ayuda al dependiente.
- El sistema debe registrar las acciones realizadas por los clientes para su posterior análisis.
- El administrador del sistema debe poder añadir múltiples imágenes a los productos de forma rápida y sencilla.

Existen otros requisitos funcionales que serían deseables pero que han quedado fuera del alcance de este proyecto por falta de tiempo:

- El sistema debe permitir al cliente crear un listado de productos que desea comprar.
- El sistema debe permitir al cliente pagar su compra.

Requisitos de hardware y software

- La aplicación debe funcionar sobre la plataforma Openbravo Mobile, que es un entorno web, y que permite sincronizar datos con el ERP Openbravo.
- La aplicación debe diseñarse para funcionar en una pantalla táctil.

Requisitos de calidad

- El uso de la aplicación debe ser intuitivo incluso para usuarios no habituados a este tipo de tecnología.
- La aplicación debe ser robusta y fiable.

3.2. Casos de uso

A partir de los requisitos funcionales, se puede deducir el diagrama de casos de uso de la figura 4.

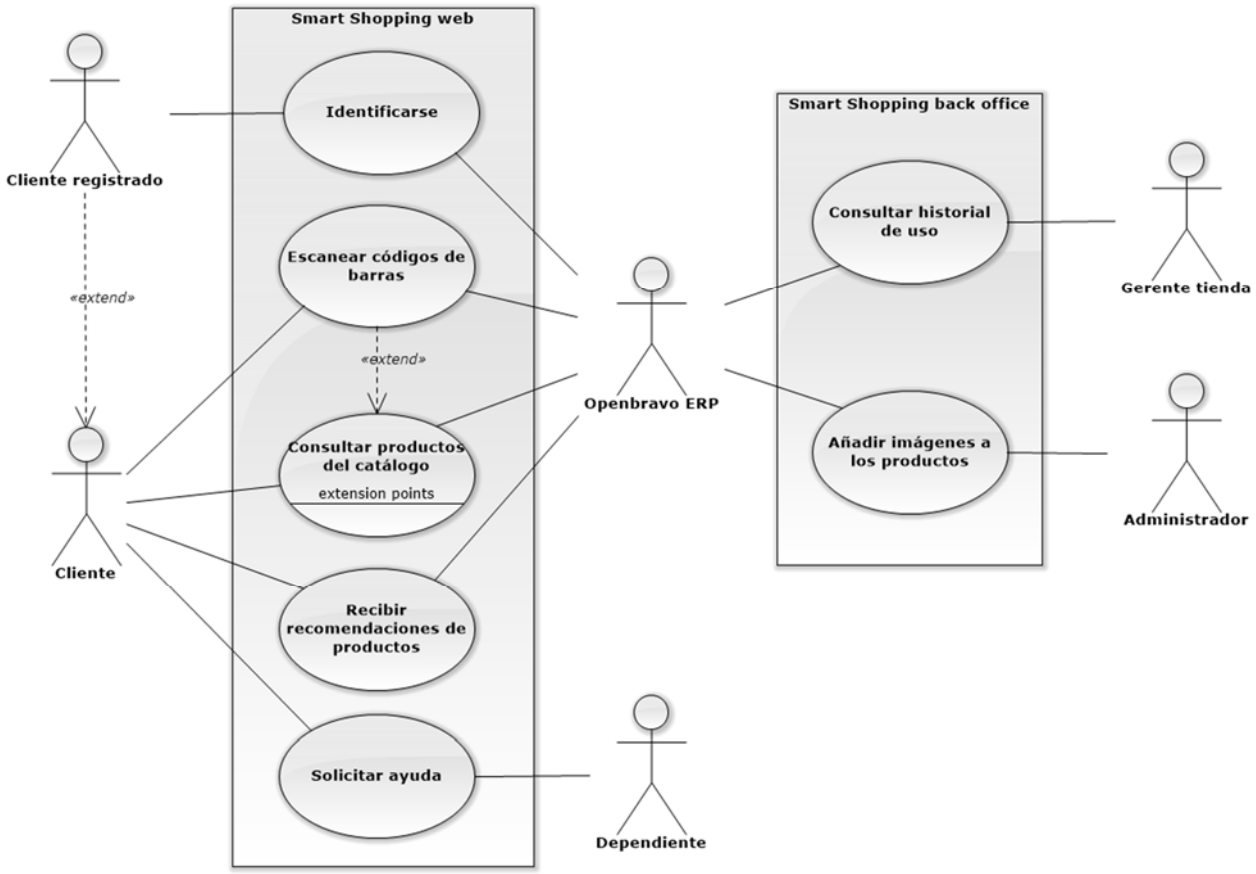

Figura 4. Diagrama de casos de uso

En este diagrama se distinguen los casos de uso que se desarrollan en la aplicación web, que es la que usan los clientes de la tienda, y las que se desarrollan en el back office, es decir en el ERP Openbravo.
También se pueden observar los diferentes actores que intervienen en el uso de la aplicación. En todos los procesos es de vital importancia la conexión con el ERP:

- Para que el cliente pueda identificarse debe obtenerse el listado de clientes registrados de la empresa
- La información de los productos, sus códigos EAN, precios, descuentos, disponibilidad deben obtenerse también del ERP.
- El ERP debe almacenar toda la información de uso de la aplicación Smart Shopping para ser posteriormente utilizada en los informes de negocio.

3.3. Especificación de casos de uso

Casos de uso del back office:

Código	SSW-01
Nombre	Identificarse
Ámbito	Smart Shopping web
Actor principal	Cliente registrado
Actor secundario	Openbravo ERP
Descripción	En cualquier momento el usuario que dispone de tarjeta de cliente puede pasarla por el escáner de códigos de barras para que el sistema pueda reconocerlo. Esto no debe ser obligatorio, pero si debe fomentarse porque aporta información valiosa a la empresa. Tiene que haber una opción de finalizar sesión para cuando el cliente ha terminado de usar la aplicación. En cualquier caso, tras un período de inactividad, la sesión debe terminar automáticamente.

Código	SSW-02
Nombre	Consultar productos del catálogo
Ámbito	Smart Shopping web
Actor principal	Cliente
Actor secundario	Openbravo ERP
Descripción	La utilidad de la aplicación más importante para el cliente es poder visualizar los productos del catálogo, con sus fotografías, descripción, precio, promociones y disponibilidad. Los productos se definen en el ERP y están organizados en categorías. También tienen características como marca, color, talla y sexo que deben tenerse en cuenta en la navegación. Por ejemplo, es necesario una opción de filtrar productos por sexo porque la mayor parte de clientes estará interesado en ver sólo ropa de hombre o sólo ropa de mujer, y no las dos al mismo tiempo. La navegación debe ser fluida y sencilla para cualquier tipo de usuario. Otro aspecto importante es que todas las consultas que se realicen deben quedar registradas para su posterior análisis.

Código	SSW-03 (extiende a SSW-02)
Nombre	Escanear códigos de barras
Ámbito	Smart Shopping web
Actor principal	Cliente
Actor secundario	Openbravo ERP
Descripción	El cliente puede utilizar la aplicación para obtener información de un producto que ha encontrado físicamente. Escaneando el código de barras, la aplicación debe mostrar toda la información de ese producto en concreto, es decir: descripción del producto, imágenes, precio, promociones, otras tallas y colores y su disponibilidad.

Código	SSW-04
Nombre	Recibir recomendaciones de productos
Ámbito	Smart Shopping web
Actor principal	Cliente
Actor secundario	Openbravo ERP
Descripción	La aplicación debe ayudar al cliente a encontrar productos que le puedan interesar y la manera de hacerlo es mostrar sugerencias de productos durante la navegación. El éxito de esta técnica dependerá de lo personalizadas que estén estas sugerencias. Algunos de los posibles enfoques son: - Mostrar productos que manualmente se han identificado como relacionados. Por ejemplo, unos zapatos que combinan con un bolso o con un cinturón. - Mostrar productos que comparten alguna característica. Vestidos del mismo color, zapatos de la misma talla o marca, etc. - Mostrar productos visitados anteriormente por el mismo cliente. - Mostrar productos basados en el historial de compras. Por ejemplo, productos populares, productos que se suelen comprar juntos o productos que suelen comprar clientes con características similares al usuario de la aplicación.

Código	SSW-05
Nombre	Solicitar ayuda
Ámbito	Smart Shopping web
Actor principal	Cliente
Actor secundario	Dependiente
Descripción	Cuando el cliente se encuentra en el probador puede necesitar la ayuda de un dependiente para hacer una consulta o pedir una talla diferente. A través de la aplicación se podrá solicitar esa ayuda. El dependiente recibirá una alerta a través del terminal de caja o de terminales móviles.

Casos de uso del back office:

Código	SSB-01
Nombre	Consultar historial de uso
Ámbito	Smart Shopping back office
Actor principal	Gerente de la tienda
Actor secundario	Openbravo ERP
Descripción	Uno de los objetivos que se quieren alcanzar con esta aplicación es comprender mejor el comportamiento de los clientes. Todas las consultas que realizan los clientes en la aplicación deben quedar registradas para ser analizadas utilizando herramientas OLAP. Algunos de los datos que quedarán registrados son: datos del cliente (si está registrado), fecha y hora de inicio, duración de la visita, categoría, producto, color, talla, precio, descuento, etc.

Código	SSB-02
Nombre	Añadir imágenes a los productos
Ámbito	Smart Shopping back office
Actor principal	Administrador de la aplicación
Actor secundario	Openbravo ERP
Descripción	El ERP Openbravo tiene soporte para definir los productos de forma muy flexible, cumpliendo todos los requisitos necesarios para esta aplicación. Sin embargo, la gestión de las imágenes no es suficiente, porque sólo se permite una imagen por producto. Por lo tanto, es necesario desarrollar una funcionalidad que permita definir varias imágenes para un mismo producto de forma fácil y rápida, ya que se trata de una operación que habrá que realizar con mucha frecuencia.

Capítulo 4

Diseño

4.1. Estructura general de la aplicación

4.2. Modelo de datos

4.3. Diseño de la interfaz

4.4. Venta cruzada

En este capítulo se justifican las principales decisiones tomadas en cuanto a la estructura, funcionamiento general y diseño de la aplicación.

4.1. Estructura general de la aplicación

El la figura 5 se muestra el diagrama de actividades que a grandes rasgos muestra las principales operaciones que realiza la aplicación.

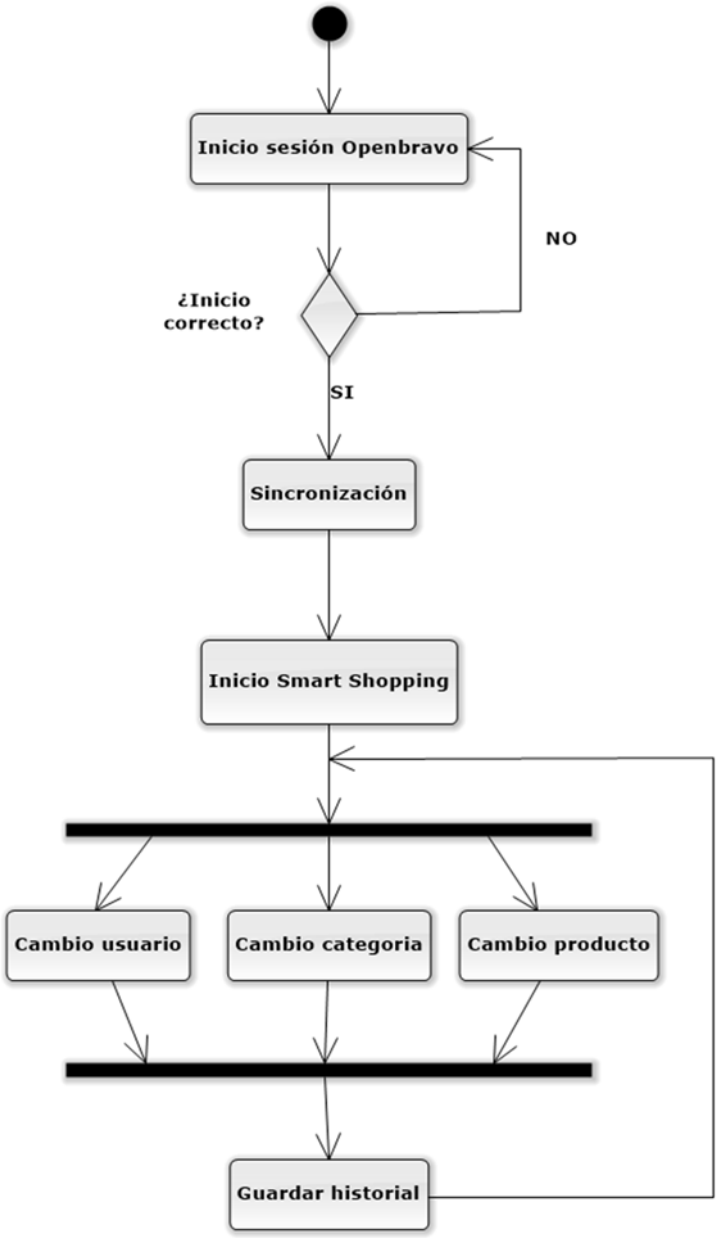

Figura 5. Diagrama de actividades

La primera parte del diagrama se lleva a cabo con la ayuda de Openbravo Mobile, que proporciona soporte a las acciones de inicio de sesión y sincronización con el ERP.

Al iniciar la aplicación se solicita acceder con un usuario válido del ERP Openbravo, sin lo cual no sería posible llevar a cabo el siguiente paso que es una operación que requiere autentificación.
La sincronización consiste en obtener todos los datos necesarios del ERP como productos, imágenes, categorías, precios, clientes, preferencias de configuración, etc. Estos datos se almacenan en una base de datos local, en el cliente. De este modo, si se pierde la conexión, la aplicación puede seguir funcionando con normalidad.

Una vez completada la carga de datos comienza la aplicación Smart Shopping. El usuario tiene varias opciones para interactuar con la aplicación, y en cada una deja un registro que se envía al ERP para que la almacene en la base de datos general. Para disminuir la carga de la red, se puede enviar el historial cada cierto número de acciones en lugar de hacerlo registro por registro.

4.2. Modelo de datos

La entidad con la que se trabaja en esta aplicación es el producto, que se corresponde con los artículos que una tienda pone a la venta. La aplicación está esencialmente destinada a tiendas de moda, lo cual implica algunas características especiales de los productos. Un mismo modelo de prenda puede estar disponible en diferentes tallas y colores. Todas ellas comparten atributos como marca, precio, nombre, descripción, etc. y difieren en otros como talla, color, código EAN/UPC, etc. Para gestionar este tipo de productos de forma ágil, es necesario un sistema que permita definir una sola vez los atributos comunes, pero con la suficiente flexibilidad para permitir cambiar los que no lo sean.

Figura 6. Ejemplo variantes de producto

Otra peculiaridad que observamos y que es fundamental en este proyecto es la gestión de las imágenes. En primer lugar, queremos tener un número variable de imágenes de cada producto, algo que no está soportado en Openbravo. Por otra parte, las variantes del mismo color comparten las mismas imágenes.

Openbravo permite definir productos de forma muy flexible. Por ejemplo, nos permite definir nuestras propias características, como *Talla*, *Color* y *Sexo* que después nos permitirán clasificar los productos. A continuación se explicará la manera en la que Openbravo gestiona los productos, para después especificar las modificaciones que deben realizarse.

La manera de definir productos con variantes en Openbravo es la siguiente:

- Creamos un producto "Genérico" que contiene todos los atributos comunes y viene a ser una plantilla con la que se crearán los demás. Es importante destacar que el genérico no se corresponde con ningún producto físico y por lo tanto no se puede vender.
- Indicamos que características tiene nuestro producto, en nuestro caso *Sexo*, *Talla* y *Color*. Y cuales definen las variaciones de ese producto (*Talla* y *Color*, porque *Sexo* es común).
- Mediante un proceso automático creamos las *Variantes*. Para ello tenemos que indicar que combinaciones de talla y color existen. El sistema crea las variantes utilizando la plantilla (el producto genérico), asignando a cada uno los valores correspondientes de talla y color.
- Por último es necesario revisar cada variante creada para completar los datos faltantes, como el código EAN/UPC.

Tras estudiar las posibilidades del sistema se ha determinado que el sistema cubre todas las necesidades de la aplicación excepto la gestión de imágenes que es insuficiente y debe ser extendida.

Teniendo en cuenta que la operación de dar de alta nuevos productos es muy frecuente, es necesario optimizar el proceso para que sea lo más rápido posible, lo cual implica no duplicar tareas innecesariamente. El sistema actual permite asignar una imagen al producto genérico que después se copia a todas las variantes. Siguiendo el ejemplo de las camisetas, si hubiera 4 colores y 3 tallas, tendríamos 12 variantes, de las cuales 9 se crearían con la imagen equivocada y habría que modificarlas una por una. Si además no se trata de una imagen, sino que hay 3 o 4 imágenes por color, el problema se hace considerable.

Por lo tanto el sistema debe permitir, en cada producto genérico, definir varias imágenes por color. Esto implicará crear una funcionalidad nueva mediante el desarrollo de un nuevo módulo de Openbravo. Para ello será necesario estudiar la forma en la que Openbravo almacena los productos en la base de datos y añadir las tablas y campos que sean necesarios. Posteriormente se crearán los componentes de la interfaz que permitan manipular esos nuevos datos.

4.3. Diseño de la interfaz

La interfaz de la aplicación será un catálogo que permita visualizar los productos de forma rápida pero también obtener información detallada de cada uno. Tiene que tener la posibilidad de distinguir las prendas por sexo, ya que lo más habitual será interesarse sólo por ropa de hombre o de mujer. Se podrá consultar la información detallada de un producto, pudiendo ver tallas y colores disponible, y se mostrarán productos relacionados que puedan interesar al cliente.

También debe tratarse de un diseño neutro, que pueda adaptarse fácilmente a la imagen corporativa de cada tienda.

Se han definido dos secciones diferenciadas:
Categorías. Muestra un listado de categorías de producto. Seleccionando cada una de ellas, se muestra una imagen de cada uno de los productos dentro de esa categoría.
Detalle de producto. Muestra todas las imágenes del producto, la información del mismo y los productos relacionados.

A continuación se muestra el diseño de cada una de las secciones:

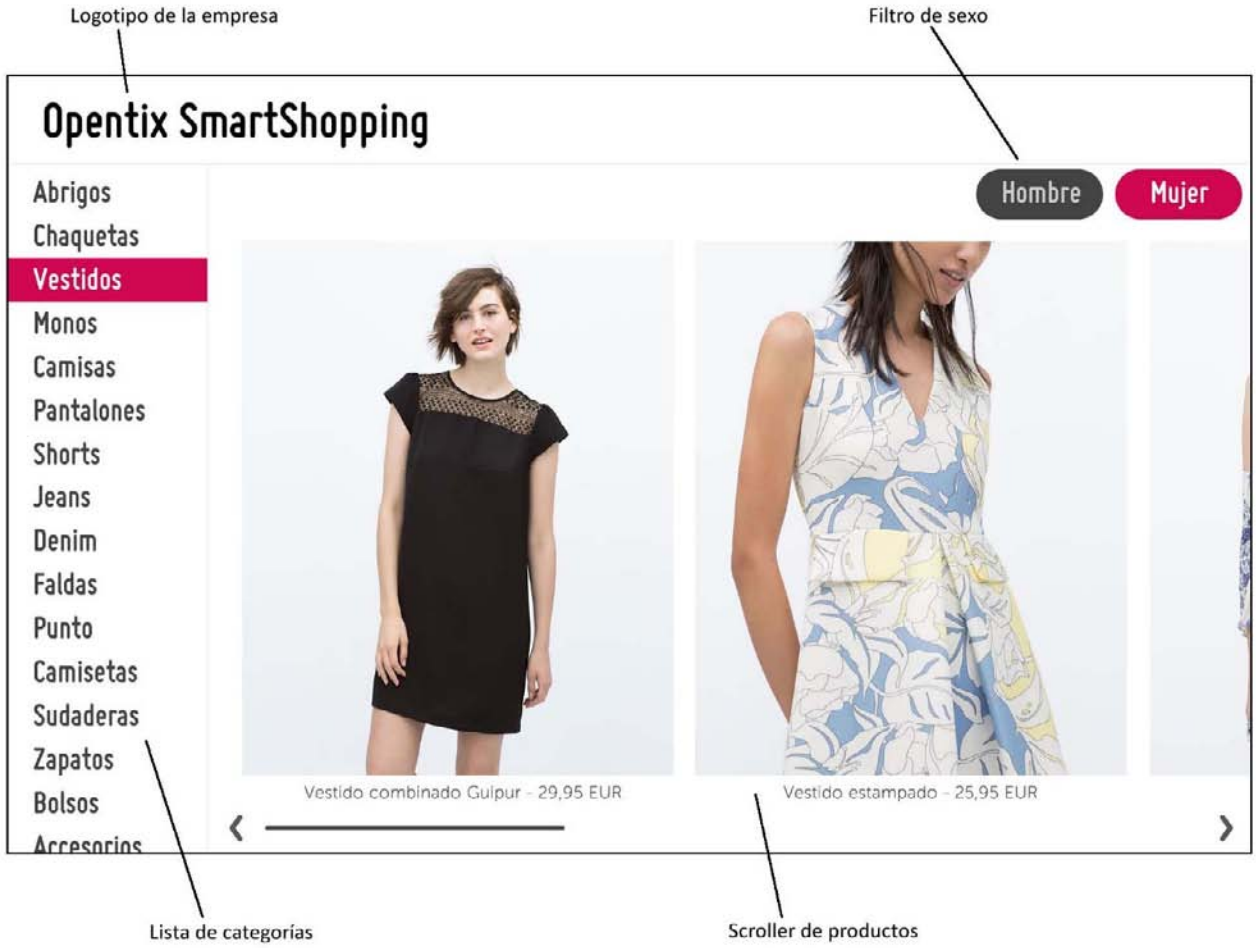

Figura 7. Diseño sección Categorías

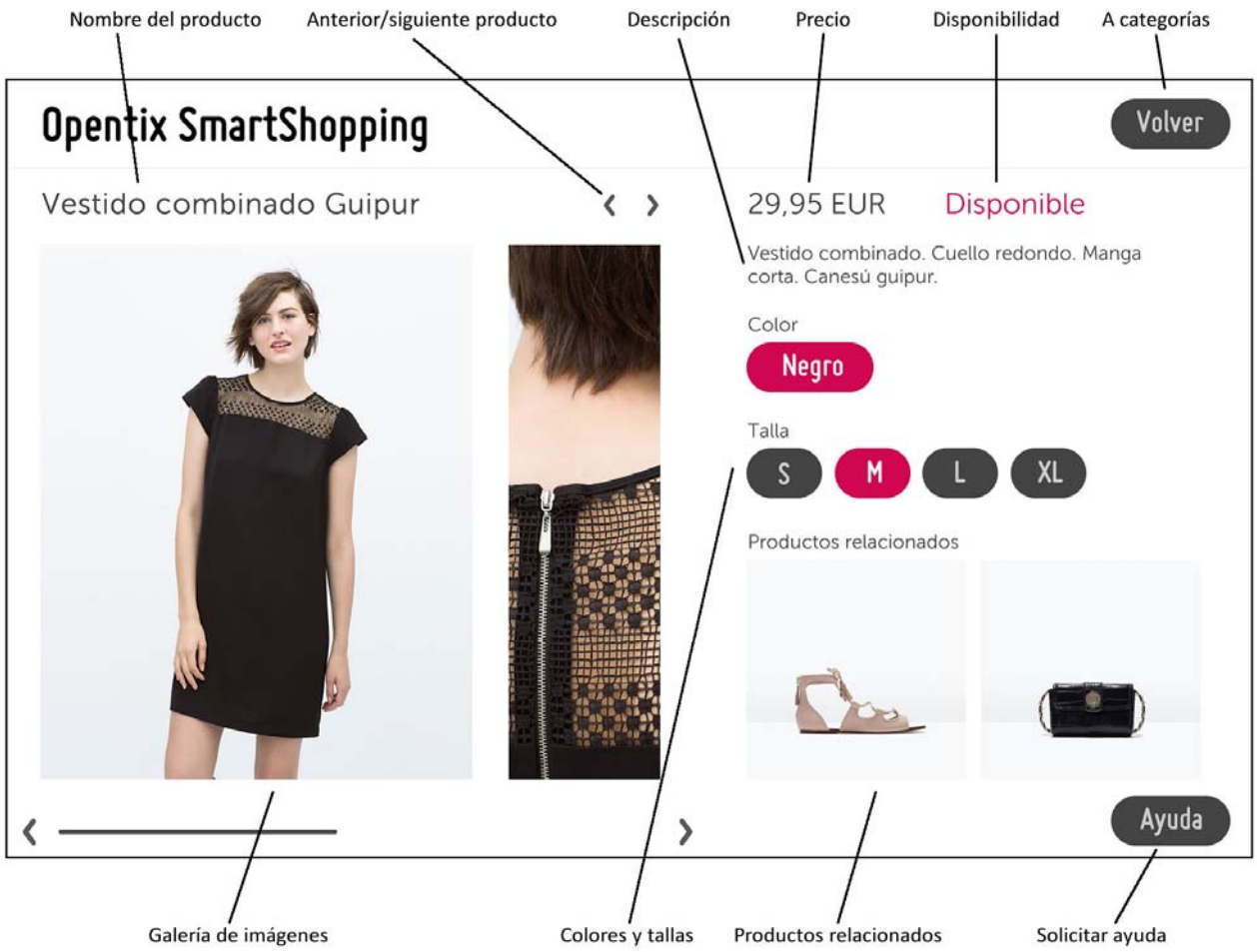

Figura 8. Diseño sección Producto

Consideraciones de usabilidad

Con el objetivo de facilitar el uso de la aplicación para todo tipo de personas se han tenido en cuenta algunas de las principales recomendaciones comúnmente aceptadas [1]:

- Uso moderado y significativo del color. Los botones en rojo indican que están seleccionados.
- Colores bien contrastados fáciles de distinguir para personas daltónicas.
- Tamaño de letra de tamaño suficiente para garantizar la legibilidad.
- Organización de elementos de izquierda a derecha y de arriba a abajo según su importancia.
- Distinguir claramente los botones de otros elementos no interactivos.
- Tamaño adecuado de los botones para el funcionamiento en pantallas táctiles.
- Botones claramente destacados para volver atrás y para solicitar ayuda.
- Densidad de contenido no excesiva.
- Contenido estructurado en zonas.
- Scroll horizontal adecuado para pantallas táctiles.

4.4. Venta cruzada

Para cumplir el requisito de generar recomendaciones para el cliente, se quiere implementar un algoritmo de venta cruzada basado en las ventas del establecimiento. Cuando el cliente este visitando un determinado producto, el sistema debe localizar y mostrar otros productos que se han comprado juntos en alguna de las transacciones guardadas. La manera más habitual de implementar este comportamiento es mediante el descubrimiento de reglas de asociación [2]. Esta técnica habitual en *data mining* proporciona afirmaciones del tipo *el 90% de las personas que compran mantequilla también compran leche* (mantequilla->leche). Por lo tanto, puede ser una buena idea colocar ambos productos en lugares cercanos dentro de un supermercado.

Para determinar el grado de interés de una regla, tenemos dos medidas posibles [3]:

- El soporte de una regla es el número de transacciones en el que se cumple dicha regla.
- La confianza de una regla a->b es el número de transacciones donde se cumple la regla, entre el número de transacciones total donde aparece a.

Es importante destacar que la regla b->a tendrá siempre el mismo soporte que a->b, pero la confianza puede ser igual o no.

El problema se puede extender si consideramos reglas del tipo a, b, ... -> x o x -> a, b, ..., lo cual aumenta considerablemente la necesidad de procesamiento cuando el número de transacciones y de productos es grande. Para el caso de esta aplicación, es suficiente con reglas de tipo a->b, donde a sería el producto que se está visualizando y b el producto recomendado.

El algoritmo que se va a implementar en esta aplicación es una versión sencilla de los que se utilizan habitualmente en las grandes cadenas de supermercados. Consta de tres pasos:

Paso 1 - Preparación de los datos

El cálculo del soporte y la confianza de cada regla requieren recorrer un gran número de veces las transacciones, por eso es importante representar las transacciones de la forma más condensada posible.

Sea T el conjunto de transacciones: $T = t_1, t_2, ..., t_n$
Y sea P el conjunto de productos, de forma que cada producto p se representa como un vector binario, con p[k]=1 si el producto se compró en la transacción t_k, y p[k]=0 si no se compró en esa transacción. Por lo tanto, despreciamos la cantidad que se compró, que no aporta más valor a las reglas.

El resultado es una matriz de bits donde los productos son las filas y las transacciones las columnas. El tamaño de la matriz para 1024 productos y 8x1024 transacciones sería de solo 1 kilobyte.

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15	t16	t17	t18	t19	t20
Pantalón	0	1	0	1	0	0	1	1	1	0	1	1	1	0	0	1	0	0	0	0
Vestido	1	0	0	0	0	0	1	1	1	1	0	0	1	0	1	0	0	0	1	1
Camisa	1	1	1	0	1	1	0	0	1	0	0	0	1	1	0	0	1	0	0	1
Bolso	0	1	1	1	0	1	1	1	0	0	0	0	0	1	1	1	1	0	0	0
Zapatos	0	1	1	1	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1

Paso 2 - Cálculo de la matriz de asociaciones

El siguiente paso será calcular el grado de interés de relación de cada par de productos. Utilizaremos como medida el soporte, es decir, el número de transacciones en el que aparecen los dos productos, que es adecuado para el tipo de negocios a los que se destina este proyecto. El resultado será una matriz de tamaño m x m, donde m es el número de productos. La posición (i,j) de la matriz sería un número entero que contendría el soporte de la regla i->j.

Como soporte(a->b) = soporte(b->a), será una matriz simétrica.

	Pantalón	**Vestido**	**Camisa**	**Bolso**	**Zapatos**
Pantalón		4	3	5	8
Vestido	4		4	3	7
Camisa	3	4		5	6
Bolso	5	3	5		6
Zapatos	8	7	6	6	

El coste computacional de la operación sería $O(m^2 \times n)$.

Este tipo de problema tiene una magnitud considerable en el caso de supermercados donde el número de productos y de transacciones es muy alto, pero en el caso que nos ocupa es perfectamente manejable. Hay que tener en cuenta que cada temporada el surtido de productos se renueva, por lo que tanto los productos como las transacciones de temporadas anteriores se pueden omitir manteniendo siempre un número bastante limitado. Por lo tanto, esta matriz se puede calcular una o varias veces al día sin afectar al rendimiento de la aplicación ni requerir hardware especial.

Paso 3 - Obtención de los productos relacionados

Para obtener un número z de productos relacionados con un producto t_o a partir de la matriz de asociaciones que ya tenemos podríamos hacerlo de dos maneras:

- Tomar los z productos que tengan mayor soporte respecto de t_o
- Elegir una lista de z productos donde la probabilidad de estar sea proporcional al soporte del producto respecto de t_o.

Mientras que el primer criterio siempre proporcionaría los mismos productos, el segundo proporcionaría cierta rotación que daría visibilidad a productos con menos transacciones. Según los datos del ejemplo, al consultar el producto *Pantalón* muchas veces, veríamos *Zapatos* con más frecuencia que *Camisa* o *Vestido*.

Posibles variaciones

A - Definición manual de reglas de asociación.
El vendedor puede estar interesado en promocionar ciertos artículos, o en establecer combinaciones de productos según su criterio y experiencia. Para implementar esto se podría definir otra matriz de asociaciones que sustituyera a la matriz extraída de las transacciones, o bien sumar ambas, de forma que se mezclarían ambos criterios.

B - Selección aleatoria
Cuando existen pocas transacciones o se trata de un producto nuevo, no será posible determinar productos relacionados, por lo que es necesario implementar un criterio de selección adicional. Puede usarse un criterio puramente aleatorio, mostrar productos de cierta categoría, los productos más vendidos, productos con descuento, etc.

C - Selección de productos basada en el cliente
Si el cliente está registrado en la tienda y utiliza su tarjeta de cliente, se puede refinar la selección de productos relacionados teniendo en cuenta sus características como edad, sexo, zona geográfica, etc. El objetivo sería determinar una serie de productos que es más probable que le gusten, basándose en las compras de clientes como características similares a las suyas.

Capítulo 5

Preparación del entorno de desarrollo

5.1. Pila de desarrollo

5.2. Modularidad

5.3. Programación en el lado del cliente: Javascript y SPAs

5.4. Programación con Openbravo Mobile

5.1. Pila de desarrollo

La preparación del entorno de desarrollo o de producción no es un asunto trivial cuando se trata de un ERP como Openbravo. Se requiere todo un ecosistema de aplicaciones que tienen que ser correctamente configuradas para trabajar conjuntamente, lo que se conoce como la pila de desarrollo. Los principales componentes que requiere Openbravo son:

- Sistema operativo
- Base de datos
- Java
- Tomcat
- Ant
- Mercurial

Figura 9. Logotipos pila de desarrollo

A continuación se describen las características de cada uno.

Sistema operativo
Openbravo recomienda su instalación sobre Linux, pero puede funcionar sobre otros sistemas UNIX, Mac OS y sobre Windows, al estar programado en Java. También es común la utilización de máquinas virtuales. Existe una colección de entornos preconfigurados en: http://sourceforge.net/projects/openbravo/

Base de datos: Openbravo soporta tanto PostgreSQL como Oracle.

PostgreSQL es un sistema de gestión de bases de datos (SGBD) objeto-relacional y de código abierto. Es altamente escalable tanto por la cantidad de datos que puede manejar como por el número de usuarios concurrentes que permite, por eso es perfectamente adecuado para entornos de empresa. Proporciona todo tipo de herramientas avanzadas como vistas, disparadores, diferentes tipos de índices, soporte para los principales lenguajes de programación, herramientas de monitorización e interfaces gráficas gratuitas como PGAdmin.

Oracle es un SGBD objeto-relacional desarrollado por Oracle Corporation, una de las principales empresas de software del mundo. Durante muchos años ha liderado el mercado de bases de datos aunque en la actualidad ha reducido su cuota de mercado por la competencia de Microsoft SQL Server y los SGBD de licencia libre como PostgreSQL y MySQL. Aun así, el SGBD de Oracle se considera el más avanzado, con numerosas funcionalidades exclusivas como soporte para documentos XML, funciones de *Business intelligence* y *Big Data*.

Habitualmente Opentix utiliza el SGBD PostgreSQL por su excelente rendimiento y por ser totalmente gratuito. En este proyecto también se ha utilizado.

Java es un lenguaje de programación de propósito general orientado a objetos. Tiene la peculiaridad de compilarse en un *bytecode* (lenguaje de bajo nivel), que puede interpretarse por una máquina virtual Java en cualquier tipo de computador o dispositivo. De este modo se obtienen las ventajas de rendimiento de un lenguaje compilado, y la portabilidad de uno interpretado.
Es uno de los lenguajes más utilizados en la actualidad, especialmente en entornos empresariales, y es el lenguaje principal en el que está programado Openbravo.

Hay dos tipos de instalaciones principales para utilizar Java en un equipo:
Java Runtime Environment (JRE), que contiene la máquina virtual para ejecutar aplicaciones Java compiladas, y las bibliotecas básicas.
Java Development Kit (JDK), que además contiene el compilador (*javac*). Es la que se utiliza en entornos de desarrollo.

Apache Tomcat es un contenedor de *servlets*. Los servlets son una de las maneras en las que las aplicaciones Java pueden funcionar. Un *servlet* es una aplicación del lado del servidor que espera peticiones del cliente y responde a ellas. Estas aplicaciones pueden ser cargadas en el arranque del servidor y permanecer en memoria a la espera de peticiones, lo cual supone una respuesta más rápida que otro tipo de arquitecturas más antiguas como los CGIs. Tomcat es el encargado de gestionar el ciclo de vida de los servlets y getionar las peticiones y respuestas.

Ant es una herramienta que permite automatizar tareas repetitivas y que se usa principalmente en proyectos Java. Permite entre otras compilar, mover ficheros, ejecutar aplicaciones y tests sobre aplicaciones Java. Las tareas de Ant se escriben en ficheros XML y son independientes del sistema donde se ejecutan, por lo que se pueden utilizar en cualquier sistema.

Openbravo proporciona una serie de tareas de Ant para facilitar el trabajo de los desarrolladores. Estas son algunas de las más importantes:

`ant install.source`
Crea la base de datos e introduce los datos de muestra. Compila la aplicación por primera vez y la despliega en la ruta de Tomcat.

`ant smartbuild`
Después de hacer modificaciones en el código de Openbravo es necesario ejecutar esta tarea para compilar y desplegar la aplicación de nuevo. Tiene la ventaja de ser incremental, es decir, que sólo actúa sobre los ficheros que han sido modificados.

`ant export.database`
Al desarrollar en Openbravo a menudo es necesario modificar la base de datos. Con esta tarea los cambios realizados en la base de datos se almacenan como archivos XML, de forma que pueden ser trasladados entre distintas instancias de Openbravo. Este proceso se explicará más adelante en el apartado de modularidad.

Mercurial es un sistema de control de versiones distribuido, que permite que múltiples desarrolladores trabajen en el mismo proyecto y llevar un registro de todas las modificaciones que se han hecho sobre el código. Por ejemplo, podemos descargar la versión más reciente de Openbravo y hacer modificaciones sobre ella. Cuando aparezca una nueva versión oficial, fusionar los cambios sin perder los que hemos realizado nosotros, siempre y cuando las dos modificaciones no se solapen. Esta es una de las herramientas más populares de este tipo junto con *Git*.

5.2. Modularidad

Una de las fortalezas del ERP Openbravo es su modularidad, es decir, la capacidad para empaquetar los desarrollos a medida para que puedan ser fácilmente distribuidos e instalados en otras instancias de Openbravo. Esta característica proporciona diferentes ventajas para usuarios y desarrolladores:

- Los desarrollos pueden ser reutilizados en diferentes proyectos, aumentando la rentabilidad y disminuyendo el coste (en caso de tener licencia comercial).
- Existe un amplio catálogo de módulos disponibles que permiten extender la funcionalidad del sistema según se necesite. En muchos casos son gratuitos, y en otros suponen un coste razonable en comparación con otros ERPs.
- El desarrollo a medida es más sencillo y fácil de mantener por tratarse de pequeños componentes independientes entre si.
- Los módulos tienen un sistema de versiones que permite la actualización de forma sencilla, y las dependencias entre módulos se gestionan fácilmente.
- El núcleo de Openbravo puede actualizarse cuando aparecen nuevas versiones, sin que la funcionalidad de los módulos se vea afectada.

Todas las modificaciones que se realizan en el sistema tienen que estar asociadas a un módulo. Los módulos pueden distribuirse individualmente o en packs, cuando se trata de varias unidades que trabajan conjuntamente para conseguir un objetivo.

Un módulo puede contener:

- Elementos del diccionario de la aplicación: ventanas, pestañas, menús, etc.
- Recursos de software: archivos java, HTML, Javascript, etc.
- Estructuras de base de datos: tablas, vistas, disparadores, etc.
- Paquetes de datos como árboles de cuentas contables, datos de ejemplo, etc.
- Informes
- Servicios web

También hay un tipo especial de módulo (*industry templates*) que permite hacer modificaciones sobre el núcleo de Openbravo mediante scripts de configuración.

Un desarrollo de Openbravo típico sería crear una tabla, que pudiéramos utilizar para almacenar cierta información, por ejemplo visitas a clientes, desde el entorno de trabajo de Openbravo. Los pasos a seguir serían los siguientes:

1. Entrando en Openbravo con permisos de administrador, crearíamos un módulo llamado *MiCRM*. Le asignaríamos un prefijo para la base de datos "micrm_" e indicaríamos que se encuentra en desarrollo, de esta forma el compilador sabrá que hay que revisar los ficheros fuente para recompilar en caso de haber modificaciones.
2. Crearíamos una tabla en la base de datos con los campos deseados utilizando SQL. Es necesario incluir ciertos campos obligatorios que especifica Openbravo y que tienen relación con la auditoría (fecha de creación, creado por, etc.) y la gestión multiusuario y multiorganización.
3. Hay que indicar las restricciones como claves primarias y claves ajenas. En este caso tendremos una clave ajena a la tabla de clientes. Seguiremos la terminología de Openbravo llamando AD_BPartner_ID a nuestra clave ajena (AD_BPartner es la tabla que almacena los clientes).
4. Desde la administración de Openbravo importamos la tabla de la base de datos. El sistema reconoce los tipos de datos y la relaciones con otras tablas. También se ocupa de crear la clase Visita.java que representará esta tabla.
5. Para poder introducir datos en la tabla es necesario crear una nueva pestaña (tab), que es una sección dentro de las ventanas de Openbravo.

6. La pestaña puede estar alojada en una nueva ventana, o puede ser una pestaña de segundo nivel dentro de la ventana de clientes. De esta manera, cuando estuviéramos en la ficha de un cliente podríamos ver y modificar las visitas que hemos hecho a ese cliente en concreto.
7. Por último, con una tarea de Ant podemos exportar este módulo, e instalarlo fácilmente en el ERP de cualquier cliente añadiendo esta nueva funcionalidad.

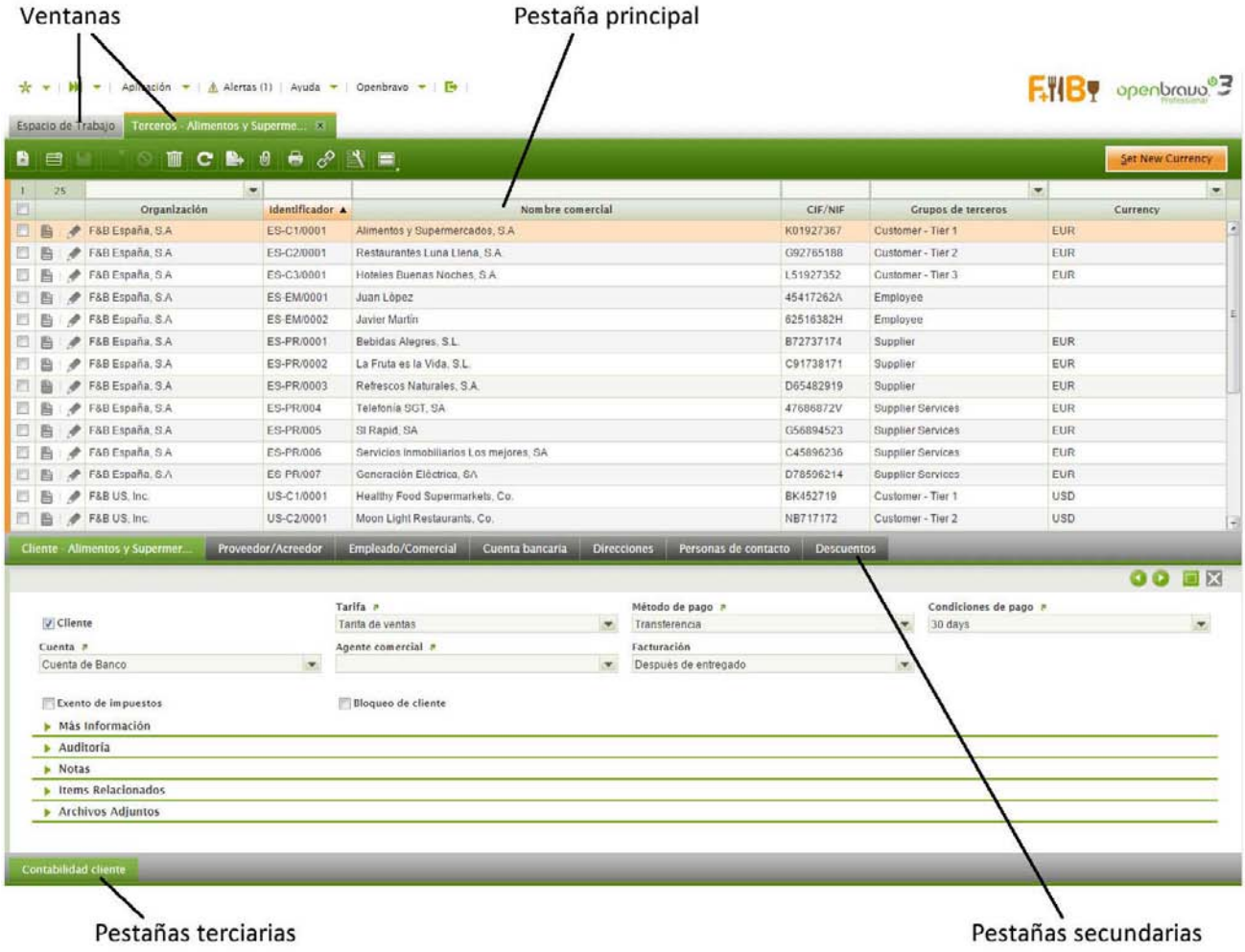

Figura 10. Secciones de la interfaz de Openbravo

La aplicación que se ha desarrollado se ha empaquetado en un sólo módulo que contiene los siguientes elementos:

- Definición de la tabla de galería de imágenes y de la tabla de historial de uso
- Ventanas y pestañas para visualizar y modificar estas tablas
- Archivos de la parte del cliente con el código de la aplicación
- Servicios web para dar soporte a la aplicación

Este módulo tiene dependencias con el módulo Openbravo Mobile que contiene las librerías que dan soporte a aplicaciones para dispositivos móviles. También tiene dependencias con *Openbravo Retail* que es un módulo comercial que proporciona diferentes funcionalidades para el comercio minorista como promociones.
A su vez estos módulos tienen dependencias con el núcleo de Openbravo que es un requisito que hay que definir directa o indirectamente.

5.3. Programación en el lado del cliente: Javascript y SPAs

El enfoque tradicional de la web ha sido una arquitectura cliente - servidor en la que el servidor elaboraba y entregaba los contenidos para que un cliente pasivo los mostrara. Cada vez que se hace clic sobre un enlace, se produce una comunicación que puede durar segundos y mientras se muestra una página en blanco. Este comportamiento a menudo produce frustración en el usuario y abandonos del sitio web. Aunque la experiencia de usuario se ha mejorado bastante desde la aparición de AJAX, las comunicaciones siguen siendo el cuello de botella.

Desde hace algunos años existe una alternativa que mejora significativamente la experiencia de usuario que son las aplicaciones en una sola página. En inglés, *Single Page Applications* (SPAs). Consiste en una aplicación que entrega el servidor para que se ejecute en el cliente. Al encontrarse la lógica de la aplicación en la parte del cliente, muchas interacciones del usuario no requieren comunicación con el servidor, eliminando el tiempo de carga y proporcionando una experiencia fluida.

Existen tres tecnologías para el desarrollo de este tipo de aplicaciones:

Adobe Flash, que es un software comercial muy popular en el pasado para animaciones gráficas pero actualmente incompatible con algunos dispositivos.
Java Applets. Requieren la instalación de Java en el navegador como un plugin y disparan mensajes de seguridad que han evitado que su uso se extendiera.
Javascript (o ECMAScript). Es una estándar abierto ampliamente extendido en todos los navegadores y dispositivos actuales sin necesidad de plugins adicionales. En los últimos años ha pasado de ser una simple herramienta para la manipulación del DOM, a disponer de cientos de librerías para todo tipo de usos, tanto en el lado del cliente como en el del servidor. La mejora en el rendimiento de las máquinas virtuales sobre las que se ejecuta también ha sido un factor decisivo.

Además, respecto a una aplicación instalable, las SPAs tienen la ventaja de no requerir ningún tipo de actualización, puesto que con cada ejecución la aplicación es descargada por completo.

El ERP Openbravo está desarrollado como una SPA, de forma que se accede a su interfaz a través de un navegador web. Esta es una tendencia creciente debido a la mejora de las tecnologías del lado del cliente. Smart Shopping también es una SPA en la que la mayor parte de la lógica se ejecuta en el dispositivo final.

Programación dirigida por eventos

Una de la característica de Javascript que le hace idóneo para las SPAs es su capacidad de adaptarse a la arquitectura dirigida por eventos. En una interfaz de usuario, la secuencia en la que tienen que ocurrir las cosas no está definida, porque depende de la interacción del usuario. Con este tipo de programación, los elementos de la interfaz pueden suscribirse a determinados eventos como clics, movimientos del ratón, entradas de teclado, cambios en variables, recepción de datos, etc. y reaccionar a ellos. Es decir, que programamos las reacciones a un determinado evento, normalmente iniciado por el usuario.

Las ventajas de este tipo de programación son la simplicidad conceptual y la eficiencia, puesto que ofrece muchas posibilidades de paralelización, aprovechando las capacidades de los multiprocesadores actuales.
Las desventajas son que en aplicaciones complejas, puede ser complicado entender el código y prever cual va a ser la reacción a un evento. Las librerías Enyo y Backbone que se han utilizado en este proyecto, pretenden dar respuesta a este problema. En la sección *Arquitectura de la aplicación* se explica como se han combinado estas librerías para establecer una forma de trabajo ordenada.

5.4. Programación con Openbravo Mobile

El ERP Openbravo proporciona un potente *framework* para el desarrollo de aplicaciones móviles conectadas con el ERP. Sobre este *framework* se ha desarrollado la aplicación objeto de este proyecto. A continuación se muestran los diferentes elementos que componen Openbravo Mobile para después describir su papel en el funcionamiento de la aplicación.

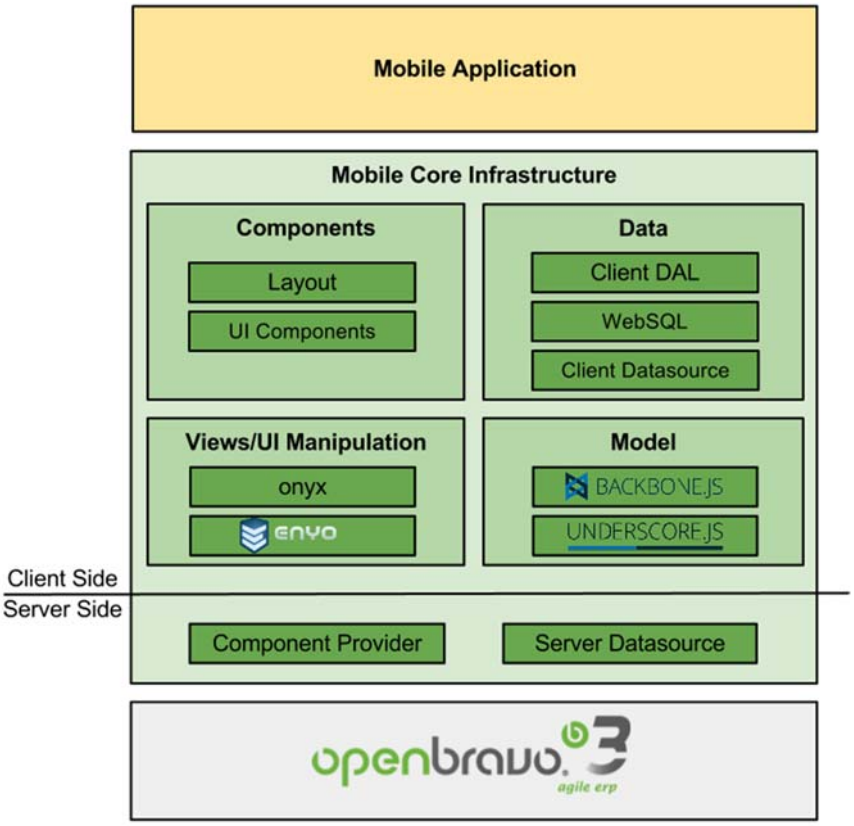

Figura 11. Arquitectura Openbravo Mobile

Layout y UI components. Son elementos prediseñados listos para ser usados como menús, botones, estructuras de página (*layout*), etc. En esta aplicación no se han utilizado porque no se adaptaban a las necesidades del proyecto, con excepción de la pantalla de login.

Client DAL. Proporciona funciones para acceder a los datos almacenados en la base de datos del cliente. Por ejemplo, para recuperar un producto a partir de su id utilizaríamos la siguiente sentencia:

```
OB.Dal.find(      OB.Model.Producto,      // Clase a recuperar
      { product_id: '3000-01' },          // condición
      function(coleccion) { ... },        // se ejecuta en caso de éxito
      function(excepcion) { ... } );      // se ejecuta en caso de error
```

Tiene la peculiaridad de ser asíncrona, es decir, que el programa se seguiría ejecutando y cuando los datos estuvieran listos, se llamaría a la función correspondiente.

WebSQL. Es uno de los métodos definidos por la W3C para almacenar datos en la parte del cliente. Los datos se almacenan en el navegador del cliente en forma de base de datos relacional y pueden ser accedidos utilizando una variante de SQL. El propósito de esto es que sea posible trabajar sin conexión a internet. Una

vez se accede a la aplicación por primera vez y se crea la base de datos local se puede perder la conexión a internet y la aplicación seguiría funcionando con normalidad.

Client Datasource. Son los métodos que llaman a los servicios web del servidor para recuperar los datos. Se ejecutan en el arranque de la aplicación. También se pueden usar para enviar datos al servidor, como en el caso del historial de uso de esta aplicación.

Enyo. Es una librería de Javascript creada por HP para sus dispositivos móviles, y liberada en 2012 como software libre. Estas son sus principales características:

- Es independiente de la plataforma, por lo que puede ejecutarse con resultados casi iguales en dispositivos móviles o de sobremesa, y en los diferentes navegadores actuales.
- Es fácilmente extensible. Los desarrolladores pueden crear nuevos componentes y ponerlos a disposición de los demás.
- Está diseñado para reducir la complejidad a través de la reutilización de código y la encapsulación.
- Es ligero y rápido, para proporcionar una agradable experiencia de usuario.

El concepto principal de la programación en Enyo es definir objetos llamados componentes que se anidan unos dentro de otros (igual que las etiquetas dentro de un documentos HTML). Los componentes pueden pertenecer a una de las clases proporcionadas por Enyo, o a las que el programador defina. Cada uno se renderiza como un elemento HTML y se le aplica un estilo CSS. También pueden definir métodos y variables.

Onyx es un conjunto de clases Enyo básicas con su propia hoja de estilos.

Backbone es otra librería Javascript que proporciona múltiples herramientas para el desarrollo de SPAs. En el entorno de Openbravo Mobile se utiliza para definir modelos de datos. Una de las ventajas es que cuando un atributo cambia de valor, se dispara un evento que cualquier componente puede capturar. Este mecanismo permite que las diferentes partes de la página se actualicen sin necesidad de recargar la página.

Underscore es una librería Javascript que define métodos para trabajar con colecciones de forma asíncrona. Es una dependencia de Backbone.

En el lado del servidor

Component provider es una clase Java que recoge la petición del navegador y envía todos los archivos necesarios al cliente: HTML, CSS, Javascript y cualquier otro archivo necesario. Se pueden definir dependencias de forma que si la aplicación Smart Shopping depende de Openbravo Mobile, cuando se llame a la primera se enviarán todos los archivos de la segunda (por ejemplo, enyo.js y backbone.js).

Server datasource es la manera que tiene Openbravo de acceder a su información. Es la parte que garantiza la seguridad, los permisos y la consistencia de los datos.

Capítulo 6

Implementación y pruebas

6.1. Implementación de la galería de imágenes

6.2. Comunicación con el servidor

6.3. Arquitectura de la aplicación cliente

6.4. Historial de uso y análisis OLAP

6.5. Lectura de códigos de barras

6.6. Pruebas

6.1. Implementación de la galería de imágenes

A continuación se explica la estructura de la base de datos de Openbravo en relación a los productos. Distinguiremos como se almacenan los productos genéricos de las variantes. Por simplicidad, solo se han incluido los campos relevantes.

Productos genéricos
El esquema de las tablas de la base de datos para productos genéricos es el siguiente:

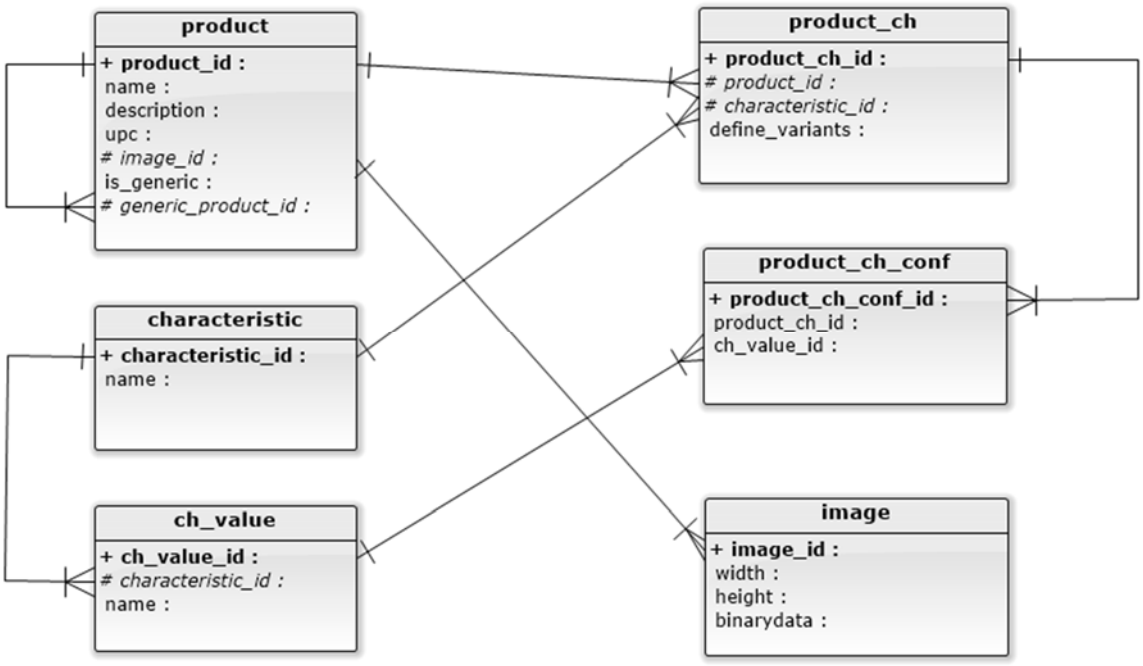

Figura 12. Esquema base de datos productos genéricos

La tabla *Product* contiene los datos principales del producto.

- *image_id* es una clave ajena a la tabla donde se almacenan las imágenes.
- El campo *is_generic* es un booleano que indica si el producto es genérico.
- *generic_product_id* será nulo en el caso de productos genérico

La tabla *characteristic* contiene las características que queremos definir para nuestros productos. En nuestro caso *Talla*, *Color* y *Sexo*.

La tabla *ch_value* contiene los valores posibles para las características. Por ejemplo, para la característica *color* definiremos *azul* y *rojo*.

La tabla *product_ch* es parte de la configuración del producto genérico. Al asignar la característica *Color* a un producto genérico para que se creen variantes, se utiliza esta tabla.

- define_variants es un parámetro de configuración que indica si se tienen que crear diferentes variantes a partir de esta característica o no. Por ejemplo, las características *Color* y *Talla* si que definen variantes en el caso de las camisetas, mientras que la característica *Sexo* no, porque todas las variantes tienen el mismo valor.

La tabla *product_ch_conf* almacena los posibles valores que puede tomar un producto y una característica. Por ejemplo, para el producto *Camiseta* y la característica *Color* los valores posibles son *Azul* y *Rojo*, de entre toda la lista de colores existentes.
Esta tabla no sería estrictamente necesaria tal cual, pero contiene otros campos que permiten diferentes opciones en Openbravo y que no se han incluido porque no se van a utilizar en este caso. Sin embargo, más adelante veremos como se ha utilizado para la galería de imágenes de este proyecto.

Variantes
El esquema de las tablas de la base de datos para las variantes de producto es el siguiente:

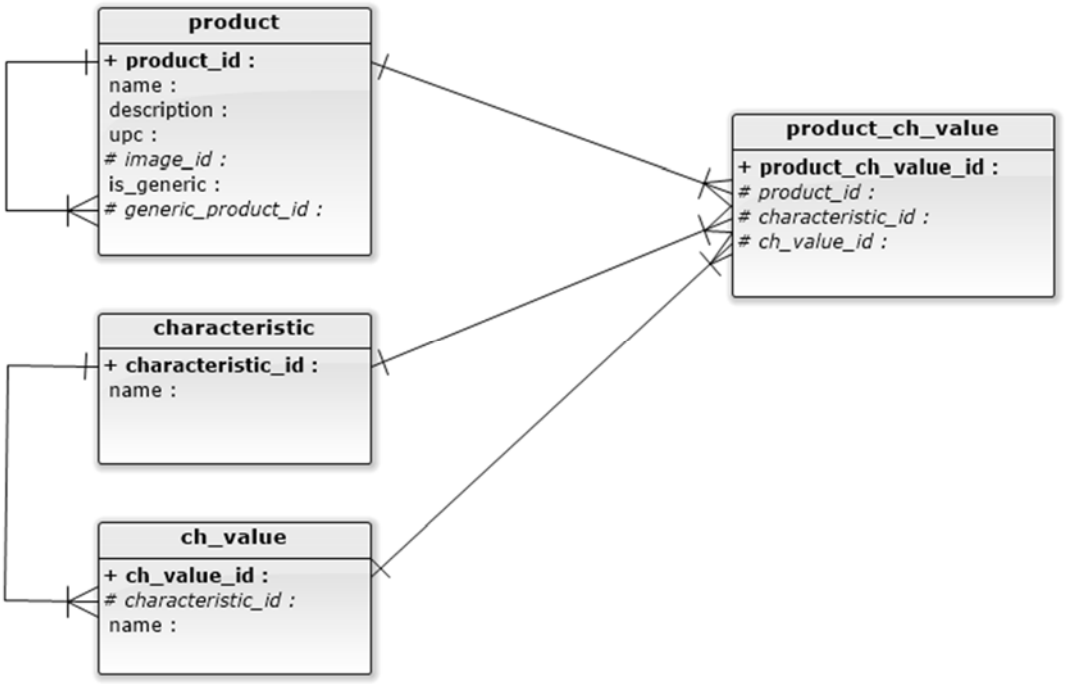

Figura 13. Esquema base de datos variantes de productos

Las tablas *product*, *characteristic* y *ch_value* son las mismas que se han descrito anteriormente. La diferencia con las variantes sería que el campo *is_generic* está marcado como *false* in en *generic_product_id* está el identificador del producto genérico correspondiente.

La tabla *product_ch_value* contiene los valores de cada característica que tiene una variante, es decir, un registro para *Color*, otro para *Talla* y otro para *Sexo*.

A continuación se muestra un ejemplo de como quedaría la base de datos para el ejemplo de las camisetas.

Product

product_id	name	description	upc	image_id	is_generic	generic_product_id	
camiseta_gen	Camiseta	Camiseta manga corta		img_01	true		
camiseta_01	Camiseta	Camiseta manga corta	123456789996		false	camiseta_gen	(Rojo, M, Unisex)
camiseta_02	Camiseta	Camiseta manga corta	123456789997		false	camiseta_gen	(Rojo, L, Unisex)
camiseta_03	Camiseta	Camiseta manga corta	123456789998		false	camiseta_gen	(Azul, M, Unisex)
camiseta_04	Camiseta	Camiseta manga corta	123456789999		false	camiseta_gen	(Azul, L, Unisex)

Characteristic

characteristic_id	name
char_01	Color
char_02	Talla
char_03	Sexo

Ch_value

ch_value_id	characteristic_id	name	
val_01	char_01	Rojo	(Color, Rojo)
val_02	char_01	Azul	(Color, Azul)
val_03	char_01	Blanco	(Color, Blanco)
val_04	char_01	Negro	(Color, Negro)
val_05	char_02	M	(Talla, M)
val_06	char_02	L	(Talla, L)
val_07	char_02	XL	(Talla, XL)
val_08	char_03	Unisex	(Sexo, Unisex)
val_09	char_03	Hombre	(Sexo, Hombre)
val_10	char_03	Mujer	(Sexo, Mujer)

Product_ch

product_ch_id	product_id	characteristic_id	define_variants	
prod_ch_01	camiseta_gen	char_01	true	(Camiseta, Color)
prod_ch_02	camiseta_gen	char_02	true	(Camiseta, Talla)
prod_ch_03	camiseta_gen	char_03	false	(Camiseta, Sexo)

Product_ch_conf

product_ch_conf_id	product_ch_id	ch_value_id	
prod_ch_conf_01	prod_ch_01	val_01	(Camiseta, Color, Azul)
prod_ch_conf_02	prod_ch_01	val_02	(Camiseta, Color, Rojo)
prod_ch_conf_03	prod_ch_02	val_05	(Camiseta, Tallar, M)
prod_ch_conf_04	prod_ch_02	val_06	(Camiseta, Tallar, L)
prod_ch_conf_05	prod_ch_03	val_08	(Camiseta, Sexo, Unisex)

Product_ch_val

product_ch_value_id	product_id	characteristic_id	ch_value_id	
prod_ch_val_01	camiseta_01	char_01	val_01	(Color, Rojo)
prod_ch_val_02	camiseta_01	char_02	val_05	(Talla, M)
prod_ch_val_03	camiseta_01	char_03	val_08	(Sexo, Unisex)
prod_ch_val_04	camiseta_02	char_01	val_01	(Color, Rojo)
prod_ch_val_05	camiseta_02	char_02	val_06	(Talla, L)
prod_ch_val_06	camiseta_02	char_03	val_08	(Sexo, Unisex)
prod_ch_val_07	camiseta_03	char_01	val_02	(Color, Azul)
prod_ch_val_08	camiseta_03	char_02	val_05	(Talla, M)
prod_ch_val_09	camiseta_03	char_03	val_08	(Sexo, Unisex)
prod_ch_val_10	camiseta_04	char_01	val_02	(Color, Azul)
prod_ch_val_11	camiseta_04	char_02	val_06	(Talla, L)
prod_ch_val_12	camiseta_04	char_03	val_08	(Sexo, Unisex)

Figura 14. Ejemplo de registros en la base de datos

Galería de imágenes

Para implementar la galería de imágenes partimos de las siguientes premisas:

- La entidad a la que vamos a asignar imágenes es *producto genérico - color.*
- Una entidad tendrá 1 o muchas imágenes.
- Las imágenes se almacenan en la tabla *image* que queremos utilizar.

Si observamos las figuras anteriores podemos determinar que la tabla que representa la entidad *producto genérico-color* es *product_ch_conf.* Si quisiéramos una sola imagen sería suficiente un campo clave ajena a la tabla *image*, pero como queremos tener un número indefinido, tenemos que crear una tabla intermedia que llamaremos *galería*.

El modelo de la base de datos queda de la siguiente manera:

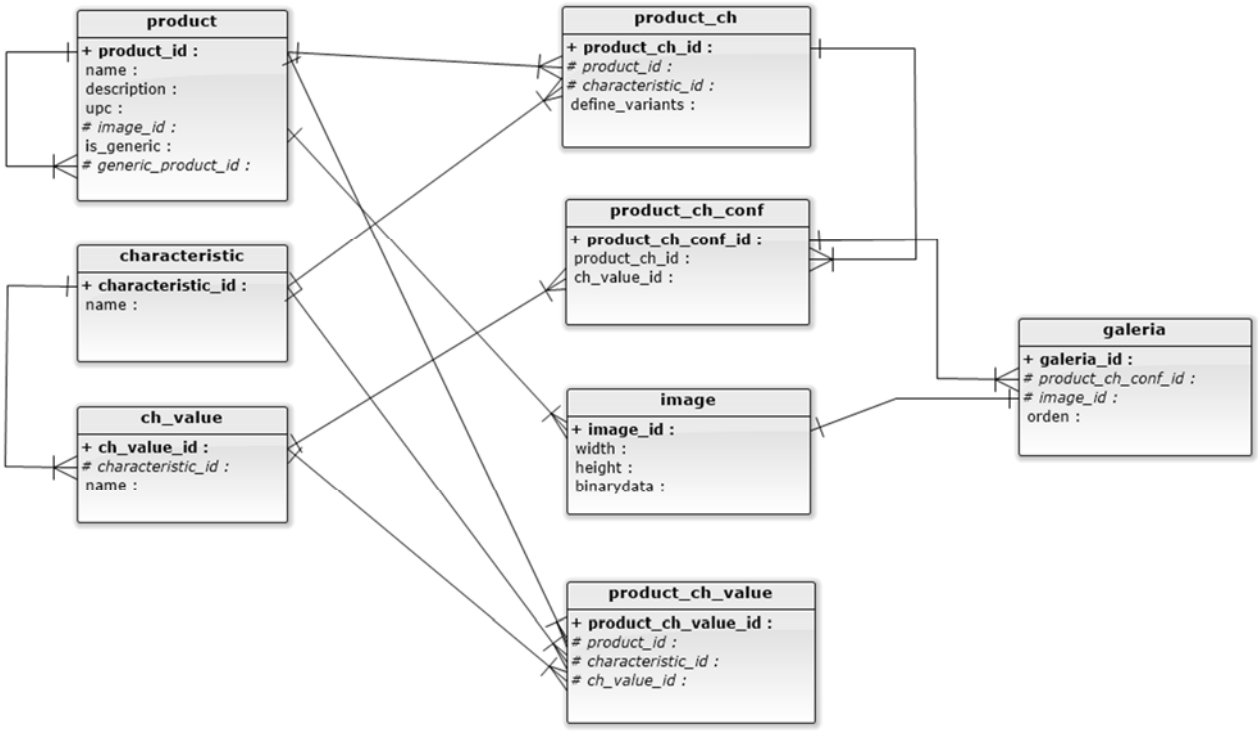

Figura 15. Esquema definitivo de la base de datos de productos

Añadir la tabla al entorno de trabajo de Openbravo

Para poder trabajar con esta tabla, se ha creado una pestaña, que a su vez se ha incluido como pestaña secundaria de la ventana Productos, de forma que cuando se edita un producto genérico tenemos la opción de añadir imágenes a su galería.

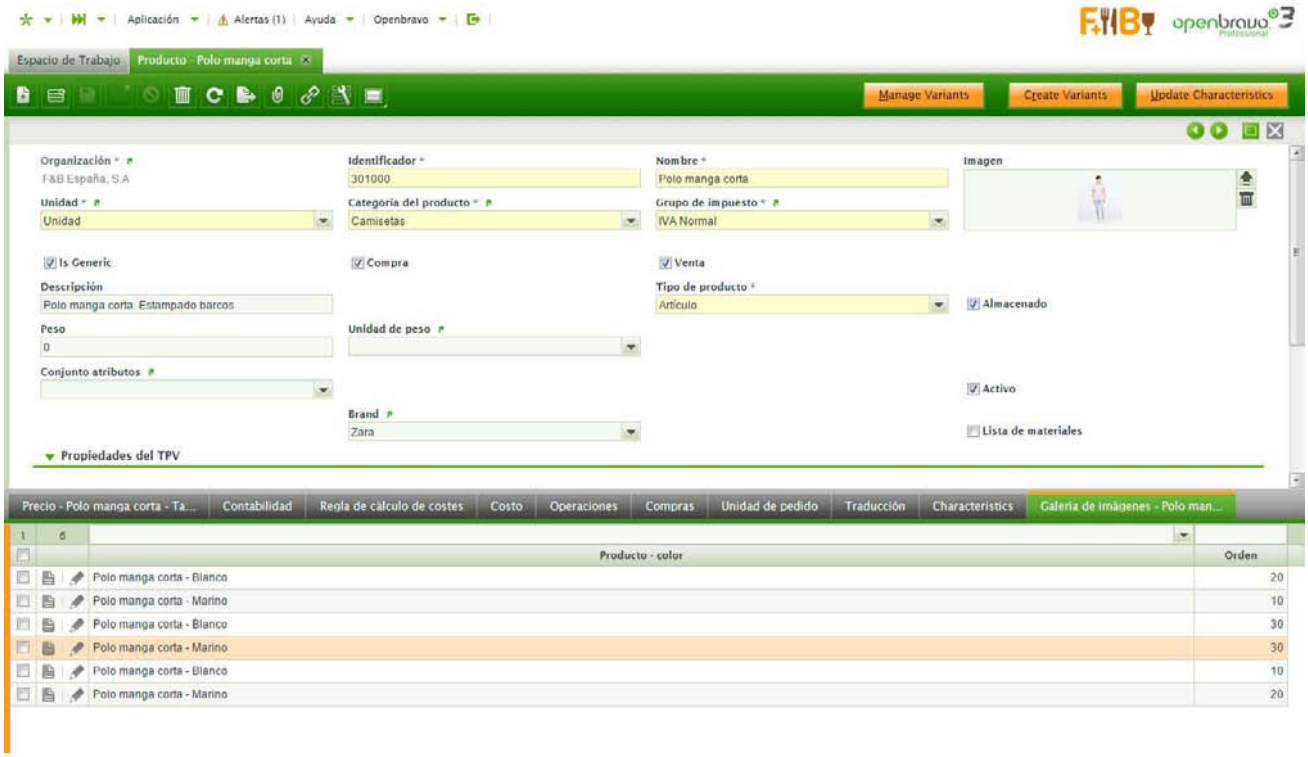

Figura 16. Vista de la galería en modo tabla

Figura 17. Vista de la galería en modo formulario

6.2. Comunicación con el servidor

Autentificación

Aunque la aplicación funcione en el lado del cliente, es necesario establecer una comunicación entre cliente y servidor para intercambiar datos. Esta conexión debe ser autentificada utilizando un usuario de Openbravo. Además, las acciones de lectura y modificación de datos que la aplicación podrá realizar dependerán de los permisos que tenga concedidos el usuario.

Openbravo Mobile gestiona el proceso de autentificación. Cuando la aplicación arranca, se comprueba si hay una sesión activa, y sino lanza la pantalla de inicio de sesión.

Al iniciar sesión con un usuario y contraseña de Openbravo, se establece un código de sesión que se utilizará en todas las comunicaciones, y que se almacena como una cookie. Esta sesión tendrá asociado un contexto que incluye:

- usuario y organización, que quedará reflejado en los registros de la base de datos creados o modificados,
- rol, que determinará que operaciones se pueden realizar y a que tablas se puede acceder

Acceso a datos

Una vez establecida la sesión la aplicación podrá llamar a dos tipos de procesos en el lado del servidor. Estos tipos corresponden a clases abstractas de Java que proporciona Openbravo Mobile para ser extendidas por los desarrolladores de aplicaciones móviles.

ProcessHQLQuery
Sirve para ejecutar una consulta sobre la base de datos en el servidor, y devolver el resultado a la aplicación como un objeto JSON. Las consultas en Openbravo se definen en *Hibernate Query Language* (HQL), que es un lenguaje similar a SQL pero orientado a objetos. *Hibernate* es lo que se conoce como un *Object Relational Mapping* (ORM). Es una capa intermedia de software que se encarga de la transformación entre los objetos de Java y las bases de datos relacionales.

Los procesos de tipo *ProcessHQLQuery* que se han definido para esta aplicación son los que se encargan de la carga de datos de categorías, productos, imágenes de la galería y promociones. Todos se ellos se llaman al inicio de la aplicación y con los datos obtenidos se crea la base de datos en el cliente.

JSONProcessSimple
Sirve para ejecutar cualquier acción, puede ser consulta o no, en el lado del servidor. En esta aplicación se ha utilizado para almacenar el historial en la base de datos. La aplicación envía un objeto JSON y el proceso lo recibe, lo transforma en entidades de Openbravo y lo almacena en la base de datos utilizando *Hibernate*.

La aplicación puede saber si el proceso ha funcionado correctamente o no por la respuesta que recibe. El formato de la respuesta también es un objeto JSON con el atributo *Status* que toma el valor 0 si el proceso a funcionado correctamente o -1 si no ha sido así.

6.3. Arquitectura de la aplicación cliente

En este capítulo se explicará como se ha estructurado la aplicación tomando como ejemplo la página de selección de categorías. Hay dos partes claramente diferenciadas: la vista y el modelo. La parte del controlador que faltaría para que se tratara de un patrón MVC corriente, como veremos está dividida entre la lógica de la vista y la lógica del modelo.

Vista

La capa de presentación de la aplicación se ha desarrollado con Enyo, que gestiona la creación y manipulación de las etiquetas HTML.

En primer lugar es necesario determinar los elementos que componen la página, y las relaciones que tienen entre ellos. En el esquema *wireframe* podemos ver claramente el anidamiento.

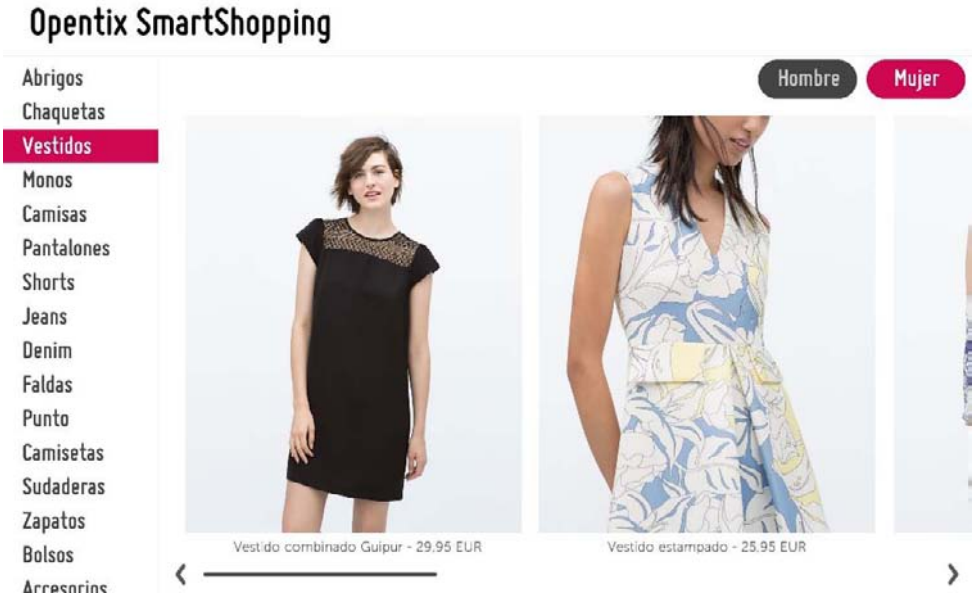

Figura 18. Diseño de la sección Categorías

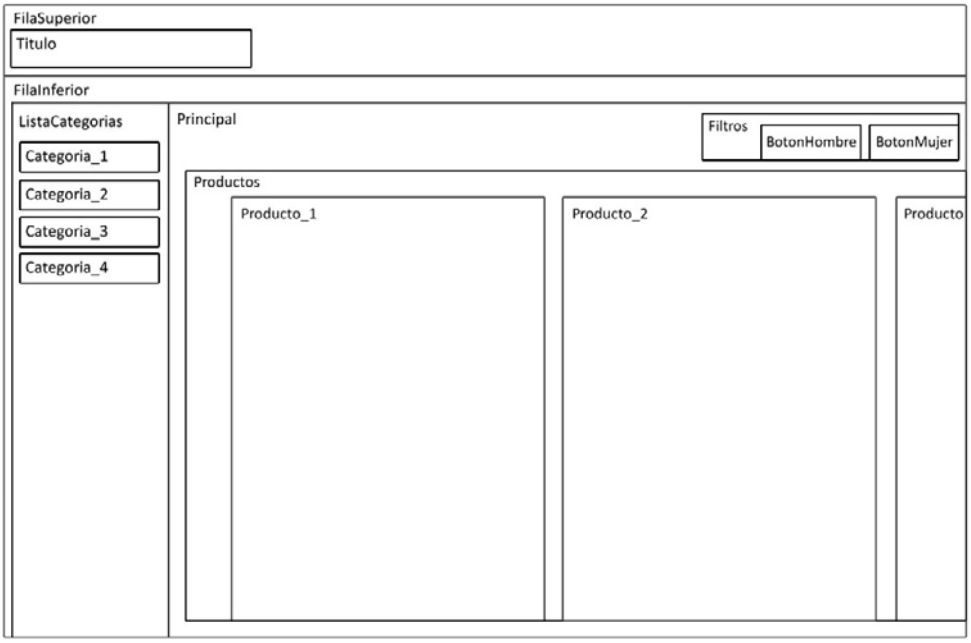

Figura 19. Esquema wireframe de la Sección Categorías

Cada uno de estos elementos corresponderá a un componente Enyo. Entre las clases predefinidas que proporciona esta librería podemos encontrar la clase *Button* que proporciona funcionalidad y aspecto de botón, y que hemos utilizado en los botones de selección de categoría y de sexo. El componente *Productos* será de la clase *Scroller* que la convertirá en un panel deslizante. La clase *Repeater* permite crear elementos similares a partir de una colección de datos, lo cual es útil para *ListaCategorias*.

Modelo

El modelo de la aplicación es un objeto de tipo Backbone, en el que están definidos atributos que representan de estado de la aplicación. Una de las ventajas del modelo Backbone es que genera eventos cuando alguno de sus atributos es modificado. Los componentes Enyo pueden suscribirse a estos eventos, es decir, ser notificados cuando se produce un cambio en la aplicación.

En la figura 20 se presenta un esquema simplificado de los componentes que forman la sección Categorías.

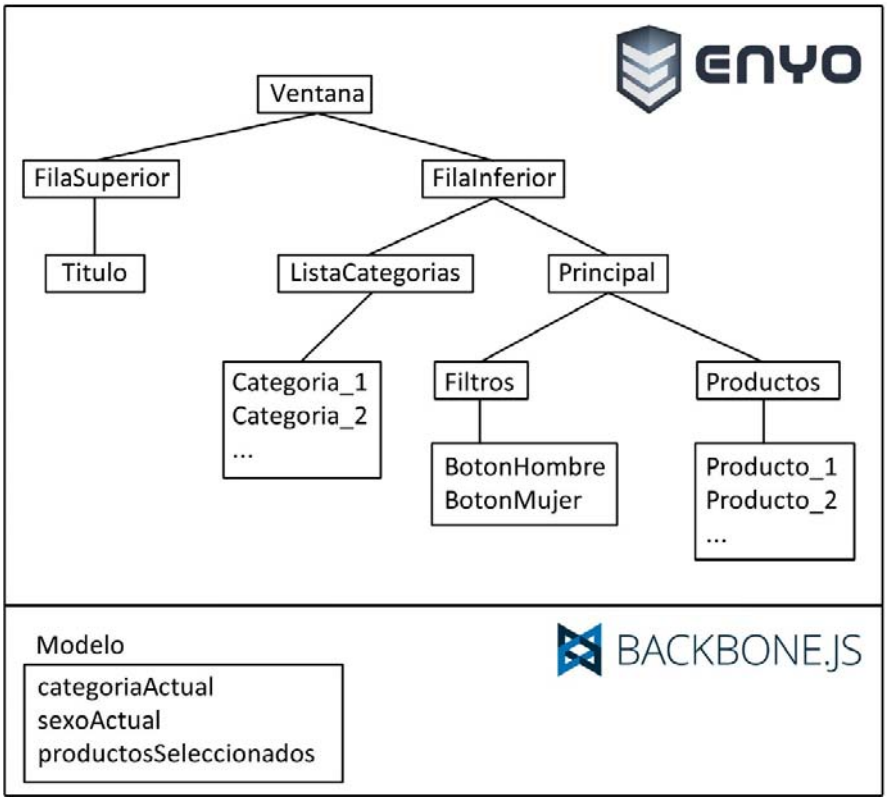

Figura 20. Esquema de componentes de la aplicación cliente

Por ejemplo, en la sección de categorías, un cambio de categoría tendría la siguiente secuencia:
1 - el usuario hace clic en el botón *Camisas*
2 - el botón cambia el atributo *categoriaActual* del modelo con el valor *Camisas*
3 - el cambio de *categoriaActual* dispara el método del modelo *cargarProductosSeleccionados* que recupera de la base de datos local los productos que cumplen con el criterio de categoría y sexo seleccionados y los almacena en el atributo *productosSeleccionados* del modelo.
4 - el cambio en *productosSeleccionados* dispara un método del componente *Principal* que redibuja la parte de la vista que tiene que cambiar para mostrar los nuevos productos

En resumen, tenemos por una parte el modelo, cuyos atributos describen el estado actual de la aplicación y tiene métodos para recuperar datos cuando el usuario lo requiere. Por otra parte están los componentes de Enyo que forman la interfaz gráfica y reaccionan a los cambios en el modelo.

6.4. Historial de uso y análisis OLAP

Una de las partes fundamentales de la aplicación Smart Shopping es la capacidad para registrar todas las operaciones que se desarrollan en ella para que puedan ser analizadas posteriormente y ayuden a entender mejor el comportamiento y los gustos del consumidor.

El funcionamiento es sencillo. Como se ha visto en el apartado anterior, el estado de la aplicación está almacenado en los atributos del modelo. Cada vez que el usuario realiza alguna acción en la interfaz, el estado se almacena en una colección. Cuando la colección de estados alcanza cierto tamaño se envía al ERP para ser almacenada en una tabla de la base de datos.

A continuación se explica la construcción el cubo de información que nos permitirá realizar análisis OLAP.

Dimensiones

Las dimensiones que intervienen en el historial de uso son las siguientes:

Producto. Es el artículo del catálogo que el usuario visita. Cuando el usuario está en la sección Categorías el producto es nulo. La aplicación almacena el identificador del producto, y el resto de atributos se extraen de la tabla de productos del ERP.

Sesión. Una sesión es un período de tiempo continuo durante el cual un único usuario está utilizando la aplicación. Las sesiones nos permiten distinguir a unos usuarios de otros. La sesión se inicia cuando el usuario toca la pantalla o escanea un código de barras por primera vez, y termina cuando presiona el botón de *Salir* o tras un período de inactividad configurable (30-60 segundos). Si el usuario escanea su tarjeta de cliente durante la sesión, todos los movimientos de esa sesión se asociarán a ese cliente.

Sección. Es la sección de la aplicación que se está visitando *Categorías o Producto*. En los dos casos hay una categoría activa que hay que tener en cuenta.

Tienda. Es el establecimiento donde está instalada la terminal Smart Shopping.

Usuario. Puede ser un cliente registrado que utiliza su tarjeta de cliente para acceder, o un usuario anónimo.

Medidas

Las medidas que vamos a considerar son las siguientes:

Número de visitas y duración de la visita. Llamamos visita a la visualización de una página de producto o de categoría. El número de visitas es el número de líneas del historial.
Precio original, descuento y precio final. Los precios que se van a utilizar en el análisis son los que provienen de la aplicación, puesto que es el precio indicado en el momento de uso.

Paquete de información

En la figura 21 se muestra como queda el paquete de información, que ilustra los datos que se van a almacenar.

Historial de uso Smart Shopping				
Producto	**Sesión**	**Sección**	**Tienda**	**Usuario**
Nombre Color Talla Sexo Categoría	Fecha Hora	Sección Categoría	Nombre Localidad Provincia	Sexo Edad Localidad Provincia
Medidas: Duración de la visita/sesión, nº de visitas/sesiones, precio original, descuento, precio final				

Figura 21. Paquete de información Historial de uso

Diseño del cubo

Teniendo en cuenta las relaciones entre los datos, podemos construir el diagrama en estrella. El resultado se muestra en la figura 22.

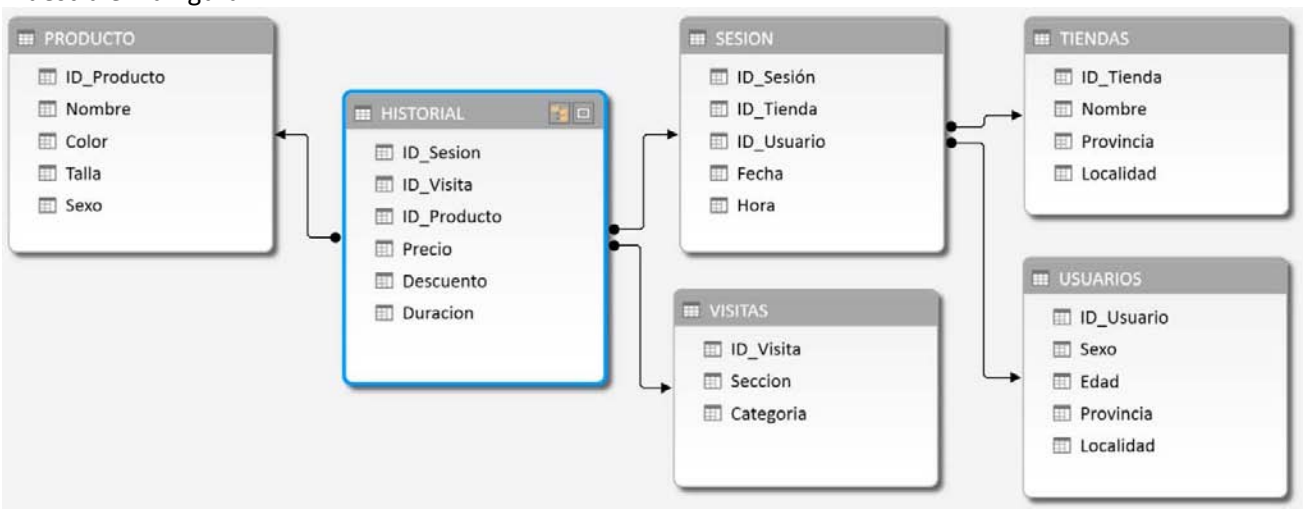

Figura 22. Diagrama en estrella

Por simplicidad y rendimiento, podemos añadir los atributos de la dimensión Visitas a la tabla Historial. El resultado final se muestra en la figura 23.

Figura 23. Diagrama en copo de nieve

Ejemplos de uso

Para ilustrar las oportunidades de negocio que generaría este sistema se han creado unos datos de muestra, sobre el que se han aplicado consultas OLAP. Aunque se ha utilizado Microsoft Excel para este ejemplo, Openbravo tiene un módulo de *Business Intelligence* que proporciona resultados similares.

Algunas de las preguntas que podríamos responder con esta información son estas:

- ¿Qué productos son más populares en cada provincia y por cada sexo?

Visitas por producto	Castellón	Valencia	Total general
Hombre	**155**	**120**	**275**
Pantalón chino con cinturón	29	25	54
Polo piqué	26	18	44
Pantalón denim slim	20	24	44
Polo manga corta	21	19	40
Náutico piel piso contraste	21	4	25
Gafas de sol de pasta	11	11	22
Sombrero paja cinta	10	8	18
Polo estampado	6	9	15
Bandolera doble cremallera	11	2	13
Mujer	**150**	**144**	**294**
Pantalón pitillo cremallera bajo	24	25	49
Sandalia tacón cadenas	14	26	40
Alpargata piel punta	23	16	39
Camiseta texto lentejuelas	20	19	39
Camiseta texto strass	21	11	32
Pantalón estampado	12	12	24
Camiseta estampado chica	11	12	23
Shopper tejido	9	9	18
Shopper print	8	9	17
Saca piel	8	5	13
Unisex	**18**	**19**	**37**
Deportivo clásico detalle	18	19	37
Total general	**323**	**283**	**606**

Figura 24. Ejemplo BI - Productos más populares por provincia y sexo

- ¿Qué volumen de mujeres y hombre pasa por cada tienda?

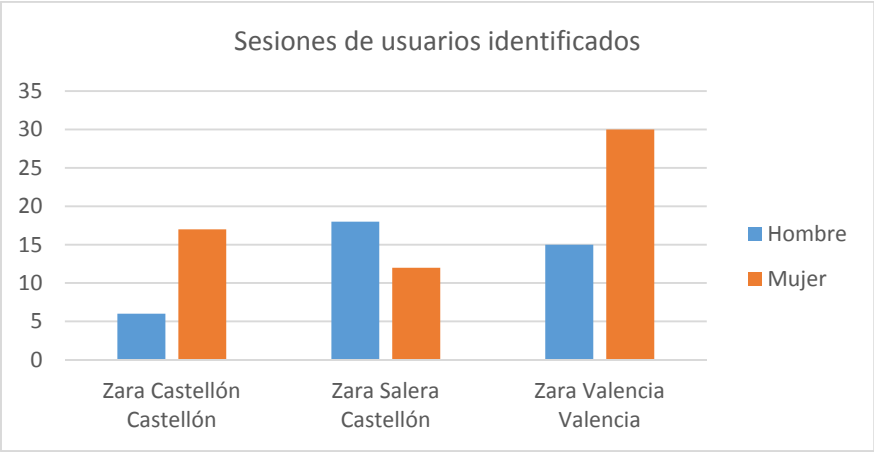

Figura 25. Ejemplo BI - Sesiones por sexo

- ¿Qué rango de edad se interesa más por los complementos?

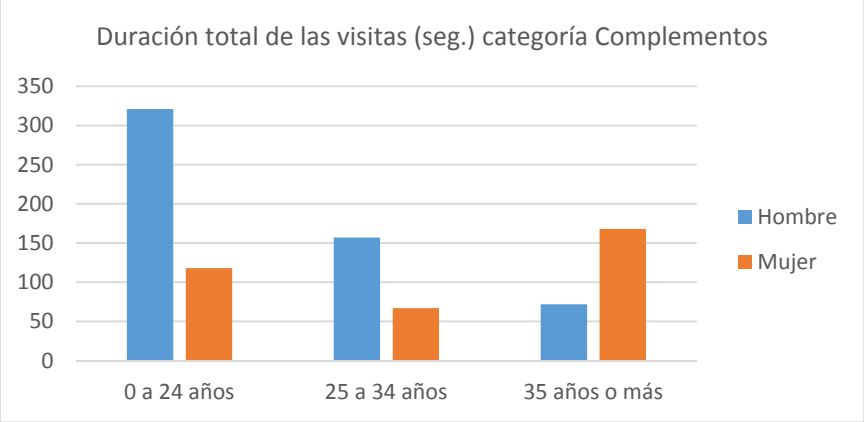

Figura 26. Ejemplo BI - Tiempo de visita categoría complementos

- ¿A qué hora se producen más visitas?

Figura 27. Ejemplo BI - Sesiones por hora del día

- ¿Qué rangos de precio son más populares en cada rango de edad?

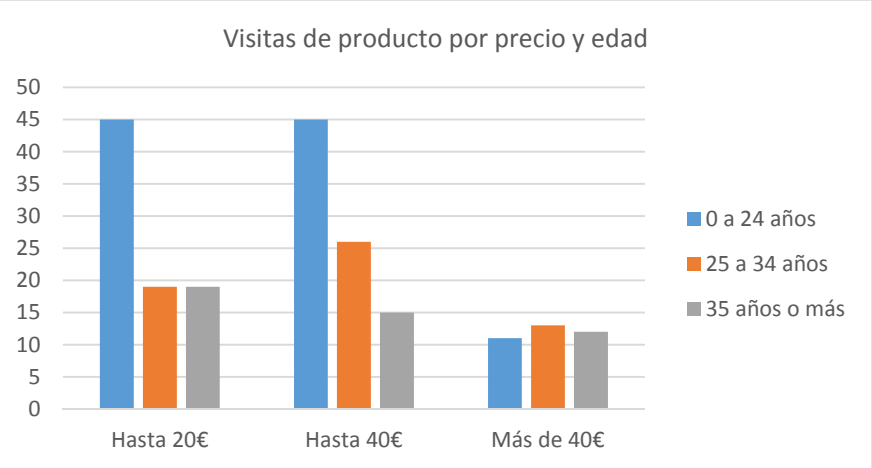

Figura 28. Ejemplo BI - Visitas por precio y edad

6.5. Lectura de códigos de barras

Una de las posibilidades que ofrece la aplicación Smart Shopping es que el cliente pueda escanear el código de barras del producto que ha escogido. De este modo puede ver las fotografías del producto, comprobar el precio y la disponibilidad de otras tallas y colores y ver otros artículos relacionados.

Figura 29. Ejemplo BI - Lector de código de barras

Los lectores de códigos de barras son dispositivos de entrada que convierten una imagen en un código numérico. Para la aplicación el funcionamiento es el mismo que el de un teclado. La manera de implementarlo es colocar un campo de formulario fuera de la página, para que no sea visible. Este campo siempre tiene que tener el foco siempre sobre él. El foco es una característica de HTML que indica que elemento está activo en cada momento. Cuando hay varios campos de formulario podemos cambiar el foco utilizando el tabulador o haciendo clic sobre uno de los campos. Mediante Javascript podemos hacer que el foco siempre este posicionado sobre el campo de entrada, y que reaccione cuando se ha introducido un código.

Cuando se ha introducido un código de barras correspondiente a un producto, la aplicación muestra toda la información de ese producto.

Identificación de usuarios

Otra función clave es la identificación de usuarios. Los clientes de una tienda a menudo tienen la posibilidad de obtener una tarjeta de cliente a cambio de proporcionar cierta información personal. Esta tarjeta suele dar ciertas ventajas como descuentos, suscripción a boletines de novedades, tiquets de aparcamiento, etc. Desde el punto de vista de la empresa, se trata de una información muy valiosa porque permite asociar ventas con datos demográficos. Aplicando técnicas de *data mining* se puede dividir el conjunto de clientes en segmentos más o menos uniformes, lo que permite una mayor adaptación al cliente y en consecuencia, aumento de ventas.

La aplicación Smart Shopping también permite que el usuario se identifique, de forma que los registros de uso puede vincularse con esos datos demográficos y proporcionar una valiosa herramienta para la toma de decisiones.

La forma que se propone es que las tarjetas de cliente tengan un código de barras identificativo, que pueda ser leído por el lector, pero podría adaptarse fácilmente para utilizar códigos QR, NFC, bandas magnéticas, etc.

6.6. Pruebas

Para comprobar el funcionamiento de la aplicación fue necesaria la creación de un conjunto de datos de ejemplo que consta de 20 productos distribuidos en cuatro categorías, con un total de 53 variaciones de tallas y colores, y 79 imágenes.

Con este paquete de datos se cubren los casos más frecuentes de configuración de producto. Permite testear la usabilidad del alta de productos en el ERP y la usabilidad de la aplicación. También es lo suficientemente grande como para comprobar el rendimiento de la aplicación en dispositivos móviles.

También se diseño una prueba consitente en una secuencia de operaciones sobre la aplicación cliente para comprobar el correcto funcionamiento de todos los elementos. Esta prueba se iba completando a medida que se iba avanzando en el desarrollo de la aplicación y se ejecutaba cada vez que se introducía una funcionalidad nueva. Para comprobar los aspectos de usabilidad y rendimiento también era necesario realizar la prueba en un dispositivo móvil tipo tablet de 10 pulgadas, similar al que se usaría en un entorno real.

Estos son los pasos de la prueba:

1. Iniciar la aplicación en el navegador desde una ventana en modo incógnito (de este modo nos aseguramos que la caché está vacía).
2. La aplicación nos pide usuario y contraseña de Openbravo.
3. Se inicia la aplicación en la sección Categorías. Seleccionamos varias categorías comprobando que los productos son los correctos y que el scroll funciona correctamente.
4. Seleccionamos un producto al azar, que nos transporta a la sección correspondiente.
5. Se muestran los colores y las tallas disponibles de ese modelo. En la categoría camisetas se tiene que mostrar un descuento del 20%.
6. Comprobamos el scroll y los botones de talla y color (las imágenes son diferentes para cada color).
7. Comprobamos los botones de producto siguiente y anterior.
8. Pulsamos en uno de los productos relacionados, que nos conduce a la pantalla de ese producto.
9. Pulsamos el botón atrás, que nos conduce de vuelta a la sección Categorías.
10. Pulsamos el filtro de Hombre o Mujer.
11. Volvemos al paso 3, comprobando en cada paso que sólo se muestran los productos del género seleccionado.
12. Escribimos varios códigos de producto. Si corresponde a un producto la aplicación muestra en pantalla ese producto, coincidiendo en talla y color.
13. Introducimos un número de cliente y hacemos algunos cambios de categoría y de producto.
14. Esperamos 30 segundos a que se cierre la sesión automáticamente.
15. En el ERP, comprobamos que el historial se ha guardado correctamente.

Capítulo 7

Desarrollos adicionales

7. Desarrollos adicionales

Las etapas del proceso de desarrollo de este y de cualquier otro proyecto de software no acostumbran a desarrollarse de forma lineal, sino que son un proceso iterativo en el que se van observando posibilidades de mejora que requieren revisar las etapas anteriores. Debido a las limitaciones de tiempo algunas de estas mejoras no se han podido materializar. A continuación se exponen las más importantes con el fin de servir de referencia futura en caso de retomarse el proyecto.

Solicitar ayuda al dependiente. Este requisito se fijó en el inicio del proyecto y pero solo se ha realizado una simulación básica de su funcionamiento. La idea es que el dispositivo se utilice en los probadores de las tiendas, y que el cliente pueda solicitar ayuda al personal de la tienda. Por ejemplo, podría pedir una talla o color diferente a la que se está probando o alguna prenda que combine. Desde la aplicación se podría mandar un mensaje con el producto solicitado que sería recibido como una alerta en el terminal de caja o en terminales móviles que llevarían los asistentes de tienda.

Compras. Con el objetivo de agilizar el proceso de compra, el terminal Smart Shopping podría permitir realizar el pago de los productos adquiridos. Hay diferentes aspectos que habría que tener en cuenta para implementar este proceso: el medio de pago, los dispositivos anti-robo de las prendas, el doblado de las prendas y la dispensación de bolsas, o la acogida que tendría este sistema por parte de directivos y empleados de las tiendas y por supuesto de los clientes. En este último aspecto, cabe recordar que las ventas online se han convertido en algo cotidiano para mucha gente, y su crecimiento es constante en los últimos años.

Tanto si se tratara de un sistema totalmente automático como si necesitara intervención humana, desde el punto de vista del negocio, podría suponer una oportunidad de diferenciación y mejora en la atención al cliente. Por una parte, el personal de tienda pasaría menos tiempo realizando cobros que podría dedicar a asistir a los clientes en los probadores proporcionando una experiencia de compra mucho más personalizada y satisfactoria.

Capítulo 8

Conclusiones

8. Conclusiones

A lo largo de este documento se ha expuesto el desarrollo de un proyecto de software desde su concepción hasta su implementación. Para garantizar el éxito del proyecto y obtener un producto de calidad, se han aplicado técnicas de gestión de proyectos de software adaptadas al contexto y a la complejidad de la aplicación, buscando un equilibrio entre lo ágil y lo riguroso.

En relación al desarrollo de software, ha sido una parte importante del proyecto el estudio de las tecnologías sobre las que se debía construir la aplicación, y que venían impuestas por la empresa. Cabe destacar que tanto el ERP Openbravo, como el ecosistema Java y las SPAs Javascript son tecnologías pujantes que abarcan buena parte del panorama actual del desarrollo de software. Se han aplicado las prácticas comúnmente aceptadas como el diseño modular que permiten obtener un producto robusto, fácil de mantener y de extender. También se ha tenido en cuenta el rendimiento en el almacenamiento de datos, las comunicaciones y la ejecución de la aplicación.

En todo momento se ha puesto el foco en el proceso de negocio que se deseaba implementar, teniendo en cuenta tanto al negocio al que está destinado la aplicación, como al usuario final. Se ha justificado la necesidad de este proyecto como un sistema innovador que redistribuye los recursos para ofrecer un valor añadido al consumidor, y que permite generar información valiosa para la toma de decisiones. En definitiva, que puede suponer una inversión rentable si se implanta correctamente.

Desde el punto de vista del autor, se han puesto en práctica los conocimientos aprendidos a lo largo del grado, y se ha adquirido una visión global y más profunda de los sistemas de información en la empresa, que se componen de multitud de aspectos técnicos pero también económicos y de marketing.

Como resultado se ha obtenido una aplicación que, aunque no está completamente terminada, si es funcional y permite demostrar su potencial a posibles clientes.

Capítulo 9

Bibliografía

9. Bibliografía

1. Nielsen J., Loranger H. - *Prioritizing Web Usability*
2. Li N., Yang Y., Yan X. - *Cross-Selling Optimization for Customized Promotion*
3. R. Agrawal, T. Imielinski, A. Swami - *Mining association rules between sets of items in large databases*
4. Tan, Steinbach, Kumar - *Introduction to Data Mining*
5. Mikowski M., Powell J. - *Single Page Web Applications*
6. Sutton, R. - Enyo: *Up and Running*

Webs y blogs de referencia

Openbravo

Desarrollo en Openbravo	http://wiki.openbravo.com/wiki/Development_Model
Configuración de la pila de desarrollo	http://wiki.openbravo.com/wiki/Development_Stack_Setup
Modularidad en Openbravo	http://wiki.openbravo.com/wiki/Modularity

Pila de desarrollo

PostgreSQL	http://www.postgresql.org
Oracle	http://www.oracle.com/es
Java	https://es.wikipedia.org/wiki/Java_(lenguaje_de_programación)
Apache Ant	http://ant.apache.org
Mercurial	https://mercurial.selenic.com

Openbravo Mobile

Arquitectura	http://wiki.openbravo.com/wiki/Mobile_Architecture_Overview
Guía de desarrollo móvil	http://wiki.openbravo.com/wiki/Category:Mobile_Developers_Guide
Iván Perdomo blog	https://katratxo.wordpress.com
WebSQL	http://www.w3.org/TR/webdatabase
Comunicación con el servidor	http://wiki.openbravo.com/wiki/Mobile_server_side_query_components