UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DESARROLLO DE UNA APLICACIÓN DE AYUDA AL COMERCIO LOCAL PARA DISPOSITIVOS ANDROID

Carlos Domingo Barrero

Tutor: Francisco José Martínez Zaldívar

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-20

Valencia, 9 de septiembre de 2020

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

Durante la última década, el auge de las compras por internet y la aparición de grandes plataformas como Amazon o AliExpress han provocado una caída drástica de las ventas e ingresos de las tiendas locales. Además, el mundo ha sido golpeado por la pandemia de COVID-19, provocando cambios drásticos en la sociedad y en la forma de vida de las personas, así como una crisis económica sin precedentes que puede provocar el cierre definitivo de numerosos comercios locales.

Con el fin de ayudar a esos comercios a salir adelante, en este TFG se va a desarrollar una aplicación móvil para dispositivos Android, con la que las tiendas podrán poner a disposición de los usuarios su catálogo de productos, sus precios y sus ofertas. Los usuarios de la aplicación podrán buscar el producto que deseen y ver los comercios cercanos donde pueden encontrarlo, sin tener que salir de casa. Para ello se van a utilizar herramientas como Android Studio o los Servicios de Google Maps.

Resum

Durant l'última dècada, l'auge de les compres per internet i l'aparició de grans plataformes com Amazon o AliExpress han provocat una caiguda dràstica de les vendes i ingressos de les botigues locals. A més, el món ha sigut colpejat per la pandèmia de COVID-19, provocant canvis dràstics en la societat i en la forma de vida de les persones, així com una crisi econòmica sense precedents que pot provocar el tancament definitiu de nombrosos comerços locals.

Amb la finalitat d'ajudar aqueixos comerços a tirar avant, en aquest TFG es desenvoluparà una aplicació mòbil per a dispositius Android, amb la qual les botigues podran posar a la disposició dels usuaris el seu catàleg de productes, els seus preus i les seues ofertes. Els usuaris de l'aplicació podran buscar el producte que desitgen i veure els comerços pròxims on poden trobar-lo, sense haver d'eixir de casa. Per a això s'utilitzaran eines com Android Studio o els Serveis de Google Maps.

Abstract

During the last decade, the “boom” of online shopping and the appearance of large platforms such as Amazon or AliExpress have produced a dramatic drop in sales and revenue of local shops. Also, recently the world has been hit by the COVID-19 pandemic, causing drastic changes in society and lifestyle, as well as an unprecedented economic crisis that can cause the final closure of numerous local businesses.

In order to help these local shops to overcome the crisis, this Final Degree Project consist on the development of a mobile application for Android devices, in which shop owners can show their product catalog, prices and discounts. Users can search for any desired products and visualize nearby shops where they can find them, without leaving home. For the development, tools like Android Studio or Google Maps Services will be used.

Índice

Capítulo 1. Introducción

Los grandes avances tecnológicos surgidos en las últimas décadas en materia de telecomunicaciones, electrónica e informática han supuesto toda una revolución en la forma de vida de personas, empresas e instituciones de todo el mundo. La creación de microprocesadores cada vez más potentes y eficientes, y el desarrollo de estándares y tecnologías de comunicación cada vez más rápidos, han hecho posible la evolución de los teléfonos móviles: de ser simples dispositivos de comunicación que sólo permitían hacer llamadas y enviar mensajes de texto de unos pocos caracteres, a ser auténticos ordenadores de bolsillo que permiten una infinidad de usos y posibilidades mediante el uso de aplicaciones e Internet.

El surgimiento de redes sociales, formas de mensajería instantánea y gratuita como WhatsApp o Telegram y aplicaciones móviles de todo tipo ha hecho que, hoy en día, los teléfonos inteligentes o *smartphones* sean un dispositivo esencial e imprescindible para la gran mayoría de la población mundial. Pero la revolución no solo ha afectado a los teléfonos móviles, sino que ha llegado a dispositivos de todas las clases con el *Internet of Things* (IoT): televisiones, relojes, cámaras, robots aspiradores, herramientas de domótica… Todos ellos se conectan de forma inalámbrica a las redes Wi-Fi de nuestras casas y pueden ser controlados desde cualquier parte del mundo mediante aplicaciones instaladas en nuestros teléfonos móviles.

Según el informe sobre consumo, uso y tendencias del móvil en España realizado por "DiTrendia", el 97% de los españoles con acceso a internet dispone de un smartphone, que se ha convertido en el dispositivo más empleado para acceder a internet. De hecho, 1 de cada 3 usuarios únicamente utiliza su smartphone para ello. Se estima que en 2019 cada persona utilizó el teléfono móvil, de media, 3 horas y 22 minutos diarios, siendo WhatsApp la aplicación más utilizada, seguida de cerca por navegadores, redes sociales y aplicaciones de compras [1].

Las compras a través de Internet han aumentado de forma exponencial en la última década, llegando a suponer un volumen de ventas de 41.509 millones de euros en España en el año 2018, incrementándose en un 32.4% respecto al año anterior. Un 67.4% de los internautas realizó alguna compra on-line durante dicho año [2].

Como contrapartida, el comercio de proximidad y las tiendas de barrio viven cada vez una situación más difícil. Factores como la comodidad de comprar desde cualquier sitio, disponer de un catálogo de productos prácticamente infinito, poder comparar precios entre tiendas en cuestión de segundos o la entrega de pedidos en la puerta de casa, han hecho que cada vez más gente, sobre todo los jóvenes, opte por realizar sus compras on-line. Según datos de la Unión de Profesionales y Trabajadores Autónomos (UPTA), solo durante el tercer trimestre de 2019 cerraron 5310 comercios de proximidad [3]. Esta situación está provocando la "muerte" de los barrios, ya que cada vez es más frecuente encontrar locales sin ningún tipo de uso, incluso dentro de centros comerciales, y no parece que esta situación se vaya a revertir: la llegada de la pandemia de COVID-19 ha provocado que este cambio en el hábito de consumo se haya agravado. Según el INE, tras la declaración del Estado de Alarma, en marzo de 2020 las ventas en el comercio minorista se desplomaron un 14.7% respecto al mismo mes del año pasado, mientras que en abril y mayo el descenso fue del 31.6% y 20.1% respectivamente [4]. Tras el fin del Estado de Alarma, el descenso es menor (3.3% y 3.7% en junio y julio), pero la tendencia no se revierte. Sin embargo, durante ese mismo periodo, las ventas por internet aumentaron drásticamente, llegando a un 64.1% de incremento en mayo [5].

Actualmente, los comercios de proximidad pueden utilizar las redes sociales para publicitarse y llegar a miles de clientes, pero esto implica la creación de perfiles en redes con las que puede que no estén familiarizados y conseguir que sus potenciales clientes les sigan. Además, las redes sociales no ofrecen una estructura que permita a los comercios exponer sus productos y servicios con claridad, y a los clientes encontrarlos con facilidad. También entran en juego factores como el posicionamiento, que pueden hacer que sus publicaciones no aparezcan, o aparezcan en exceso, molestando a los clientes y consiguiendo el efecto contrario.

Capítulo 2. Objetivos

Por todos los motivos expuestos anteriormente, el objetivo de este Trabajo Final de Grado es aplicar todos los conocimientos adquiridos durante mi etapa universitaria para desarrollar una aplicación para dispositivos móviles que ofrezca a los comercios locales una plataforma con la que puedan mostrar sus productos de una forma clara y sencilla a sus potenciales clientes. Los clientes pueden ver de un simple vistazo todas las ofertas y precios publicados de un determinado producto o servicio a su alrededor y, a su vez, pueden ayudar a los comercios publicando las ofertas que vean cuando visiten sus tiendas, si es que no están ya publicadas. De esta forma se pretende introducir a los comercios locales en el mundo de internet más allá de las redes sociales y hacerlos más atractivos de cara al público, intentando incrementar sus ventas y revertir la situación tan dramática que están viviendo.

Para llegar al objetivo final, se pensó en la estructura y modelo que debía tener la aplicación, siendo el más adecuado el modelo cliente-servidor, por lo que el objetivo inicial se puede dividir en tres objetivos:

- Diseñar e implementar una base de datos relacional donde se guarde toda la información necesaria para el funcionamiento de la aplicación móvil.
- Crear un servidor que implemente una API REST, que hará de intermediario entre la aplicación móvil y la base de datos, gestionando las peticiones y devolviendo la información solicitada.
- Diseñar una aplicación para dispositivos móviles con una estructura e interfaz sencilla.

Además, con este Trabajo de Fin de Grado pretendo reforzar y ampliar los conocimientos que he adquirido en la universidad, aprendiendo a usar nuevas herramientas y lenguajes de programación que, hasta ahora, no he tenido la oportunidad de estudiar, y que están muy demandados en el mercado laboral.

Capítulo 3. Metodología de trabajo

3.1 Gestión del proyecto

Dentro del desarrollo del software encontramos múltiples metodologías diferentes, desde las clásicas como el modelo en cascada, donde se establecen una serie de fases a ejecutar de forma rígida y lineal, y no se empieza una hasta que termina la anterior, hasta metodologías ágiles donde el cliente interviene activamente, se sigue un ciclo de desarrollo iterativo, dinámico y flexible, y se realizan pruebas, correcciones y cambios de forma continua hasta llegar al producto final.

Para este Trabajo de Fin de Grado, puesto que no hay un equipo de desarrollo ni un cliente como tal ya que es un trabajo individual ideado y desarrollado por la misma persona, se ha empleado el modelo de desarrollo en cascada, en el que se establecen cinco fases: requisitos, diseño, implementación, verificación y mantenimiento. A estas fases añadí una fase extra de aprendizaje, que se desarrolló de manera simultánea a las demás y que se extendió hasta prácticamente el final del trabajo, ya que pude comprobar que necesitaba ampliar mis conocimientos para poder desarrollarlo correctamente. En esta fase realicé una búsqueda de información, guías y tutoriales sobre los entornos de desarrollo y lenguajes de programación que se van a usar para el desarrollo de la aplicación.

En la fase de requisitos se describen las características y necesidades de la aplicación que se va a desarrollar sin entrar en aspectos técnicos, las cuales ya han sido descritas en los capítulos anteriores. A partir de los objetivos, el trabajo se ha dividió en tres grandes bloques:

1. Desarrollo de la base de datos
2. Desarrollo del servidor y la API REST.
3. Desarrollo de la aplicación móvil

Antes de pasar a la fase de diseño, se deben establecer las plataformas para las que va a desarrollarse la aplicación y las tecnologías que se van a utilizar para cada uno de los bloques.

En este caso, los conocimientos adquiridos en las asignaturas de Programación y Aplicaciones Telemáticas han sido determinantes para decantarme por Android como la plataforma para la que desarrollar la aplicación. Además, es una plataforma con la que estoy ampliamente familiarizado, ya que llevo utilizando dispositivos que utilizan este sistema operativo desde 2010. El hecho de no disponer de ningún dispositivo Apple y desconocer su funcionamiento también fue un factor decisivo para elegir desarrollar únicamente para Android en vez de utilizar alguna herramienta multiplataforma. Para el desarrollo he utilizado el entorno de desarrollo oficial de Google *Android Studio*, ya que integra todas las herramientas necesarias para el desarrollo de aplicaciones Android, así como un emulador que facilita las labores de testeo.

Para la base de datos relacional he optado por utilizar el sistema MySQL de Oracle en su versión *Community*, ya que es una tecnología de código abierto ampliamente extendida y distribuida bajo licencia pública GNU, y se adecúa a las necesidades de la aplicación. El uso de esta tecnología me permite poner en práctica los conocimientos adquiridos en la asignatura de Sistemas Telemáticos para la Gestión de la Información y en mis prácticas en empresa. Para el diseño y gestión de la base de datos he utilizado la herramienta oficial *MySQL Workbench*, que provee una interfaz gráfica de configuración y uso bastante sencillo y simplifica en gran medida la creación de la base de datos.

Por último, para el desarrollo del servidor y la API REST barajé dos posibilidades. La primera era crear un servidor Apache y utilizar el lenguaje de programación PHP para desarrollar la API mediante la herramienta XAMPP, que ya utilicé en la asignatura de Sistemas Telemáticos para la Gestión de la Información. Además, la combinación PHP + MySQL es una de las más utilizadas en el desarrollo de aplicaciones web y es ampliamente aceptada por servidores en la nube, por lo que, en caso de decidir alojar el servidor en la nube, esta opción sería la más fácil.

La segunda opción era implementar la API utilizando Node.js y JavaScript. Esta tecnología es mucho más reciente y, a diferencia de PHP, es muy útil a la hora gestionar peticiones multihilo de forma asíncrona y de desarrollar programas escalables. Además, tiene soporte para servidor incorporado, por lo que no sería necesario instalar ninguna herramienta adicional para crear el servidor. Sin embargo, es una tecnología en general más complicada, requiere más código y no está tan aceptada por servidores en la nube.

Aunque, a priori, la primera opción parece más recomendable, hubo un factor determinante que hizo que me decantara por utilizar Node.js. Una de las metas que me marqué para el desarrollo de este Trabajo de Fin de Grado era adquirir conocimientos que me fueran útiles de cara al mercado laboral, y, en ese aspecto, Node.js y JavaScript son tecnologías que llevan años en pleno auge, mientras que PHP cada vez se utiliza menos. Según el informe anual de 2019 elaborado por StackOverflow[1] basado en encuestas a desarrolladores, JavaScript es el lenguaje de programación más popular a nivel mundial, utilizado por un 67.8% de los desarrolladores, mientras que PHP sólo es utilizado por el 26.4%. El mismo informe dice que Node.js es una de las tecnologías más demandadas y es utilizada por el 49.9% de los desarrolladores [6]. Puesto que ambas opciones implican el uso de tecnologías que son nuevas para mí y debía aprenderlas desde cero, opté por aprender JavaScript y Node.js.

Puesto que es una aplicación que se va a desarrollar de manera conceptual para el Trabajo de Fin de Grado y no se va a poner en producción, por simplicidad, decidí que el servidor se ejecutara de forma local en mi propio ordenador, en vez de subirlo a la nube.

Una vez definidos los requisitos, descargué e instalé todas las herramientas necesarias para el desarrollo y pasé a la fase de diseño, en la que se desarrollan con detalle los bloques definidos anteriormente. En primer lugar, se definió la base de datos: tablas, campos necesarios, tipos de datos, restricciones y relaciones. Después, los otros dos bloques se dividieron en pequeños bloques correspondientes a cada una de las funcionalidades que se quiere implementar en la aplicación. Todos estos aspectos se definirán con mayor detalle más adelante.

En la fase de implementación fui programando cada una de las partes diseñadas en las fases anteriores, con la ayuda de tutoriales. En primer lugar, creé la base de datos. Posteriormente creé el servidor e implementé la base de la API REST, y por último fui desarrollando la aplicación móvil de forma simultánea al resto de la API, ya que cada una de las funcionalidades de la aplicación requiere su correspondiente funcionalidad en la API.

En la fase de verificación realicé las pruebas necesarias para comprobar que todas las funcionalidades programadas se ejecutan correctamente, y corregí todos los fallos que encontré.

Por último, en la fase de mantenimiento modifiqué algunos aspectos visuales de la aplicación con los que no estaba del todo conforme y reestructuré parte del código para dejarlo más claro.

[1] StackOverflow es una comunidad web de programación fundada en 2008 con más de 50 millones de usuarios, donde se plantean y responden dudas.

3.2 Distribución en tareas

Las tareas en las que se ha dividido la elaboración del trabajo se corresponden con las fases descritas en el apartado anterior:

1. Requisitos:
    - Búsqueda de ideas para la aplicación.
    - Búsqueda de información sobre tecnologías y lenguajes de programación.
    - Definición de funcionalidades.
    - Elección de plataformas, lenguajes de programación y metodología de trabajo.
    - Descarga de programas y entornos de desarrollo necesarios.
2. Diseño:
    - Diseño de la estructura de la base de datos.
    - Diseño de la estructura de la aplicación y división en pequeñas tareas según las funcionalidades definidas.
    - Diseño del servidor y la API REST.
3. Implementación:
    - Creación de la base de datos.
    - Creación del servidor y la base de la API REST.
    - Programación simultánea de la aplicación y el resto de la API.
4. Verificación:
    - Comprobación de la correcta funcionalidad del proyecto.
    - Corrección de errores.
5. Mantenimiento:
    - Modificación de la interfaz de la aplicación
    - Reestructuración del código
6. Aprendizaje:
    - Búsqueda y lectura de guías sobre el uso de Android Studio.
    - Búsqueda y lectura de guías de instalación y uso de Node.js.
    - Búsqueda y visualización de tutoriales de aprendizaje de JavaScript.
    - Búsqueda y visualización de guías y tutoriales sobre desarrollo Android en Java.
7. Desarrollo de la memoria del Trabajo de Final de Grado.

3.3 Diagrama temporal

La elaboración de este Trabajo de Fin de Grado se ha compaginado con la realización de prácticas en empresa. Eso, sumado a la situación de excepcionalidad vivida en los últimos meses por la pandemia de COVID-19 ha hecho que el tiempo disponible haya sido mucho más limitado de lo previsto inicialmente. En el siguiente diagrama se puede ver el tiempo empleado en cada una de las tareas:

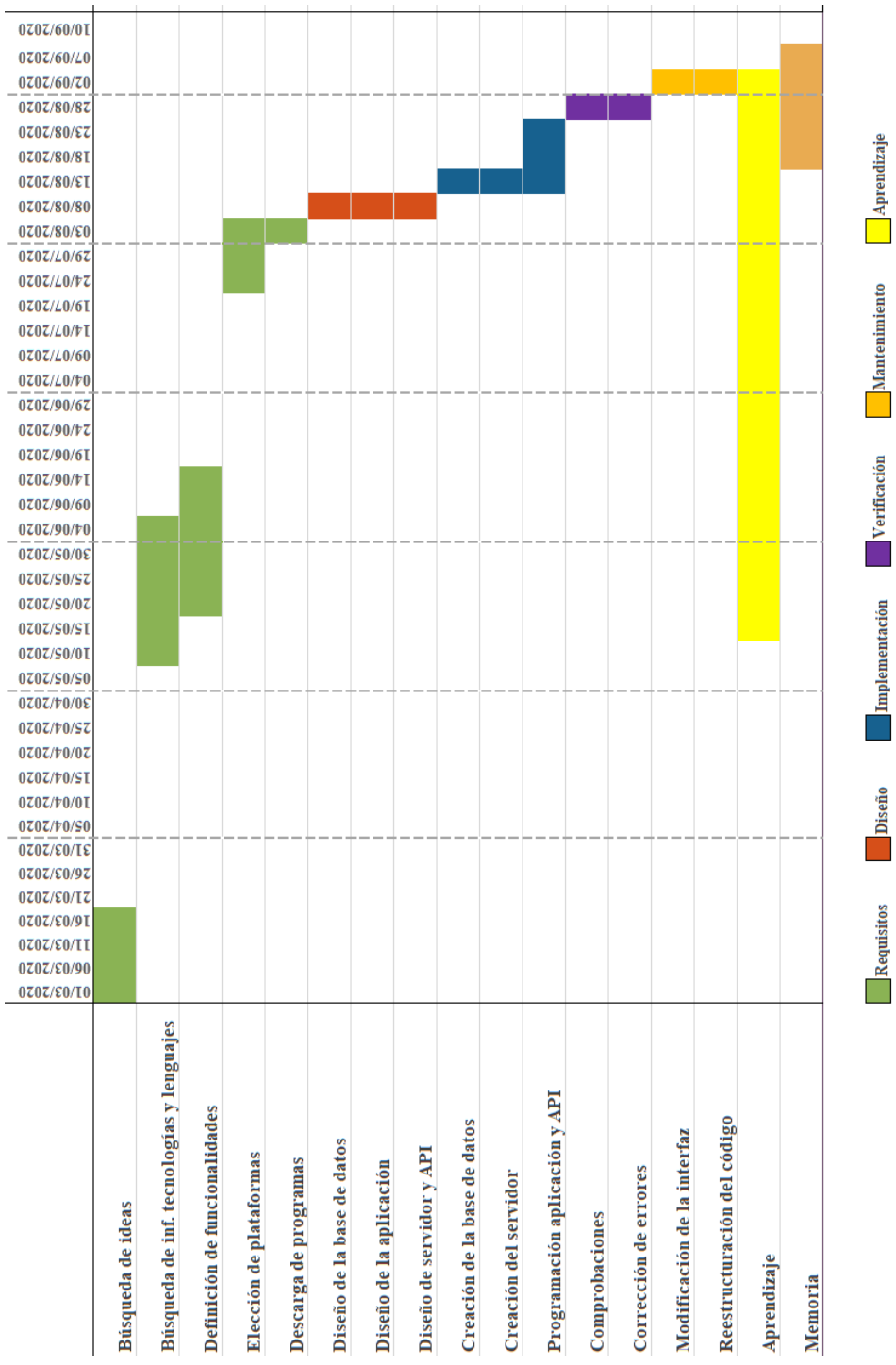

Figura 1. Diagrama temporal

Capítulo 4. Funcionalidad de la aplicación

En este capítulo se van a describir con detalle las funciones de la aplicación que se desarrolla en este Trabajo de Fin de Grado.

La aplicación, llamada "ComercioLocalApp", pretende ser una plataforma en la que comercios y clientes puedan compartir ofertas y comparar precios de una manera rápida, sencilla y eficaz. Por lo tanto, se establecen dos tipos de usuario: cliente y tienda.

4.1 Usuario de tipo cliente

Los usuarios de tipo cliente pueden realizar las siguientes acciones:

- **Registro:** los clientes se pueden registrar en la aplicación como usuario de tipo cliente introduciendo un nombre de usuario, una contraseña, un correo electrónico y sus datos personales.
- **Iniciar sesión:** los clientes pueden acceder a la aplicación introduciendo el usuario y la contraseña con la que se registraron.
- **Buscar ofertas:** los clientes pueden acceder al buscador de ofertas, donde podrán introducir la cadena de texto que quieren buscar, ver los resultados en un mapa y filtrarlos según el tipo de producto o la distancia a la que se encuentra.
- **Ver ofertas**: una vez hayan realizado una búsqueda, podrán ver con detalle la oferta que han seleccionado.
- **Añadir ofertas**: los clientes pueden añadir ofertas que hayan visto en alguna tienda para compartirla con el resto de usuarios de la aplicación y que todos puedan beneficiarse de ella. Para ello tendrán que seleccionar el producto en oferta, la tienda que lo ofrece y rellenar un formulario con la información necesaria.
- **Añadir tiendas**: en el caso de que un cliente quiera añadir una oferta de una tienda que no está registrada en la aplicación, podrá añadirla manualmente. La tienda quedará registrada como "no verificada" para informar a los usuarios de que la información asociada a esa tienda no es oficial y puede contener fallos.
- **Añadir productos**: cuando un usuario quiera añadir una oferta y el producto no esté registrado en la aplicación, podrá añadirlo introduciendo los datos necesarios en un formulario.
- **Gestionar ofertas**: los clientes podrán ver, modificar y eliminar las ofertas que hayan añadido.
- **Ver perfil**: los clientes pueden ver los datos personales con los que se han registrado en la aplicación.
- **Actualizar el perfil**: los clientes podrán modificar algunos datos de su perfil para mantenerlo actualizado.
- **Eliminar cuenta**: los clientes podrán darse de baja de la aplicación y eliminar sus datos y ofertas registradas.
- **Cerrar sesión**: los clientes podrán cerrar la sesión y salir de la aplicación.

4.2 Usuario de tipo tienda

Los propietarios de tiendas pueden realizar las siguientes acciones:

- **Registro:** los propietarios se pueden registrar en la aplicación como usuario de tipo tienda introduciendo un nombre de usuario, una contraseña, un correo electrónico y sus datos personales.
- **Iniciar sesión:** los propietarios pueden acceder a la aplicación introduciendo el usuario y la contraseña con la que se registraron.
- **Buscar ofertas:** los propietarios pueden acceder al buscador de ofertas, donde podrán introducir la cadena de texto que quieren buscar, ver los resultados en un mapa y filtrarlos según el tipo de producto o la distancia a la que se encuentra.
- **Añadir tiendas:** los propietarios pueden registrar sus comercios en la aplicación. Para ello tendrán que seleccionar en un mapa la ubicación de la tienda, rellenar un formulario con la dirección y otro con los datos de la tienda. Se puede dar el caso de que una misma persona sea la propietaria de varios comercios, por lo que el mismo usuario podrá añadir tantas tiendas como tenga a la misma cuenta sin necesidad de registrarse múltiples veces.
- **Modificar tiendas:** un propietario puede modificar los datos de una tienda registrada.
- **Eliminar tienda:** el propietario puede eliminar las tiendas creadas y todos sus datos asociados.
- **Añadir ofertas**: los propietarios pueden añadir ofertas a sus tiendas. Para ello tendrán que seleccionar la tienda, el producto en oferta, y rellenar un formulario con la información necesaria.
- **Crear catálogos**: un propietario puede añadir el catálogo de productos de su tienda a la aplicación. Para ello tiene que seleccionar la tienda, seleccionar el producto e introducir el precio.
- **Añadir productos**: cuando un propietario quiera añadir un producto a su catálogo y el producto no esté registrado en la aplicación, podrá crearlo introduciendo los datos necesarios en un formulario.
- **Gestionar ofertas**: los propietarios podrán ver, modificar y eliminar las ofertas que hayan añadido.
- **Ver perfil**: los propietarios pueden ver los datos personales con los que se han registrado en la aplicación.
- **Actualizar el perfil**: los propietarios podrán modificar algunos datos de su perfil para mantenerlo actualizado.
- **Eliminar cuenta**: los propietarios podrán darse de baja de la aplicación y eliminar todos sus datos asociados.
- **Cerrar sesión**: los propietarios podrán cerrar la sesión y salir de la aplicación.

Capítulo 5. Diseño de la aplicación

En este capítulo se va a explicar más detalladamente la parte de diseño del proyecto. Como ya se ha nombrado en apartados anteriores, se ha diseñado una aplicación basada en el modelo cliente-servidor de tres capas. En este modelo, como su nombre bien indica, hay tres partes bien diferenciadas:

1. Cliente: es la parte que inicia el envío de peticiones al servidor, esperando de él una respuesta. Su función principal es interactuar con el usuario y ofrecerle la información que solicite de manera transparente. En este caso, el cliente es la aplicación móvil.
2. Servidor: implementa la lógica de negocio de la aplicación, es decir, se encarga de procesar las peticiones del cliente, enviarlas a la base de datos, procesar la respuesta y enviarla de vuelta al cliente. En este caso, el servidor está alojado de forma local e implementa una API REST.
3. Base de datos: es la última capa del modelo. En ella se almacena toda la información necesaria para el funcionamiento de la aplicación. Al igual que el servidor, la base de datos también se aloja de forma local.

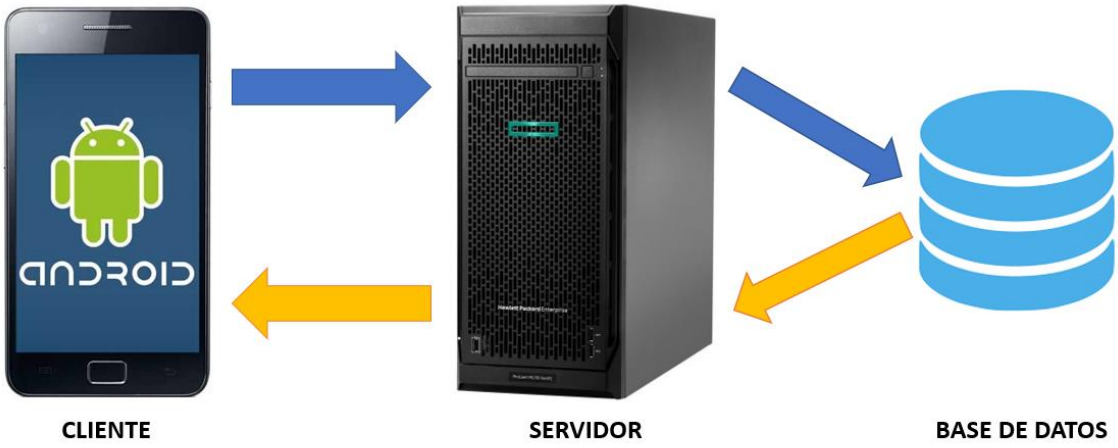

Figura 2. Modelo cliente-servidor de tres capas

El intercambio de solicitudes e información entre el cliente y el servidor se hace mediante el protocolo HTTP, mientras que las solicitudes entre el servidor y la base de datos se hacen con el protocolo MySQL. En el siguiente apartado se profundiza más sobre este tema.

5.1 Tecnologías utilizadas en la aplicación

Antes de entrar en detalles del diseño de la aplicación, conviene explicar algunas de las tecnologías utilizadas para su desarrollo, puesto que en mayor o menor medida han influido en el diseño.

5.1.1 REST

REST son las siglas de *Representational State Transfer* (Transferencia de Estado Representacional). Es un término que define una forma de arquitectura de software utilizada para crear una interfaz que conecta sistemas que utilizan el protocolo HTTP para sus comunicaciones, con el fin de obtener o realizar operaciones sobre datos. Es una tecnología ampliamente extendida, pues prácticamente todas las aplicaciones y webs utilizan en mayor o menor medida una interfaz REST.

Una interfaz o API REST debe cumplir con estas cuatro características:

1. **Protocolo cliente-servidor sin estado**: todos los mensajes HTTP generados en la comunicación contienen la información necesaria para realizar la petición, de forma que cada petición se trata de manera totalmente independiente y no tiene relación con ninguna anterior.
2. **Uso de operaciones**: al utilizar el protocolo HTTP, se hace uso de las operaciones definidas por éste, también llamadas "Verbos HTTP". Entre todas las operaciones, en REST se utilizan principalmente cuatro:
    a. GET: se utiliza para recuperar datos de un recurso.
    b. POST: se utiliza para crear o enviar datos a un recurso, produciendo un cambio en el estado o en el servidor.
    c. PUT: se utiliza para reemplazar o actualizar el recurso.
    d. DELETE: se utiliza para borrar el recurso.
3. **Identificación de recursos mediante URI**: Un URI es un Identificador Uniforme de Recursos. Los URI tienen un formato estándar en el que se incluye el esquema, la máquina donde se aloja el recurso, el directorio, el archivo y el fragmento al que se quiere acceder.
4. **Uso de hipermedios**: los hipermedios permiten que el usuario pueda navegar entre recursos mediante los enlaces adecuados y ejecutar acciones concretas sobre los datos.

Para comunicarse, el cliente simplemente crea y envía al servidor una petición HTTP en la que incluye, entre otras cosas, una cabecera, un método, un URI, y si corresponde, un cuerpo donde se incluye información adicional necesaria para procesar la petición. El servidor traduce la petición, la procesa y devuelve al cliente una respuesta en función del resultado obtenido.

Las respuestas del servidor incluyen un código de estado de tres cifras, que cambia según el resultado de la operación. La primera cifra define la clase de la respuesta: si la respuesta es meramente informativa, la primera cifra será un 1, si la operación ha sido correcta será un 2, si se ha redirigido, un 3, si ha sido incorrecta, un 4, y si se ha producido un fallo en el servidor será un 5. Las otras dos cifras cambian según el resultado de la respuesta. Por ejemplo, el código 200 indica que la petición se ha procesado y resuelto correctamente, el código 400 indica que la sintaxis de la petición es incorrecta y no se ha podido procesar, el 401 indica que el cliente no está

autorizado para recibir la respuesta solicitada, el 404 indica que no se ha encontrado el recurso solicitado, etc.

Además del código de estado, en la respuesta se puede incluir un cuerpo, donde se incluye la información solicitada por el cliente.

El uso del modelo REST ofrece una gran ventaja: es fácilmente escalable y adaptable a cualquier otro tipo de cliente, ya que es la lógica de negocio del servidor la que se encarga de procesar las peticiones. En el caso de este proyecto, el cliente es una aplicación para dispositivos Android, pero podría crearse un cliente para cualquier dispositivo o plataforma de una manera rápida y sencilla, siempre que éste tenga capacidad para interpretar el protocolo HTTP y por lo tanto pueda consumir la API del servidor.

5.1.2 Android y Java

Android es un sistema operativo lanzado por Google en 2008, ideado inicialmente para teléfonos móviles táctiles y posteriormente ampliado a tabletas, automóviles, televisores, receptores y relojes inteligentes [7]. También existen versiones compatibles con arquitecturas x86 instalables en ordenadores, pero su uso es meramente anecdótico. Hoy en día es el sistema operativo para smartphones más utilizado en el mundo, acaparando un 86.1% de la cuota de mercado, frente al 13.9% de iOS [8].

El desarrollo de aplicaciones para Android se hace, por norma general, en Java, aunque también se puede desarrollar en otros lenguajes como C/C++. Sin embargo, en los últimos años, Google está promoviendo el uso del lenguaje de programación Kotlin para el desarrollo de aplicaciones. En 2017 lo integró en el entorno de desarrollo oficial Android Studio y hoy en día ofrece toda la documentación, ejemplos y ayuda para el desarrollo de aplicaciones tanto para Java como para Kotlin [9]. Este lenguaje está ganando popularidad y está previsto que desplace a Java como lenguaje de desarrollo mayoritario para Android en los próximos años.

Para el desarrollo de la parte del cliente se va a utilizar el lenguaje de programación Java, ya que es el que aprendí en las diferentes asignaturas del Grado. Java fue creado por Sun Microsystems en 1995 y actualmente es propiedad de Oracle. Es un lenguaje de programación orientado a objetos cuya principal ventaja es que es multiplataforma. En la actualidad sigue siendo uno de los lenguajes más populares, gracias entre otras cosas a ser el lenguaje de desarrollo nativo para Android. Según el estudio de StackOverflow que ya se ha citado con anterioridad, el 41.1% de los desarrolladores lo utiliza. Sin embargo, en los últimos años está perdiendo cuota frente a otros lenguajes como Python, que ya le ha superado en popularidad [6], y probablemente siga perdiendo cuota si Kotlin se consolida como lenguaje de desarrollo preferente para Android.

5.1.3 Node.js y JavaScript

Node.js es un entorno de ejecución open-source y multiplataforma, que permite ejecutar código JavaScript de forma asíncrona en la parte del servidor [10]. Esto permite a los desarrolladores crear aplicaciones más livianas, con un alto grado de escalabilidad y manejar miles de conexiones simultáneamente en el mismo servidor. Como ya se ha nombrado en el capítulo 3, en los últimos años se ha convertido en una de las herramientas más utilizadas por los desarrolladores.

JavaScript es un lenguaje de programación interpretado y orientado a objetos, creado en 1995 [11]. Es utilizado generalmente para crear páginas web dinámicas, aunque en la actualidad

también se utiliza en la parte del servidor. Al ser un lenguaje de programación interpretado, no necesita compilación para ejecutar los programas.

5.1.4 MySQL

MySQL es un sistema de gestión de bases de datos (DBMS) relacional de código abierto perteneciente a Oracle. Fue creado en 1995 por MySQL AB, que posteriormente fue comprada por Sun Microsystems en 2008 y, a su vez, Sun fue comprada por Oracle en 2010 [12].

Al ser un sistema de código abierto perteneciente a una empresa privada, MySQL tiene un sistema de doble licencia pública y privada, por lo que se distribuye una versión gratuita llamada *Community* bajo licencia pública GNU v2, y versiones comerciales llamadas *Enterprise* de licencia privada, que incluyen funcionalidades extra como monitorización o asistencia técnica, para las empresas que quieran utilizar este sistema en sus productos.

Un sistema de bases de datos relacional almacena los datos en tablas. Cada tabla tiene definidos unos campos, y a su vez los campos tienen definido un tipo de variable (numérico, fecha, texto…). Los campos pueden incluir claves y restricciones. Por ejemplo, hay campos que no pueden ser nulos, que son de clave única (es decir, que no pueden existir dos registros con el mismo valor para dicho campo), auto-incrementales, con valores por defecto…

Pero lo que diferencia a sistema relacional de los demás son las relaciones entre tablas. Las relaciones pueden ser de varios tipos, entre los que destacan tres:

- **1 a 1:** se da cuando el registro de una tabla esta relacionado únicamente con uno de la otra tabla, y viceversa. Este tipo de relaciones no es común ya que se podría crear una única tabla con la información de las dos, por lo que se usa generalmente para dividir tablas por razones de tamaño, seguridad o para aislar datos.
- **1 a muchos:** en este tipo de relaciones, un registro de una tabla puede corresponderse con muchos de la otra tabla. Es el tipo de relación más común.
- **Muchos a muchos:** esta relación se da cuando muchos registros de la primera tabla pueden corresponderse con muchos registros de la segunda. Para realizar este tipo de relaciones se crea una tabla intermedia en la que se incluyen las claves primarias de ambas tablas.

Para crear relaciones, en primer lugar, se debe asignar al campo (o combinación de campos) necesario el valor de clave primaria o principal (Primary Key, o PK en inglés). Este campo actúa como identificador único de un registro, por lo que debe tener un valor único y no puede ser nulo. Para definir la relación, se establece una clave foránea (Foreign Key o FK) entre los campos de ambas tablas que se van a relacionar.

5.2 Diseño del cliente: aplicación Android

La parte de cliente consiste en una aplicación para dispositivos Android. Tal y como se ha especificado en el capítulo anterior, la aplicación debe implementar una serie de funcionalidades para dos tipos de usuario distintos: cliente y tienda.

Las aplicaciones para Android se distribuyen en *Activities.* Una *Activity* se puede definir como una pantalla de la aplicación, que se divide en dos partes: la parte gráfica, que es un archivo XML que contiene el código que crea la interfaz con la que interactuará el usuario, y la parte lógica, que es una clase de Java que contiene todo el código necesario para implementar la funcionalidad requerida. Por lo tanto, lo lógico es pensar que cada una de las funcionalidades de la aplicación necesitará una Activity. Sin embargo, viendo que algunas de las funcionalidades son compartidas por los dos tipos de usuario y hay otras que son similares, el número de Activities necesarias se puede reducir considerablemente. Este aspecto se tratará en el capítulo de desarrollo de la aplicación

Para diseñar la aplicación de una forma más fácil y ordenada, he dividido las funcionalidades en tres grupos: inicio de sesión y registro, funcionalidades del usuario de tipo cliente y funcionalidades del usuario tipo tienda.

5.2.1 Diseño de las funcionalidades de inicio de sesión y registro

Esta es la parte inicial de la aplicación, y serán las primeras pantallas que vean los usuarios.

Esta parte es común a los dos tipos de usuario, ya que ambos deben registrarse e iniciar sesión para poder acceder a la aplicación. La interfaz es sencilla y solo consta de unos formularios donde introducir los datos requeridos y unos botones para ejecutar las acciones.

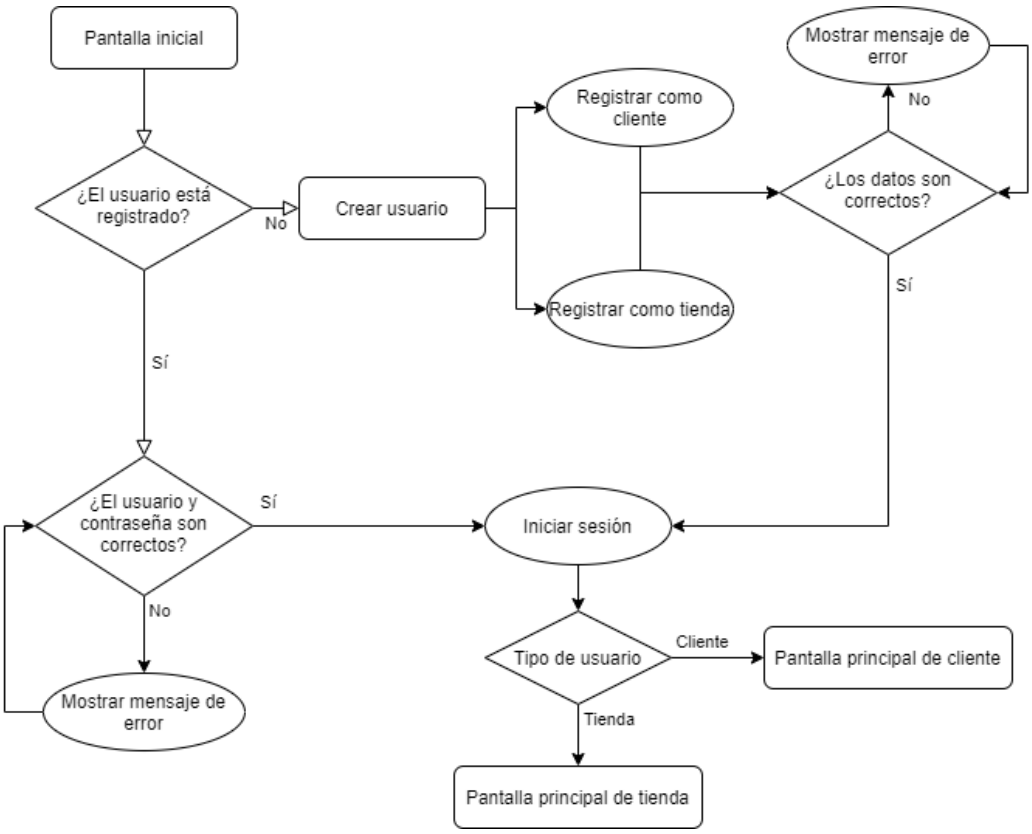

Figura 3. Esquema del diseño de inicio de sesión y registro

5.2.2 Diseño de las funcionalidades del usuario de tipo cliente

Una vez un cliente ha iniciado sesión, la aplicación le redirige a la pantalla principal del cliente, donde puede seleccionar una de las cuatro opciones disponibles: buscar, gestionar sus ofertas, ver su perfil y cerrar sesión.

5.2.2.1 Buscar

La pantalla mostrará en la parte superior una caja de texto donde introducirá el producto o marca que quiere buscar, otra caja de texto donde introducirá el radio de búsqueda en metros, una lista desplegable con la que podrá filtrar los resultados por tipo de producto y un botón de buscar. En el resto de la pantalla aparecerá un mapa centrado en la ubicación del usuario, donde se mostrarán los resultados de la búsqueda. Si la búsqueda se quiere realizar desde otra ubicación, se podrá cambiar haciendo una pulsación larga en el punto deseado. La ubicación de búsqueda se mostrará con un marcador con el diseño por defecto que provee Google.

Los resultados de la búsqueda se deben mostrar en forma de marcador en el mapa. Los marcadores serán pequeños rectángulos que contendrán el precio correspondiente al producto u oferta encontrado, y su color variará en función del resultado: si es una oferta creada por la propia tienda, si es una oferta creada por un cliente, o si es un precio de catálogo (es decir, que no está de oferta).

Al pulsar el marcador, se debe mostrar una pequeña tarjeta con los datos principales de la oferta o del precio correspondiente. Si es una oferta, al pulsar en la tarjeta se mostrará una nueva pantalla donde se verá la oferta en detalle.

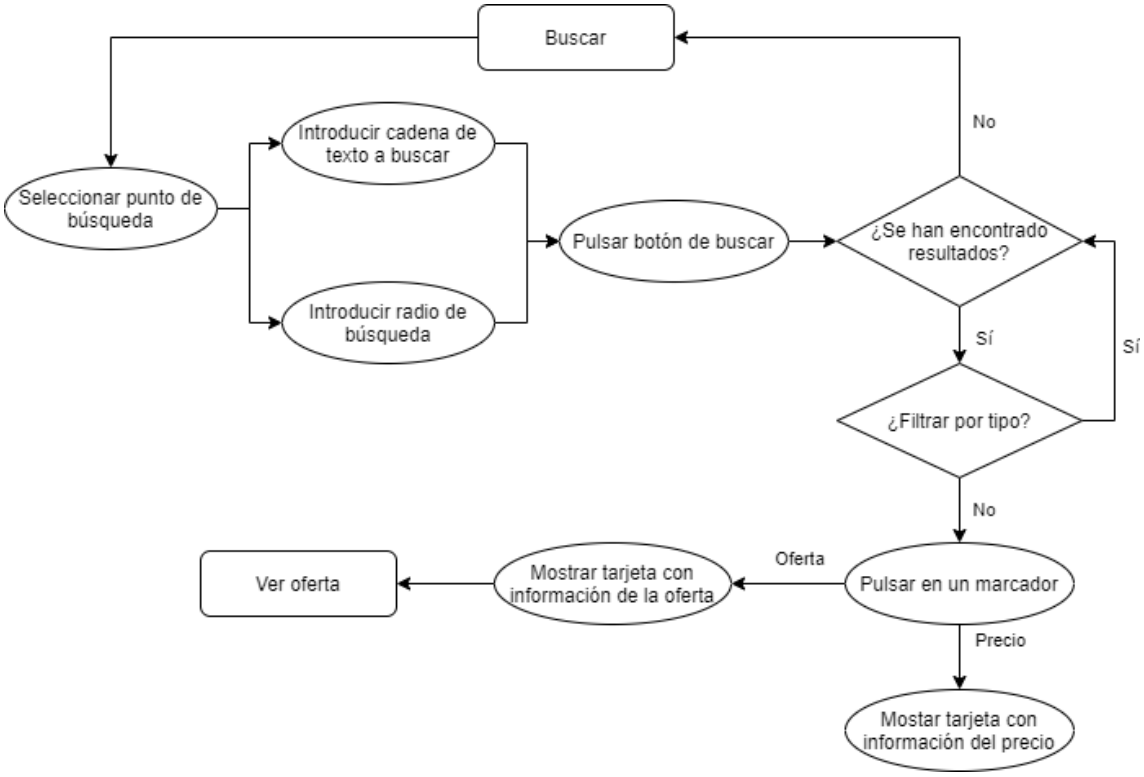

Figura 4. Esquema del diseño de la funcionalidad de buscar

5.2.2.2 Gestionar ofertas

Al seleccionar la opción de gestionar ofertas, el cliente accede a una pantalla donde se mostrará en la parte superior un botón para añadir una nueva oferta, y debajo una lista con tarjetas con información básica de todas las ofertas que haya creado.

Si pulsa en las tarjetas de las ofertas accederá a una nueva pantalla donde se mostrará toda la información de la oferta y dos botones para modificar o eliminar la oferta. Si pulsa en modificar, se mostrará un formulario donde podrá cambiar los datos que crea convenientes. Si pulsa en eliminar, se mostrará una alerta pidiendo confirmación.

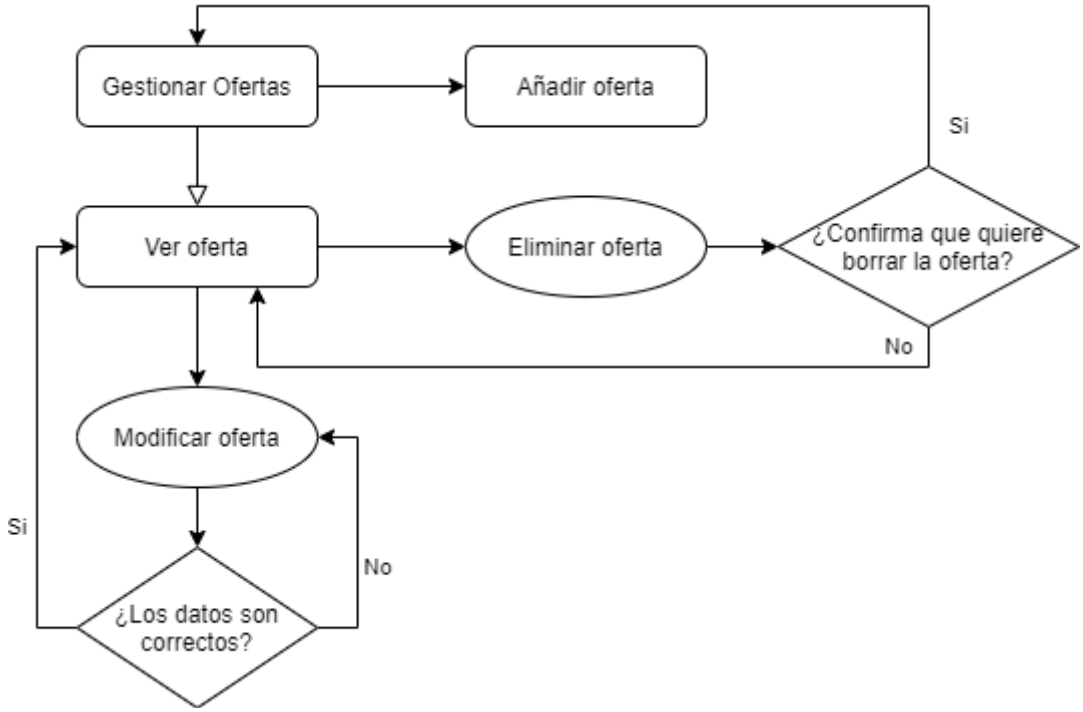

Figura 5. Esquema del diseño de la funcionalidad de gestionar ofertas de un cliente

Si pulsa en añadir oferta, accederá a una pantalla donde tendrá que seleccionar la tienda donde ha visto la oferta. Para ello se mostrará una pantalla similar a la de "Buscar", pero en este caso la búsqueda devolverá tiendas en vez de ofertas. En la parte inferior se mostrará un botón para crear una tienda, para que el cliente pueda crear la tienda en caso de que no esté registrada en la aplicación.

Si el cliente encuentra la tienda, pulsará en el marcador correspondiente y se mostrará una tarjeta con la dirección. Al pulsar en la tarjeta, la tienda quedará seleccionada y se pasará a una nueva pantalla donde tendrá que seleccionar el producto que ha visto en oferta.

Si el cliente no ha encontrado la tienda, pulsará en el botón de crear tienda y pasará a una nueva pantalla donde tendrá que seleccionar la ubicación de la tienda y rellenar los datos básicos. Una vez creada, pasará a la pantalla de seleccionar producto.

La pantalla de seleccionar producto será similar a la de gestionar ofertas, pero en este caso se mostrará un botón para añadir un producto al catálogo de la tienda, una lista desplegable para filtrar por tipo de producto y una lista con tarjetas que muestran la información básica de cada producto que hay en el catálogo de la tienda.

Si el producto no está en el catálogo de la tienda, el cliente puede introducirlo pulsando en el botón de añadir producto. En ese caso, se mostrará una nueva pantalla donde podrá buscar entre todos los productos registrados en la aplicación que no estén ya incluidos en el catálogo de la tienda. Si no lo encuentra, puede crear un nuevo producto pulsando en el botón de la parte inferior. Se mostrará una pantalla con un formulario en el que tendrá que introducir la información del producto.

Una vez seleccionado el producto, pasará a una pantalla donde se mostrarán los detalles del producto y le pedirá que introduzca el precio habitual al que está en la tienda, no el de la oferta. Después se mostrará la pantalla final, donde tendrá que introducir todos los datos de la oferta.

En el siguiente esquema se ha simplificado la parte de "Buscar", puesto que es idéntica a la mostrada en su esquema correspondiente.

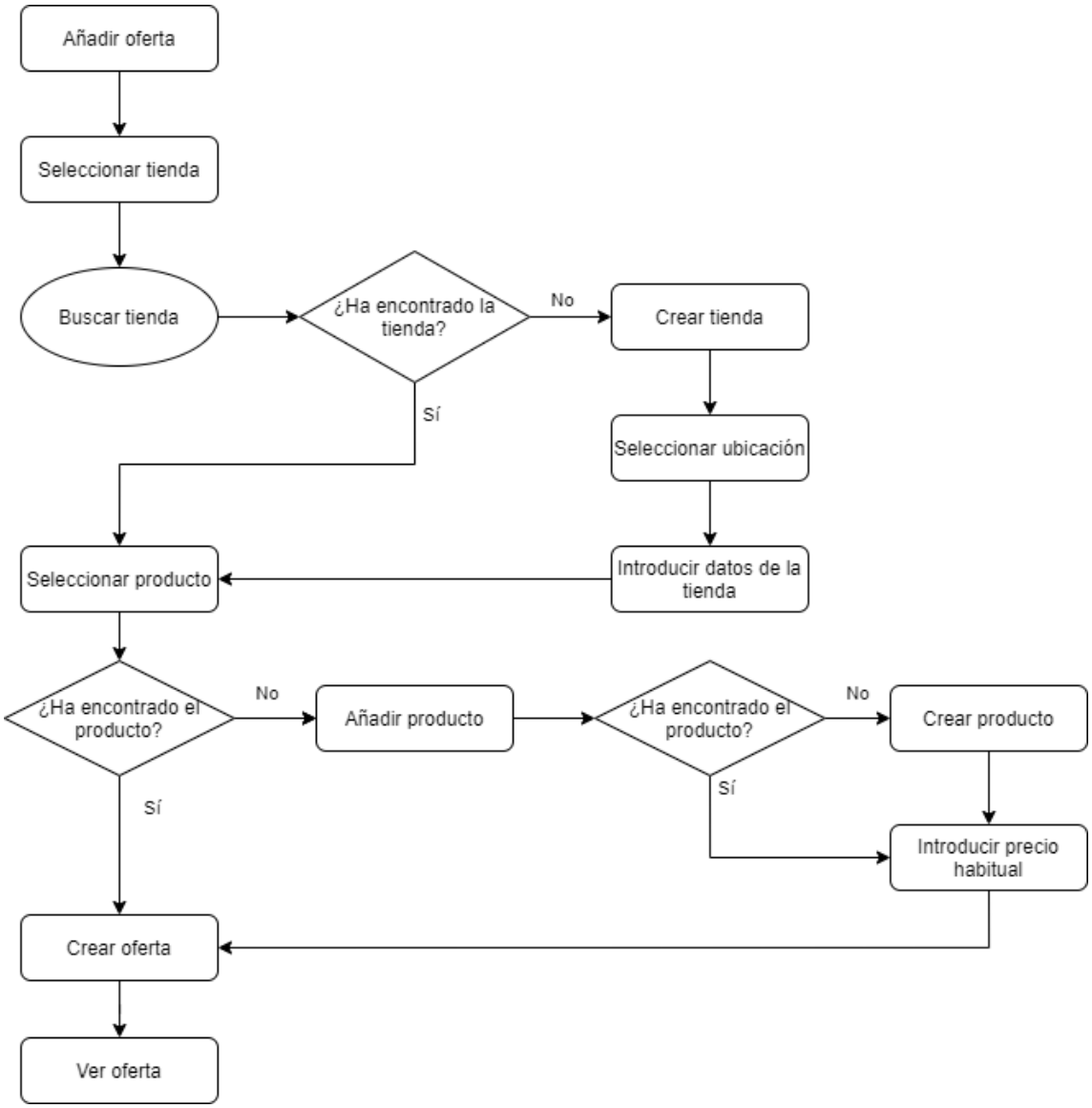

Figura 6. Esquema del diseño de la funcionalidad de añadir oferta de un cliente

5.2.2.3 Ver perfil

Si el cliente selecciona la opción de ver su perfil, accede a una pantalla donde simplemente se mostrarán los datos que indicó al registrarse. En la parte inferior se mostrarán dos botones: uno para actualizar sus datos y otro para eliminar su cuenta.

Si pulsa el botón de actualizar los datos se mostrará un formulario donde podrá introducir los nuevos datos. Si pulsa el botón de eliminar la cuenta, se mostrará una alerta pidiendo confirmación. Si confirma, se eliminarán todos los datos asociados a la cuenta: datos personales, datos de acceso, tiendas y ofertas añadidas.

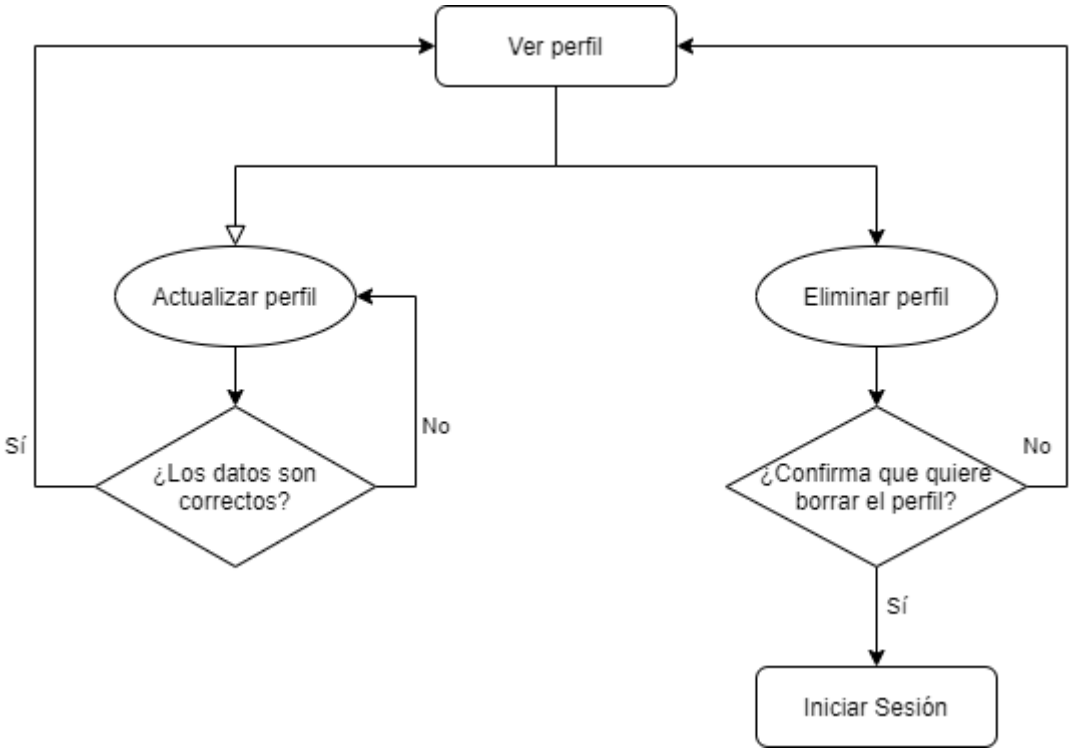

Figura 7. Esquema del diseño de la funcionalidad de ver perfil

5.2.2.4 Cerrar sesión

Al seleccionar la opción de cerrar sesión, simplemente se mostrará una alerta pidiendo confirmación y, en caso afirmativo, se redirigirá al usuario a la pantalla inicial de iniciar sesión.

5.2.3 Diseño de las funcionalidades del usuario de tipo tienda

Una vez un propietario ha iniciado sesión, la aplicación le redirige a la pantalla principal del usuario de tipo tienda, donde puede seleccionar una de las seis opciones disponibles: buscar, gestionar sus tiendas, gestionar sus ofertas, gestionar sus catálogos de productos, ver su perfil y cerrar sesión.

5.2.3.1 Buscar

La funcionalidad de buscar es exactamente la misma que para el usuario de tipo cliente descrita anteriormente, por lo que no se va a volver a explicar.

5.2.3.2 Gestionar tiendas

La pantalla de gestión de tiendas será similar a la de gestionar ofertas del cliente que se ha descrito en el apartado anterior. En este caso, habrá un botón de crear tienda y la lista de tarjetas mostrará las tiendas del usuario, con su nombre y dirección.

Al pulsar en la tarjeta de una tienda, se mostrará una nueva pantalla con los datos de la tienda, un mapa con la ubicación y en la parte inferior un botón para modificar los datos de la tienda y otro botón para eliminarla. Si pulsa en el botón de modificar tienda, pasará a una pantalla donde podrá modificar la ubicación, y después pasará a otra donde podrá cambiar los datos de la tienda. Si pulsa en el botón de eliminar tienda se abrirá una alerta donde se le pedirá confirmación. Si confirma, se eliminará la tienda junto a su catálogo de productos y ofertas asociadas.

Si pulsa en el botón de crear tienda, pasará a una nueva pantalla donde tendrá que seleccionar la ubicación de la tienda y rellenar los datos de dirección (calle, número, código postal, etc). Después pasará a una nueva pantalla donde tendrá que introducir el nombre de la tienda, el CIF de la empresa y el número de teléfono de la tienda. Una vez introducidos los datos, pasará a la pantalla de ver tienda, donde podrá modificar los datos o eliminarla.

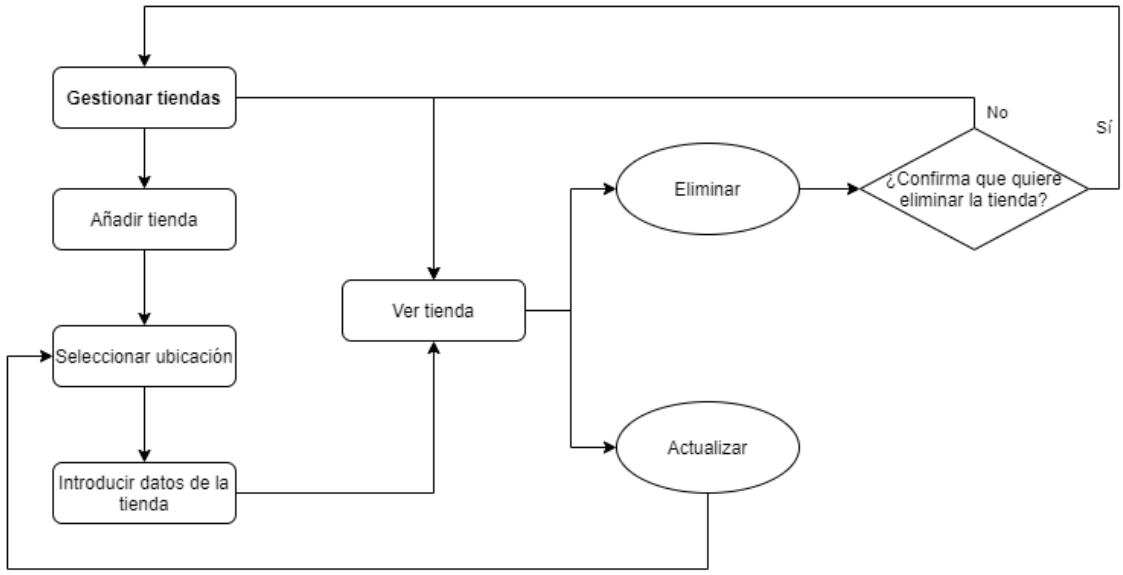

Figura 8. Esquema del diseño de la funcionalidad de gestionar tiendas de un propietario

5.2.3.3 Gestionar ofertas

La pantalla de gestión de ofertas de un propietario será similar a la de gestionar ofertas del cliente que se ha descrito en el apartado anterior, con la diferencia de que, en este caso, se muestra en la parte superior una lista desplegable donde el usuario puede seleccionar la tienda que quiere gestionar. Por lo tanto, la pantalla tendrá una lista desplegable con las tiendas, un botón para añadir ofertas y una lista de tarjetas con las ofertas añadidas.

Cuando el usuario pulse en la lista desplegable, se mostrará la lista de tiendas que ha añadido a la aplicación. Al seleccionar una, se mostrará debajo la lista de ofertas que ha añadido para esa tienda en forma de tarjetas.

Al pulsar en la oferta, entrará en la pantalla de ver oferta, donde se mostrará la información detallada de la oferta y podrá modificarla o eliminarla.

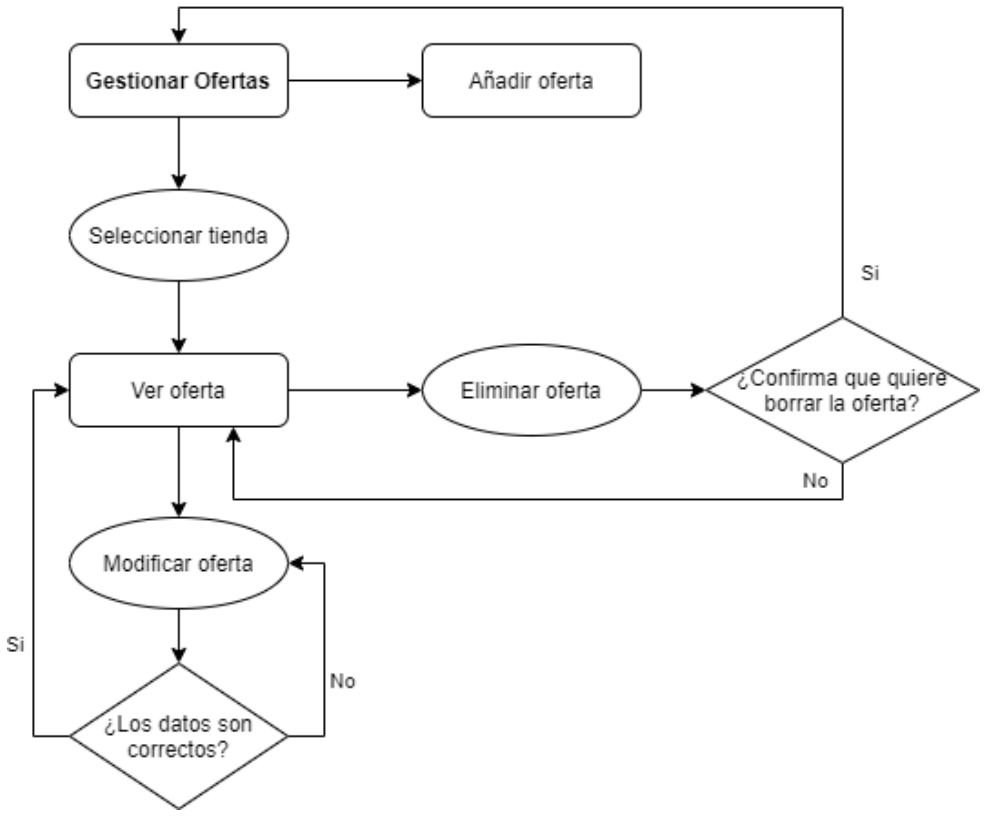

Figura 9. Esquema del diseño de la funcionalidad de gestionar ofertas de un propietario

Para añadir una nueva oferta a la tienda seleccionada podrá pulsar en el botón de añadir oferta. Se mostrará una nueva pantalla donde aparece el catálogo de productos de la tienda y una lista desplegable con la que podrá filtrar los productos por tipo. Si el producto en oferta ya está en el catálogo de la tienda, pulsará en él y pasará a nueva pantalla donde se muestra la plantilla de la oferta. Ahí tendrá que introducir el título de la oferta, el precio, la fecha de inicio de la oferta, la fecha de fin de la oferta y una descripción más detallada de la oferta. Una vez creada la oferta, la aplicación regresa a la pantalla inicial de gestionar ofertas.

Si el producto en oferta no está en el catálogo, tendrá que añadirlo pulsando en el botón de añadir producto. Esta funcionalidad se describirá más adelante. Una vez creado, la aplicación le llevará a la pantalla de crear la oferta, descrita anteriormente.

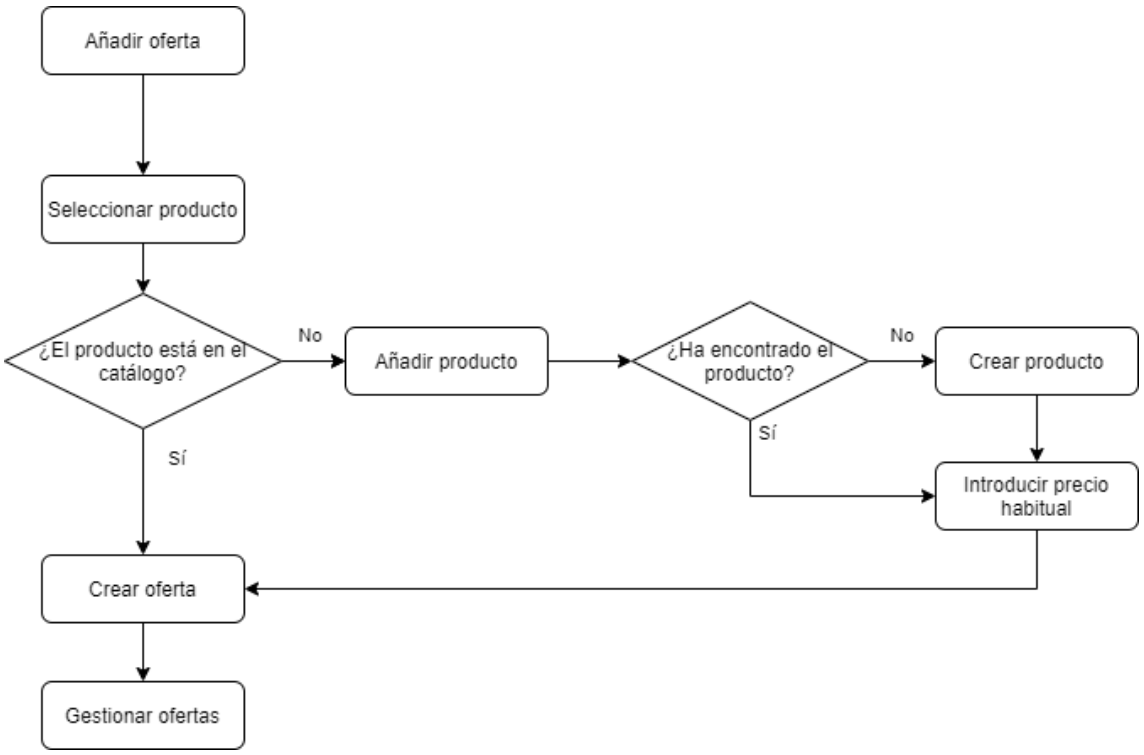

Figura 10. Esquema del diseño de la funcionalidad de añadir oferta de un propietario.

5.2.3.4 Gestionar catálogos de productos

En esta opción, el usuario podrá ver los catálogos de productos de sus tiendas. La pantalla consta de una lista desplegable con las tiendas del usuario, una lista de tarjetas con todos los productos del catálogo de esa tienda, donde podrá ver información básica de cada producto, una lista desplegable para filtrar los productos según su tipo y un botón para añadir productos.

El usuario tendrá que seleccionar, en primer lugar, la tienda que quiere gestionar en la lista desplegable. Después se mostrará debajo la lista de productos del catálogo de la tienda seleccionada. Si quiere, podrá seleccionar un tipo en la lista desplegable de tipos para filtrar los resultados.

Al pulsar en uno de los productos, pasará a una nueva pantalla donde verá los detalles del producto y tendrá la opción de modificar el precio y eliminar el producto del catálogo de esa tienda. El usuario no podrá modificar ni eliminar el producto de la aplicación, ya que puede estar en el catálogo de las tiendas de otros usuarios. Si pulsa el botón de modificar, se mostrará un cuadro de texto donde podrá introducir el precio actualizado. Si pulsa el botón de eliminar, se mostrará una alerta pidiendo confirmación. Si acepta, se eliminará el producto del catálogo de la tienda seleccionada y todas las ofertas asociadas.

Si el usuario quiere añadir un producto al catálogo de una tienda, tendrá que pulsar el botón de añadir producto. Se abrirá una nueva pantalla con un buscador de productos, donde podrá introducir el nombre o la marca del producto que quiere añadir. Al pulsar en el botón de buscar se generará debajo una lista de tarjetas con información básica de todos los productos que se hayan encontrado, y podrá filtrar por tipo eligiéndolo en una lista desplegable. Al pulsar en un producto, pasará a una nueva pantalla donde verá los detalles del producto y podrá introducir el

precio al que lo vende de forma habitual. Si pulsa el botón de confirmar, se añadirá el producto al catálogo y pasará a la pantalla de ver producto que ya se ha descrito antes.

Si el producto no está registrado en la aplicación, podrá crearlo pulsando el botón de crear producto. Pasará a una pantalla donde tendrá que introducir el nombre, la marca, la descripción del producto, seleccionar el tipo de producto mediante una lista desplegable y, por último, introducir una URL que contenga una imagen del producto. Una vez creado, la aplicación le redirigirá a la pantalla de introducir el precio, y una vez introducido verá la pantalla con la información del producto y las opciones de modificar o eliminar del catálogo.

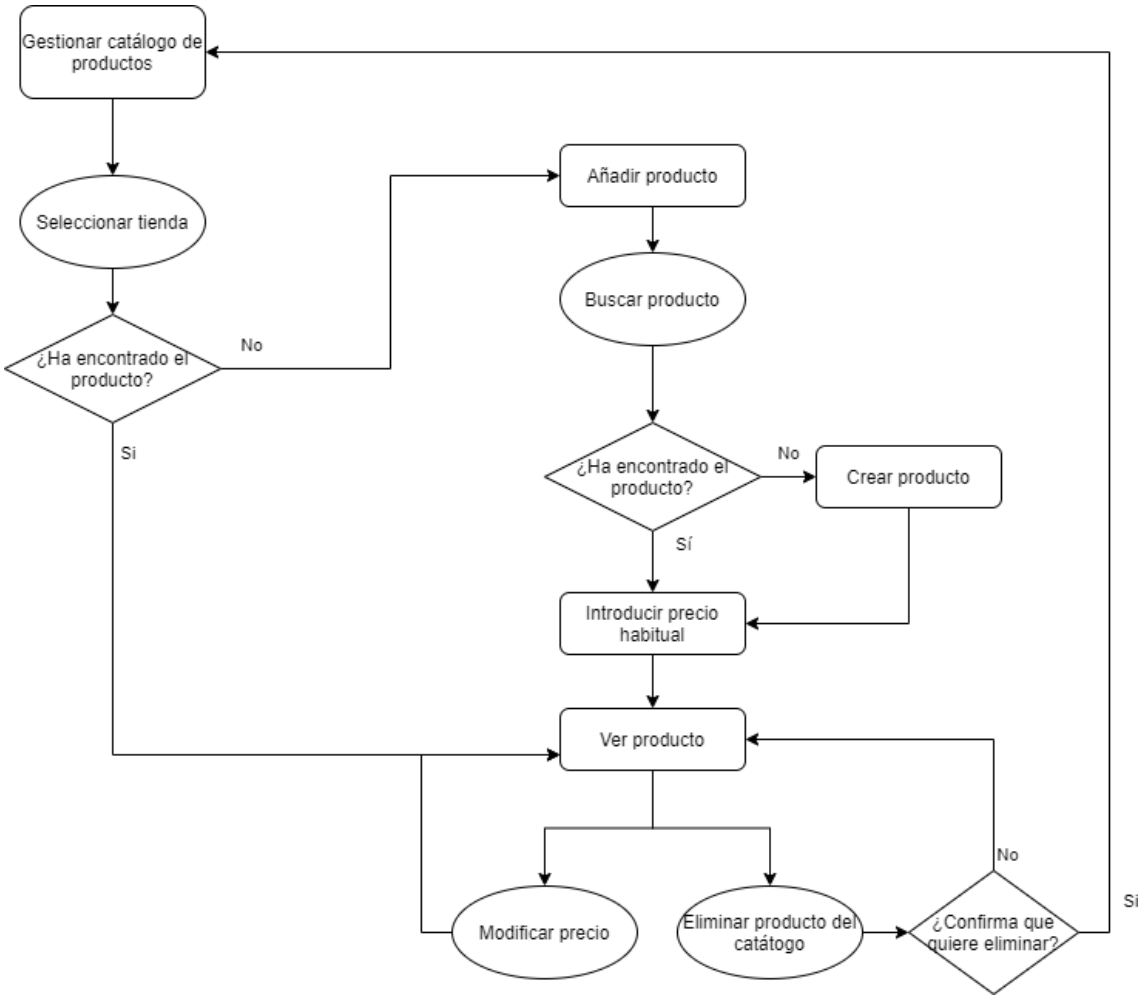

Figura 11. Esquema del diseño de la funcionalidad de gestionar catálogo de un propietario

5.2.3.5 Ver perfil

La funcionalidad de ver perfil de un usuario de tipo tienda es similar a la de un usuario de tipo cliente. La información que se podrá actualizar son los datos personales del propietario y los datos de acceso, no los de las tiendas, que se cambiarán en su propia sección.

Al eliminar el perfil se borrarán también todas las tiendas, catálogos de precios y ofertas asociadas al usuario. No se eliminarán los productos, ya que pueden estar en los catálogos de otras tiendas.

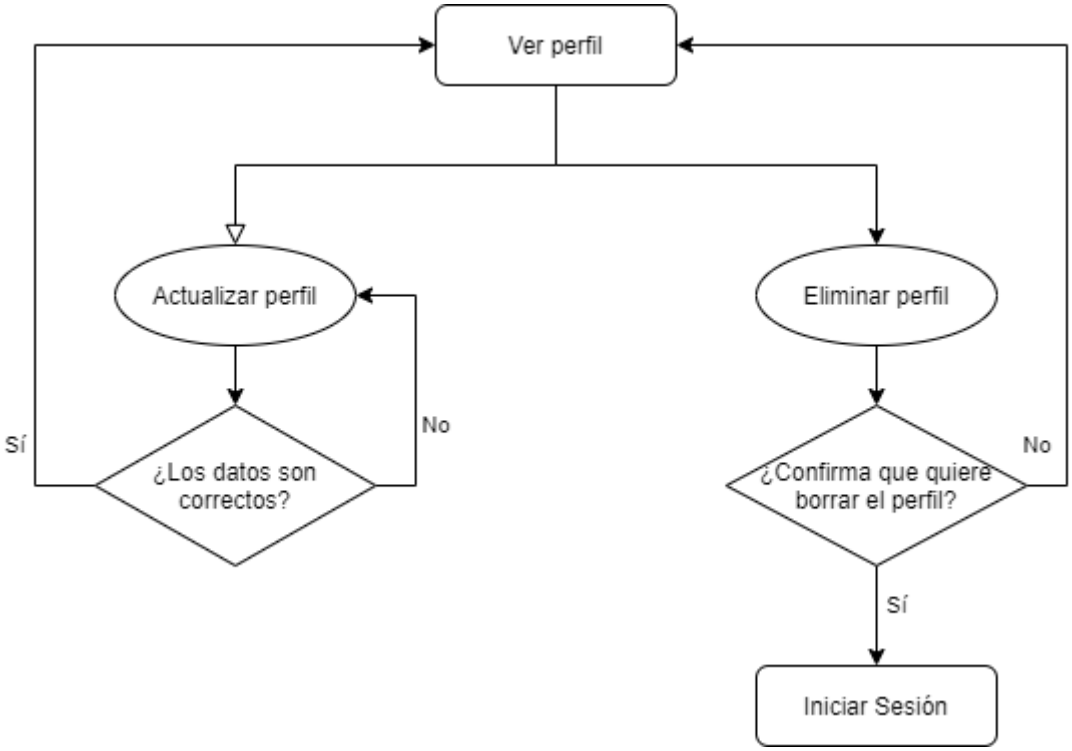

Figura 12. Esquema del diseño de la funcionalidad de ver perfil

5.2.3.6 Cerrar sesión

Al seleccionar la opción de cerrar sesión, simplemente se mostrará una alerta pidiendo confirmación y, en caso afirmativo, se redirigirá al usuario a la pantalla inicial de iniciar sesión.

5.3 Diseño del servidor: API REST

La parte del servidor tendrá que ser capaz de recibir las peticiones del cliente, procesarlas y enviarlas a la base de datos. Después tendrá que realizar el proceso contrario: tendrá que recibir la respuesta de la base de datos, procesarla y enviarla a la aplicación.

Para realizar este proceso se van a diferenciar cinco partes: parser (o "traductor"), rutas, autentificación, funciones y modelos:

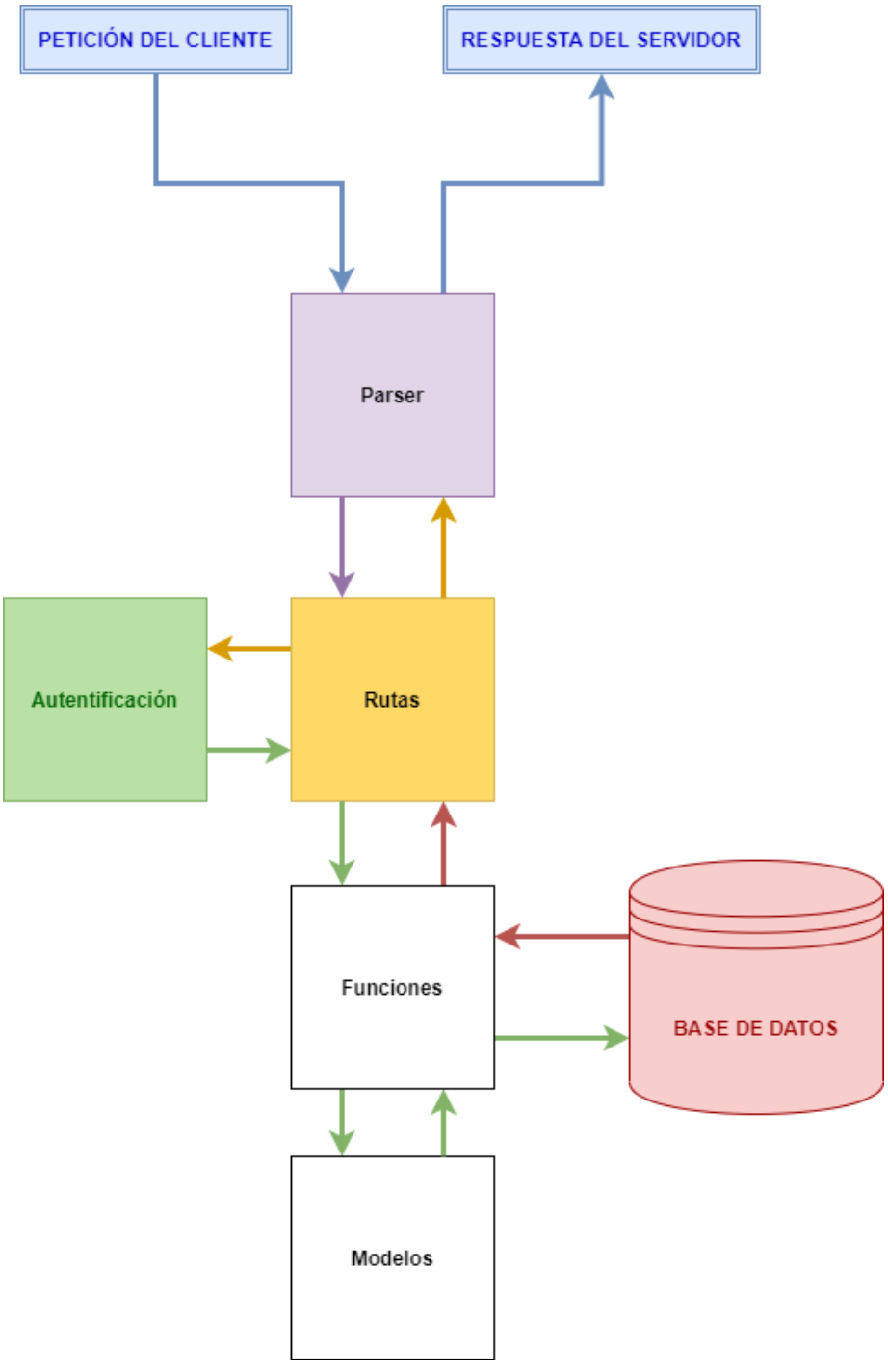

Figura 13. Esquema del diseño del servidor

5.3.1 Parser

Un parser es un programa o, en este caso, un módulo que analiza los datos que recibe y los transforma para que puedan ser entendidos por el receptor.

En el servidor, el parser va a ser el encargado de recibir la petición HTTP del cliente, trocearla y guardarla en variables para que el resto de los procesos del servidor puedan tratarlas. A su vez, tendrá que hacer el proceso contrario cuando el servidor tenga que devolver una respuesta al cliente.

5.3.2 Rutas

En la explicación sobre REST que se hizo en el apartado anterior, se habló de las URI. Cuando el cliente envía una petición, incluye un método y una URI, que apunta a una ruta del servidor. Por lo tanto, el servidor deberá tener un archivo de rutas donde se especifique qué debe hacer según el método que se solicita y la ruta a la que se esté apuntando.

Una vez esclarecida la ruta, el servidor llamará a una función y le pasará los parámetros que sean necesarios para poder ejecutarla. Los parámetros pueden venir en la URI o en el cuerpo (*body*) de la petición según el método que estén solicitando. Por ejemplo, si el servidor recibe una petición con el método GET, tendrá que buscar los parámetros en la URI. Si se recibe una petición POST, los parámetros irán incluidos en el cuerpo de la petición.

Cuando la función se haya terminado de ejecutar, devolverá a la ruta el resultado de la operación, que incluirá el código de estado y el cuerpo de la respuesta. La ruta devolverá una respuesta al cliente incluyendo el código de estado y el cuerpo de la respuesta. El cuerpo de la respuesta enviado por la ruta no tiene por qué coincidir con el devuelto por la función, ya que quizás solo sea necesario enviar una parte del cuerpo, o simplemente un mensaje.

5.3.3 Autentificación

Para que la aplicación tenga un mínimo de seguridad, el servidor debe implementar un módulo de autentificación. Este módulo será el encargado de cifrar y descifrar las credenciales que envíe el cliente para iniciar sesión o registrarse en la aplicación y proteger las rutas para que no puedan ser accesibles si no se ha iniciado sesión o no se tienen privilegios suficientes.

En este caso, se ha optado por implementar un módulo de autentificación mediante Base64 y JSON Web Token (JWT). Base64 sólo se utilizará para evitar que el usuario y la contraseña del cliente viajen por la red en texto plano sin ningún tipo de codificación.

Cuando el servidor reciba una petición de registro de un nuevo usuario, descodificará las credenciales y cifrará la contraseña mediante “Salted hashing”. Con este método se generan bytes aleatorios (salt) que se combinan con la contraseña y, posteriormente, se genera un hash de dicha combinación. De esta manera, en la base de datos no se almacenarán las contraseñas en texto plano, sino que se almacenará el hash generado.

Cuando el servidor reciba una petición de inicio de sesión, descodificará las credenciales enviadas en la petición en Base64, extraerá la contraseña, recuperará el hash de la contraseña de la base de datos y comparará ambas. Si coinciden, generará un token de sesión que enviará al cliente, y éste deberá introducirlo en la cabecera de todas las peticiones que envíe al servidor hasta que cierre sesión para acreditar que tiene permiso para acceder a las rutas que solicite.

Evidentemente, Base64 es un método de codificación y no de cifrado, y cualquier elemento codificado con Base64 se puede descodificar fácilmente (hay miles de herramientas para ello), por lo que no aporta seguridad. Para que el envío de las credenciales fuera realmente seguro y

evitar que cualquiera pudiera interceptar, ya no solo las credenciales, sino todos los datos que se envíen a través de la aplicación, habría que implementar el uso del protocolo HTTPS y certificados de seguridad. Sin embargo, al ser un desarrollo meramente conceptual que no se va a poner en producción y se va a ejecutar de forma local en mi ordenador personal, no se va a implementar esta capa de seguridad.

5.3.3.1 JWT - JSON Web Token

JWT es un estándar abierto que utiliza JSON para crear tokens de acceso. Estos tokens están formados por una cabecera, un payload (o contenido) y una firma, separadas por un punto y codificadas en Base64 [13]:

- **Cabecera**: incluye el algoritmo que se ha utilizado para generar la firma y el tipo de token.
- **Payload**: contiene la información de los datos del usuario, los privilegios que tiene y la fecha de caducidad del token.
- **Firma**: incluye la firma que se utilizará para comprobar que el token es válido.

Para generar la firma es necesario crear una clave privada que se almacenará en el servidor y nunca deberá ser revelada.

5.3.4 Funciones

Cuando se accede a una ruta, ésta hace una llamada a una función. La función será la encargada de realizar la petición a la base de datos y recuperar la información que le envíe en la respuesta, así como de devolver a la ruta el resultado de la operación.

Por lo tanto, las funciones contendrán toda la lógica para generar la petición, recibir los resultados y procesarlos, generando una respuesta que enviará a la ruta y que contendrá el código de estado correspondiente y el cuerpo de la respuesta, si procede.

5.3.5 Modelos

Para que el servidor sepa cómo está diseñada la base de datos, es necesario crear modelos. Un modelo incluye el nombre de la tabla a la que hace referencia, los campos, el tipo de variable de cada campo, las restricciones, los valores por defecto (si corresponde) y las relaciones con otras tablas. Por lo tanto, será necesario crear un modelo para cada tabla de la base de datos.

Las funciones tendrán que acceder al modelo o modelos que sean necesarios para hacer la petición a la base de datos y construir la respuesta que devolverá a la ruta que la invocó.

5.4 Diseño de la base de datos

En este apartado se va a explicar el diseño de la base de datos: las tablas que se crearán, los campos, las restricciones y las relaciones entre tablas.

En primer lugar, hice un diseño inicial que separaba los datos según el tipo de usuario: clientes y propietarios tenían sus propias tablas y compartían algunas comunes. Sin embargo, me di cuenta de que ese diseño era poco eficiente y complicaría en exceso la programación de la aplicación, además de que era fácilmente simplificable uniendo las tablas e incluyendo en ellas un campo que sirviera para saber si el registro pertenece a un cliente o a un propietario.

Finalmente, creé un diseño en el que hay ocho tablas: usuarios, datos de usuario, tiendas, ubicaciones, precios, productos y tipos de producto. En los siguientes apartados las explico con detalle.

5.4.1 Usuarios

La tabla de usuarios contiene los datos de inicio de sesión y el tipo de usuario:

USUARIOS		
CAMPO	**TIPO**	**DESCRIPCIÓN**
idUsuario	Int	Identificador del usuario
User	Varchar	Nombre de usuario
Pass	Varchar	Contraseña (hash) del usuario
Email	Varchar	Email del usuario
Estado	Int	Estado de la cuenta de usuario
TipoUsuario	Int	Tipo de usuario de la cuenta.

Tabla 1. Estructura de la tabla Usuarios

La clave primaria de la tabla es el campo idUsuario, que actúa como identificador. Respecto al resto de campos, ninguno puede ser nulo, y user y email deben ser únicos. De esta forma se evita que se puedan registrar varios usuarios con el mismo nombre y correo.

El campo estado indica el estado en el que se encuentra la cuenta del usuario. Si es un 0, la cuenta está activa, si es un 1 la cuenta está inactiva.

El campo TipoUsuario indica el tipo de usuario de la cuenta. Si es 0 será un usuario de tipo cliente y si es 1 será un usuario de tipo tienda.

5.4.2 Datos de usuario

La tabla de usuarios contiene los datos personales del usuario:

DATOS_USUARIO		
CAMPO	**TIPO**	**DESCRIPCIÓN**
idUsuario	Int	Identificador del usuario
Nombre	Varchar	Nombre del usuario
Apellido1	Varchar	Primer apellido del usuario
Apellido2	Varchar	Segundo apellido del usuario
FechaNacimiento	Int	Fecha de nacimiento del usuario
Teléfono	Varchar	Teléfono del usuario
FechaRegistro	Timestamp	Fecha de registro del usuario

Tabla 2. Estructura de la tabla Datos_Usuario

La clave primaria de la tabla es el campo idUsuario, que servirá para establecer una relación 1 a 1 con la tabla de usuarios. Ningún campo puede ser nulo.

El campo FechaRegistro es de tipo Timestamp. Este tipo de variable de MySQL es simplemente una fecha que coge el valor del momento en el que se crea el registro.

5.4.3 Tiendas

La tabla de tiendas contiene los datos las tiendas registradas en la aplicación:

TIENDAS		
CAMPO	**TIPO**	**DESCRIPCIÓN**
idTienda	Int	Identificador de la tienda
idUsuario	Int	Identificador del usuario que creó la tienda
idUbicacion	Int	Identificador de la ubicación de la tienda
NombreTienda	Varchar	Nombre de la tienda
CIF	Varchar	CIF de la tienda
Teléfono	Varchar	Teléfono de la tienda
FechaRegistro	Timestamp	Fecha de creación de la tienda
Verificacion	Int	Identifica si la tienda está verificada o no.

Tabla 3. Estructura de la tabla Tiendas

La clave primaria de la tabla es el campo idTienda, que sirve como identificador. Del resto de campos, cabe destacar el campo de verificación. Este campo sirve para saber de forma

directa si la tienda ha sido creada por el propietario o si la ha añadido un cliente. Este campo será clave a la hora de programar la aplicación, pues según su valor las activities cambiarán a unas u otras

5.4.4 Ubicaciones

La tabla de ubicaciones contiene las direcciones y coordenadas de las tiendas:

UBICACIONES		
CAMPO	**TIPO**	**DESCRIPCIÓN**
idUbicacion	Int	Identificador de la ubicación
TipoVia	Varchar	Tipo de vía (calle, avenida, plaza…)
NombreVia	Varchar	Nombre de la vía
Numero	Varchar	Número del local
Adicional	Varchar	Información adicional (Piso, puerta, bajo…)
CodPostal	Varchar	Código postal
Localidad	Varchar	Localidad
Provincia	Varchar	Provincia
Coordenadas	Point	Latitud y longitud de la tienda

Tabla 4. Estructura de la tabla Ubicaciones

La clave primaria de la tabla es el campo idUbicación. En esta tabla cabe destacar el campo Coordenadas, que es de tipo Point. Este tipo de variable de MySQL es en realidad un vector de tamaño 2, de números con signo y decimales. Simplifica en gran medida las operaciones en las que hay que hacer cálculos de distancias entre puntos.

5.4.5 Ofertas

La tabla de ofertas contiene los datos de las ofertas:

OFERTAS		
CAMPO	**TIPO**	**DESCRIPCIÓN**
idOferta	Int	Identificador de la oferta
idUsuario	Int	Identificador del usuario que ha creado la oferta
idProducto	Int	Identificador del producto asociado a la oferta
idTienda	Int	Identificador de la tienda asociada a la oferta
Titulo	Varchar	Título de la oferta
Descripcion	Varchar	Descripción de la oferta
Precio	Decimal	Precio de la oferta
FechaAnadido	Timestamp	Fecha de creación de la oferta
FechaInicio	Date	Fecha en la que empieza la oferta
FechaFin	Date	Fecha en la que expira la oferta

Tabla 5. Estructura de la tabla Ofertas

La clave primaria de la tabla es el campo idOferta, que sirve como identificador. El campo Estado indica si la oferta está activa o inactiva.

5.4.6 Productos

La tabla de productos contiene los datos de los productos añadidos a la aplicación:

PRODUCTOS		
CAMPO	**TIPO**	**DESCRIPCIÓN**
idProducto	Int	Identificador del producto
idTipo	Int	Identificador del tipo de producto
Nombre	Varchar	Nombre del producto
Marca	Varchar	Marca del producto
Descripcion	Varchar	Descripción del producto
ImagenURL	Varchar	URL de una imagen del producto

Tabla 6. Estructura de la tabla Productos

La clave primaria de la tabla es el campo idProducto, que sirve como identificador. El campo ImagenURL servirá para cargar en la aplicación imágenes de los productos, a falta de implementar un mecanismo de subida de imágenes al servidor.

5.4.7 Tipos de producto

La tabla de tipos de producto simplemente contiene los tipos o categorías en las que se engloban los productos (alimentación, electrónica, ropa...):

TIPOS_PRODUCTO		
CAMPO	**TIPO**	**DESCRIPCIÓN**
idTipo	Int	Identificador del tipo de producto
Nombre	Varchar	Nombre del tipo de producto

Tabla 7. Estructura de la tabla Tipos_Producto

La clave primaria de la tabla es el campo idTipo, que sirve como identificador.

5.4.8 Precios

La tabla de precios es una tabla intermedia que sirve de unión para establecer una relación muchos a muchos entre la tabla de tiendas y la de productos. Contiene los precios asociados a un producto y una tienda:

PRECIOS		
CAMPO	**TIPO**	**DESCRIPCIÓN**
idProducto	Int	Identificador del producto
idTienda	Int	Identificador del tipo de producto
idUsuario	Int	Nombre del producto
Precio	Decimal	Marca del producto
FechaAnadido	Timestamp	Descripción del producto

Tabla 8. Estructura de la tabla Precios

Esta tabla no contiene clave primaria, pues es una tabla intermedia.

5.4.9 Diseño final y relaciones

El diseño final de la base de datos es el siguiente:

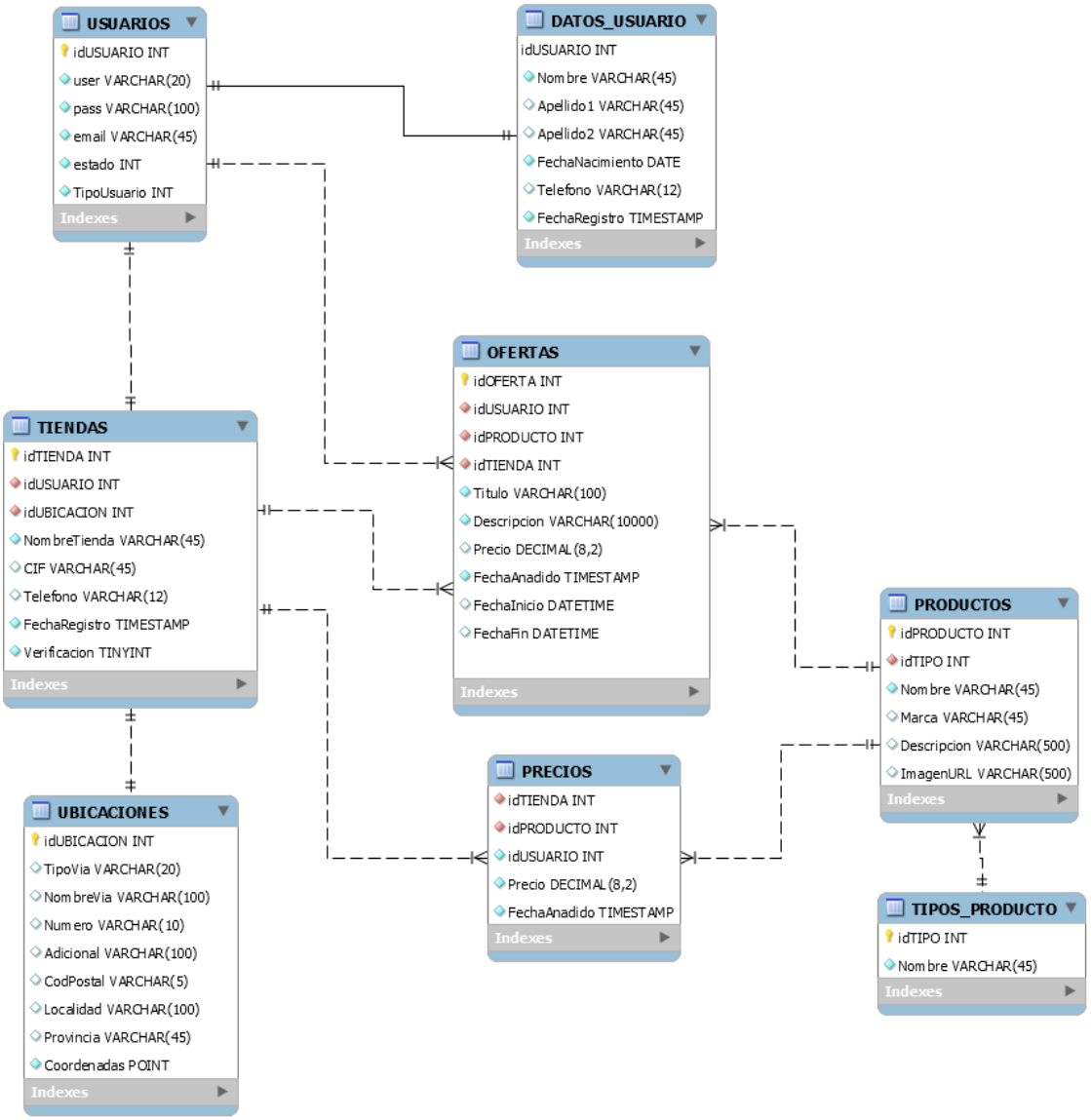

Figura 14. Diseño final de la base de datos

En la imagen se pueden ver las relaciones creadas entre las tablas:

- La tabla Usuarios se une con una relación 1 a 1 a la tabla Datos_Usuario, a través del campo idUsuario.
- La tabla Usuarios se une con una relación 1 a muchos a la tabla Ofertas, a través del campo idUsuario. Es decir, un usuario puede haber creado muchas ofertas.
- La tabla Usuarios se une con una relación 1 a 1 a la tabla Tiendas, a través del campo idUsuario. Esto significa que una tienda solo puede haber sido creada por un usuario.
- La tabla Tiendas tiene una relación 1 a 1 con la tabla Ubicaciones, a través del campo idUbicacion.

- La tabla Tiendas está unida a la tabla Ofertas con una relación 1 a muchos mediante el campo idTienda. Es decir, que una tienda podrá tener muchas ofertas, pero una oferta solo puede estar asociada a una tienda.
- La tabla Productos está unida a la tabla Ofertas con una relación 1 a muchos a través del campo idProducto. Es decir, que un producto podrá tener asociadas muchas ofertas, pero una oferta solo puede tener asociado un producto.
- La tabla Productos está unida a la tabla Tipos_Producto mediante una relación 1 a muchos con el campo idTipo. Esto significa que un tipo de producto puede tener asociados muchos productos, pero un producto solo puede estar asociado a un tipo.
- La tabla Tiendas está asociada a la tabla Productos mediante una relación muchos a muchos establecida a través de la tabla intermedia de Precios y los campos idTienda e idProducto.

Capítulo 6. Desarrollo y resultados del trabajo

En este capítulo se va a explicar la creación y programación de cada una de las partes de la aplicación.

6.1 Creación de la base de datos

Para crear la base de datos se ha utilizado la herramienta oficial MySQL Workbench. Esta herramienta permite diseñar, crear, gestionar y administrar las bases de datos de manera visual, simplificando en gran medida dichas tareas. La herramienta va incluida en el paquete MySQL Community Edition, que contiene todo lo necesario para crear el servidor de base de datos, por lo que no es necesario instalar ninguna herramienta adicional.

Cuando instalamos el paquete, nos pide configurar la conexión al servidor de bases de datos. En este caso, al ser desarrollo en local, seleccionaremos la opción "Development computer" y dejaremos los valores por defecto: usar el protocolo TCP/IP y utilizar el puerto 3306. Después nos pide crear una contraseña para la cuenta de administrador. Estos son los datos que tendremos que introducir en la interfaz visual y en el servidor de la API REST para conectar con la base de datos.

Una vez instalado y configurado MySQL, arrancamos la herramienta MySQL Workbench. Se mostrará la ventana del programa, donde nos dará a elegir entre crear conexiones a la base de datos, crear diagramas de bases de datos y realizar migraciones.

6.1.1 Conexión a la base de datos

En primer lugar, creamos la conexión a la base de datos que utilizará el programa para conectar con nuestro servidor MySQL:

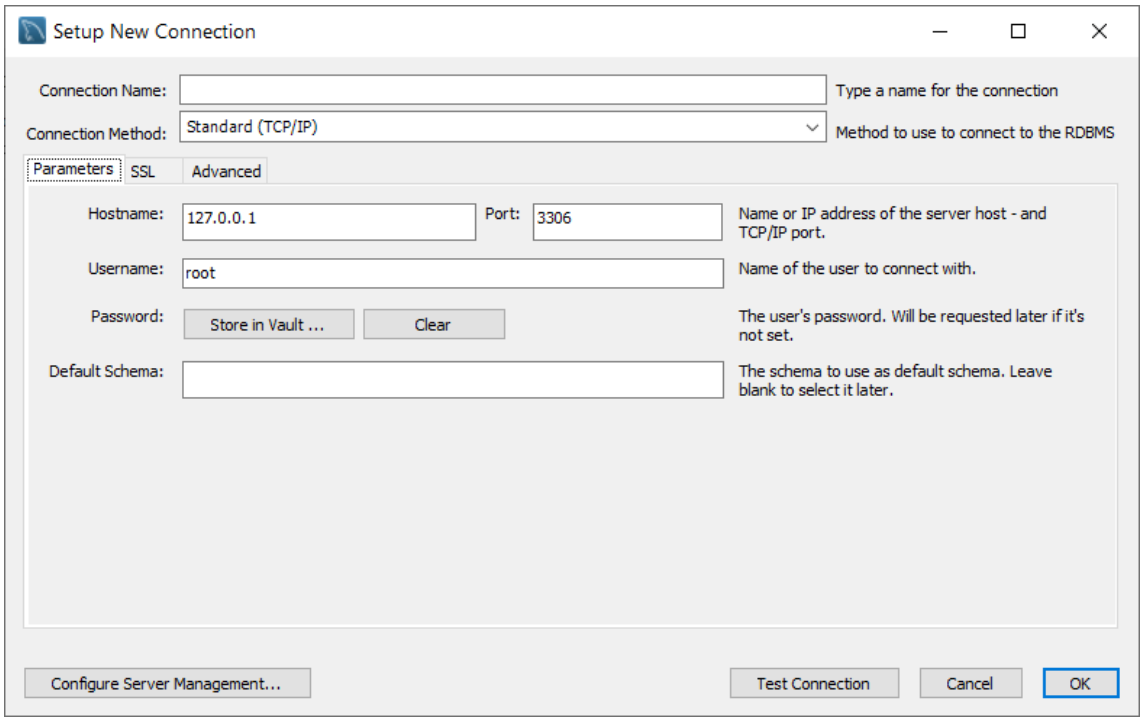

Figura 15. Configuración de la conexión al servidor de base de datos

Introducimos un nombre para identificar la conexión y elegimos conexión mediante TCP/IP. Después introducimos la IP donde se aloja el servidor, que en este caso es el propio ordenador, y

el puerto que hemos establecido al instalar el paquete. Por último, tenemos que introducir el usuario y contraseña de acceso al servidor, que serán las de administrador.

6.1.2 Creación del esquema

Una vez creada la conexión podremos ver las bases de datos (llamadas *Schemas* o esquemas) que hay en el servidor, con sus tablas, vistas, funciones y procedimientos. De momento únicamente aparece el esquema que crea el sistema, por lo que tendremos que crear uno nuevo pulsando el botón de añadir esquema. El nombre elegido para la base de datos es "appcomerciolocal" y se utilizarán los valores por defecto.

Para rellenar nuestra base de datos podemos crear las tablas y las relaciones manualmente, mediante las opciones que nos da la herramienta, mediante sentencias SQL, o de manera gráfica mediante la herramienta de creación de diagramas. Para acceder a la herramienta gráfica vamos a la pantalla inicial, seleccionamos la opción "Models" y creamos uno nuevo. Seleccionamos nuestro esquema y se abrirá una nueva pestaña con la herramienta visual.

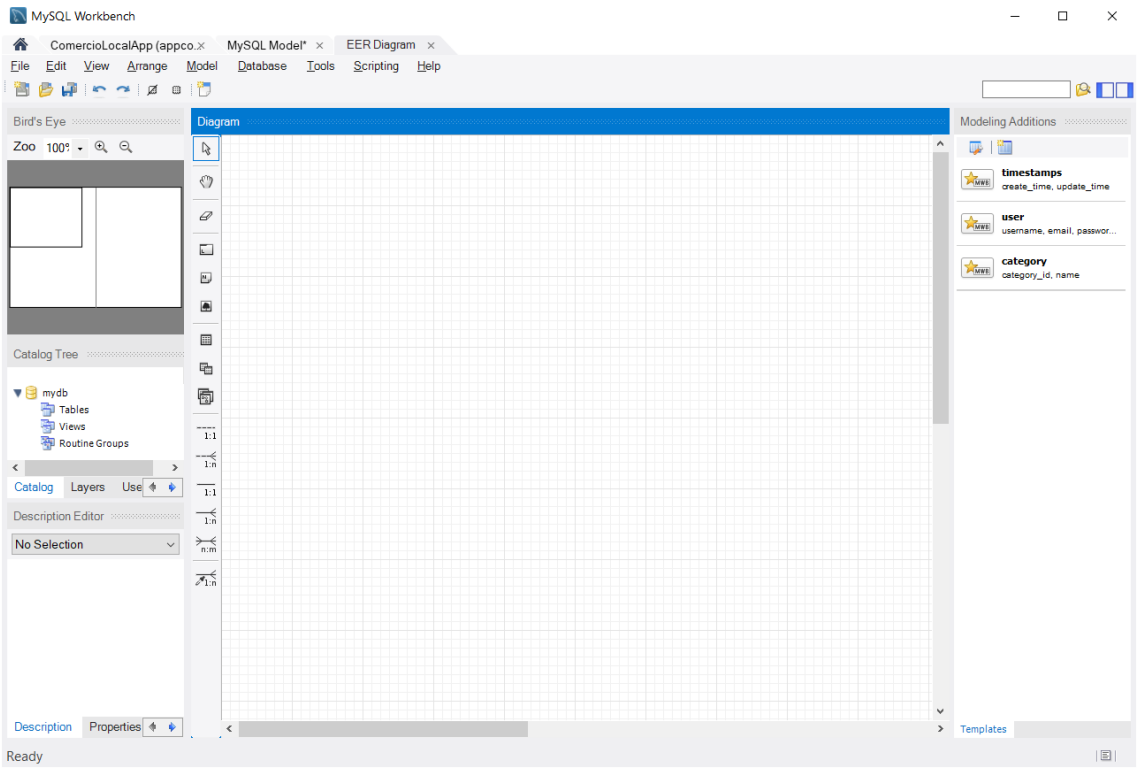

Figura 16. Creación de un diagrama de base de datos con MySQL Workbench

El diagrama se creará utilizando los botones que aparecen en la parte izquierda de la interfaz, que permiten crear las tablas, vistas y relaciones.

Para crear una nueva tabla, pulsamos en el botón de Tablas y después en una zona libre del diagrama. Haciendo doble clic se abre en la parte inferior una ventana donde se pueden añadir las columnas de la tabla, con su tipo de variable, restricciones y valores por defecto.

Para crear relaciones entre tablas, hay que pulsar el botón de la relación que corresponda y después hacer clic en los campos en los que se quiere crear la Foreign Key.

Una vez creado el diagrama, habrá que trasladarlo a la base de datos creada. Para ello se pulsa en la opción "Forward Engineer", que generará el script SQL que creará la base de datos y nos dará opción a ejecutarlo. Al ejecutarlo, la base de datos estará creada y lista para funcionar.

6.2 Creación del servidor

La parte del servidor se ha creado con Node.js mediante el entorno de desarrollo Microsoft Visual Studio Community Edition. Este IDE es gratuito y compatible con numerosos lenguajes de programación, como JavaScript, que es el que se necesita para desarrollar esta parte.

El desarrollo del servidor se ha hecho de forma conjunta junto al del cliente para Android, ya que cada funcionalidad del cliente requiere su parte correspondiente en el servidor. De esta forma se me ha hecho mucho más sencillo desarrollar la aplicación, al poder dividir el desarrollo en partes más pequeñas e ir probándolas de forma aislada.

6.2.1 Herramientas necesarias

A continuación, se describen las herramientas que se han utilizado para la creación del servidor.

6.2.1.1 Node.js

Para desarrollar en Node.js es necesario instalar primero el entorno de ejecución, que se puede descargar de su página web. Node ofrece una versión LTS (Soporte a largo plazo) que es la recomendada, y una versión actualizada que ofrece las últimas características [14]. En este caso se ha instalado la versión LTS.

Este paquete incluye la herramienta Npm, que es el sistema de gestión de paquetes que utiliza Node.js por defecto. Con esta herramienta se podrán instalar aplicaciones Node.js que se encuentren en el repositorio desde la ventana de comandos [15].

6.2.1.2 Microsoft Visual Studio Community Edition

Para hacer más fácil la tarea de programación, se ha utilizado el entorno de desarrollo Microsoft Visual Studio en su versión gratuita Community. En principió utilicé la herramienta Notepad++ y la ventana de comandos, pero con este IDE puedo tenerlo todo integrado en la misma ventana, visualizar la estructura de carpetas y archivos, etc.

Para poder utilizarlo para desarrollar aplicaciones Node.js, se debe seleccionar esta opción durante la instalación del programa y tener instalado el entorno de ejecución.

6.2.1.3 Postman

Postman es una herramienta gratuita que permite crear peticiones HTTP que se enviarán a la API del servidor. En este Trabajo de Fin de Grado se va a utilizar para probar las rutas creadas en la API y ver de manera clara y sencilla las respuestas que devuelve nuestro servidor. De esta manera es mucho más sencillo comprobar que el servidor funciona correctamente y corregir errores.

6.2.2 Frameworks Node.js

El desarrollo en Node.js es mucho más sencillo si se utiliza un marco de desarrollo (o Framework) y módulos. De esta manera podemos utilizar numerosas funciones que, de otra manera, serían largas y difíciles de programar.

6.2.2.1 Express

El Framework utilizado para el desarrollo del servidor es Express. Este es el Framework más utilizado en Node.js y ofrece un conjunto básico de características que permiten crear un servidor HTTP, el manejo y direccionamiento de rutas, conexiones con bases de datos, uso de middlewares, etc. [15].

6.2.2.2 Sequelize

Sequelize es un Framework ORM (Object-Relational Mapping) que permite leer las tablas de la base de datos y convertirlas en datos compatibles con el servidor. Dicho de una forma más clara, convertirá la información que reciba de la base de datos en objetos y variables que luego podremos utilizar y manejar como queramos. Ofrece soporte para múltiples tipos de bases de datos, como MySQL, SQLite, PostgreSQL…

Por lo tanto, este Framework el encargado de procesar las peticiones y respuestas que se hagan en el uso de la aplicación.

6.2.3 Módulos Node.js

Además de los Frameworks descritos, para el desarrollo del servidor se han utilizado los siguientes módulos:

1. **Morgan**: es un módulo de Express que registra las peticiones HTTP que llegan al servidor y las muestra por consola. Gracias a él podremos saber qué esta pasando en el servidor.
2. **JSONWebToken**: es un módulo que permite implementar en el servidor la autentificación mediante JWT, explicada en un capítulo anterior.
3. **BcryptJS**: es una librería cuya principal función es realizar funciones de cifrado y hash. En el servidor será el encargado de realizar el hash de las contraseñas y manejarlas.
4. **MySQL2**: es el cliente MySQL para Node.js, necesario para que el Framework Sequelize pueda establecer la conexión con la base de datos.
5. **Body-Parser**: este módulo se utiliza para convertir datos a formato JSON. Será necesario para manejar los cuerpos de las peticiones y respuestas.
6. **Nodemon**: es una herramienta de Npm que monitoriza los cambios que se producen en el código del servidor. Cuando se produce algún cambio, reinicia automáticamente el servidor para que se ejecute con los nuevos cambios aplicados, evitando así tener que cerrarlo e iniciarlo de nuevo cada vez que programamos algo nuevo.

6.2.4 Creación del proyecto y estructura de carpetas

Para crear el proyecto, se inicia Visual Studio y vamos a Archivo => Nuevo => Nuevo proyecto. Seleccionamos la opción “Crear una aplicación de consola en blanco de Node.js”, le ponemos el nombre al proyecto, elegimos la carpeta donde queremos guardarlo y le damos a Finalizar.

El IDE nos habrá creado un proyecto con la estructura básica de Node.js. Esta estructura incluye dos archivos que se encuentran en la carpeta raíz de nuestro proyecto:

- **Package.json**: es un archivo en formato JSON que incluye los datos del proyecto. Nombre, versión, descripción, autor, scripts, licencia y dependencias.
- **Index.js**: es el archivo principal del proyecto y será la puerta de entrada al servidor. Por lo tanto, cuando iniciemos el servidor, todo lo que se incluya en este archivo se ejecutará.

Como se explicó en el capítulo de diseño, para crear el servidor se tienen que implementar rutas, modelos y funciones:

- **Rutas**: se crearán todas en el mismo archivo, que se llamará rutas.js y estará en la raíz del proyecto.
- **Modelos**: se creará un modelo por cada tabla, donde se incluirán todos los campos, tipos de variable, restricciones y relaciones con el resto de tablas. En la base de datos del proyecto se han creado ocho tablas, por lo que será necesario crear ocho archivos con extensión ".js" distintos, que tendrán el nombre de la tabla de referencia e irán dentro de la carpeta "modelos". Dentro de esta carpeta también se creará un archivo llamado "index.js" (no confundir con el archivo index de la carpeta raíz), donde se incluirá la configuración de la conexión a la base de datos y que servirá de acceso a los modelos al resto de archivos del proyecto.
- **Funciones**: las funciones se agruparán según la funcionalidad del cliente Android que vayan a implementar, que he dividido en cinco grupos. Cada grupo tendrá su propio archivo de extensión ".js" y estarán dentro de la carpeta "funciones":
    - **Login**: incluye las funciones necesarias para que el usuario inicie sesión.
    - **Registro**: incluye las funciones necesarias para registrar un nuevo usuario.
    - **Perfil**: incluye las funciones necesarias para gestionar el perfil del usuario (ver, modificar y eliminar).
    - **Tienda**: incluye todas las funciones necesarias para gestionar las tiendas y los catálogos de productos.
    - **Oferta**: incluye todas las funciones necesarias para gestionar las ofertas

Por último, para implementar la autentificación, el módulo JsonWebToken necesita la creación de una carpeta llamada "config", donde se incluye un archivo de formato JSON en el que se guardará la clave secreta.

Por lo tanto, la estructura final del proyecto del servidor quedará de la siguiente forma:

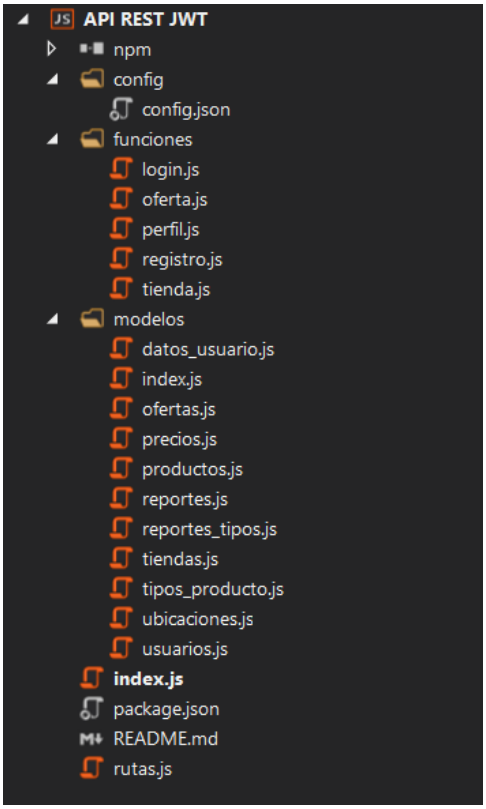

Figura 17. Estructura de archivos del servidor

6.2.5 Programación del servidor

Una vez clara la estructura del proyecto, se procede a la creación y programación del servidor y la API REST.

6.2.5.1 Instalación y configuración del Framework Express y sus módulos

El primer paso a la hora de programar el servidor es descargar e instalar los Frameworks y módulos necesarios, que se han descrito con anterioridad. Para esta tarea haré uso del administrador de paquetes de Node.js llamado Npm.

Para instalar cualquier paquete que se encuentre en el repositorio de Node.js, sólo es necesario ejecutar en la consola el siguiente comando:

```
npm install --save <nombre del paquete>
```

Si se quiere instalar varios paquetes, se pueden concatenar sus nombres en el mismo comando, evitando de esta forma tener que ejecutar el mismo comando reiteradamente. Npm buscará en su repositorio los paquetes y los descargará al proyecto.

Una vez instalados los paquetes, debemos importarlos al proyecto. En primer lugar, importaremos el Framework Express, que es la base del proyecto. Para ello debemos ir al archivo index.js de la raíz del proyecto y escribir las siguientes líneas, donde se crea una variable en la que se importa el framework y otra donde se inicializa:

```
var express = require('express');
var app = express();
```

Después se importan los módulos, que necesitan Express para poder ejecutarse. Estos módulos en Node.js se denominan “Middleware”, y son grupos de funciones que siempre se van a ejecutar entre la petición y la respuesta del servidor. El proceso es similar, solo que, en el caso de los módulos en vez de inicializarlos en una variable, tendremos que indicarle a Express que los use. Por ejemplo, para importar el módulo body-parser tendremos que incluir las siguientes líneas:

```
var bodyparser = require('body-parser');
app.use(bodyparser.json());
app.use(bodyparser.urlencoded({extended: true}));
```

De esta forma le indicamos a Express que queremos que use el módulo body-parser cada vez que llegue una petición para analizar las peticiones HTTP y convertir los datos a JSON, y viceversa.

El proceso para importar el resto de módulos es idéntico, por lo que no se va a explicar.

6.2.5.2 Creación del servidor

Como ya se ha dicho con anterioridad, crear un servidor con Node.js es muy sencillo. Una vez instalado y configurado Express, solo hay que añadir las siguientes líneas al archivo index.js para crear el servidor en el puerto que queramos:

```
var puerto = process.env.PORT || 4000;
app.listen(puerto);
console.log('Escuchando en ' + puerto);
```

Con el comando app.listen se crea el servidor, que procesará todas las peticiones que se reciban por el puerto 4000.

6.2.5.3 Configuración del Framework Sequelize y conexión con la base de datos

La configuración de Sequelize se hará en el archivo index.js de la carpeta modelos. Para importarlo, al igual que se ha explicado en el apartado anterior, se crea una variable llamada "Sequelize". Después creamos una nueva variable llamada "sequelize" (en minúsculas), donde se inicializará el framework.

Para inicializarlo, debemos pasarle como parámetros el nombre de nuestra base de datos, el usuario y la contraseña de acceso. Posteriormente se incluyen todos los parámetros de configuración, como el host donde está alojada la base de datos, el tipo, la zona horaria, etc. También permite crear un pool de conexiones para mejorar el rendimiento del servidor:

```
var sequelize = new Sequelize('appcomerciolocal', 'user', 'pwd', {
    host: 'localhost',
    dialect: 'mysql',
    define: {
        timestamps: false,
        freezeTableName: true
    },
    dialectOptions: {
        useUTC: false, // for reading from database
    },
    timezone: 'Europe/Madrid',
    pool: {
        max: 5,
        min: 0,
        idle: 10000
    }

});
```

En este caso, le estamos indicando a sequelize que la base de datos está en local, es de tipo MySQL y hemos creado un pool de conexiones con la base de datos donde no hay un mínimo de conexiones, un máximo de 5 y las conexiones se cierran cuando no se hayan utilizado durante 10.000 milisegundos.

6.2.5.4 Creación de los modelos

Sequelize es una herramienta muy potente que permite, entre otras cosas, sincronizar los datos de su configuración y los modelos del servidor con la base de datos. De esta forma, no sería necesario crear la base de datos de forma manual como se ha hecho, ya que se podrían programar los modelos en el servidor y Sequelize se encargaría de crear la base de datos con todas las tablas y parámetros que hayamos incluido. Sin embargo, esta tarea es bastante más complicada que crear la base de datos de manera visual como se ha hecho, por lo que se va a realizar el proceso inverso: se van a crear los ficheros con los modelos de cada tabla a partir de la base de datos creada.

Para ello necesitamos instalar el módulo "sequelize-auto" de forma similar al resto de módulos. Una vez instalado, lo invocamos en la ventana de comandos indicando los parámetros de conexión a la base de datos y la ruta donde se guardarán los ficheros, y el módulo generará los modelos con el formato requerido por Sequelize para que funcionen correctamente en el servidor. Tras ejecutarlo, importamos los modelos al proyecto y los colocamos en la carpeta "modelos".

Para que Sequelize reconozca los ficheros de los modelos, primero hay que importarlos en el archivo index.js de la carpeta modelos. Por ejemplo, para importar el modelo de la tabla Usuarios se debe incluir esta línea, donde se importa el modelo en una variable llamada "Usuarios":

```
let Usuarios = require(path.join(__dirname, 'usuarios')) (sequelize,
Sequelize);
```

Para que se pueda acceder al modelo desde el resto de la aplicación, debemos exportarlo:

```
exports.Usuarios = Usuarios;
```

Cada vez que necesitemos acceder a la tabla de usuarios desde otros ficheros, solo tendremos que importar el modelo en el fichero como si fuera un módulo más, y tendremos disponibles para usar todas las funciones de Sequelize para el módulo.

6.2.5.5 Creación de funciones

Para que las rutas sepan qué hacer cuando las llaman, deben invocar a una función. Las funciones creadas para implementar la API del servidor se pueden clasificar en dos tipos:

1. **Funciones que devuelven una query:** cuando una ruta llama a una de estas funciones, le pasará unos parámetros y la función le devolverá una query (o sentencia) SQL, que la ruta ejecutará contra la base de datos para obtener un resultado.
2. **Funciones que devuelven una promesa**: cuando una ruta llama a una de estas funciones, le pasará unos parámetros y la función creará una promesa, en la que se ejecutará una función de un modelo y se resolverá o rechazará según el resultado de la ejecución. La promesa será devuelta a la ruta, que generará una respuesta según el resultado de la promesa.

Como se ha dicho anteriormente, las funciones se han organizado en cinco ficheros según el tipo de funcionalidad que implementan. En cada fichero será necesario importar los modelos y módulos necesarios. Por ejemplo, el fichero de funciones login, utilizado para iniciar sesión, requiere importar el módulo BcryptJS para comprobar la contraseña del usuario. Otros ficheros de funciones, como los de perfil o oferta solo requieren importar los modelos. A su vez, si en

alguno de los ficheros hay funciones que utilizan varios modelos de sequelize, será necesario especificar las relaciones entre modelos para obtener los resultados esperados.

Para verlo de una forma más clara, se va a explicar el contenido del fichero de funciones "perfil.js", que es el más sencillo:

En primer lugar, se importan los modelos en la variable "modelos":

```
var modelos = require('../modelos');
```

Después se especifican las relaciones entre los modelos que van a utilizar las funciones:

```
modelos.Usuarios.hasOne(modelos.DatosUsuario,
{ foreignKey: 'idUSUARIO', sourceKey: 'idUSUARIO' });

modelos.DatosUsuario.belongsTo(modelos.Usuarios,
{ foreignKey: 'idUSUARIO', targetKey: 'idUSUARIO' });
```

En la primera línea estamos indicando que en la variable "modelos" hay un objeto llamado "Usuarios", que tiene una relación 1 a 1 con el modelo "DatosUsuario" mediante una Foreign Key en el campo "idUSUARIO".

En la segunda estamos indicando que existe la misma relación, pero en sentido contrario. En este caso se utiliza el método "belongsTo" ya que es la parte dependiente de la relación.

Por último, se crean las funciones. En el caso de este fichero hay una única función:

```
exports.ObtenerDatosUsuario = function (idUSUARIO) {
    var promesa = new Promise(function (resolve, reject) {
        modelos.DatosUsuario.findOne ({
            where: { idUSUARIO: idUSUARIO },
            include: [{ model: modelos.Usuarios, attributes: ['user', 'email',
'estado', 'TipoUsuario'] }]
        })
            .then(function (datos_usuario) {
                if (!datos_usuario) {
                    reject({ status: 404, mensaje: 'No existe el usuario' });
                } else {
                    console.log('Usuario encontrado. Seguimos procesando');
                    //console.log(datos_usuario);
                    resolve({ status: 200, datos_usuario });
                }
            })
            .catch(function (err) {
                console.log(err);
                reject({ status: 500, mensaje: 'Error interno' });
            })
    })
    return promesa;
}
```

La función sirve para obtener los datos de un usuario y requiere que se le pase como parámetro del id del usuario.

Dentro de la función se crea una promesa, donde se hace una llamada al método "findOne" del modelo "DatosUsuario" y además se incluyen algunos campos del modelo "Usuarios". Después, en función del resultado, se resuelve la promesa enviando el código de estado 200 y el objeto donde se almacena el resultado que devuelve el método findOne, o se rechaza y se envía un mensaje de error. Por último, se devuelve la promesa.

Como se ha podido observar, Sequelize proporciona a los modelos una serie de funciones y métodos que servirán para lanzar sentencias contra la base de datos sin necesidad de crear sentencias SQL. Esto es muy útil si no se maneja el lenguaje SQL, o si se quiere añadir una capa de seguridad contra ataques SQL Injection. En el ejemplo mostrado, se utiliza el método findOne del modelo DatosUsuario, que sirve para obtener el registro de dicha tabla donde el id de usuario sea el que se ha pasado a la función. Sería equivalente a ejecutar la siguiente sentencia SQL:

```
SELECT d.*, u.user, u.email, u.estado, u.TipoUsuario
FROM Datos_Usuario d, Usuarios u
WHERE d.idUSUARIO = u.idUSUARIO
AND d.idUSUARIO = "idUsuario pasado a la función"
```

Sin embargo, cuando se requiere hacer una consulta compleja, utilizar estos métodos puede ser un proceso muy engorroso y frustrante, por lo que es más sencillo hacerla manualmente.

6.2.5.6 Creación de rutas

Las rutas van a ser las encargadas de recibir las peticiones y devolver las respuestas al cliente. Cuando se reciba una petición, se buscará la ruta en el servidor. La ruta hará una llamada a una función, que le devolverá una serie de datos, y procesará esos datos para crear una respuesta que enviará de vuelta al cliente.

Para el manejo de las rutas del servidor, se ha creado en el directorio raíz el fichero "rutas.js". Para que las rutas sean accesibles, se ha incluido en el fichero index.js del directorio raíz un middleware que llama a las rutas.

Express ofrece soporte para varias formas de enrutamiento: directo, modular, mediante métodos… En este caso se ha optado por utilizar la clase de Express "Router", que permite crear manejadores de rutas montables y modulares [17].

El primer paso para la creación del fichero es importar la clase Router de Express, las funciones, los módulos de autentificación y la configuración del método de autentificación. Una vez importados, se pueden crear las rutas.

Todas las rutas se crean con la misma sintaxis:

```
rutas.metodo('/direccion', function (req, res) {...});
```

"Rutas" hace referencia a la variable donde se ha importado la clase Router, el método será uno de los cuatro métodos HTTP que se nombraron en la explicación de REST (GET, POST, PUT o DELETE), y la dirección será la ruta donde tendrá que llamar el cliente para obtener un resultado. En algunas ocasiones, las direcciones pueden incluir parámetros necesarios para llegar

al resultado requerido. Por ejemplo, si se quiere obtener la ubicación de una tienda en concreto, habrá que incluir el parámetro idTienda en la ruta. Para indicar que es un parámetro, se añaden dos puntos al inicio del parámetro:

```
rutas.get('/tiendas/:idTIENDA/ubicación', function(req, res)…
```

Cuando se hace una llamada a un método POST, la petición incluirá un cuerpo con los parámetros que se quieren tratar en la base de datos. Estos parámetros serán accesibles desde la variable req. En la variable res se incluirán los datos que se enviarán en la respuesta. Estos datos serán siempre un código de estado y un objeto JSON, o en su defecto un mensaje.

En el siguiente ejemplo se muestra la ruta utilizada para crear una oferta:

```
rutas.post('/oferta', function (req, res) {
    const { idUSUARIO, idPRODUCTO, idTIENDA, Titulo, Descripcion, Precio,
FechaInicio, FechaFin } = req.body;
    const newOferta = { idUSUARIO, idPRODUCTO, idTIENDA, Titulo, Descripcion, Precio,
FechaInicio, FechaFin };

    console.log('Se va a insertar la oferta: ');
    console.log(newOferta);
    oferta.CrearOferta(newOferta.idUSUARIO, newOferta.idPRODUCTO, newOferta.idTIENDA,
newOferta.Titulo, newOferta.Descripcion, newOferta.Precio, newOferta.FechaInicio,
newOferta.FechaFin)
        .then(function (result) {
            res.status(result.status).send(result.nuevaOferta)
        }, function (error) {
            res.status(error.status).send(error.mensaje);
        });
});
```

En el ejemplo se observa que dentro de req.body (es decir, dentro del cuerpo de la petición) están todos los parámetros que se tienen que insertar en la base de datos para crear la oferta. Después se crea una variable con dichos parámetros y se hace la llamada a la función oferta.CrearOferta. Cuando la función devuelve un resultado, la ruta lo analiza y añade en la variable res (response, o respuesta) el código de estado y el objeto que le ha devuelto la función.

Como se ha explicado anteriormente, las rutas pueden llamar a funciones que devuelven promesas o queries. En el primer caso, la llamada a la base de datos y el procesado de la respuesta lo hace la función, mientras que la ruta solo fabrica la respuesta y la envía. Cuando se llama a una función que devuelve una query, la función simplemente devuelve una cadena de caracteres que contiene la sentencia SQL necesaria para realizar la petición, y la ruta será la encargada de mandarla a la base de datos, procesarla, fabricar la respuesta y enviarla al cliente.

6.3 Creación de la aplicación Android

El cliente para Android se ha creado con el entorno de desarrollo oficial Android Studio, utilizando Java como lenguaje de programación. Este IDE incluye todas las herramientas necesarias para desarrollar el cliente, depurar y una máquina virtual de Android que me servirá para probar la aplicación.

Para crear el proyecto, abrimos la aplicación y seleccionamos la opción "Create New Project". El programa nos da a elegir entre varios tipos de plantillas que crean aplicaciones por defecto, o la opción que se va a elegir, que es crear un proyecto vacío. En la siguiente pantalla nos pide ponerle el nombre, la localización del proyecto, el lenguaje de programación y la versión de Android para la que se quiere desarrollar. Dependiendo de la versión elegida, nuestra aplicación será compatible con un numero mayor o menor de dispositivos. Por ejemplo, la versión seleccionada para este proyecto ha sido la versión 6.0, que, como indica el programa, hará que la aplicación sea compatible con el 84.9% de los dispositivos Android que hay en el mundo. Si elegimos la última versión, nos indica que nuestra aplicación será compatible con menos de un 1% de los dispositivos.

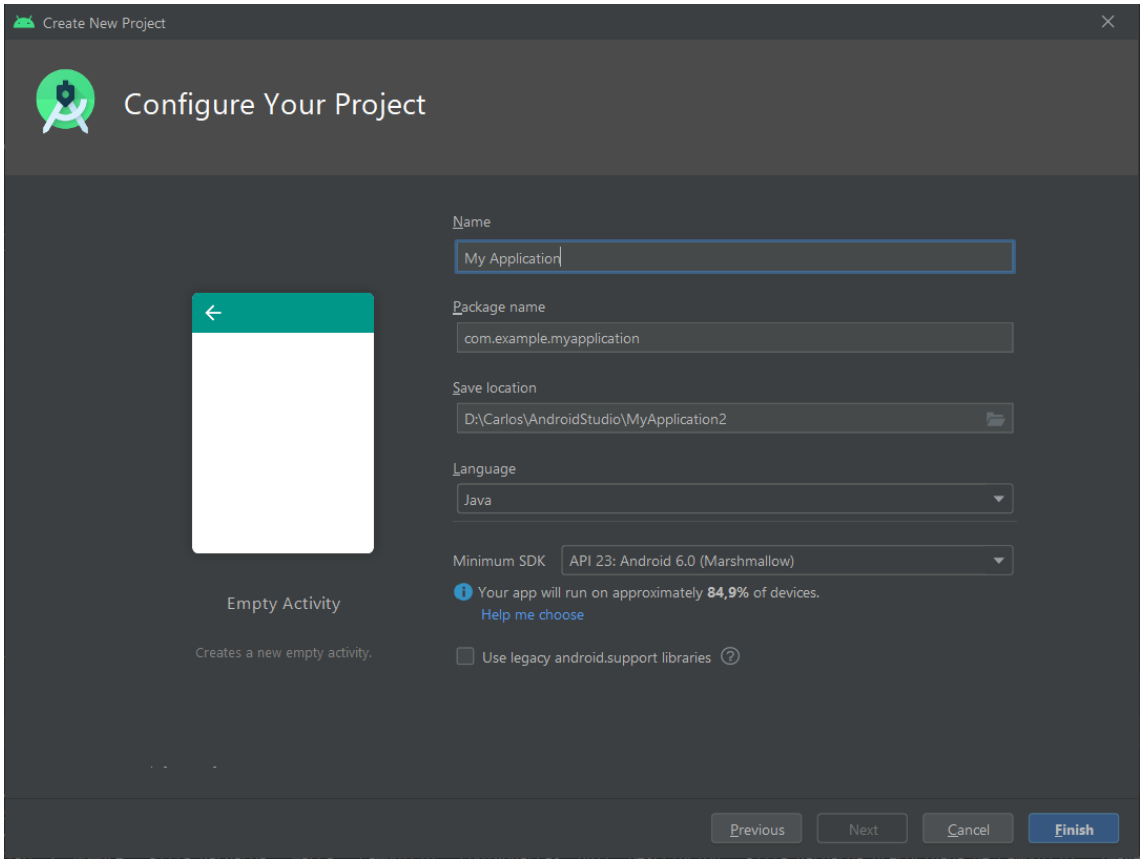

Figura 18. Creación de un nuevo proyecto en Android Studio

6.3.1 Librerías

Las librerías que han sido necesarias para desarrollar el proyecto son muy similares a las que agregaron al servidor:

- **Retrofit**: es una de las librerías más utilizada en el desarrollo de Android. Nos permite realizar peticiones a la API REST del servidor, simplificando el proceso.
- **GSON**: sirve para convertir objetos Java en JSON y viceversa.
- **Picasso**: es un gestor de imágenes que permite cargarlas en la aplicación, redimensionarlas y modificarlas.
- **RxJava**: se encarga de gestionar todas las tareas de gestión de hilos. Se utiliza en combinación con Retrofit para evitar callbacks anidados simplificar los procesos.

Para agregar las librerías al proyecto, habrá que ir al archivo de Gradle nombrado en el apartado anterior y añadirlas en la parte de dependencias.

6.3.2 Estructura de la aplicación

Para explicar la estructura básica de la aplicación, se va a utilizar la siguiente imagen, que muestra la estructura de carpetas del proyecto recién creado.

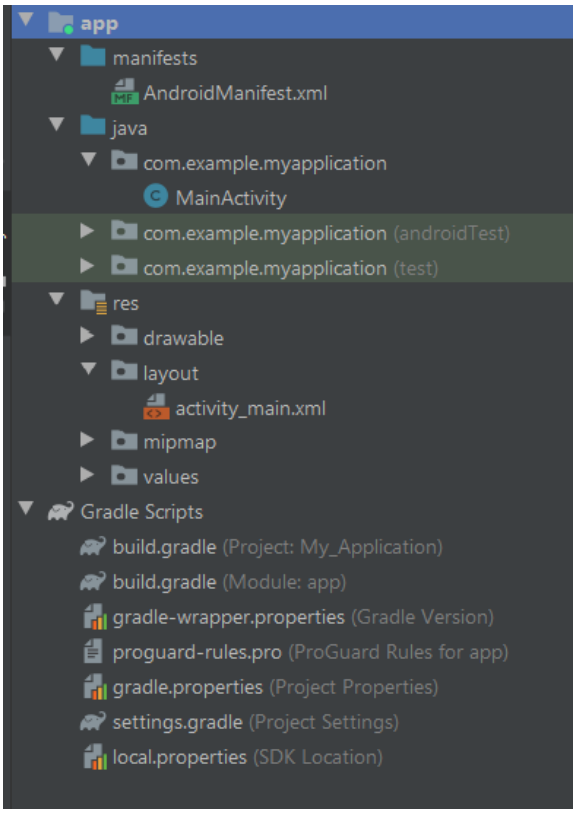

Figura 19. Estructura de carpetas del proyecto de Android Studio

En la carpeta manifests está el archivo AndroidManifest.xml. Este archivo contiene los datos básicos de la aplicación: nombre, icono, tema, activities, permisos que necesita la aplicación, etc.

En la carpeta que tiene el nombre del paquete de nuestro proyecto, se crearán todas las activities y clases de Java necesarias para ejecutar la aplicación. Dentro se pueden crear nuevas carpetas para organizar mejor la aplicación.

En la carpeta res (resource) estarán todos los recursos gráficos a los que accederá nuestra aplicación. Dentro de esta carpeta encontramos, entre otras, la carpeta drawable, donde se colocarán todas las imágenes y vectores, la carpeta layout, donde estarán las plantillas gráficas de cada activity.

Por último, en Gradle Scripts están todos los archivos que utiliza Gradle, que es el sistema de compilación utilizado por Android Studio. En el archivo build.gradle de la app podemos ver las dependencias y librerías implementados en nuestra aplicación.

Para desarrollar la aplicación, he dividido las clases Java en cuatro categorías, por lo que he creado cuatro carpetas diferentes:

6.3.2.1 Directorio raíz

En este directorio estarán las clases pertenecientes a las Activities de la aplicación. A su vez, este directorio contendrá el resto de carpetas que contienen las clases Java.

6.3.2.2 IO (Entrada/Salida)

En esta carpeta estarán las clases Java necesarias para la conexión con el servidor. Estas clases vienen impuestas por la librería Retrofit, que las necesita para poder gestionar las peticiones.

La primera clase es la clase **ApiAdapter**. Esta clase contiene la base para crear las peticiones. Para ello, se debe crear un interceptor, que es una función de Retrofit que monitoriza, escribe y envía peticiones, y un constructor del servicio de API, en el que se incluye el interceptor y la dirección del servidor.

El segundo archivo de la carpeta no es una clase sino una interfaz de java. Se llama **ApiService** y sería el equivalente al archivo de rutas del servidor. En este archivo se definen todas las rutas con sus correspondientes métodos HTTP, parámetros y cabeceras, así como los modelos donde se guardarán los datos que se reciban.

Por último, está el archivo **LoginService**, que es una clase de Java donde se implementan las llamadas al servidor para iniciar sesión.

6.3.2.3 Model

Al igual que en el servidor, es necesario crear un modelo para cada una de las tablas de la base de datos, solo que, en este caso, los modelos serán constructores de objetos donde se alojarán los datos correspondientes a cada tabla.

Dentro de esta carpeta hay otro directorio llamado Response (respuesta), donde habrá modelos que permitirán guardar los resultados de algunas de las peticiones que contienen datos de varias tablas y no se corresponden con ningún modelo de tabla concreto.

6.3.2.4 Util

En esta carpeta estarán las utilidades y adaptadores. Los adaptadores son clases que sirven para generar recursos dinámicos en las activities, como vistas de tarjetas o marcadores de Google maps.

6.3.3 Activities

Como ya se ha explicado en capítulos anteriores, una aplicación de Android se compone de Activities, que tienen una parte visual (layout) y una parte lógica (clases de Java). Esta aplicación consta de más de 20 activities con sus respectivos layouts y otras 20 clases de Java necesarias para generar algunas partes gráficas y las peticiones al servidor, por lo que no es posible entrar a explicar con detalle cómo se ha diseñado la parte gráfica y lógica de cada una. Como muchas de las activities tienen estructuras tanto gráficas como lógicas similares, voy a explicar los aspectos más destacados de cada una de las partes.

6.3.3.1 Parte gráfica: layouts

En este apartado se van a explicar los aspectos principales de la parte gráfica de las activities de la aplicación, los layouts.

Un layout es un contenedor de elementos gráficos, que conforma la parte visual de las pantallas de una aplicación [18]. En un layout se pueden insertar elementos gráficos de todo tipo: texto, imágenes, botones, cuadros de texto, listas desplegables, otros layouts, etc. Estos ficheros tienen formato XML y se colocan en la carpeta res/layout. Android Studio ofrece numerosos tipos de layouts, que se distinguen por la manera de gestionar la colocación de los elementos, pero para crear la aplicación se han utilizado dos:

- **Constraint Layout**: se ha utilizado para crear los layouts de todas las activities. En este tipo de layout, los elementos se colocan en la pantalla mediante "constraints" (restricciones), que son conexiones entre elementos. Así, para crear la pantalla tendremos que interconectar los elementos entre ellos, ajustando distancias y márgenes hasta conseguir el resultado deseado.
- **Linear Layout**: se ha utilizado para crear las vistas de las tarjetas donde se mostrará la información de tiendas, ofertas o productos. En este tipo de layout solo se permite elegir si los elementos se colocarán en vertical, horizontal o en malla y ajustar las distancias.

Para cargar el layout en su actividad o vista correspondiente, habrá que indicarlo en su clase java. De igual manera, si se quiere acceder a los elementos del layout para, entre otras cosas, editarlos, extraer el texto que contienen, cambiar su visibilidad o asignarles acciones cuando se pulsa en ellos, se deben crear sus respectivas variables en la clase java que corresponda e importarlos.

Para generar los layouts de la aplicación se han utilizado los siguientes elementos:

- **TextView**: se utilizan para mostrar líneas de texto en la aplicación.
- **EditText**: cuadros de texto donde podremos escribir al pulsar en ellos.
- **Button**: botones a los que habrá que asignar una acción.
- **Spinner**: listas desplegables donde se podrá elegir una opción. Se pueden generar de forma estática creándolos como un recurso en la carpeta res, o de forma dinámica asignando los valores en las clases java que los carguen. En el caso de la aplicación, siempre van a ser dinámicos.
- **ScrollView**: es un contenedor que permite desplazamiento. Se utiliza cuando hay muchos elementos en pantalla y es necesario deslizar el dedo para poder verlos todos.
- **RecyclerView**: es un contenedor que permite mostrar listas de elementos, reciclando las vistas que no se muestran para optimizar el funcionamiento de la aplicación. En la aplicación se van a utilizar para mostrar las tarjetas de tiendas, ofertas y productos.
- **CardView**: son tarjetas que agrupan información. Se utilizarán para mostrar la información de tiendas, ofertas y productos.

6.3.3.2 Generación dinámica de objetos. RecyclerView y Spinner

Algunos elementos que se deben mostrar en las Activities no se pueden generar directamente en el layout ya que no tienen unos valores fijos ni hay un número fijo de elementos, sino que deben crearse de forma dinámica en función del resultado que se obtenga de una petición al servidor. Este es el caso de los RecyvlerView y los Spinner.

Para mostrar las listas de tarjetas en las activities y generarlas de forma dinámica, se hace uso de la combinación RecyclerView + CardView. Para ello, es necesario crear un "Adapter", que será el encargado de transformar los datos en tarjetas e insertarlos en el RecyclerView.

El funcionamiento de esta combinación es el siguiente:

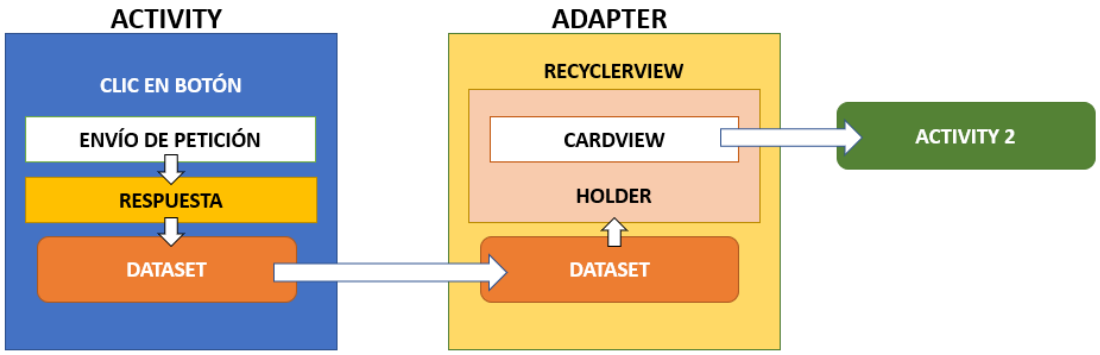

Figura 20. Funcionamiento de la combinación RecyclerView + CardView

En las actividades donde se generan listas de tarjetas, al hacer clic en un botón o seleccionar un elemento de una lista desplegable, se envía una petición al servidor donde se le pide una cierta información (listado de tiendas del usuario, ofertas, catálogo de productos de una determinada tienda, etc.). Dicha información se almacena en un dataset, que se enviará al Adapter.

El Adapter es una clase de Java que extiende la clase RecyclerView. Dentro del Adapter tenemos la case ViewHolder, donde se cargarán todos los elementos visuales de la tarjeta. Con el método onCreateViewHolder creamos el ViewHolder y le asignamos el layout de la tarjeta, y con el método onBindViewHolder asignamos los valores del DataSet que ha recibido el Adapter a cada uno de los elementos visuales de la tarjeta. De esta forma, se crearán las tarjetas de forma dinámica y se mostrarán en el RecyclerView.

Además, se puede implementar el método onClickListener para que cuando el usuario hace clic en una de las tarjetas, le lleve a una nueva actividad.

Por su parte, los Spinner se utilizarán únicamente para mostrar las tiendas de un propietario o mostrar los tipos de producto. En este caso, se manda una petición al servidor, que nos devuelve un objeto con los datos que queremos mostrar en la lista desplegable. A partir de este objeto se crea un Array que contendrá el nombre de los tipos o de las tiendas, y un HashMap, donde se asignará a cada nombre el ID que le corresponda.

Posteriormente, se crea un Adapter para el Spinner, donde se le asigna un layout y el array creado anteriormente, y mediante el método setOnItemSelectedListener podremos realizar acciones cada vez que elijamos un elemento en la lista desplegable. En el caso de esta aplicación, al seleccionar un elemento de la lista se mandará una petición al servidor para que nos devuelva la lista de productos o de ofertas de una tienda, o se haga un filtrado de productos u ofertas según el tipo de producto.

6.3.3.3 Parte lógica: envío de peticiones y recepción de respuestas

Como se ha apuntado anteriormente, el envío y recepción de peticiones se hace mediante las funciones que aporta la librería Retrofit. Para poder realizar esta tarea, necesitamos hacer uso de la clase ApiAdapter y la interfaz ApiService, cuyo funcionamiento se ha descrito en el apartado de la estructura de la aplicación.

En todas las activities y clases donde sea necesario realizar peticiones al servidor, habrá que crear una llamada (Call), donde se indicará el objeto donde se quiere almacenar la respuesta, la ruta a la que se quiere enviar la petición (almacenada en la interfaz ApiService) y los parámetros que se enviarán en la petición.

Después, con el método enqueue de Call, ponemos en cola la petición. Para poder recibir la respuesta debemos implementar el método onResponse, donde se establecerán las acciones que se realizarán cuando se reciba la respuesta del servidor, y el método onFailure, donde se asignarán las acciones a realizar en caso de que no se reciba respuesta.

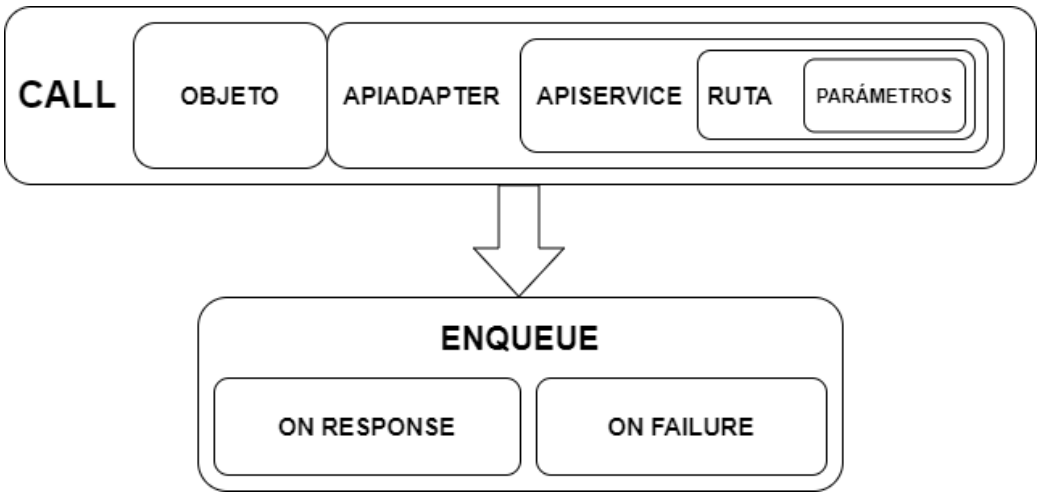

Figura 21. Esquema del funcionamiento de una llamada al servidor

6.3.4 Google Maps

La principal funcionalidad de esta aplicación es poder ver en un mapa las ofertas y precios cercanos de un producto en concreto, por lo que será necesario implementar los servicios de Google Maps en la aplicación.

Para ello, crearemos una nueva Activity en la que seleccionaremos la plantilla por defecto de Google Maps. De esta forma, Android Studio nos crea todos los archivos necesarios. Antes de poder utilizar Maps, debemos comprobar que están agregadas las dependencias de Google Maps en el archivo build.gradle de la aplicación. Después, tendremos que agregar permisos de acceso a la ubicación en el archivo AndroidManifest, y tendremos que introducir el valor de la API_KEY de Google Maps. Para conseguir esta clave, el propio IDE nos genera un enlace que nos lleva a la web de desarrolladores de Google, donde crearemos una API de Maps a la que agregaremos nuestro proyecto, y nos dará la clave.

Para agregar mapas a nuestras activities usaremos un Fragment donde se cargará el mapa. Un Fragment es un componente que amplía la funcionalidad de una activity y tiene su propio layout. Se pueden combinar varios Fragments en la misma activity e interactuar entre ellos, haciendo que la activity cambie completamente de apariencia e implementando funcionalidades completamente

distintas sin salir de ella. Aunque son realmente útiles, en el caso de esta aplicación, solo se han utilizado Fragments para agregar mapas.

Para añadir un mapa a una activity, debemos crear un Fragment en su layout, y en la clase Java habrá que asignar el Fragment al uso de Google Maps. Después tan sólo hay que crear el método onMapReady para tener el mapa funcional.

En los mapas se van a implementar tres funcionalidades: uso de marcadores para mostrar las ubicaciones de las tiendas y del usuario, detección automática de la ubicación del usuario y cambio de ubicación mediante un clic largo en un punto de la pantalla.

6.3.4.1 Generación dinámica de marcadores en Google Maps

Esta funcionalidad consiste en mostrar en el mapa marcadores con distintas formas y valores en función de la información que representen, y mostrar una tarjeta con información al pulsar en ellos. Por lo tanto, se implementará de forma muy parecida a la de generar listas de tarjetas de forma dinámica.

Una vez recibida la respuesta de la búsqueda, para crear los marcadores solo habrá que acceder a las coordenadas de cada punto, crear el icono y añadir el marcador al mapa. Para mostrar la tarjeta con la información, el proceso es idéntico al de las listas de tarjetas descrito anteriormente. Será necesario crear un Adapter al que se le pasará el resultado de la búsqueda, y éste nos devolverá la tarjeta. La única diferencia es que, en vez de mostrarse en un RecyclerView, las tarjetas se mostrarán al pulsar en el marcador.

6.3.4.2 Detección automática de la ubicación del usuario

Para detectar la ubicación del usuario y que se centre el mapa en ella, tendremos que utilizar el método setMyLocationEnabled. Al ponerlo en true, en el mapa se mostrará un punto azul con nuestra ubicación y un botón que centrará el mapa en ella.

Ahora, si queremos que la detección sea automática y el mapa cargue directamente en nuestra ubicación, tendremos que utilizar la clase LocationServices para recuperar la última ubicación registrada en el teléfono. Por último, con el método moveCamera, podemos hacer que el mapa se mueva a la ubicación que queramos, que, en este caso, será la del usuario.

6.3.4.3 Cambio de ubicación al dejar el dedo pulsando en el mapa

Esta funcionalidad se implementa de manera muy sencilla utilizando el método onMapLongClick. Este método nos permite realizar cualquier acción cuando dejemos el dedo pulsando en algún punto del mapa. En este caso, el mapa nos devolverá las coordenadas del punto que estemos pulsando, las guardaremos en una variable y crearemos un marcador que indique la nueva ubicación.

6.3.5 Implementación de las funcionalidades del cliente Android

En el capítulo 5 se describieron todas las funcionalidades que debía implementar el cliente según el tipo de usuario, y se hizo un apunte: varias funcionalidades son compartidas por ambos usuarios o muy parecidas, por lo que el número de activities se podría reducir considerablemente. Para ello, en la tabla de tiendas se creó el campo "verificación". Este campo tendrá valor 0 cuando la tienda haya sido creada por un cliente, y 1 cuando haya sido creada por el propietario. Este valor será el que haga que al pulsar un botón se cambie a una activity u otra.

En los siguientes apartados se va a mostrar cómo se han implementado dichas funcionalidades y se van a describir brevemente las activities que las componen.

6.3.5.1 Inicio de sesión y registro

Estas funcionalidades se han implementado en las activities IniciarSesion, CrearUsuario y CrearDatosUsuario. En estas activities simplemente se muestran cuadros de texto donde los usuarios tendrán que escribir sus datos:

Figura 22. Pantallas de iniciar sesión y crear usuario

Al pulsar en el botón de crear usuario se carga la activity CrearUsuario, donde nos pedirá que introduzcamos usuario, email y contraseña, y que seleccionemos el tipo de usuario que queremos crear. Después pasará a la activity CrearDatosUsuario, donde nos pedirá los datos personales y finalizará el registro. Después nos redirigirá a la pantalla principal correspondiente a nuestro tipo de usuario.

6.3.5.2 Pantallas principales: menú de usuario

Las pantallas principales de cada usuario no tienen ninguna función más allá de iniciar una nueva activity o cerrar la sesión del usuario. Corresponden a las activities ClientePrincipal y TiendaPrincipal.

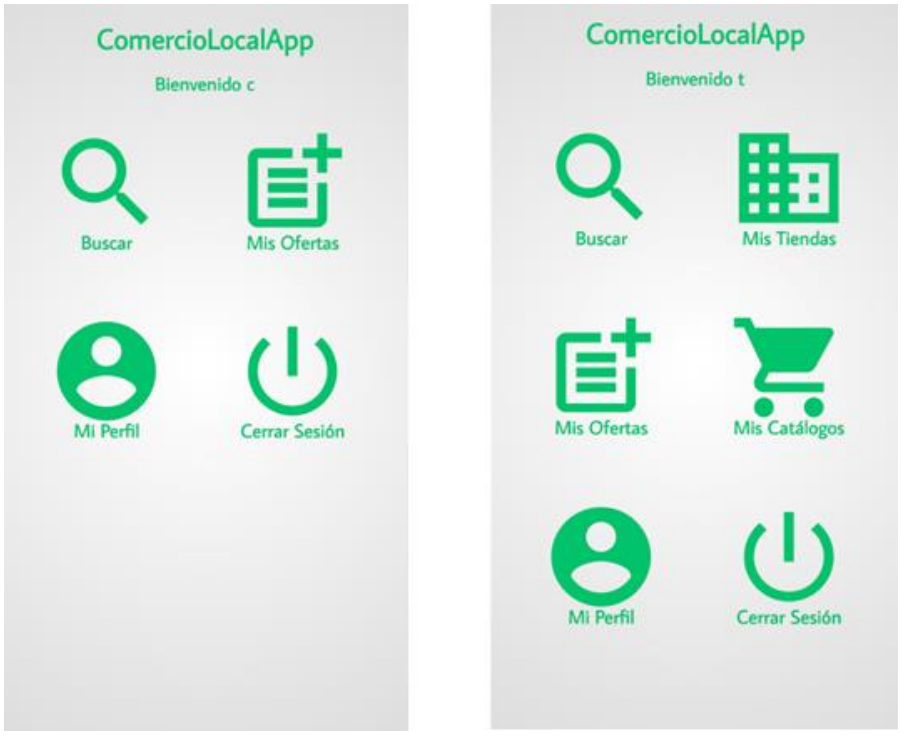

Figura 23. Pantallas principales del usuario de tipo cliente y tipo tienda

6.3.5.3 Buscar

La funcionalidad de búsqueda es compartida por los dos tipos de usuario. Se ha implementado en la activity "Buscar". En esta Activity se puede introducir una cadena de texto en el cuadro de búsqueda, y un número en el cuadro de "radio". Al pulsar el botón de buscar, se mostrarán en el mapa todos los resultados mediante marcadores indicando el precio. Los marcadores naranjas serán ofertas, y los azules precios de venta habituales.

Si pulsas en un marcador, aparecerá la información de la oferta:

Figura 24. Tarjeta de información de una oferta

Al pulsar en la tarjeta, se lanzará la activity VerOferta, donde se muestran los detalles de la oferta:

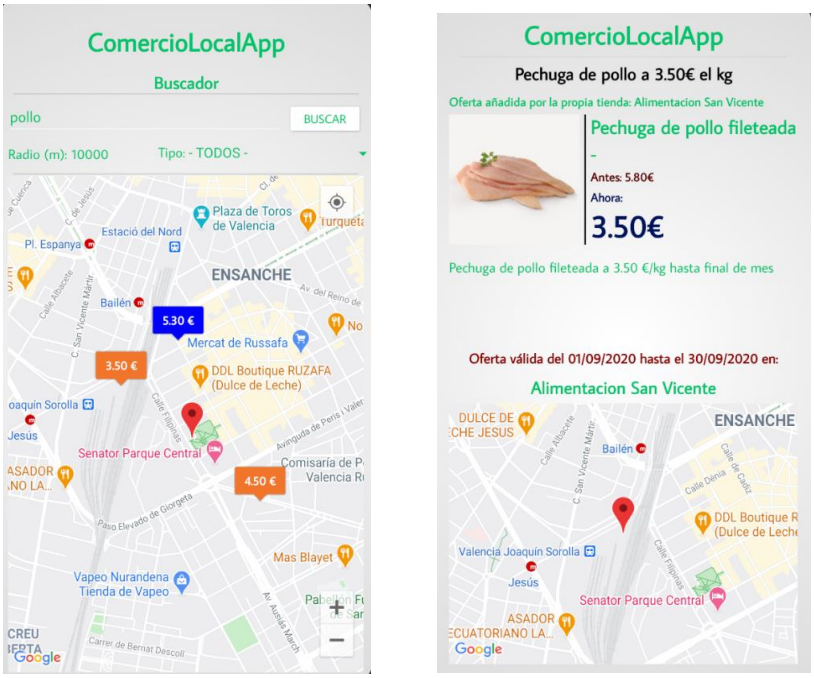

Figura 25. Pantallas de Buscar y Ver oferta

6.3.5.4 Gestionar ofertas

La funcionalidad de gestionar ofertas se ha implementado en las activities ClienteOferta y TiendaOfertas. Ambas pantallas son iguales, con la salvedad de que la del usuario de tipo tienda incluye una lista desplegable donde puede seleccionar la tienda que quiere gestionar. En la parte inferior de la pantalla se muestra la lista de tarjetas con las ofertas que el usuario haya añadido. Si se pulsa en la tarjeta, se pasa a la activity GestionarOferta, donde se da opción a eliminarla o modificarla:

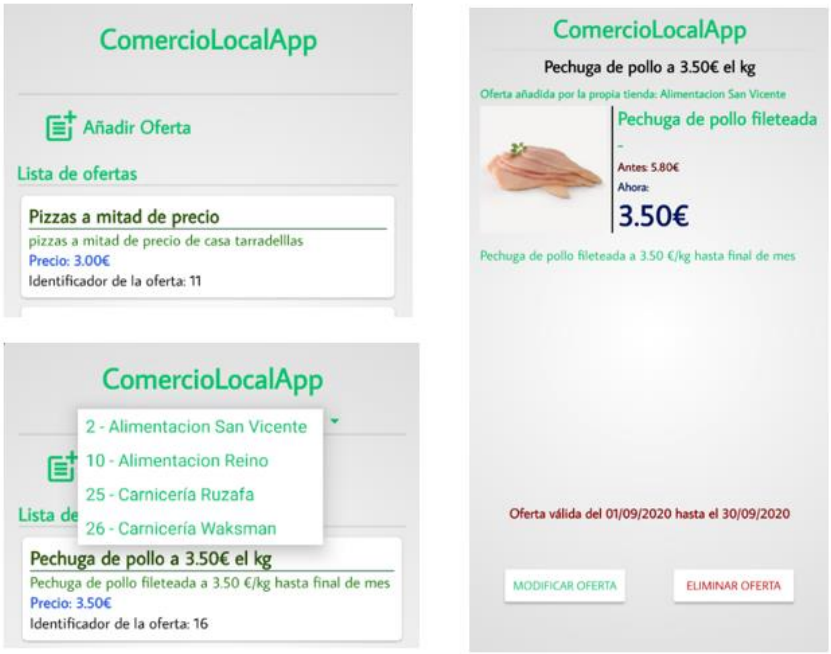

Figura 26. Pantallas de ClientesOferta, TiendasOferta y GestionarOferta

Al pulsar en el botón de Crear oferta, se lanza la activity ClienteAnadirOferta1 o TiendaAnadirOferta1, según el tipo de usuario. En el primer caso, la pantalla mostrara un buscador de tiendas y un mapa, y pedirá al usuario que busque la tienda donde ha visto la oferta, o que cree una nueva si no existe. Al seleccionar una tienda, se lanzará activity Cliente SeleccionarProducto donde elegirá el producto que está de oferta. En el segundo, mostrará una pantalla similar a la de la activity Catálogo, donde se puede elegir en una lista desplegable la tienda, y debajo el catálogo de dicha tienda.

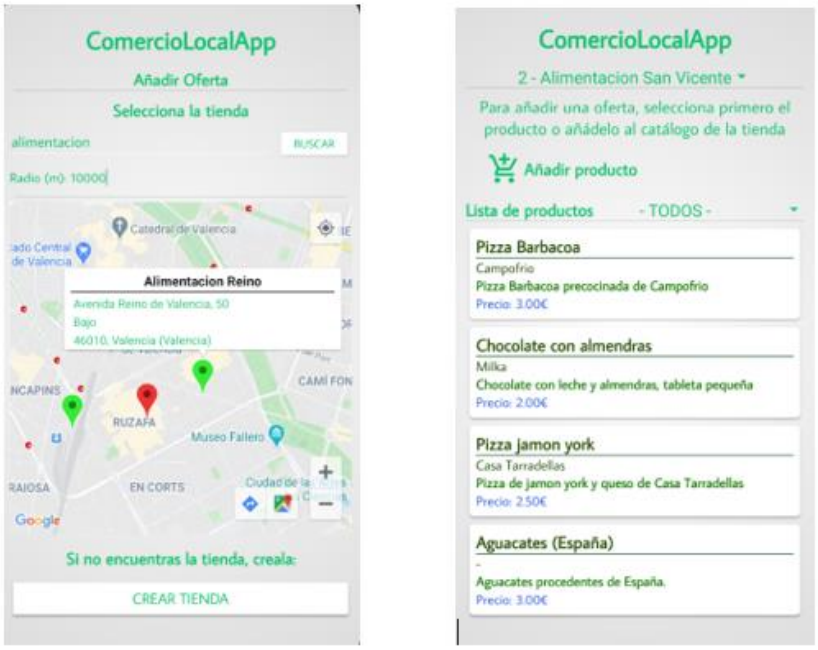

Figura 27. Pantallas de Añadir Oferta para cliente y tienda

Cuando se seleccione un producto, independientemente del tipo de usuario, se lanzará la activity AnadirOferta2, donde se muestra la plantilla con todos los datos que el usuario tendrá que rellenar:

Figura 28. Pantalla de AnadirOferta2

6.3.5.5 Gestionar tiendas

Esta funcionalidad solo está disponible para usuarios de tipo tienda, y se implementa en la activity MisTiendas. En la pantalla se muestra un botón para crear una nueva tienda y una lista de tarjetas con las tiendas del usuario. Al pulsar en las tarjetas se lanza la activity VerTienda, donde se muestran los detalles de la tienda y se da opción a modificarla o eliminarla.

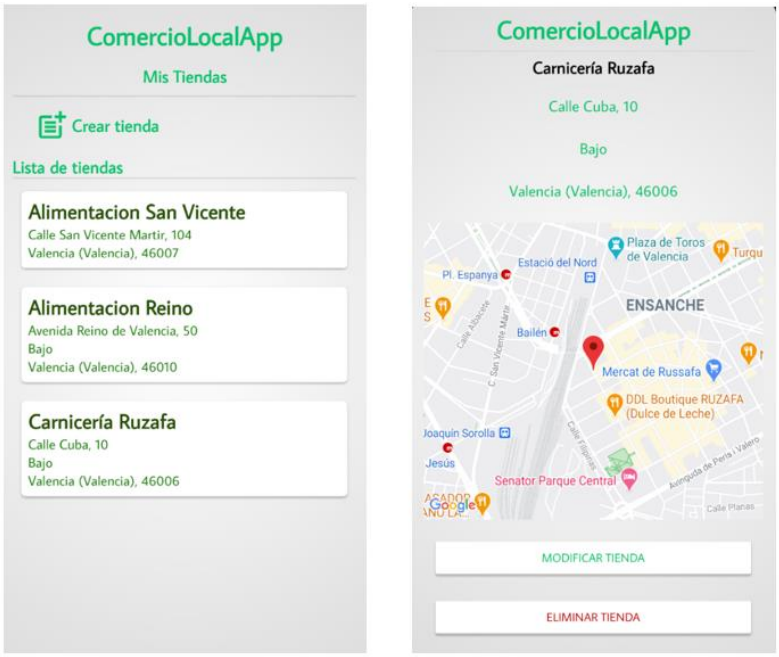

Figura 29. Pantallas de MisTiendas y VerTienda

Si se pulsa en el botón de crear tienda, se lanza la activity CrearTienda, donde se mostrará un formulario para rellenar con la dirección de la tienda y debajo un mapa para seleccionar la ubicación, realizando una pulsación larga con el dedo. Después se pasa a la activity CrearTiendaDatos, donde se pide introducir los datos de la tienda.

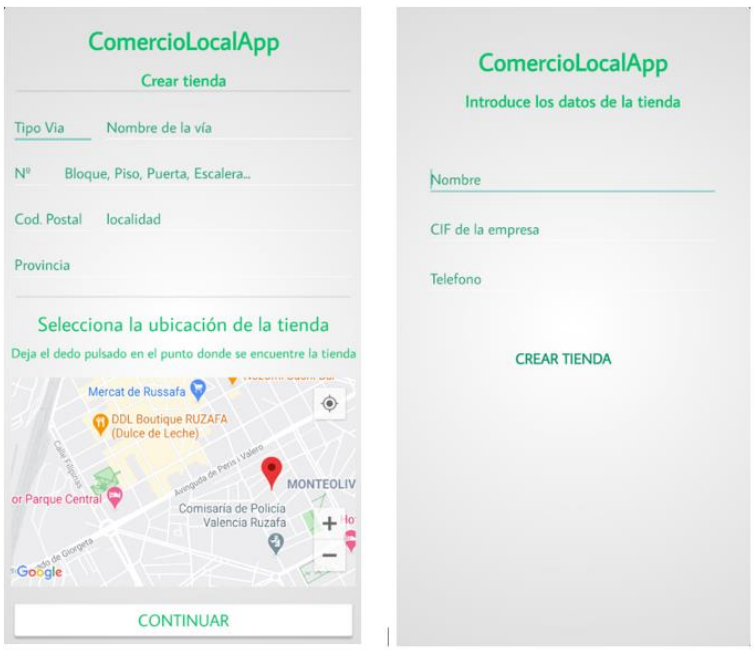

Figura 30. Pantallas de Crear Tienda

6.3.5.6 Gestionar catálogos

La funcionalidad de gestionar catálogos solo está disponible para usuarios de tipo tienda. Se implementa en la activity Catalogo. En esta pantalla se muestra, en primer lugar, una lista desplegale para seleccionar la tienda cuyo catálogo se quiere gestionar. Al seleccionar un elemento de la lista, se mostrarán, en forma de tarjeta, los productos que se hayan añadido en el catálogo. Si se pulsa en una de las tarjetas se lanzará la activity VerProducto, donde también se da opción a modificar el precio o eliminarlo del catálogo.

Figura 31. Pantallas de Catálogo y VerProducto

Si se pulsa en añadir producto, se lanzará la activity AnadirProducto, donde se mostrará un buscador de productos. Al introducir una cadena de texto y pulsar el botón de buscar, se mostrará la lista de tarjeta con los productos encontrados. Al pulsar en uno de ellos, se lanzará la activity AnadirProducto_Precio, donde se verán los datos del producto y se pedirá introducir el precio de venta habitual.

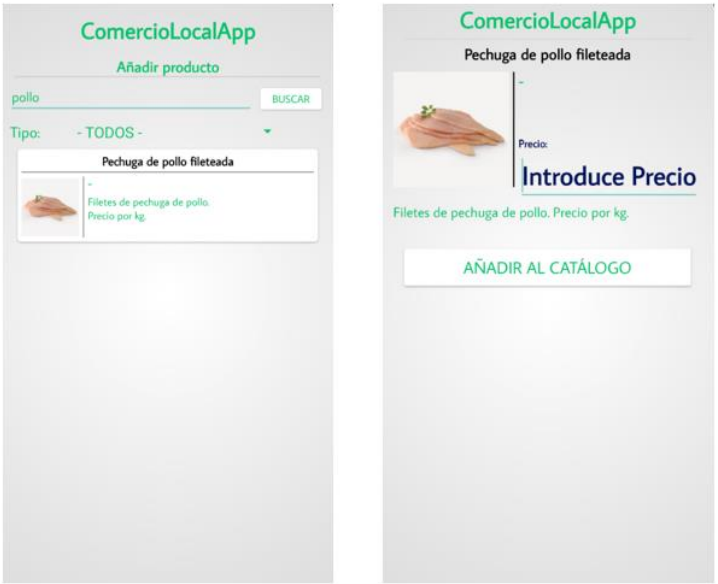

Figura 32. Pantallas de AnadirProducto y AnadirProducto_Precio

Si se pulsa en crear producto, se lanzará la activity CrearProducto, donde se pedirá al usuario que introduzca los datos del producto y una URL que apunte a una imagen del producto. Tras crearlo se lanzará la activity AnadirProducto_Precio, ya descrita.

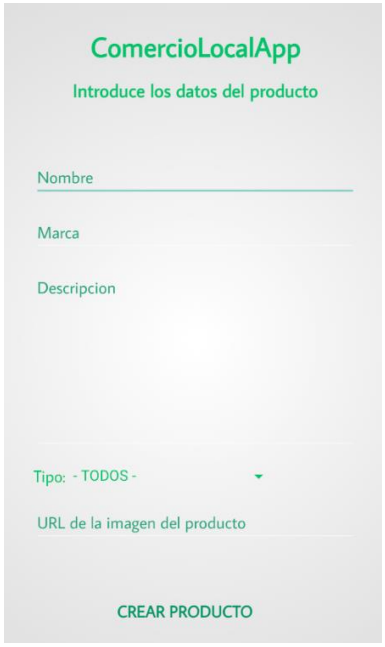

Figura 33. Pantalla de CrearProducto

6.3.5.7 VerPerfil

Por último, la funcionalidad de ver perfil es común a ambos usuarios. Se implementa en la activity MiPerfil, y solamente muestra los datos de registro y personales del usuario. En la parte inferior se da opción a modificar los datos o eliminar la cuenta:

Figura 34. Pantalla de VerPerfil

Capítulo 7. Conclusiones y propuesta de trabajo futuro

7.1 Conclusiones

Al principio de este Trabajo de Fin de Grado se marcó un objetivo principal, que era crear una aplicación para Android con la que los comercios de proximidad pudieran llegar a sus potenciales clientes de una forma más fácil y directa. Ese objetivo se dividió en tres objetivos más pequeños, que eran desarrollar cada una de las partes de la aplicación de acuerdo con la estructura cliente-servidor de tres etapas. Después se añadió otro objetivo, que era aprender a manejar nuevas herramientas y nuevos lenguajes de programación de cara a introducirme en el mercado laboral.

Con la elaboración de este Trabajo de Fin de Grado, he podido aplicar mis conocimientos sobre SQL y Java más allá de las prácticas en empresa, he ampliado mis conocimientos sobre programación para Android y he adquirido nociones sobre otras tecnologías muy demandadas actualmente en el mercado laboral como son Node.js y JavaScript.

Además, he aprendido cómo funcionan la mayoría de las aplicaciones móviles y web que utilizamos hoy en día, que hacen uso de una API REST.

Durante la realización de este Trabajo de Fin de Grado, he tenido una etapa de aprendizaje en la que he podido constatar que hay una gran cantidad de información en la red sobre programación, tutoriales, ejemplos y gente dispuesta a resolver las dudas que les surgen a desconocidos de manera totalmente desinteresada, lo cual facilita en gran medida las tareas de desarrollo de aplicaciones. También debo destacar que esto no siempre ha sido algo positivo, ya que me he topado con una gran cantidad de cursillos y tutoriales desactualizados, otros que no sirven para aprender porque solo muestran código sin explicar absolutamente nada y, sobre todo, cursillos que se quedan a medias y te obligan a buscar alternativas e incluso a empezar de cero.

Por lo tanto, y pese a las dificultades encontradas, considero que todos los objetivos que marqué para la elaboración del Trabajo de Fin de Grado se han cumplido, ya que se ha conseguido crear la aplicación utilizando mis conocimientos previos y los adquiridos durante la realización del trabajo.

7.2 Propuesta de trabajo futuro

La aplicación ha sido creada como un prototipo, pero considero que es una buena idea que podría ayudar a mucha gente y revitalizar las ciudades, por lo que de cara al futuro se podría considerar la opción de sacarla al mercado, pero para ello habría que resolver una serie de carencias que he observado durante la elaboración de este trabajo.

Lo primero y más básico sería crear nuevas capas de seguridad, implementando el uso de HTTPS, por ejemplo, y mejorar la interfaz gráfica para hacerla más atractiva al usuario.

En cuanto a la funcionalidad de la aplicación, la principal carencia que he observado es la falta de control sobre las ofertas y tiendas que crean los usuarios, ya que cualquier usuario podría añadir tiendas u ofertas falsas. Para solventar este problema se podría crear un sistema de verificación real, en el que los usuarios de tipo tienda acreditaran que son los dueños de las tiendas antes de poder utilizar la aplicación, y con el que se comprobara que las ofertas y tiendas que añaden los usuarios de tipo cliente fueran reales. El sistema de verificación se podría complementar con un sistema de moderación mediante reportes, con el que los usuarios pudieran advertir de ofertas falsas o incorrectas.

También se tendría que crear una funcionalidad para que los propietarios pudieran reclamar tiendas, pues se puede dar el caso de que un propietario quiera registrar una tienda que

ya ha sido registrada previamente por un cliente. En ese caso, el usuario tienda podría reclamarla y agregarla a su cuenta.

Otras funcionalidades que añadirían valor a la aplicación serían las siguientes:

- Búsqueda y registro de productos mediante un escáner de códigos de barra.
- Creación de listas de la compra, en las que los usuarios pudieran añadir los productos que vean de oferta, y que se podría complementar con la creación de rutas en el mapa.
- Añadir comentarios de los usuarios a las ofertas y valoración de tiendas.
- Crear gráficas con historiales de precios, ofrecer estadísticas a las tiendas, etc.
- Crear un sistema para que los usuarios puedan subir imágenes al servidor.

Por último, más allá del mundo de la programación, para lanzar la aplicación al mercado sería necesario un equipo de asesoramiento legal y jurídico que ayudara, entre otras cosas, con la gestión y tratamiento de la Ley de Protección de Datos, ya que esta aplicación pide al usuario que introduzca datos muy sensibles.

Capítulo 8. Bibliografía

[1] DiTrendia, “Todas las estadísticas sobre móviles que deberías conocer – MWC19”

https://mktefa.ditrendia.es/blog/todas-las-estad%C3%ADsticas-sobre-m%C3%B3viles-que-deber%C3%ADas-conocer-mwc19

[2] Observatorio Nacional de las telecomunicaciones y de la S.I. “Informe anual 2018 – La sociedad en red. Transformación digital en España (Edición 2019)”

https://www.ontsi.red.es/sites/ontsi/files/2019-10/InformeAnualLaSociedadEnRedEdic2019.pdf

[3] UPTA “Desplome del comercio, 1566 establecimientos han cerrado en septiembre” https://upta.es/desplome-del-comercio-1-566-establecimientos-han-cerrado-en-septiembre/

[4] Instituto Nacional de Estadística. “Indice de Comercio al por Menor (ICM) https://www.ine.es/prensa/icm_tabla1.htm

[5] Heraldo de Aragón. “El comercio online pierde fuelle con la reapertura de las tiendas” https://www.heraldo.es/noticias/aragon/2020/06/30/el-comercio-online-pierde-fuelle-con-la-reapertura-de-las-tiendas-1382991.html

[6] StackOverflow. “Developer Survey Results 2019” https://insights.stackoverflow.com/survey/2019

[7] Wikipedia. “Android” - https://es.wikipedia.org/wiki/Android

[8] Statista. “Android e iOS dominan el mercado de los smartphones” https://es.statista.com/grafico/18920/cuota-de-mercado-mundial-de-smartphones-por-sistema-operativo/

[9] Xataka “Ni Java ni C++, Kotlin pasa a ser el lenguaje preferido por Google para desarrollar en Android”

https://www.xatakandroid.com/programacion-android/no-hara-falta-aprender-java-para-programar-android-kotlin-pasa-a-ser-preferido-google

[10] Serquo. “Explicamos lo que es Node.js para técnicos y principantes” https://serquo.com/blog/node-js/

[11] Wikipedia. “JavaScript” - https://es.wikipedia.org/wiki/JavaScript

[12] OpenWebinars. “Qué es MySQL” - https://openwebinars.net/blog/que-es-mysql/

[13] En Mi Local Funciona. “Construyendo una web API REST segura con JSON Web Token” https://enmilocalfunciona.io/construyendo-una-web-api-rest-segura-con-json-web-token-en-net-parte-i/

[14] Web oficial de Node.js – https://nodejs.org/es/

[15] Wikipedia. “Npm” - https://es.wikipedia.org/wiki/Npm

[16] Medium.com. “Introducción a Express JS” https://medium.com/@aarnlpezsosa/introducci%C3%B3n-a-express-js-a1ebe16dbcf4

[17] Web oficial de Express JS. “Routing” - https://expressjs.com/es/guide/routing.html

[18] CornerShop. “Explorando ConstraintLayout” - https://tech.cornershop.io/android-explorando-constraintlayout-1-4f39beef9f82