Facultad de Ciencias

Desarrollo de una aplicación móvil de comercio electrónico para un bazar
(Development of an e-commerce mobile app for an electronics shop)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Javier de las Heras Zamora

Directora: Patricia López Martínez

Julio – 2017

Resumen

Gracias a la evolución de internet y de la tecnología es posible realizar operaciones desde cualquier lugar del mundo y a cualquier hora. Esto, que hace años era impensable, ha provocado una evolución muy importante para el comercio, con la aparición del comercio electrónico.

Son muchas las empresas que se están adaptando a este nuevo modelo de negocio, siendo cada vez más difícil encontrar negocios o comercios que no dispongan de tienda online. La esencia de este nuevo modelo de negocio viene de la mano de los smartphones. Y es que hoy en día la mayoría de la población (en los países desarrollados) accede a internet a través de estos dispositivos. Por ello ha crecido enormemente esta estrategia de comercio electrónico y qué mejor forma de aprovecharla por parte de las empresas que crear sus propias aplicaciones móviles para la venta de sus productos.

Nace aquí la idea origen de este Trabajo Fin de Grado, la de crear una aplicación móvil para la empresa Bazar Canarias que permita principalmente la compra de productos, además de consultar el catálogo y calcular la ruta hacia sus tiendas. De esta manera cualquier persona con un dispositivo Smartphone y acceso a internet podrá hacer uso de los servicios de esta aplicación móvil.

La empresa dispone de una página web ya implementada, por lo que como aspecto básico del trabajo se diseñará e implementará un servicio REST que permita obtener información de la base de datos ya existente en el servidor y así mantener ambas aplicaciones (web y móvil) sincronizadas en todo momento.

Palabras clave: Aplicación móvil, comercio electrónico, Smartphone, servicios REST.

Abstract

Thanks to the evolution of the Internet and technology, it is possible to carry out operations from anywhere in the world and at any time. This is something that was unimaginable years ago and that has had a massive impact on the way of doing business, with the emergence of electronic commerce.

Most companies are trying to adapt to this new trend in business, being very difficult to find businesses that do not have an online store nowadays. The essence of this new business model is coming from the hand of the smartphones. Indeed, today a major part of the world's population (at least in the developed countries) use internet through these devices. Thus, the online business sector is growing exponentially and the best way for the companies to benefit from this growing industry is to create their own mobile Apps for the sale of their products.

This is how the idea of this work started. Creating a mobile application (App) for the company Bazar Canarias, to allow online purchase of products as a main feature, in addition to the possibility to consult its catalogue and calculate the shortest way to get to the physical shop. Any customer with a Smartphone and access to internet will be able to use the services of this App.

The firm already has an established website, so as a basic aspect of the work will be to design and implement a REST service to obtain information from the database already existing on the server and thus keep both applications (web and mobile) synchronized at all times.

Keywords: Mobile App, e-commerce, Smartphone, REST services.

Agradecimientos

Este Trabajo Fin de Grado cierra una etapa muy especial de mi vida. Ha sido un periodo de trabajo y de esfuerzo muy intenso, sobre todo por el poco tiempo disponible para realizar este proyecto, pero que me ha aportado muchas cosas positivas. Es por ello por lo que quiero agradecer a todas las personas que han estado a mi lado, en mayor o menor medida, su ayuda y apoyo.

En especial a mis padres y a mi familia, por darme la oportunidad de estudiar en la universidad y de cursar un año en el extranjero. Habéis conseguido inculcarme unos valores que me han ayudado a orientar mi futuro profesional, pero sobre todo por saber aconsejarme y apoyarme siempre.

A todos los compañeros y amigos con los que he compartido momentos, por comprenderme, ayudarme y sobre todo aguantarme durante todos estos años.

A todos los profesores que he tenido, por todo lo que me han enseñado en este tiempo y su disposición para ayudarme siempre. Sin todos ellos nada de esto hubiera sido posible.

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

Vivimos en una sociedad en la que la tecnología cada vez está tomando más protagonismo. Esto se traduce en grandes avances, como son el caso de los Smartphones y de internet, que van acompañados el uno del otro. Podemos darnos cuenta como cada vez los teléfonos móviles son más potentes y más completos, se están convirtiendo en una herramienta indispensable para el ser humano y esto se debe en parte también al avance de internet y las comunicaciones, que luchan por el objetivo de permanecer siempre conectados.

A raíz de estos avances tecnológicos surge una nueva idea de mercado denominada comercio electrónico, también conocido como *e-commerce*, que radica en la compraventa de productos o servicios a través de internet. Las empresas pioneras en el comercio electrónico comenzaron creando sus páginas web de venta de artículos, pero al igual que la tecnología, han ido evolucionando de forma exponencial. Según datos publicados por la Comisión Nacional de los Mercados y la Competencia (CNMC) respecto al año 2015: "El comercio electrónico movió en España la cifra récord de 5.414,1 millones de euros en el primer trimestre del año, lo que supone un incremento del 21,5 % respecto al mismo periodo de 2015, en el que creció el 24,5 %" [1].

Cuando comenzó la idea del comercio electrónico la totalidad de las personas accedían a ello mediante ordenadores, pero poco a poco esto ha ido cambiando y ahora son los Smartphones los dispositivos que prefiere la gente para conectarse a internet. Una de las principales razones para que se produzca este cambio es la posibilidad de acceder a internet en cualquier momento y desde casi cualquier lugar de mundo. Otra razón de peso es la comodidad y tamaño que ofrecen estos dispositivos a diferencia de los ordenadores. Es por ello por lo que nace la idea del desarrollo de aplicaciones móviles que permitan el comercio electrónico.

Dentro de este marco, este proyecto pretende desarrollar una aplicación móvil para dar una mayor facilidad al usuario final en la compra de productos de la empresa Bazar Canarias, fácil acceso al catálogo de productos y geolocalización de sus tiendas.

El proyecto no comienza de cero porque ya existe una página web con su correspondiente servidor y base de datos implementados. Con la creación de esta nueva aplicación móvil se pretende unificar y anexar todo el trabajo de manera que se intentará reutilizar al máximo el trabajo ya realizado en la medida de lo posible. Esto se logrará con la ayuda de un servicio web, que se diseñará de manera que sea el que mejor se adapte a las necesidades del proyecto.

1.1 Motivación y contexto

El proyecto comenzó con una idea clara, que es el crecimiento de la empresa gracias al desarrollo de una aplicación móvil. Con una aplicación de este tipo logramos que todos los clientes puedan acceder a comprar productos en cualquier momento y desde cualquier lugar. Además, podrán observar todo el catálogo actualizado desde la aplicación y calcular la ruta hasta la tienda más cercana.

La empresa ya cuenta con una página web para la compra de productos, a la cual se puede acceder desde el teléfono móvil. Sin embargo, se decidió crear una aplicación móvil porque tiene muchas ventajas respecto a la página web a nivel de accesibilidad y rendimiento. Como consecuencia de que ha sido desarrollada de forma nativa, la aplicación es capaz de tener acceso a todo el hardware del dispositivo, acceder a todas las librerías graficas del sistema operativo como pueden ser los botones del dispositivo, permitir el envío de notificaciones y optimizar al máximo el rendimiento del sistema operativo.

No obstante, como ya se dispone de dicha página web y de su correspondiente base de datos, parte del trabajo se va reutilizar aprovechando la base de datos ya creada por medio de un servicio web de tipo REST. De esta manera se logra explotar toda la información existente a través de peticiones basadas en el protocolo HTTP.

1.2 Objetivos

El objetivo del presente proyecto es, por tanto, el diseño e implementación de una aplicación móvil multiplataforma para la empresa Bazar Canarias que permita a los usuarios:

- Consultar el catálogo de productos, pudiendo además realizar búsquedas por nombre del producto.
- Añadir productos al carrito de compra para su posterior compra ó realizar compras de productos previamente añadidos al carrito de compra.
- Añadir comentarios sobre los productos adquiridos.
- Calcular la ruta hacia las tiendas físicas gracias a la geolocalización.

La aplicación se conectará con la base de datos que da soporte a la página web ya existente a través de un servicio REST. La aplicación móvil accederá a los diferentes recursos de la API REST para obtener información de la base de datos como pueden ser los productos, información acerca de los usuarios o sobre el carrito de compra. El servicio REST es el encargado de establecer la comunicación entre la aplicación móvil y el servidor garantizando la coherencia de la información con la página web.

Por otro lado, la aplicación desarrollada será multiplataforma. Hasta hace relativamente poco tiempo las aplicaciones se desarrollaban de forma nativa, es decir se desarrollaban y se optimizaban específicamente para un sistema operativo y una plataforma concretos. Esto se hacía así para optimizar el rendimiento de la aplicación y para poder aprovechar al máximo las prestaciones del teléfono. También así obtener una interfaz mucho más adaptada al sistema operativo al cual el usuario está acostumbrado. Sin embargo, hoy en día el desarrollo de aplicaciones ha evolucionado mucho, el desarrollo multiplataforma ha tomado mucho protagonismo ya que cada vez es menos complejo desarrollar en múltiples plataformas simultáneamente y con resultados idénticos a las aplicaciones nativas. Esto se logra gracias a que los compiladores compilan directamente a código ensamblador nativo. Estas son las llamadas aplicaciones generadas, que son aplicaciones donde el desarrollo se realiza usando técnicas y lenguajes específicos de la herramienta, y luego se genera la aplicación en el lenguaje de la plataforma destino para ser compilada con las herramientas nativas.

Como objetivo de este proyecto cabe destacar además el aprendizaje de nuevas herramientas y tecnologías, como son las necesarias para el diseño y desarrollo de aplicaciones móviles multiplataforma, de las cuales el alumno no tenía ningún tipo de formación previa; así como del desarrollo de servicios web, concretamente basados en tecnología REST, que tampoco había sido abordado por el alumno a lo largo de sus estudios.

1.3 Organización de la memoria

A partir de este capítulo, la memoria se va a organizar en los siguientes apartados:

- En el capítulo 2 se van a tratar las tecnologías y herramientas utilizadas para el desarrollo de la aplicación móvil y el servicio REST, así como la metodología seguida en el desarrollo del proyecto.
- En el capítulo 3 se tratará el análisis y la especificación de requisitos, así como de los casos de uso, entrando en detalle en aquellos que tienen mayor complejidad. Por último, se especificarán los requisitos de interfaz de usuario que permiten obtener una aproximación real de cómo va a ser la aplicación.
- El capítulo 4 corresponde al diseño del sistema en el cual se describirá la arquitectura de la aplicación, los patrones utilizados, el diseño de la API REST y el diseño detallado de la aplicación móvil.
- La parte correspondiente a la implementación tanto de la aplicación móvil como del servicio web se expone en el capítulo 5. Se describirá y se entrará en detalle acerca de cómo las herramientas han sido utilizadas para realizar estas implementaciones.
- Después de la implementación sigue la parte de pruebas, expuesta en el capítulo 6, en el que se detallarán las pruebas unitarias realizadas para probar el servicio REST y probar éste dentro de la aplicación móvil. También se mostrarán las pruebas de integración aplicadas a la aplicación móvil y las pruebas de aceptación.
- Por último, un apartado con las conclusiones obtenidas al finalizar el proyecto y los trabajos futuros o mejoras que se podrían realizar una vez terminado.

2. Material y métodos utilizados

En este capítulo se van a tratar la metodología utilizada para el desarrollo de esta aplicación móvil, así como las tecnologías y las diferentes herramientas empleadas en el modelado y desarrollo de la aplicación móvil y el servicio REST.

2.1 Metodología

La metodología propuesta para el desarrollo de esta aplicación móvil se trata de una metodología de diseño clásica estilo modelo en cascada [2], pero aplicada de modo no estricto, ya que se permite la vuelta atrás, es decir la retroalimentación en función de las necesidades que vayan surgiendo durante el desarrollo de la aplicación. Se ha utilizado este tipo de metodología porque es la que más se adapta al estilo de trabajo empleado ya que al ser un trabajo individual no tenía mucho sentido aplicar metodologías más modernas y colaborativas como las metodologías ágiles [2].

Esta metodología consta de cinco fases como se muestra en la figura 1, denominadas: requisitos, diseño, construcción, pruebas y despliegue. Todas las fases tienen una elevada importancia siendo el objetivo común mejorar todos los aspectos para construir un software sin errores y de calidad.

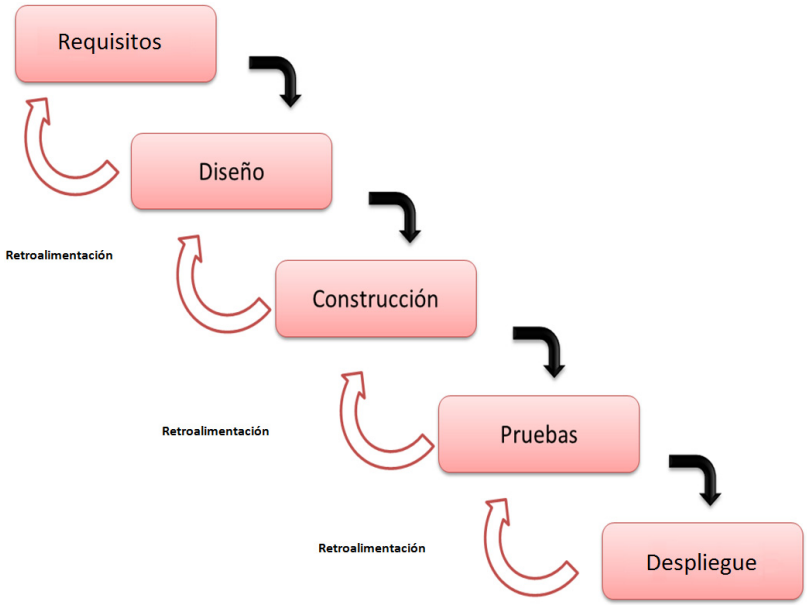


Figura 1. Metodología en cascada con retroalimentación (figura tomada de [3]).

La fase de requisitos es común a todo el proyecto, es decir, se efectúa de manera conjunta tanto para el desarrollo de la aplicación móvil como para el desarrollo del servicio REST. Sin embargo, para las fases de diseño, construcción y pruebas el trabajo se ha dividido en modo incremental, abordando primero el desarrollo del servicio REST, y a continuación el de la aplicación móvil. Se ha definido de esta manera debido a que son dos desarrollos completamente distintos, aunque complementarios.

2.2 Tecnologías y herramientas

En este apartado se expondrán las diferentes tecnologías y herramientas utilizadas en el desarrollo del proyecto. En la figura 2 se pueden observar los logos de todas las herramientas utilizadas en este proyecto.

Figura 2. Herramientas empleadas en este proyecto.

2.2.1 Modelado

Magic Draw

MagicDraw UML [4] es una herramienta de modelado empleada para ayudar en las actividades de diseño y desarrollo de software basado en lenguajes orientados a objetos. Está dirigida a arquitectos, ingenieros, analistas y programadores. Tiene una interfaz muy amigable que permite la creación de diagramas y modelos fácilmente. Es compatible con el estándar UML 2.3 además de los lenguajes de programación más populares (Java, C ++ y C #).

Se ha utilizado esta herramienta para diseñar los diagramas de casos de uso, el diagrama de componentes, diagrama de clases, interfaces y diagrama de despliegue.

2.2.2 Desarrollo de la aplicación móvil

Xamarin Forms

Xamarin [5] es una herramienta de software libre para desarrolladores de aplicaciones móviles que permite programar la aplicación en lenguaje C#, y que luego se encarga de traducir el código a lenguaje nativo para las distintas plataformas más importantes como son iOS, Android y Windows Phone. Proporciona, por tanto, un conjunto de herramientas de desarrollo multiplataforma que cubre todas las diferentes opciones desde un único código como base. Los desarrolladores pueden, de esta forma, crear aplicaciones completamente nativas desde un solo código.

Las aplicaciones de Xamarin tienen acceso a toda la gama de funcionalidades expuestas por la plataforma y el dispositivo, incluyendo funcionalidades específicas de cada

plataforma como *iBeacons* o *Android Fragments*. También aprovechan la aceleración de hardware específica de la plataforma y se compilan para el rendimiento nativo.

Xamarin permite la creación de proyectos que utilizan bibliotecas de clases portables (PCL por sus siglas en inglés), esto significa que la gran mayoría del código (60-70%) se comparte entre las diferentes plataformas. Pero no solo eso, para el desarrollo de la aplicación móvil se va a utilizar Xamarin Forms, que además de lo comentado anteriormente permite compartir el código de la interfaz de usuario, de manera que haya una interfaz común para las distintas interfaces multiplataforma, aumentado así el porcentaje de código compartido hasta el 80-90%.

Las herramientas Xamarin están disponibles gratuitamente para todos los usuarios de Visual Studio.

Visual Studio

Es un entorno de desarrollo integrado para sistemas operativos Windows que nos permite desarrollar aplicaciones para Android, iOS, Mac, Windows, la Web y la nube. Soporta múltiples lenguajes de programación como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby y PHP. Dispone de una versión gratuita.

Las grandes ventajas de utilizar Visual Studio [6] son que permite escribir código con rapidez gracias a su función de autocompletado, es muy sencillo depurar y emitir diagnósticos gracias a su programa de corrección de fallos, identifica la línea donde está el problema y tiene una interfaz muy amigable la cual se puede extender y personalizar según las preferencias del programador.

Android SDK

Como indican sus propias siglas es un kit de desarrollo software que, entre otras funcionalidades, permite visualizar programas realizados para Android gracias al emulador del sistema Android incorporado. Android SDK [7] incluye un depurador de código, documentación detallada acerca de la interfaz de programación de aplicaciones, la cual puede encapsular una biblioteca donde se especifica el comportamiento de un procedimiento, tutoriales y un simulador de teléfonos Android.

Android SDK está basado en el lenguaje Linux, utiliza una máquina virtual para la ejecución de las aplicaciones y ofrece soporte para componentes como GPS, 3G, WIFI, Bluetooth y pantallas táctiles entre otros. Hoy en día es capaz de soportar cualquier componente disponible en Smartphones de alta gama.

Por todas las características descritas anteriormente se ha elegido este emulador de Android para aplicaciones desarrolladas con Xamarin en Visual Studio.

NinjaMock

NinjaMock [8] es una herramienta gratuita para crear *mockups* y *wireframes* de manera sencilla, además la herramienta permite compartirlos con otras personas y descargarlos localmente. Tiene una amplia gama de accesorios y permite personalizar el diseño a medida.

2.2.3 Desarrollo del servicio REST

REST

REST (Representational State Transfer) [9] [10] es un estilo arquitectónico utilizado inicialmente para definir arquitecturas Web, y que en los últimos años ha ido evolucionando y tomando protagonismo en la implementación de servicios web.

Se caracteriza por basar el diseño del servicio en la existencia de un conjunto de recursos, los cuales son elementos de información que hacen referencia a conceptos clave de la aplicación o servicio diseñado. Cada recurso posee un identificador único global mediante el cual puede ser accedido públicamente. Los servicios web basados en REST no definen un conjunto específico de métodos u operaciones a medida para acceder a los recursos, sino que todos los recursos se acceden de la misma manera, a través de lo que se denomina la interfaz uniforme, que en la mayoría de los casos corresponde a los métodos (o verbos) del estándar HTTP: GET, POST, PUT, DELETE, etc. Cuando se accede a un recurso se obtiene una representación del mismo, generalmente en formato XML o JSON, aunque es posible utilizar más formatos.

En REST la obtención de datos o la ejecución de operaciones sobre datos, se realiza sin necesidad de hacer uso de una capa opcional, como puede ser SOAP o la gestión de sesiones a través de cookies. Esto permite a cualquier cliente interactuar con servidores HTTP sin necesidad de realizar ninguna configuración especial. Otra característica muy importante es que no se guarda el estado en el servidor, esto permite procesar más peticiones en menor tiempo y convertirlo en más eficiente. Por el contrario, a diferencia de otros servicios web, REST introduce una mayor complejidad en la fase de diseño de los servicios, ya que se deben diseñar los recursos en base al conjunto de operaciones o métodos estándar permitidos, y esto de entrada puede resultar costoso según la funcionalidad del servicio. Sin embargo, REST se está convirtiendo en la opción más utilizada para el desarrollo de servicios web en la actualidad y por ello se ha decidido crear una API REST en este proyecto.

Slim framework

Se va a utilizar Slim framework [11] para implementar el servicio REST. Es un framework de aplicaciones web PHP que permite escribir velozmente tanto aplicaciones web sencillas pero muy potentes como APIs. Slim también facilita la comunicación con el cliente gracias a la validación de las peticiones y la captura de lo que viene antes y después de cada petición.

Slim recibe peticiones HTTP, invoca una rutina de devolución de llamada, y devuelve una respuesta HTTP. Para ello se basa en el protocolo HTTP [12] que es un protocolo que permite la comunicación entre dos elementos software, generalmente un navegador web (cliente) y un servidor web. HTTP define la sintaxis y la semántica que utilizan estos elementos para comunicarse. El propósito fundamental es la transferencia de información basado en un esquema de petición/respuesta. Además, HTTP es un protocolo sin estado, es decir no guarda ninguna información de las conexiones que se han realizado anteriormente, tras la respuesta el servidor cierra inmediatamente la conexión.

Sublime Text

Es un editor de texto y de código fuente multiplataforma que se puede obtener de forma gratuita. Soporta múltiples lenguajes de programación y tiene una interfaz de usuario muy amigable que facilita la programación gracias a su funcionalidad de autocompletado y su coloreado de palabras en función del lenguaje en que se esté programando.

Sublime Text [13] ha sido utilizado en el desarrollo de esta aplicación como editor de texto para programar el servicio REST en lenguaje PHP.

Apache

Apache [14] es un servidor web HTTP de código abierto. Es software libre mantenido y desarrollado por una comunidad abierta de desarrolladores disponible para sistemas Unix, Windows e iOS. Es el servidor web más utilizado del mundo, y en concreto, la aplicación web de comercio electrónico ya disponible está realizada con un programa, denominado Oscommerce [15], que corre sobre un servidor Apache.

Se ha empleado Apache 2.4 para replicar la parte correspondiente al servidor, de manera que se pudieran hacer pruebas en el propio equipo de trabajo de forma local.

Advanced REST Client

ARC (Advanced REST Client) [16] es una extensión de Google Chrome que permite lanzar peticiones sobre un servicio o API para realizar pruebas de manera rápida y eficaz. Además de las peticiones habituales como son GET, POST, PUT, DELETE es posible lanzar cualquier tipo de petición. Permite el paso de parámetros en las peticiones y retorna el resultado esperado. ARC ha sido utilizado para realizar pruebas sobre la API REST diseñada y comprobar su correcto funcionamiento.

PHP

Es un lenguaje de programación de código abierto empleado generalmente para desarrollo web, sobre todo para páginas con contenido dinámico. Se diferencia de otros lenguajes en que permite la programación en el lado del servidor, quiere decir que permite procesar peticiones a través de scripts en el servidor web y así generar como respuesta páginas HTML. El objetivo del lenguaje PHP [17] es permitir a los desarrolladores web escribir fácilmente páginas complejas generadas dinámicamente.

Es el lenguaje escogido para implementar la API REST debido a su sencillez. Sus características encajan perfectamente en las necesidades requeridas por el servicio web.

3. Análisis y especificación de requisitos

En este apartado se describe el procedimiento seguido para la especificación de los requisitos del proyecto. Para ello se organizaron varias reuniones con el cliente para determinar el funcionamiento de la aplicación y se diseñaron los posibles escenarios de ejecución.

Se ha aproximado mucho el diseño y funcionamiento de la aplicación con la página web para una mayor facilidad e intuición del manejo de ésta.

3.1 Especificación de requisitos funcionales

Los requisitos funcionales describen el funcionamiento propio de la aplicación, es decir cómo va a interaccionar con el entorno, cuáles van a ser sus respuestas y sus estados. En la tabla 1 se describen los requisitos funcionales de la aplicación móvil a desarrollar.

Tabla 1. Definición de requisitos funcionales de la aplicación.

Identificador	Descripción del Requisito
RF01	El usuario podrá consultar el catálogo de productos de la tienda.
RF02	El usuario contará con un buscador de productos que permita buscar productos por su nombre.
RF03	El usuario podrá registrarse en la aplicación indicando su nombre, apellido, email, que será utilizado como identificador del usuario, número de teléfono, contraseña y dirección de envío (calle, código postal, población, provincia y país).
RF04	El usuario registrado podrá modificar los parámetros de su cuenta, tanto sus datos personales como la dirección de envío.
RF05	El usuario registrado podrá añadir productos del catálogo a su carrito/cesta de compra.
RF06	El usuario registrado podrá eliminar productos de su carrito/cesta de compra.
RF07	El usuario registrado podrá escribir comentarios en los productos adquiridos.
RF08	El usuario registrado podrá realizar compras a través de la aplicación, es decir, el usuario registrado puede confirmar su carrito/cesta de compra.
RF09	El usuario podrá consultar en la aplicación la localización de las tiendas físicas.
RF10	El usuario podrá consultar la ruta hasta la tienda más cercana (la aplicación la calculará a través de Google Maps).
RF11	La aplicación mostrará un mensaje al usuario para indicarle que la aplicación no puede ejecutarse a consecuencia de no tener conexión a internet.

3.2 Especificación de requisitos no funcionales

Los requisitos no funcionales describen restricciones que se imponen en la aplicación, principalmente en aspectos como rendimiento, interfaces y procesos de desarrollo. A continuación, en la Tabla 2 se describen los requisitos no funcionales de la aplicación.

Tabla 2. Definición de requisitos no funcionales de la aplicación.

Identificador	**Descripción del Requisito**	**Categoría**
RNF01	Ningún usuario podrá acceder a los datos personales de otro usuario. Un usuario solo podrá acceder a sus datos tras realizar el proceso de autenticación.	Seguridad
RNF02	La aplicación deberá cifrar las contraseñas, para garantizar la protección de las cuentas de los usuarios.	Seguridad
RNF03	La aplicación realizará las transacciones de manera segura gracias al protocolo SSL.	Seguridad
RNF04	Para que la aplicación funcione correctamente, el dispositivo (Smartphone) deberá disponer de conexión a internet, bien 3G/4G o WIFI.	Operatividad
RNF05	La aplicación debe ser soportada por los sistemas operativos Android e iOS.	Compatibilidad
RNF06	La aplicación debe ser fácil de usar e intuitiva.	Usabilidad
RNF07	La aplicación debe tener un tiempo de respuesta que sea lo suficientemente razonable para cualquier petición realizada por los usuarios.	Rendimiento
RNF08	El diseño de la aplicación debe ser modular para facilitar su evolución y mantenimiento.	Extensibilidad

3.3 Especificación de casos de uso

Los casos de uso especifican el comportamiento que se espera de la aplicación móvil, describen los modos en los que la aplicación puede ser utilizada y las características que ofrece la aplicación a disposición de los usuarios. Son utilizados para obtener una visión global de las funcionalidades de la aplicación y describir las necesidades de los usuarios tras un análisis inicial. Permiten observar de manera fácil y sencilla todas las acciones que realiza la aplicación móvil para los diferentes actores.

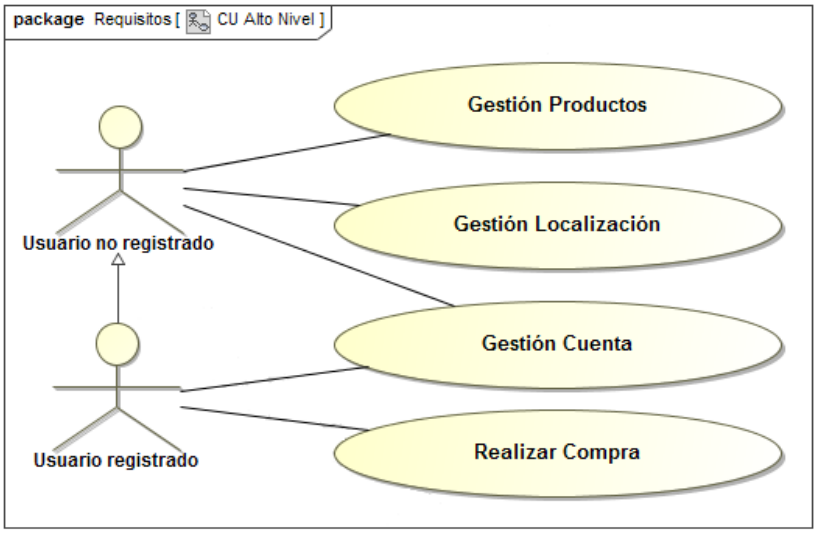


Figura 3. Diagrama de casos de uso de alto nivel.

En la figura 3 se pueden distinguir los casos de uso de alto nivel, separados por las diferentes áreas de la aplicación y los dos actores principales, que son los usuarios registrados y los usuarios no registrados, los cuales tendrán diferentes privilegios y acceso a diferentes casos de uso.

En las figuras 4, 5, 6 y 7 se puede observar el detalle de cada uno de los casos de uso de alto nivel. Se ha definido así para poder modularizar el diseño arquitectónico más adelante.

En la figura 4 se observan los casos de uso correspondientes a la gestión de productos y se determina cuales corresponden a cada actor. Lo mismo ocurre en las figuras 5 y 6 en las que se tratan los casos de uso correspondientes a la gestión de la cuenta o perfil del usuario registrado y los casos de uso correspondientes al proceso de compra. Por último, en la figura 7 se determinan los casos de uso de bajo nivel correspondientes a la localización de tiendas.

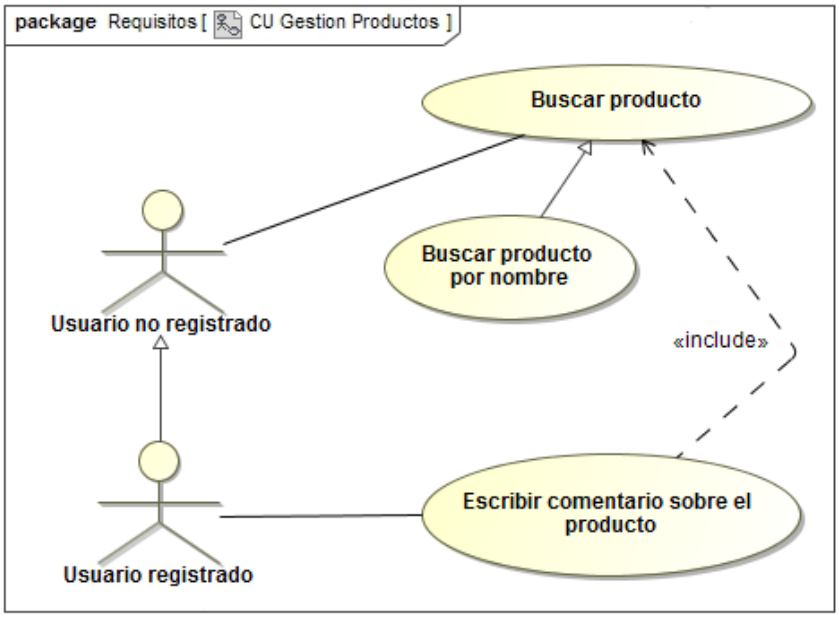


Figura 4. Diagrama de casos de uso correspondiente a la gestión de productos.

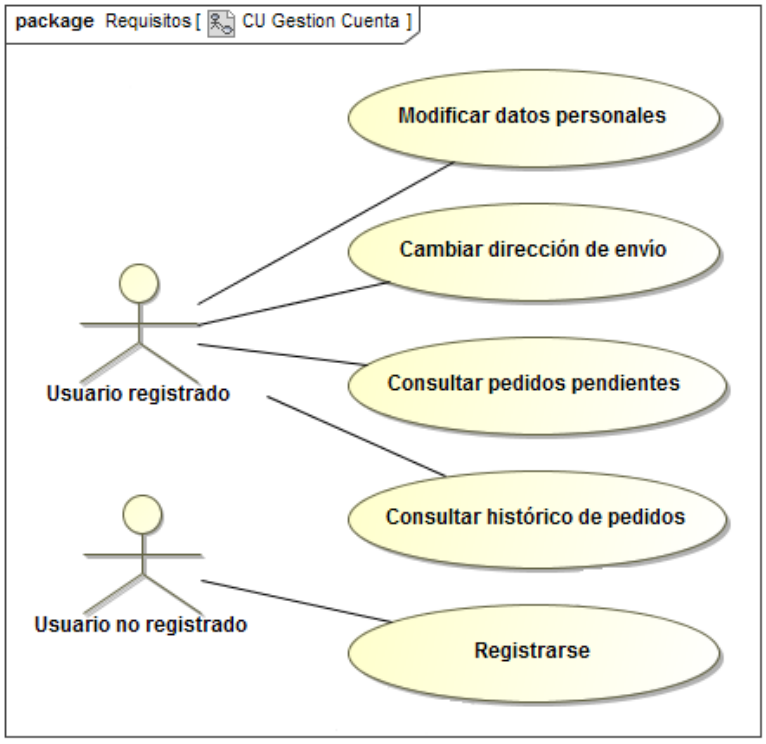


Figura 5. Diagrama de casos de uso correspondiente a la gestión de cuentas.

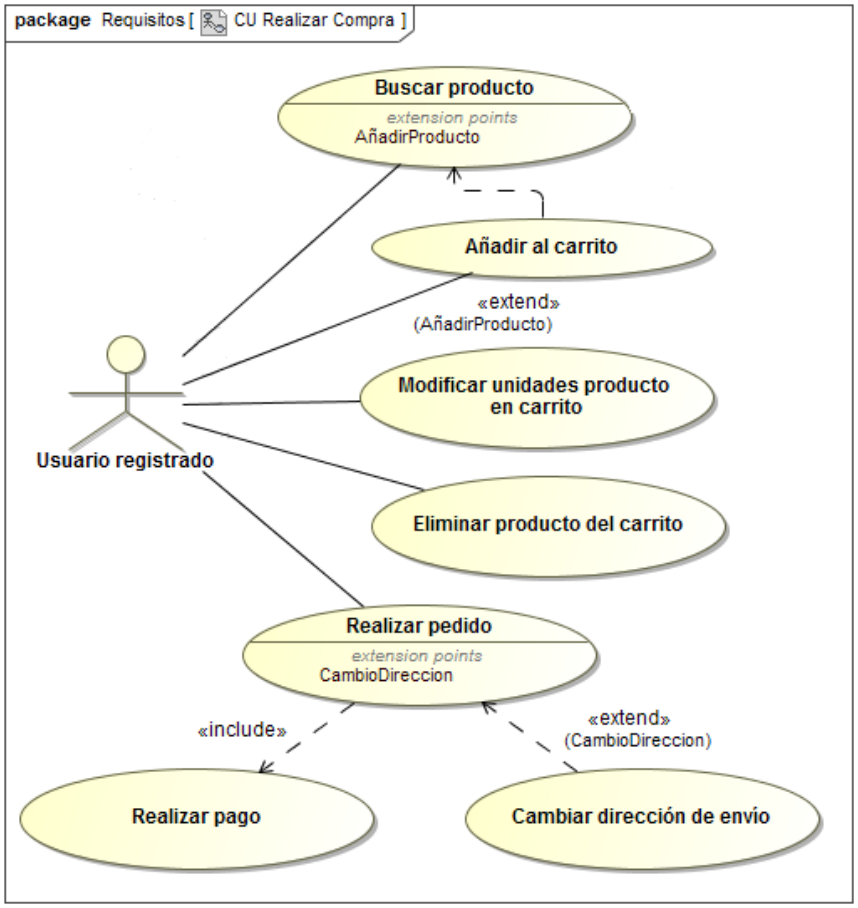


Figura 6. Diagrama de casos de uso correspondiente a la realización de una compra.

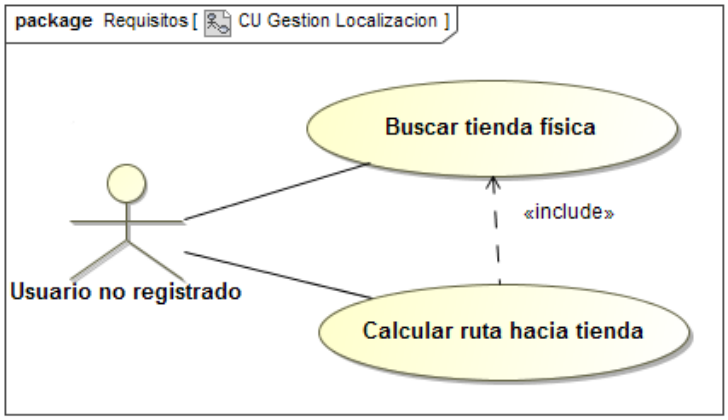


Figura 7. Diagrama de casos de uso correspondiente a la gestión de localización.

3.4 Casos de uso detallados

A continuación, se van a describir algunos casos de uso de forma detallada mediante plantillas, que permiten estructurar y describir los casos de uso para alcanzar el mayor nivel de detalle posible. Se detallan los casos de uso que no son triviales y se determina el comportamiento esperado de la aplicación en cada uno de estos casos.

Tabla 3. Especificación del caso de uso "Realizar pedido".

Identificador	CU1
Nombre	Realizar pedido
Descripción	El usuario realiza una compra a través de la aplicación móvil, confirmando su carrito de compra.
Actores	Usuario registrado
Precondición	1. El usuario debe estar logueado. 2. El usuario debe tener al menos un artículo en su cesta.
Flujo principal	1. El usuario pulsa el botón "Realizar pedido". 2. Extensión Point. CambioDirección. 3. El usuario selecciona la forma de envío marcando una de las opciones disponibles. 4. El usuario podrá añadir un comentario al envío. 5. El usuario selecciona la forma de pago. 6. Include Realizar Pago. 7. La aplicación almacena el pedido. 8. La aplicación envía un email de confirmación con todos los detalles del pedido al usuario.
Flujo alternativo	6.1 Si no se ha podido realizar el pago la aplicación lanza un mensaje de error y finaliza el caso de uso.

Tabla 4. Especificación del caso de uso "Cambiar dirección de envío".

Identificador	CU2
Nombre	Cambiar dirección de envío
Descripción	El usuario podrá cambiar la dirección de envío del pedido a la dirección que desee.
Actores	Usuario registrado
Precondición	El usuario debe haber iniciado el proceso de realización del pedido.

Flujo principal	1. El usuario pulsa el botón “Cambiar dirección”. 2. El usuario rellenará el formulario con los nuevos datos de dirección de envío. 3. El usuario pulsará el botón “Continuar”. 4. La aplicación modifica la dirección de envío del usuario para este pedido.
Flujo alternativo	3.1 Si el usuario no ha completado todos los campos obligatorios se mostrará un mensaje indicando que debe completar todos los campos requeridos.

Tabla 5. Especificación del caso de uso “Realizar pago”.

Identificador	CU3
Nombre	Realizar Pago
Descripción	El usuario confirmará y hará efectivo el pago del pedido.
Actores	Usuario registrado
Precondición	El usuario debe haber iniciado el proceso de realización del pedido
Flujo principal	1. El usuario pulsa el botón “Confirmar Pedido”. 2. La aplicación se conectará con el sistema de pago elegido y se mostrará una ventana para introducir los datos de pago correspondientes. 3. El usuario introducirá los datos para efectuar el pago y confirmará el pago. 4. La aplicación mostrará un mensaje de que el pago se ha realizado con éxito.
Flujo alternativo	2.1. Si el usuario no ha introducido correctamente los datos del pago se mostrará un mensaje de error seleccionado por cada entidad de pago. En cualquier momento el usuario podrá cancelar la compra y volver a la aplicación.

Tabla 6. Especificación del caso de uso “Calcular ruta hacia tienda”.

Identificador	CU4
Nombre	Calcular ruta hacia tienda
Descripción	El usuario podrá calcular la ruta hacia la tienda escogida desde su ubicación actual a través de la aplicación de Google Maps
Actores	Usuario no registrado
Precondición	El usuario debe tener activada la localización.
Flujo principal	1. Include Buscar tienda física 2. El usuario pulsa el botón “Calcular ruta hacia tienda” 3. La aplicación solicita acceso a la aplicación de Google Maps para poder calcular la ruta. 4. Se calcula la ruta más rápida hacia la tienda escogida. 5. La aplicación muestra la ruta más rápida.
Flujo alternativo	1.1 Si la localización esta desactivada, la aplicación lazará un mensaje solicitando al usuario que active la localización en su Smartphone.

Tabla 7. Especificación del caso de uso “Buscar producto por nombre”.

Identificador	CU5
Nombre	Buscar producto por nombre
Descripción	El usuario busca un producto en el catálogo indicando su nombre.
Actores	Usuario no registrado
Precondición	El usuario deber estar en la sección “Catálogo” de la aplicación.
Flujo principal	1. El usuario introduce la palabra a buscar en el buscador y pulsa buscar.

	2. La aplicación buscará el producto con el nombre indicado en el catálogo. 3. La aplicación mostrará un listado con todos los productos que coincidan con la palabra buscada o aquellos que al menos contengan esos caracteres.
Flujo alternativo	1.1 Si no existe ningún producto que coincida o contenga ese término la aplicación no mostrará ningún producto.

3.5 Requisitos de la interfaz de usuario

Son requisitos que describen la forma en la que la aplicación interacciona con el usuario. Estos requisitos proporcionan una primera aproximación de cómo debería ser el aspecto de la aplicación y su funcionamiento.

En el proceso de diseño y desarrollo de aplicaciones móviles una de las claves principales es el prototipado [2]. Permite acelerar la fase de desarrollo, evitar problemas futuros y obtener lo antes posible una visión cercana al resultado final. En este caso, se va a utilizar la técnica de Mockups como maquetas de prototipado.

3.5.1 Mockups

Los mockups son maquetas digitales que contienen información de cada una de las pantallas, como por ejemplo contenidos, disposición de los botones, navegación entre las pantallas, etc. También permite visualizar la línea gráfica, estructura de la información y algunas funcionalidades del proyecto.

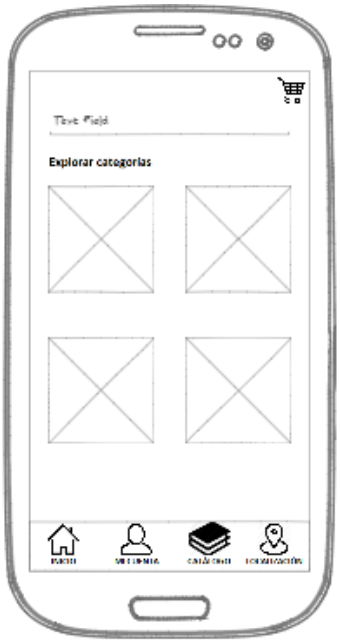
En la figura 8 se muestran varios mockups del proyecto. En ellos se representa una aproximación del contenido visual de la aplicación final, se observan los elementos que componen cada interfaz, la navegación entre las diferentes pantallas/interfaces y la funcionalidad de éstas. Gracias a los mockups se tiene una muy buena idea de cómo se verá el producto final y una idea aproximada de cómo podría funcionar.

Cada mockup de la figura 8 representa una sección distinta de la aplicación:

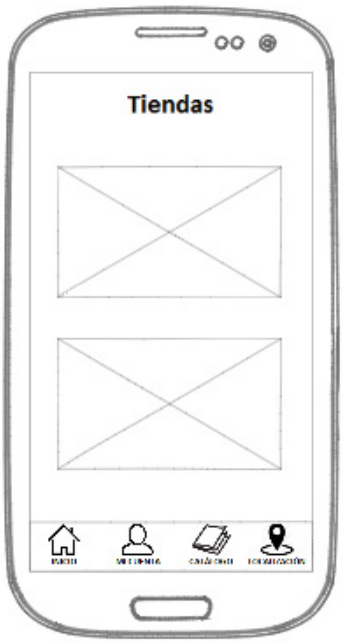
- La sección Inicio corresponde a la página de inicio cuando se abre la aplicación, donde se especifican algunas de las ofertas y productos disponibles.
- En la sección Catálogo se observan los productos existentes en el catálogo y se permite realizar búsquedas por nombre de productos específicos.
- La sección Cuenta permite loguearse a los usuarios registrados o dar de alta a usuarios no registrados. También permite realizar modificaciones sobre los datos personales a los usuarios registrados.
- Por último, la sección Localización permite obtener la localización de las tiendas y calcular la ruta hacia ellas gracias a la geolocalización.

Sección Inicio

Sección Cuenta



Sección Catálogo



Sección Localización

Figura 6. Mockups.

4. Diseño del sistema

En este capítulo se describen la arquitectura hardware y software del sistema, el diseño arquitectónico de la aplicación móvil y los patrones utilizados para su implementación, los componentes y módulos necesarios para satisfacer los requisitos previamente definidos, y el diseño del servicio REST.

4.1 Diseño arquitectónico del sistema

Un patrón arquitectónico describe la estructura de un sistema de software. Establece los componentes que forman el sistema, la relación e interacción entre ellos y las responsabilidades de cada uno. La elección del patrón arquitectónico es una elección de mucha importancia e influye directamente en las fases de diseño y construcción.

En este caso la arquitectura que se va a emplear seguirá el patrón de arquitectura cliente-servidor combinada con una arquitectura en tres capas, debido a que al utilizar dicha arquitectura es posible aislar la lógica de negocio y convertirla en una capa intermedia entre la capa de presentación y la capa de datos. A continuación, en la figura 9 se detalla el modelo de arquitectura en tres capas que se ha utilizado en el desarrollo software del proyecto:

- Capa de presentación: Proporciona la interfaz de la aplicación con el usuario final, en este caso formada por la aplicación móvil y la aplicación web ya existente. No realiza apenas tareas de procesamiento ya que se limita a pasar y recibir información de la capa de negocio. Únicamente se comunica con la capa de negocio.
- Capa de negocio: Construida por los elementos que efectúan las operaciones necesarias para el correcto funcionamiento de la aplicación. Se comunica tanto con la capa de presentación, como con la de persistencia. Se comunica con la aplicación móvil ofreciendo toda su funcionalidad en forma de servicio REST.

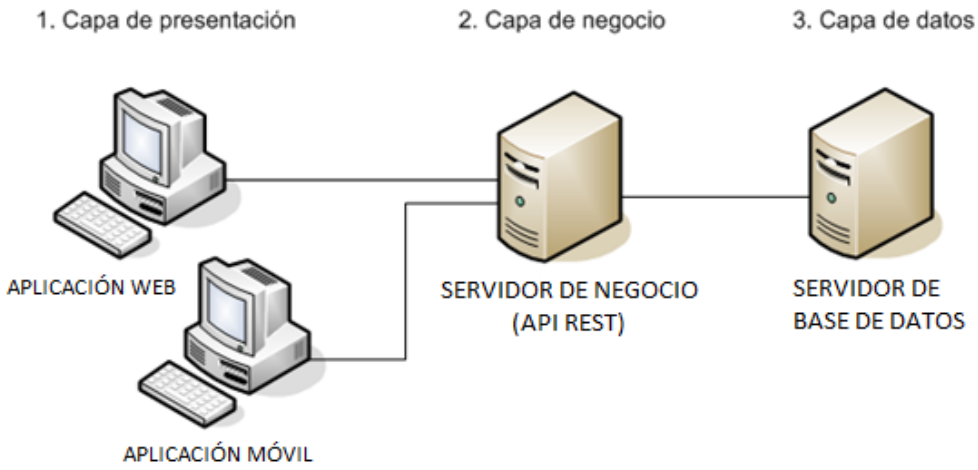


Figura 7. Modelo de arquitectura en tres capas.

- Capa de persistencia o de datos: Su objetivo es la gestión del almacenamiento de los datos. En esta capa es donde se almacena la base de datos (formada por uno o más gestores de bases de datos), en nuestro caso se emplea MySQL para su gestión.

4.2 Diseño arquitectónico de la aplicación móvil

El diseño arquitectónico de la aplicación móvil se basa en el patrón Modelo -Vista - Controlador (MVC) [16], que separa el código en (tres) bloques de distintas funcionalidades. Se trata de un patrón arquitectónico empleado para realizar el diseño modular de aplicaciones software interactivas. Consiste en separar el modelo de datos (Modelo), el modo de visualización e interacción del usuario (Vista) y la lógica de la aplicación (Controlador). A continuación, se describen las capas que lo forman:

- Modelo: Esta capa proporciona la funcionalidad necesaria para el acceso y la actualización de los datos. El rol de modelo lo va a implementar en este caso la API REST.
- Vista: Proporciona una representación gráfica e interactiva del modelo. Permite la interacción con el usuario en forma de interfaz.
- Controlador: Define la lógica de control y la funcionalidad de la aplicación. Actúa como intermediario entre la 'vista' y el 'modelo' ya que opera sobre el modelo en función de los eventos y comandos recibidos y actualiza la vista en base al resultado del procesamiento o elección del usuario.

El patrón MVC es independiente del lenguaje de programación y permite configurar la estructura global de la aplicación en la fase de diseño. Puede haber múltiples componentes de Vista para un mismo modelo y a cada vista puede ser asociado un componente Controlador. Es por todas estas consideraciones por lo que es utilizado frecuentemente en la arquitectura software de aplicaciones interactivas complejas.

El diseño arquitectónico de una aplicación se puede modelar a través de diagramas de componentes UML, que permiten modelar los componentes de software de alto nivel y, lo que es más importante, las interfaces de esos componentes. La figura 10 muestra el diagrama de componentes correspondiente a la aplicación en el que se han definido los componentes necesarios de cada capa (modelo, vista, controlador) para implementar las funcionalidades, los casos de uso de alto nivel definidos previamente, y las dependencias entre éstos.

Se ha definido una vista por cada caso de uso de alto nivel. Éstas se relacionan con los controladores por medio de interfaces. La API REST es la que comunica los controladores, que actúan de intermediarios, con los datos del modelo y así se cierra la conexión entre los componentes.

Enlazado en la figura 11 se pueden observar las interfaces de cada uno de los componentes y el detalle de las operaciones de éstas.

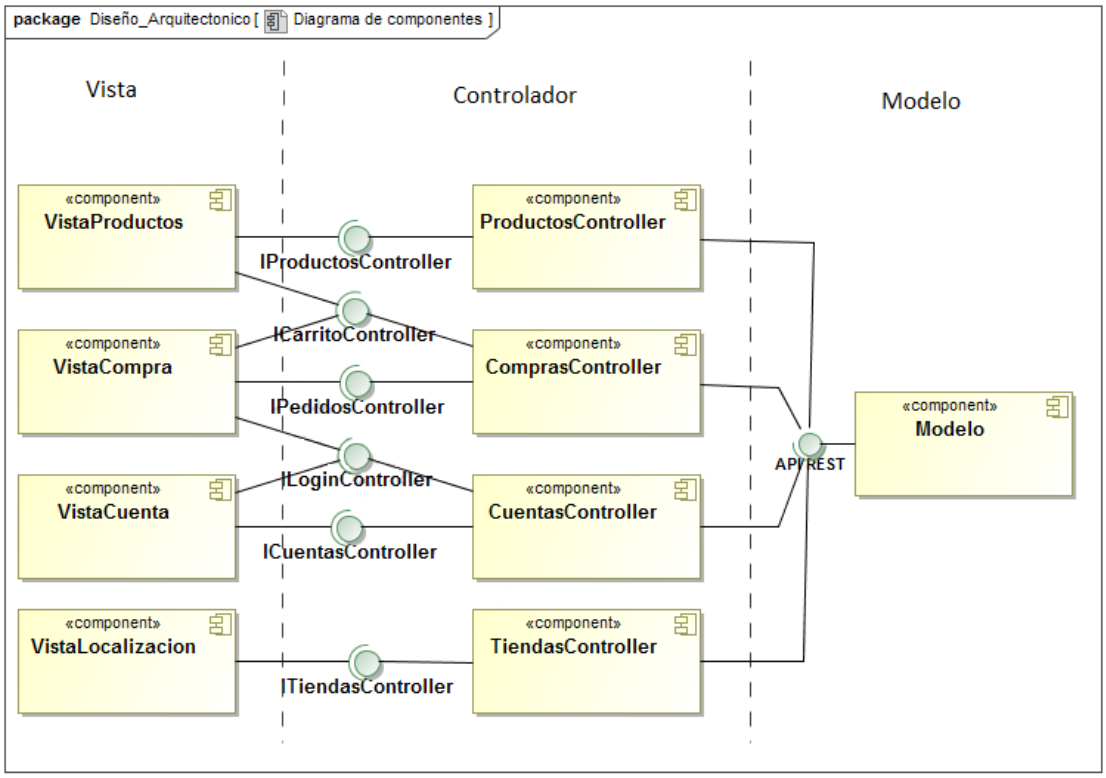


Figura 8. Diagrama de componentes del sistema.

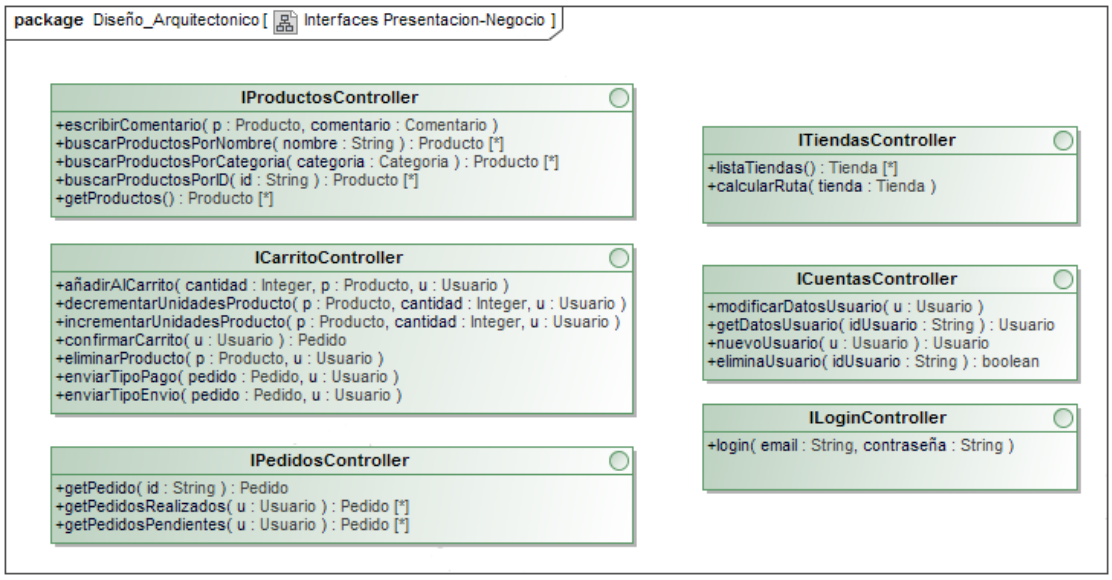


Figura 9. Interfaces del sistema.

4.3 Diseño de la API REST

Explotando el trabajo ya hecho por parte de la empresa gracias a su aplicación web, se decidió implementar una API REST [18] de forma que actúe empleando el rol de Modelo y se comunique con la base de datos ya existente. La API permite la comunicación entre el cliente y el servidor usando URIs basadas en el protocolo HTTP que han sido diseñadas e implementadas en base a las necesidades del proyecto.

Las peticiones a la API se realizan mediante los métodos HTTP, los más comunes y que han sido empleados para la implementación de la API REST son:

- GET: Solicita la representación de un recurso al servidor. El recurso es identificado unívocamente por su URI. Al ser empleado solo para la lectura de datos es considerado un método seguro. La API REST implementada para este proyecto devolverá la representación de los recursos en formato JSON.
- POST: Crea un nuevo recurso en el servidor. POST envía la información sobre el recurso a crear en el cuerpo del mensaje, también en formato JSON.
- PUT: Se utiliza para actualizar recursos en el servidor. El nuevo estado del recurso se envía en el cuerpo del mensaje HTTP y el servidor informará al cliente que el recurso ha sido actualizado correctamente.
- DELETE: Esta petición requiere que el servidor elimine el recurso especificado por el URI.

Se han diseñado tanto peticiones a nivel de usuario como de administración de sistema, aunque solo se han implementado las que corresponden al usuario. Se ha diseñado de esta manera por si en un futuro se desea implementar la lógica correspondiente al rol de administrador del sistema. En ese caso el administrador contará con más privilegios que el usuario y por tanto será capaz de realizar peticiones más sensibles como es el caso del DELETE de un producto, por ejemplo.

La tabla 8 nos muestra el diseño de la API REST donde se muestran todas las peticiones diseñadas para la aplicación, o lo que es lo mismo, los recursos que forman la API, sus correspondientes URIs y las peticiones HTTP que soporta cada uno junto a sus correspondientes códigos de respuesta.

Tabla 8. Diseño de la API REST.

RECURSO	URI RECURSO	PETICIONES	RESPUESTAS
Producto	/productos/{idProducto}	GET	HTTP 200 OK HTTP 404 NOT FOUND
		PUT	HTTP 200 OK HTTP 404 NOT FOUND HTTP 403 FORBIDDEN
		DELETE	HTTP 200 OK HTTP 404 NOT FOUND HTTP 403 FORBIDDEN
Productos	/productos QueryParams: nombre, categoria	GET	HTTP 200 OK HTTP 404 NOT FOUND
		POST	HTTP 201 CREATED HTTP 403 FORBIDDEN HTTP 409 CONFLICT HTTP 404 NOT FOUND
Usuario	/usuarios/{email}	GET	HTTP 200 OK HTTP 404 NOT FOUND
		PUT	HTTP 200 OK HTTP 404 NOT FOUND HTTP 403 FORBIDDEN
		DELETE	HTTP 200 OK HTTP 404 NOT FOUND

			HTTP 403 FORBIDDEN
Usuarios	/usuarios	GET	HTTP 200 OK HTTP 404 NOT FOUND
		POST	HTTP 201 CREATED HTTP 409 CONFLICT HTTP 404 NOT FOUND
Carrito	/carrito/{idUsuario}	GET	HTTP 200 OK HTTP 404 NOT FOUND HTTP 403 FORBIDDEN
		DELETE	HTTP 200 OK HTTP 404 NOT FOUND HTTP 403 FORBIDDEN
		POST	HTTP 201 CREATED HTTP 403 FORBIDDEN HTTP 409 CONFLICT HTTP 404 NOT FOUND
Línea de Pedido Carrito	/carrito/{idUsuario}/ {idLinea}	GET	HTTP 200 OK HTTP 404 NOT FOUND HTTP 403 FORBIDDEN
		PUT	HTTP 200 OK HTTP 404 NOT FOUND HTTP 403 FORBIDDEN
		DELETE	HTTP 200 OK HTTP 404 NOT FOUND HTTP 403 FORBIDDEN
Tienda	/tiendas/{idTienda}	GET	HTTP 200 OK HTTP 404 NOT FOUND
		PUT	HTTP 200 OK HTTP 403 FORBIDDEN HTTP 404 NOT FOUND
		DELETE	HTTP 200 OK HTTP 404 NOT FOUND HTTP 403 FORBIDDEN
Tiendas	/tiendas	GET	HTTP 200 OK HTTP 404 NOT FOUND
		POST	HTTP 201 CREATED HTTP 403 FORBIDDEN HTTP 409 CONFLICT HTTP 404 NOT FOUND
Compra (pago)	/carrito/{idUsuario}/ compra	GET	HTTP 200 OK HTTP 404 NOT FOUND HTTP 403 FORBIDDEN
		PUT	HTTP 201 CREATED HTTP 403 FORBIDDEN HTTP 404 NOT FOUND

Vamos a definir a continuación algunas de las representaciones de los diferentes recursos que se manejarán en las llamadas a la API.

Producto

El recurso Producto corresponde a cada producto perteneciente al catálogo. Para realizar una petición REST y obtener información sobre un producto en concreto deberemos realizar una petición HTTP GET al URI {URIBase}/productos/{idProducto}. Si se encuentra el recurso, es decir el producto existe, el servidor responderá con un mensaje

HTTP 200 OK en cuyo cuerpo se incluye un archivo JSON con la información correspondiente, como podemos observar en el siguiente ejemplo:


```
"producto": {
        "idProducto": "1247",
        "nombre": "Tablet Samsung Tab E",
        "disponible": "Si",
        "fechaDisponible": "01/12/2016",
        "fabricante": "Samsung",
        "tipoImpuesto": "IVA 21%",
        "precio": 171.00,
        "descripcion": "texto…",
        "cantidad": 10,
        "modelo": "TAB E 560 9,6" 8GB Blanca",
        "imagen": "tabE.jpg",
        "peso": 490.00 }
```


Usuario

El recurso Usuario corresponde a cada usuario registrado en el sistema. Para obtener información del usuario se hace una petición HTTP invocando el método GET al URI {URIBase}/usuarios/{email}. Por el contrario, si se quiere modificar información de un usuario deberemos invocar el método PUT al mismo URI. A continuación, vemos la representación de un recurso usuario en formato JSON.


```
"usuario": {
        "id": "123",
        "nombre": "Juan",
        "apellido": "Gómez Pérez",
        "email": "juangp@hotmail.com",
        "NIF": "72134567X",
        "nombreEmpresa": "null",
        "direccion": "Calle Los Ciruelos 22, 5F",
        "cPostal": "39011",
        "poblacion": "Santander",
        "pais": "España",
        "telefono": "609345678",
        "fax": "null",
        "contraseña": "xxxx" }
```


Carrito

El recurso Carrito corresponde a la lista de productos que hay en el carrito de compra de un usuario registrado. Para obtener los productos pertenecientes a un carrito se invocará el método GET al URI {URIBase}/carrito/{idUsuario}. Para añadir un nuevo producto al carrito se invocará el método POST sobre el mismo URI. Si se quiere por ejemplo modificar la cantidad de un producto, se invocará el método PUT al URI {URIBase}/carrito/{idUsuario}/{idLinea} y si queremos eliminar un producto del carrito se invocará el método DELETE en el URI {URIBase}/carrito/{idUsuario}/{idLinea}. A continuación, se muestra un ejemplo del método GET, que devuelve varios productos pertenecientes a un carrito.

```
“carrito”: [
        {
        “carritoCompraArticulo”: {
                “idLinea”: “18”,
                “idUsuario”: “8”,
                “idProducto”: “50”,
                “cantidadProducto”: “1”,
                “precio”: 171.00
                “fechaAñadido”: “01/01/2017”,
        }
        “carritoCompraArticulo”: {
                “idLinea”: “19”,
                “idUsuario”: “8”,
                “idProducto”: “55”,
                “cantidadProducto”: “1”,
                “precio”: 80.00
                “fechaAñadido”: “01/01/2017”,

} ]
```


4.4 Diseño del despliegue

Los diagramas de despliegue son una forma de representar el software tal y como va a ser distribuido sobre los componentes hardware. Estos forman el conjunto de nodos físicos sobre los cuales se distribuyen los artefactos del software. En este caso, para describir el sistema en términos de recursos hardware, se ha utilizado el diagrama de despliegue de la figura 12. En él aparece el diagrama completo, incluyendo el cliente web que ya estaba implementado. Para este proyecto se ha realizado la parte del cliente correspondiente a la aplicación móvil.

4.5 Diseño detallado

El diagrama de clases que se muestra en la figura 13 representa un esquema conceptual en el que se muestran las clases del sistema, sus atributos y las relaciones entre ellas, es decir proporciona una visión estructurada del modelo del sistema.

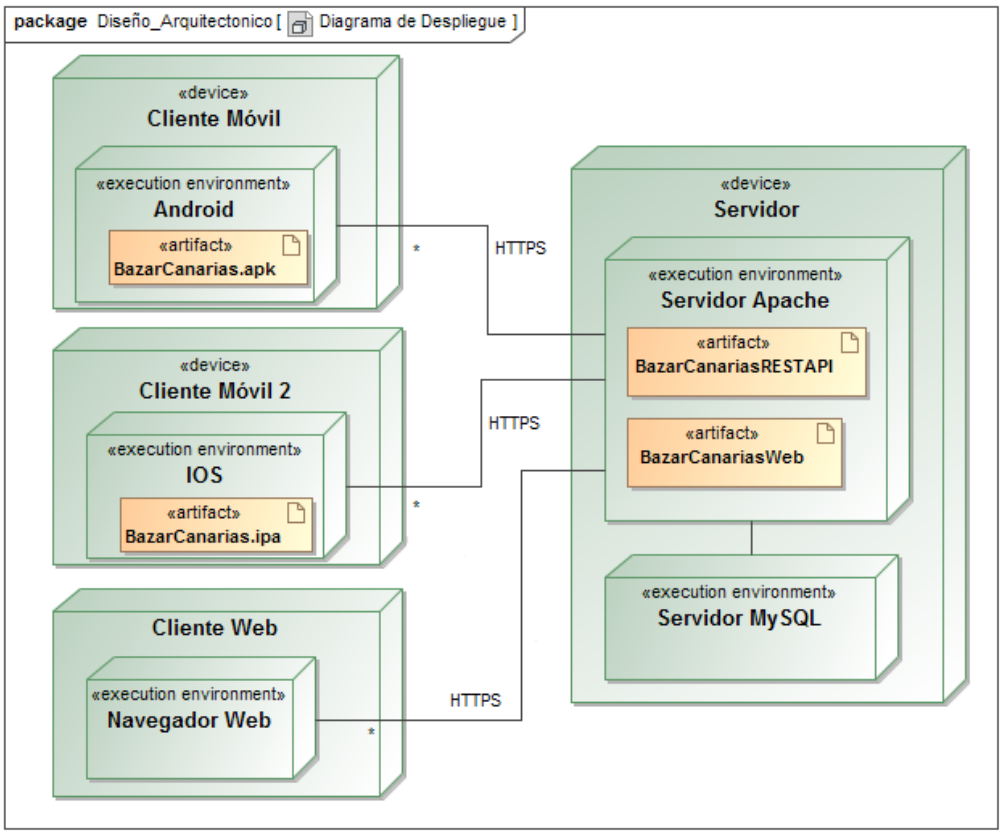


Figura 10. Diagrama de despliegue del sistema.

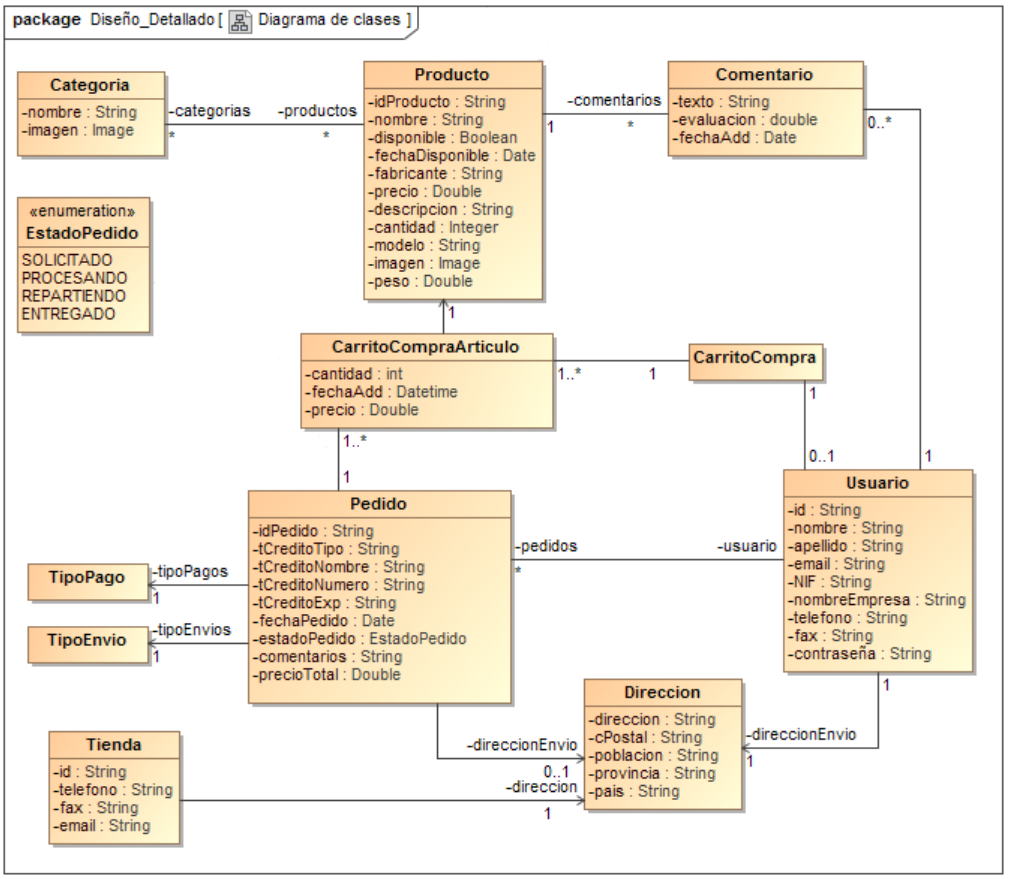


Figura 11. Diagrama de clases del sistema.

5. Implementación

Una vez completadas las fases de análisis y diseño, se procede a desarrollar la fase de implementación, que consiste en implementar todo lo especificado en el modelo de diseño. A lo largo del capítulo, se incluirán fragmentos del código para explicar cómo se han plasmado en código decisiones tomadas en las fases de diseño y el uso de las tecnologías empleadas en esta fase del desarrollo de la aplicación.

5.1 Implementación de la aplicación móvil

Para la implementación de la aplicación móvil se ha utilizado el entorno de desarrollo integrado Visual Studio con la extensión Xamarin, que permite el desarrollo de aplicaciones multiplataforma en lenguaje C#.

El código referente al desarrollo de la aplicación móvil se ha estructurado, como se puede ver en la figura 14, para facilitar el entendimiento del mismo. Se ha tratado de asemejar el diseño lo máximo posible al diagrama de componentes de la figura 10, pero hay algunas diferencias ya que cada componente en el diseño puede corresponder con más de un fichero en la implementación.

El código se divide en vistas que son las interfaces de usuario y en controladores que contienen la lógica de la aplicación. Las vistas son las encargadas de la representación de los datos a través de los elementos visuales que las componen y son empleadas por los usuarios para interactuar con la aplicación. Cómo se observa en la figura 14, para una mayor claridad a la hora de leer el código y estructurarlo se ha definido cada vista en un fichero XAML independiente (como se ha comentado anteriormente Xamarin Forms permite compartir la misma interfaz de usuario entre las diferentes plataformas).

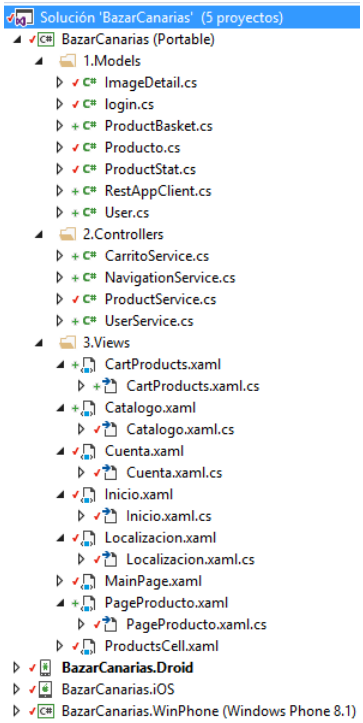


Figura 12. Organización del código de la aplicación.

Como se ha definido en el diagrama de componentes, cada vista se comunica con un controlador, que en la implementación comprende uno o más ficheros. En la figura 14 también se pueden observar los controladores codificados en lenguaje C#, cuyo código es también común a las distintas plataformas. Existen dos tipos de ficheros en este lenguaje, los ficheros con extensiones `.cs` y `.xaml.cs`. Estos últimos, son ficheros asociados a las vistas, contienen una parte de la lógica de negocio y la llamada a los ficheros con extensión `.cs`, dónde se implementa el resto de la lógica de negocio. Xamarin no permite separar los ficheros `.xaml` de los `.xaml.cs`, es por ello que la organización no es exactamente igual a la definida en el diagrama de componentes.

El código de los controladores es la parte más compleja de la aplicación, ya que en ellos se encuentran las llamadas a los servicios REST y el resto de funcionalidades de la aplicación, como la invocación de la aplicación de Google Maps entre otras. Es por ello por lo que se ha documentado y se han añadido comentarios al código como buenas prácticas de programación.

A modo de ejemplo se va a detallar, en cuanto a implementación, el proceso que se realiza para la consulta del catálogo en la aplicación móvil. En primer lugar, se accede a través de la interfaz de usuario a la vista correspondiente al Catálogo, que se muestra en la figura 15, y que se corresponde con el fichero `Catalogo.xaml` mostrado en la figura 16. Dicha interfaz muestra la barra de búsqueda de productos, la cual tiene asociado un *Binding* denominado `SearchText`. Seguido se muestra la lista de productos, que tiene asociado un *Binding* para cada elemento que se desea mostrar. La figura 16 muestra los *Bindings* (manejadores) comentados, los cuales están definidos en el correspondiente controlador asociado a la vista (`Catalogo.xaml.cs`), y constituyen el mecanismo de unión entre vista y controlador.

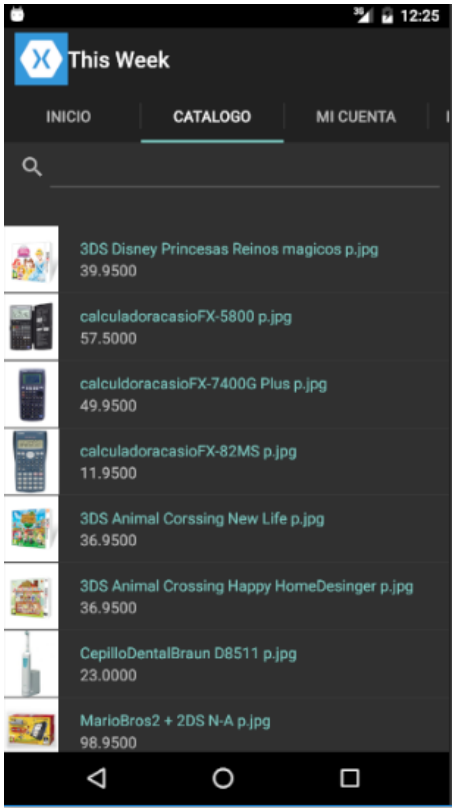


Figura 13. Vista del Catálogo.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:BazarCanarias;assembly=BazarCanarias"
             x:Class="BazarCanarias.Catalogo" Title="This Week"
             x:Name="RootPage">
  <ContentPage.Content>
    <StackLayout >
            <SearchBar Text="{Binding SearchText, Mode=TwoWay, Source={x:Reference RootPage}}" />
            <Label x:Name="Response"
          Text="[Response here]"
          />
            <ListView x:Name="lstView" ItemSelected="OnItemSelected">
                <ListView.ItemTemplate>
                    <DataTemplate>

                        <ImageCell Text="{Binding products_image}"
                         Detail="{Binding products_price}" ImageSource="{Binding FlagSource}" />

                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
        </StackLayout>
  </ContentPage.Content>
</ContentPage>
```

Figura 14. Código de la vista Catálogo.

El controlador define las variables necesarias para dar soporte a su funcionalidad. En este caso, además de los Bindings previamente mencionados, se definen dos objetos de la clase `ObservableCollection` del tipo `Producto`, una para almacenar el listado completo de productos del catálogo (`Products`) y otra para almacenar las listas filtradas (`Results`). Esta última lista es la que está automáticamente sincronizada con la lista de productos que se muestra en la vista Catalogo. También se define una variable del tipo `ProductService`, que es la clase donde se realiza la llamada a la API REST para, en este caso, obtener el listado de productos. En la figura 17 se observa una parte del código correspondiente al controlador del Catálogo (fichero `Catalogo.xaml.cs`), donde se inicializan las variables y se cargan los datos (método `Load`), haciendo uso del correspondiente método de la API REST.

```xml
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;

namespace BazarCanarias
{
    // Este es el catalogo de la aplicacion
    public partial class Catalogo : ContentPage
    {
        public ObservableCollection<Producto> Results { get; private set; }
        public ObservableCollection<Producto> Products { get; private set; }
        private ProductService _client;

        public Catalogo()
        {
            Results = new ObservableCollection<Producto>();
            Products = new ObservableCollection<Producto>();
            InitializeComponent();
            lstView.ItemsSource = Results;
            _client = new ProductService();
            Load();
        }
```

```
    public async void Load()
    {
      try
      {
          var result = await _client.getProductos();
          Results.Clear();
          foreach (var item in result)
          {
              Producto mProduct = item;
              mProduct.FlagSource =
                 ImageSource.FromUri(_client.GetImageSource(mProduct.products_image));
              Results.Add(mProduct);
              Products.Add(item);
          }
          Response.Text = string.Empty;
      }
      catch (Exception ex)
      {
              Response.Text = ex.Message;
              lstView.IsVisible = false;
      }
  }
```

Figura 15. Inicialización y carga de productos en el controlador del Catálogo.

En la figura 18 se observa el fragmento de código del controlador dónde se utiliza el binding `SearchText`. Primero se obtiene el texto introducido (producto a buscar) y en base a él, se llama al método `Filter` de la figura 18, que realiza la búsqueda de productos con esa coincidencia de nombre en el array de productos. En base a ello se actualiza el contenido de la lista `Results`, que se actualiza automáticamente en la vista.

En la clase `ProductService` se definen las URL necesarias para invocar el servicio REST, es concreto, se almacena en una variable la URL que contiene las imágenes del catálogo y en otra variable se guarda la URI del recurso Productos del servicio REST diseñado. Después se invoca el método GET para obtener la lista de productos y de nuevo el método GET para obtener las imágenes correspondientes a cada producto. En la siguiente figura 19 se observa un ejemplo de una invocación/llamada al servicio REST en la cual se retorna una lista de los productos de la tienda en formato JSON.

```
56          private string _searchText;
57          public string SearchText
58          {
59              get { return _searchText; }
60              set
61              {
62                  _searchText = value;
63                  OnPropertyChanged(nameof(SearchText));
64                  Filter();
65              }
66          }
67
68          private void Filter()
69          {
70              Results.Clear();
71
72              var query = Products.Where(item => item.Matches(SearchText)).ToArray();
73
74              foreach (var item in query)
75              {
76                  Results.Add(item);
77              }
78          }
```

Figura 16. Búsqueda de productos por nombre en el controlador del Catálogo.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BazarCanarias
{
    public class ProductService : RestAppClient
    {
        protected const string IMAGES_SERVICE = "http://www.bazarcanarias.eu/images/";


        public ProductService() : base("http://192.168.0.155/productos") { }

        //Obtenemos la lista de todos los productos
        public Task<IEnumerable<Producto>> getProductos()
        {
            return getAsJson<Producto>();
        }

        public Uri GetImageSource(string image)
        {
            return new Uri(FromUrl(IMAGES_SERVICE, image));
        }
    }
}

```

Figura 17. Fragmento de código de la llamada al método GET del recurso Producto.

Para obtener los productos en formato JSON, la clase `ProductService` extiende a la clase `RestAppClient`, en la cual se han definido métodos genéricos para realizar invocaciones sobre recursos en servicios REST. La clase debe recibir como parámetro en su construcción el URI del recurso al que se quiera acceder. A través del método `getAsJson` se consigue la representación del recurso, la lista de productos en este caso, se deserializa el objeto en formato JSON y se retorna. En la figura 20 se muestra el código del método `getAsJson` de la clase `RestAppClient` que, en este caso, retorna la representación de un recurso enviado en formato JSON.

Por otro lado, se ha hecho uso de la aplicación de Google Maps para calcular la ruta hacia las tiendas físicas. La aplicación se conecta con la aplicación de Google, y hace uso de la geolocalización para calcular la ruta entre la tienda seleccionada y la ubicación del dispositivo.

```
protected async Task<IEnumerable<T>> getAsJson<T>()
    where T : class
{
    var result = Enumerable.Empty<T>();

    using (var httpClient = new HttpClient())
    {
        httpClient.DefaultRequestHeaders.Accept.Clear();
        httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

        var response = await httpClient.GetAsync(baseUrl).ConfigureAwait(false);

        if (response.IsSuccessStatusCode)
        {
            var json = await response.Content.ReadAsStringAsync().ConfigureAwait(false);

            if (!string.IsNullOrWhiteSpace(json))
            {
                result = await Task.Run(() =>
                {
                    return JsonConvert.DeserializeObject<IEnumerable<T>>(json);
                }).ConfigureAwait(false);
            }
        }
    }
    return result;
}
```

Figura 18. Invocación de método GET sobre un recurso en la API REST.

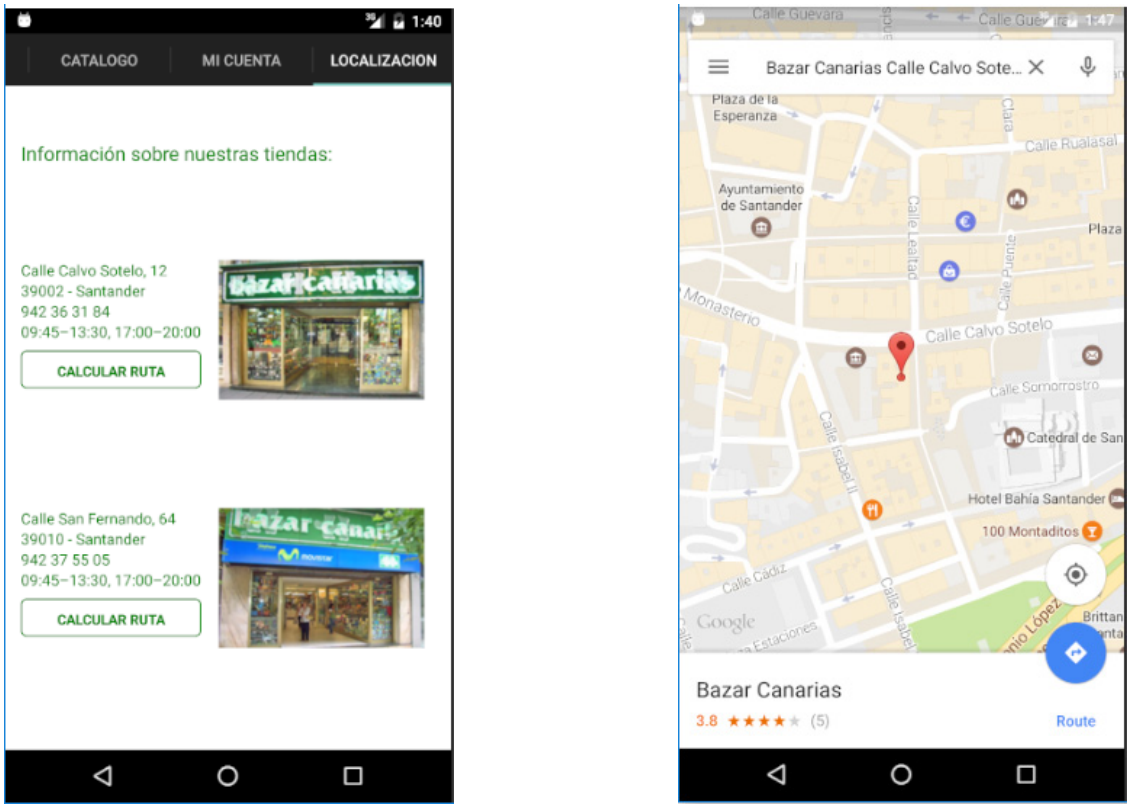


Figura 19. Vista Tiendas y Calcular ruta a tienda.

Se han encontrado dificultades en esta integración debido a que no se había implementado anteriormente nada parecido. Finalmente se ha encontrado la documentación adecuada, lo cual ha sido de gran ayuda para el desarrollo de esta integración. La parte que ha tenido mayor dificultad ha sido la creación de una API Key, la cual es necesaria para hacer uso de la aplicación de Google Maps en Android y asignar los permisos necesarios a la aplicación.

En la figura 21 se observa la vista final de las tiendas y la vista generada para la localización de una de las tiendas una vez pulsado el botón “Calcular Ruta” en la aplicación.

5.2 Implementación del servicio REST

Para la implementación de este servicio REST se va a hacer uso del framework Slim, cuya función principal es facilitar la creación de la API y establecer la comunicación entre el cliente y el servidor. Slim utiliza el lenguaje PHP. Todas las peticiones del servicio REST se ubican en un mismo módulo ya que no son un número elevado de peticiones, de forma que sea más sencillo hacer migraciones en un futuro y así esté todo organizado en un único fichero.

En la figura 22 se muestra, a modo de ejemplo, la implementación de la petición GET correspondiente a la consulta de la lista de productos. También una petición de tipo DELETE a través de la que se elimina un producto del carrito perteneciente al usuario que se especifica en el URI. Ambos métodos tienen la misma estructura, el parámetro `db` que se observa representa a la base de datos, cuyos datos de acceso se definen en un archivo denominado `dependences.php`. En base al tipo de petición y a los parámetros

de invocación del método se prepara la correspondiente query y se ejecuta en la base de datos. El resultado se almacena en una variable en el caso de GET y se traduce a formato JSON. En el caso de DELETE se elimina el producto y se guarda la respuesta para ver que todo ha sucedido correctamente.

Se ha optado por este modelo debido a su sencillez y flexibilidad a la hora de diseñar e implementar la API. Además, este módulo se puede añadir de forma muy sencilla al servidor Apache, donde reside Oscommerce, facilitando la interacción entre ambos.

```php
// get all de todos los productos
$app->get('/productos', function ($request, $response, $args) {
  $sth = $this->db->prepare("SELECT  * FROM PRODUCTS");
  $sth->execute();
  $todos = $sth->fetchAll();
  return $this->response->withJson($todos);
});

// Delete item with given id
$app->delete('/carrito/{idUsuario}/[{idLinea}]', function ($request, $response, $args) {
    $sth = $this->db->prepare("DELETE FROM customers_basket WHERE products_line_id=:idLinea
                               AND customers_id=:idUsuario");
    $sth->bindParam("idUsuario", $args['idUsuario']);
    $sth->bindParam("idLinea", $args['idLinea']);
    $sth->execute();
    return $this->response->withJson([array('success' => "success" )]);
});
```

Figura 20. Ejemplos de métodos GET y PUT de la API REST.

6. Pruebas

En este capítulo se expondrán las pruebas realizadas para evaluar el correcto funcionamiento del software. Esta es la última fase del desarrollo del proyecto y sirve para comprobar la calidad, integridad y fiabilidad del software.

Es muy importante realizar las pruebas correctamente, ya que una amplia batería de pruebas puede detectar defectos en el sistema que previamente no se habían visto. Pero no solamente eso, también así se asegura que el sistema cubre las necesidades y los requisitos definidos por el cliente.

Se van a realizar varios tipos de pruebas para cubrir todos los aspectos posibles y garantizar un software de calidad.

6.1 Pruebas unitarias

Las pruebas unitarias también llamadas pruebas modulares permiten determinar el correcto funcionamiento y la eficiencia de una unidad software aislada. Deben cumplir una serie de características para poder ser consideradas pruebas unitarias eficientes, una de las más importantes es que la ejecución de la prueba no acceda a recursos externos ni a las clases reales de las que la unidad a probar dependa ya que si no dejarían de ser pruebas unitarias. Destacan por tener tiempos de ejecución muy rápidos (cuestión de milisegundos). Además, deben cubrir la totalidad del código para garantizar que no haya partes sin probar, poder repetirse su ejecución tantas veces como se quiera y estar automatizadas. Generalmente las pruebas unitarias son tarea de los desarrolladores, ya que la persona que ha desarrollado esa parte del código conoce a la perfección y entiende el comportamiento esperado de ese código.

Se van a realizar pruebas unitarias tanto en la aplicación móvil como para probar el Servicio REST.

6.1.1 Pruebas unitarias de la aplicación móvil

Se han realizado pruebas unitarias para comprobar que el código desarrollado para la aplicación móvil es correcto y el resultado obtenido (tras la ejecución) es el esperado. Son fundamentalmente empleadas para reducir la cantidad de defectos en el código de la aplicación móvil, reducir costes en el desarrollo y detectar la cantidad de errores de una forma más rápida y efectiva.

Se ha hecho uso de los framework `NUnit` y `Xamarin.UITest` de Xamarin para realizar pruebas unitarias sobre pequeñas unidades funcionales de nuestra aplicación. Para ello se agrega un nuevo proyecto portable a la solución existente utilizando el framework `Xamarin.UITest`, que permite realizar pruebas de comportamiento de la aplicación. En este nuevo proyecto se crea una nueva clase test en la que residen todos los métodos implementados para realizar las pruebas unitarias. De esta manera y como buenas prácticas, todo queda modularizado para un mejor entendimiento.

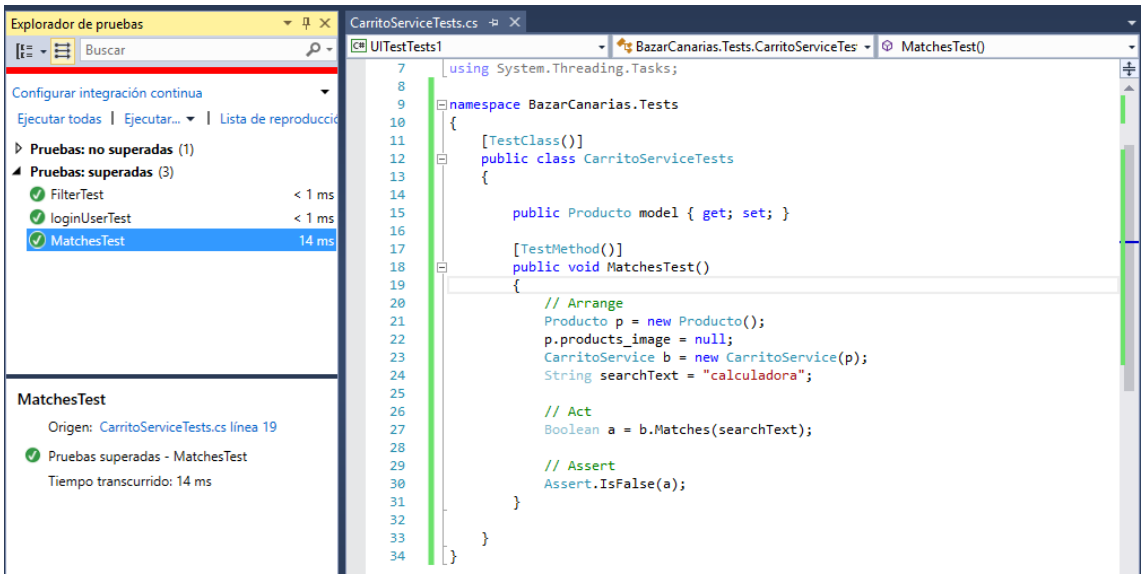


Figura 21. Código prueba unitaria método Matches.

Se ha implementado una amplia batería de pruebas unitarias para cubrir la mayor parte posible del código. En la figura 23 se observa el código de una prueba unitaria que comprueba el correcto funcionamiento del método `Matches`, que se encarga de comprobar la coincidencia de palabras cuando se realiza la búsqueda de productos por nombre. El resultado será un booleano, en este caso se ha probado la casuística de que el nombre del producto sea nulo, donde deberá devolver `False`.

Además, se debe comprobar el correcto funcionamiento de la aplicación en otras plataformas y sistemas operativos diferentes. `Xamarin.UITest` permite probar por ejemplo la apariencia de los diferentes elementos de las vistas al cambiar de dispositivo y que su comportamiento sea el esperado.

6.1.2 Pruebas unitarias del Servicio REST

A medida que se ha ido implementando el servicio REST se han ido realizando pruebas unitarias para comprobar el correcto funcionamiento de los métodos de la API. Para ello, se ha utilizado la herramienta Advanced REST Client, que permite probar todos los métodos y peticiones HTTP de la API personalizada de una forma muy sencilla. Simula un cliente REST y establece la conexión directamente en el *socket*, obteniendo un control completo sobre la conexión y sobre las cabeceras de la petición/respuesta.

Se ha realizado una batería de pruebas que cubre todos los escenarios posibles, probando todos los métodos HTTP y sus posibles respuestas. A continuación, en la figura 24 se muestra un ejemplo de una petición HTTP invocando el método GET para obtener información de un producto determinado.

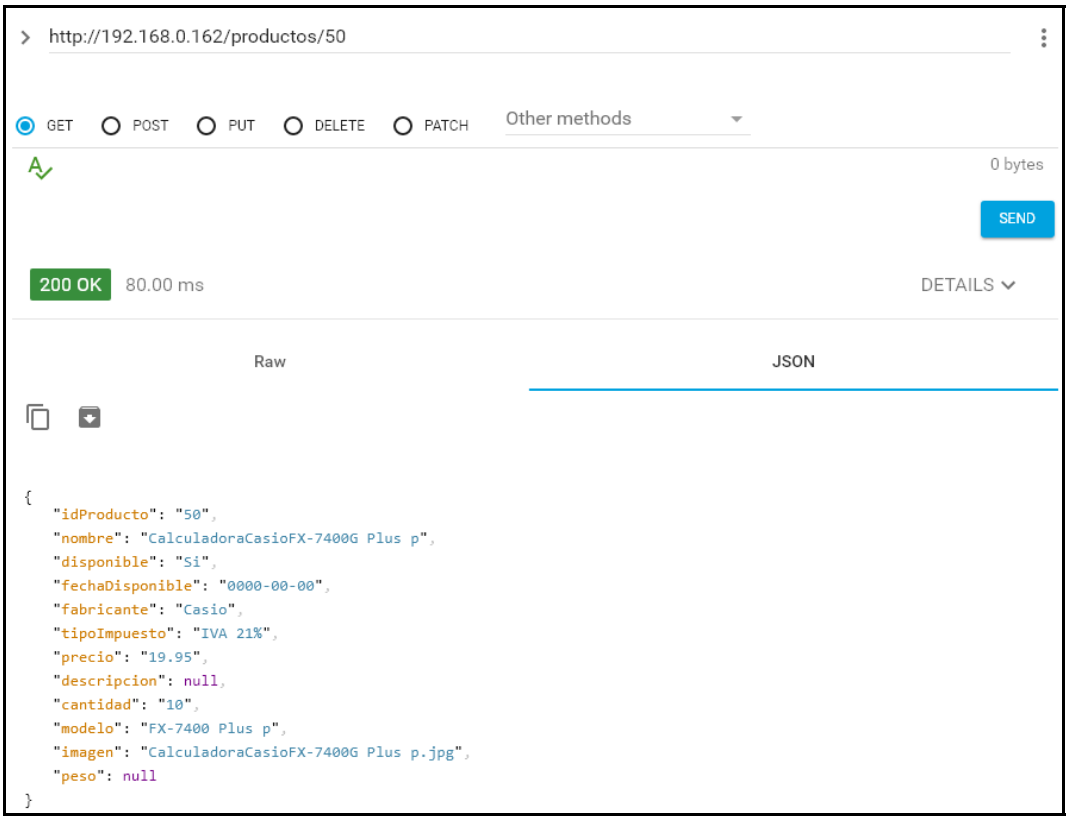


Figura 22. Invocación del método GET en la API REST.

6.2 Pruebas de integración

Las pruebas de integración se realizan una vez han sido superadas las pruebas unitarias, su función es comprobar que el conjunto de componentes de la aplicación funciona correctamente de manera conjunta, es decir comprueba las interacciones entre las unidades software.

En una primera fase las pruebas de integración se han llevado a cabo en un entorno simulado, para ello se ha creado una base de datos local en el equipo de trabajo simulando la base de datos de la empresa. Esta fase de pruebas iniciales se ha realizado probando métodos simples manualmente desde la aplicación móvil, como la búsqueda de un producto, el cual invocaba la API REST diseñada y ésta se conectaba con la base de datos simulada. De esta forma se consigue probar todos los componentes del sistema y la interacción entre ellos.

Finalmente se han realizado pruebas de integración con la base de datos real de la empresa y el resultado ha sido el esperado. Se ha realizado la misma batería de pruebas que para la base de datos local, probando todas las funcionalidades de la aplicación. Estas pruebas se han realizado de forma manual de modo que se han cubierto todos los casos descritos en la fase de análisis del capítulo 3.

6.3 Pruebas de sistema

Son las primeras pruebas una vez esté construido el sistema, comprueban a fondo la funcionalidad e integridad del sistema en un entorno lo más parejo al entorno final. Deben cubrir tanto los aspectos funcionales como los no funcionales del sistema, para ello se basan en los requisitos establecidos en la fase de análisis.

Se han realizado pruebas de sistema para cada requisito establecido, comprobando el resultado de diferentes posibles escenarios. Un ejemplo de prueba de sistema para probar la seguridad y funcionalidad del sistema ha sido realizar el login de un usuario en diferentes escenarios, “login cuando no hay conexión a internet”, “login cuando si hay conexión a internet”, “login de usuario incorrecto”, “login de usuario correcto” y derivados.

Para comprobar el correcto funcionamiento de las pruebas, se han utilizado diferentes versiones de Android y diferentes dispositivos simulados, para alcanzar una batería de pruebas lo más amplia posible.

6.4 Pruebas de aceptación

Son las pruebas de aceptación por parte del usuario, que determinan el grado de satisfacción del cliente con el producto final, es decir, si cumple las expectativas y los requisitos establecidos. Son realizadas sobre el software finalizado e integrado, en un entorno que imite al entorno de uso habitual y se basan fundamentalmente en probar la funcionalidad del sistema. Son muy importantes ya que son la última fase de pruebas antes de que el producto sea finalmente entregado al cliente. Además, abren paso a nuevas fases del proyecto como son el despliegue y el mantenimiento.

Generalmente este tipo de pruebas no son realizadas por los desarrolladores, pero en este caso como el cliente y el desarrollador son personas muy cercanas, se han llevado a cabo pruebas de aceptación del tipo Alfa, que son un tipo de pruebas de aceptación realizadas por desarrolladores en las que el cliente se limita a observar el resultado de las pruebas.

Como resultado de estas pruebas, decir que se han sugerido pequeños cambios en la funcionalidad y apariencia de las interfaces. También han aparecido nuevas ideas para desarrollarse en un futuro proyecto.

7. Conclusiones y trabajos futuros

Una vez finalizado el proyecto se ha hecho una reflexión y se han sacado una serie de conclusiones acerca del trabajo realizado, para darse cuenta de las cosas que se pueden mejorar, las dificultades que han surgido durante el desarrollo y facilitar la planificación y ampliación de este proyecto en un futuro.

7.1 Conclusiones

La finalidad de este proyecto consistía en crear una aplicación móvil para mejorar el canal de venta de la empresa Bazar Canarias gracias a la venta de sus productos a través de ésta y así conseguir llegar a nuevos clientes y fidelizar los ya existentes. También, gracias a la geolocalización, permitir a los usuarios obtener la ruta y localización de las tiendas físicas existentes.

Para lograr esto y aprovechar el trabajo ya existente, la aplicación se comunica con la base de datos creada por la empresa y que contiene toda la información actualizada. De esta manera se beneficia del trabajo hecho y es coherente con los datos de la página web.

El diseño inicial era bastante costoso y la verdad que no se ha podido implementar todo lo que se había diseñado en un principio, pero sí se han realizado las funcionalidades más importantes, que son la consulta de productos en el catálogo, añadir o borrar productos del carrito de compra, y calcular la ruta hacia las tiendas físicas. Por complejidad y tiempo se tuvieron que dejar para más adelante funcionalidades como la compra de artículos a través de la aplicación ya que requería una lógica bastante costosa y se excedía de lo requerido para este trabajo. A pesar de que no se hayan implementado estas funcionalidades, el diseño sí se ha realizado para facilitar la extensión/ampliación de la aplicación móvil en un futuro.

Se decidió utilizar una herramienta bastante novedosa como es Xamarin (y que está en continuo crecimiento gracias al impulso de Microsoft) para el desarrollo de esta aplicación móvil multiplataforma.

Gracias a este proyecto he aprendido a programar en un lenguaje de programación nuevo para mí, como es C#, he aprendido a utilizar nuevas herramientas para el desarrollo de una aplicación móvil multiplataforma y a manejar un nuevo entorno de desarrollo integrado como es Visual Studio. También he aprendido a diseñar e implementar un servicio REST, a realizar su integración con una base de datos ya existente y con la aplicación móvil. Todos estos conceptos eran nuevos para mí, desconocía esta rama de software al no haber cursado nunca asignaturas que contuvieran esta temática.

El proyecto me ha servido para enfrentarme a nuevos retos, y pese a las dificultades encontradas y los cambios realizados en el transcurso del proyecto, poder lograr el objetivo inicial.

7.2 Trabajos futuros

El mantenimiento del software a menudo incluye mejoras en aspectos como rendimiento, corrección de errores y mejoras o novedades en el software. En este apartado se van a tratar las posibles mejoras o novedades que se pueden incluir en la aplicación en el futuro.

En una primera versión se pretendía que la aplicación fuera capaz de realizar todo el proceso de compra de un producto, pero debido a la complejidad que suponen requisitos tan importantes como la seguridad finalmente se decidió acotar algunas funcionalidades que se habían diseñado previamente. Toda la parte que corresponde al administrador del sistema en la aplicación móvil como los métodos DELETE y POST de productos no se ha implementado. No obstante, se ha realizado el diseño de manera que en un futuro sea posible realizar estas ampliaciones, facilitando así la implementación de dichas funcionalidades.

Una mejora que ha surgido a raíz del desarrollo de la aplicación es la creación de un chat de ayuda para clientes a través de la aplicación móvil, por el cual los clientes registrados podrán hacer consultas sobre las características y funcionalidades de los productos. Estas mejoras requieren conexión a internet para su ejecución, y es a raíz de esta problemática cuando ha surgido una nueva mejora, que consiste en poder añadir productos al carrito de compra sin conexión a internet, de manera que se almacenen los datos localmente mientras no haya conexión y se envíen cuando la haya.

También sería deseable implementar un sistema de asesoramiento de productos para clientes basado en inteligencia artificial, que en función de los gustos de cada cliente recomiende un tipo de productos u otro. El objetivo es ayudar al cliente a encontrar el producto deseado y para ello se quiere implementar a futuro la organización y búsqueda de productos por categoría. Son trabajos que se pretenden incluir en un futuro en la aplicación móvil de manera que se siga mejorando y ampliando este proyecto.

Referencias

[1] Efe (2016). El comercio electrónico mueve 5.414 millones hasta marzo, el 21,5 % más. elDiario.es. Recuperado de http://www.eldiario.es/economia/comercio-electronico-mueve-millones-marzo_0_566593445.html.

[2] Sommerville, Ian (2011). Ingeniería del Software 7ª edición. Addison Wesley.

[3] (2009). Cap 3 - QA&V Software Development Life cycle. Recuperado de http://raknarrok.blogspot.com.es/2015/05/cap-3-software-development-life-cycle.html.

[4] MagicDraw UML. http://www.nomagic.com/products/magicdraw.html.

[5] Xamarin. https://www.xamarin.com.

[6] Visual Studio. https://www.visualstudio.com/.

[7] Android SDK. https://developer.xamarin.com/guides/android/application_fundamentals/using-the-sdk-manager/.

[8] NinjaMock. https://ninjamock.com/home/index.

[9] Fielding, Roy T.; Taylor, Richard N. (2002). Principled Design of the Modern Web Architecture. ACM Transactions on Internet Technology 2 (2): (pp. 115–150).

[10] Gregorio, Joe (2004). How to create a REST Protocol. XML.com website. Recuperado de http://www.xml.com/pub/a/2004/12/01/restful-web.html.

[11] Slim framework. https://www.slimframework.com/.

[12] World Wide Web Consortium y Internet Engineering Task Force (1999), Hypertext Transfer Protocol -- HTTP/1.1. https://tools.ietf.org/html/rfc2616.

[13] Sublime Text. https://www.sublimetext.com/.

[14] Apache. https://httpd.apache.org/.

[15] Oscommerce. https://www.oscommerce.com/.

[16] Advance REST Client. https://advancedrestclient.com/.

[17] PHP. http://php.net/manual/es/intro-whatis.php.

[18] Massé, Mark (2011), REST API Design Rulebook. O'Reilly Media.