ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

GESTIÓN DE RESERVAS Y PEDIDOS EN RESTAURANTES MEDIANTE DISPOSITIVOS MÓVILES

Autor: Álvaro Clemente Verdú

Director: Jose Miguel Ordax Cassá

Madrid

Julio 2017

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

GESTIÓN DE RESERVAS Y PEDIDOS EN RESTAURANTES MEDIANTE DISPOSITIVOS MÓVILES en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2016/17 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Álvaro Clemente Verdú Fecha: 7/ 7/ 2017

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Jose Miguel Ordax Cassá Fecha: 7/ 7/ 2017

Vº Bº del Coordinador de Proyectos

Fdo.: David Contreras Bárcena Fecha: 7/ 7/ 2017

Agradecimientos

A mis padres, porque sin su apoyo durante estos años nada de esto habría sido posible.

GESTIÓN DE RESERVAS Y PEDIDOS EN RESTAURANTES MEDIANTE DISPOSITIVOS MÓVILES

Autor: Clemente Verdú, Álvaro
Director: Ordax Cassá, Jose Miguel

RESUMEN DEL PROYECTO

Con el objetivo de aprovechar un nicho de mercado en el que las aplicaciones móviles están ganando popularidad debido a su efectividad, se va a crear una plataforma móvil online que permita a los restaurantes ofrecer un servicio más rápido, cómodo y eficiente, que disminuya costes y mejore la fidelidad de los clientes. Esto se consigue gestionando las reservas, los pedidos y los pagos a través de una aplicación móvil, eliminando las esperas y facilitando la gestión.

Palabras clave: *Aplicación móvil, plataforma, restaurante, cliente.*

1. Introducción

Los mercados de smartphones y de aplicaciones móviles han experimentado los últimos años un crecimiento sin precedentes. En 2015, el número de dispositivos móviles superó los 7,9 mil millones, más que personas hay en nuestro planeta [1]. Las descargas anuales de aplicaciones alcanzaron los 90 mil millones en 2016, y durante ese mismo año se emplearon cerca de 900 mil millones de horas utilizando este tipo aplicaciones [2]. En los últimos años se pensaba que las redes sociales, o los transportes serías las más beneficiadas por estas tecnologías. No obstante, recientemente se están encontrando nuevos nichos en los que su uso está ganando popularidad.

Uno de ellos es la restauración. En el contexto actual, el cliente exige al restaurante rapidez y comodidad, a un precio cada vez más bajo. Como primera solución, han surgido numerosas aplicaciones que facilitan el servicio a domicilio del restaurante (Just Eat, La Nevera Roja, etc.), u otras que incluso les permiten externalizar este servicio por completo (Deliveroo, Glovo).

Sin embargo, estos servicios sólo incrementan los costes operativos, y en una coyuntura en la que los márgenes de beneficio son cada vez más bajos, los restaurantes necesitan encontrar otra solución que les permita acceder a nuevos clientes, a la vez que conseguir reducir costes para ofrecer un servicio más competitivo.

2. Definición del proyecto

Como respuesta a estas necesidades, se desea diseñar una plataforma que permita a los usuarios disfrutar en un restaurante de un servicio rápido y de calidad. A su vez, esta plataforma servirá a las empresas como punto para publicitarse accediendo a nuevos clientes, distribuir ofertas, fidelización y como una herramienta que facilite la autogestión y que les permita ofrecer su servicio de forma más eficiente, reduciendo costes operativos.

Con este proyecto se busca diseñar una prueba de este concepto. Para ello se va a implementar una aplicación móvil Android que con la que el cliente pueda reservar y

sentarse en sus mesas, ver el menú y hacer pedidos, pedir la cuenta y pagar, todo desde la aplicación y de forma online, lo que permite hacer reservar y hacer pedidos desde cualquier lugar. Esto permitirá al restaurante incrementar el número de mesas a las que puede atender cada camarero, a la vez que permite que una mesa pueda ser utilizada por más clientes en un mismo turno (reduciendo el tiempo medio que está un cliente en el restaurante).

3. Descripción del sistema

La plataforma móvil se compone de dos elementos principales: una aplicación de servidor, y una aplicación móvil cliente, que permite al usuario interactuar con la plataforma. Estos dos componentes se van a comunicar a través de Internet, mediante el protocolo HTTP. La arquitectura del sistema es la siguiente:

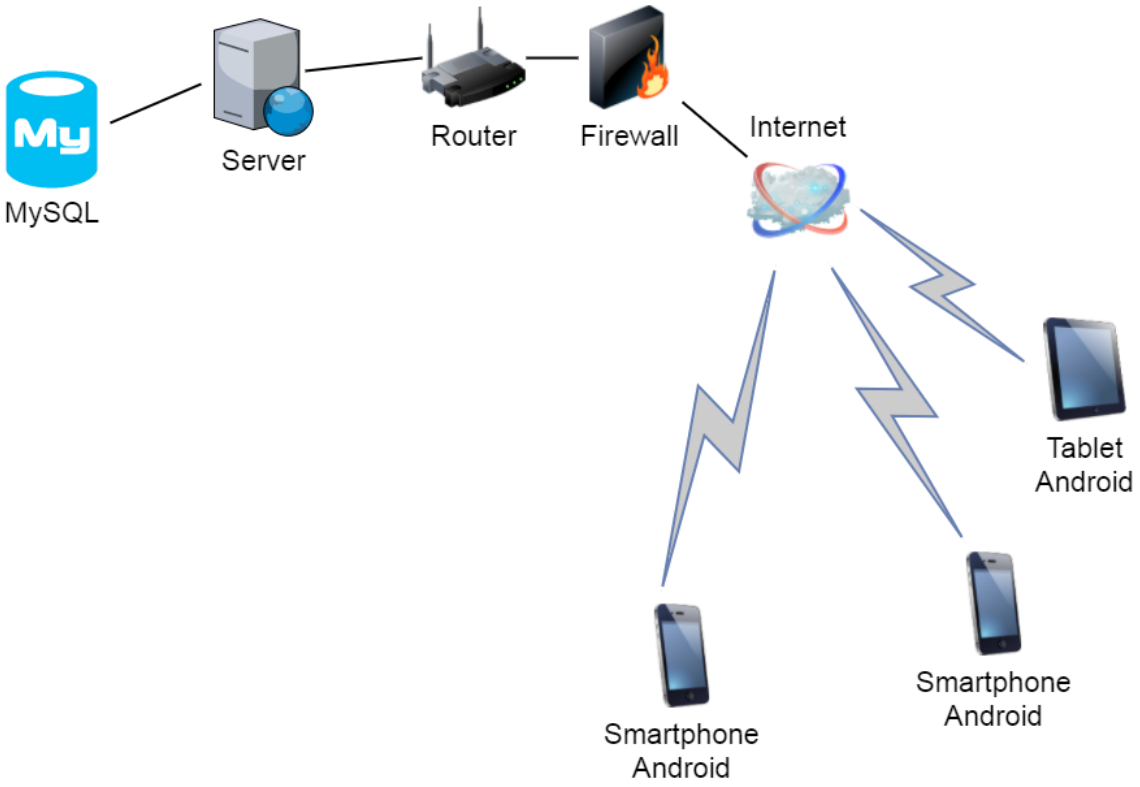

Ilustración 1. Diagrama de arquitectura del sistema

En servidor es el encargado de acceder a la base de datos, y contiene la mayor parte de la lógica de negocio del sistema, que estará implementada en el lenguaje Java. El servidor tiene múltiples funciones:

- Gestiona los usuarios de la plataforma: creación y autenticación de los usuarios, así como inicio y cierre de sesión en la plataforma.
- Gestión de las reservas: creación, gestión y eliminación de las reservas de cada restaurante.
- Gestión de los pedidos: creación, y eliminación de los pedidos que se generan en cada reserva
- Gestión del pago: mantenimiento de la cuenta del usuario, y permite el pago a través de servicios online.

- Avisos e incidencias: comunica a los restaurantes los avisos e incidencias en una reserva.

Este servidor debe tener conexión a Internet, por lo que se va a conectar a un router central, protegido detrás de un firewall, conectado con a la red.

Por otro lado, está la aplicación móvil Android. Esta aplicación se instala en el dispositivo móvil del cliente, y ofrece una interfaz gráfica con la que el usuario interactúa, y que genera peticiones HTTP para el servidor.

4. Resultados

Los resultados obtenidos tras el diseño e implementación del prototipo cumplen con los objetivos marcados inicialmente. A través de la aplicación móvil el usuario es capaz de ver y elegir el restaurante que desee, hacer reservas, hacer pedidos y pagar. Un ejemplo del uso se muestra en la siguiente ilustración:

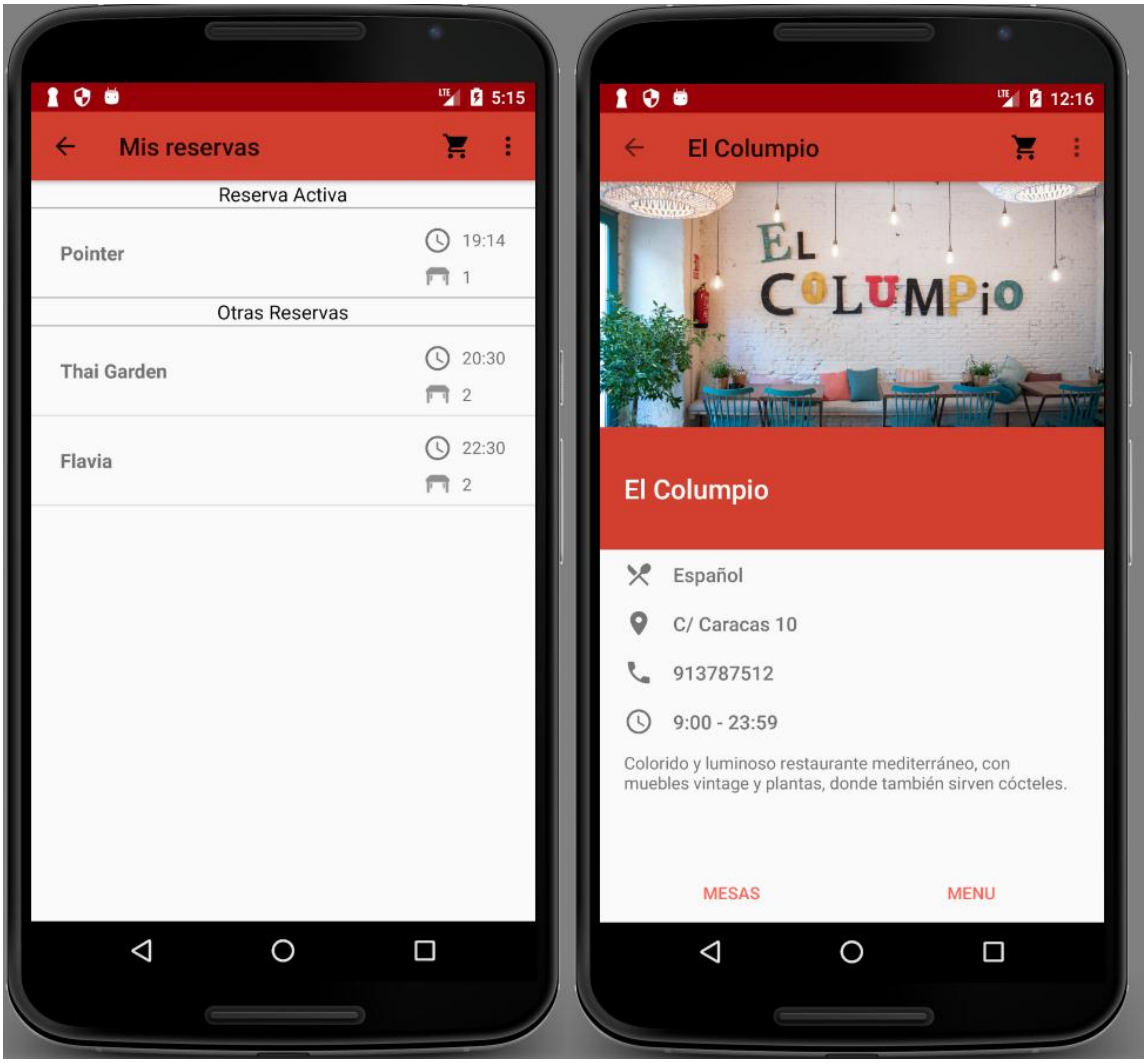

Ilustración 2. Pantallas de ejemplo de la aplicación

Por otra parte, el aspecto online de la plataforma tiene un funcionamiento correcto y rápido, siendo todos los tiempos de carga inferiores a 3 segundos. Esto se consigue gracias al reducido tiempo de procesamiento de las peticiones, y al tamaño de las respuestas. En el lado

del cliente, la información que pueda ser reutilizada (como listados de restaurantes, menú o imágenes) es cacheada para evitar peticiones redundantes.

Sin embargo, conforme el número de clientes crezca, para mantener este comportamiento serán necesarias más optimizaciones, y migrar los servidores a servicios de hosting en la nube, como Bluemix o Amazon Web Services (AWS).

5. Conclusiones

En el mercado de las aplicaciones móviles para la restauración existe aún un nicho que puede ser explotado. Los restaurantes tienen unas necesidades que pueden ser cubiertas por el servicio adecuado, y las aplicaciones móviles son unas plataformas ideales para ello. Por otra parte, los usuarios buscan comodidad y rapidez, y las funcionalidades online móviles cubren estas dos necesidades.

La solución propuesta responde a estas necesidades y ofrece al restaurante medio una oportunidad de competir con grandes cadenas y franquicias, a un nivel que no ha sido posible hasta entonces.

Por otra parte, esta aplicación hace que el servicio que recibe el cliente sea más rápido y cómodo. Esto puede fomentar el consumo, incrementando el consumo anual por cliente.

Por último, cabe destacar que los requisitos que se han planteado para esta plataforma pueden ser extrapolables para otras aplicaciones. Las necesidades que se han querido cubrir no son exclusivas de los restaurantes, y otros sectores de la hostelería (bares u hoteles), así como otros comercios pueden beneficiarse con un sistema de estas características.

6. Referencias

[1] Ditrendia: "Informe ditrendia 2016: Mobile en España y en el Mundo". Julio 2016. http://www.amic.media/media/files/file_352_1050.pdf

[2] Thompson, E: "App Annie 2016 Retrospective – Mobile's Continued Momentum". Enero 2017. https://www.appannie.com/en/insights/market-data/app-annie-2016-retrospective/

RESERVATIONS AND ORDER MANAGEMENT THROUGH MOBILE DEVICES

Autor: Clemente Verdú, Álvaro
Director: Ordax Cassá, Jose Miguel

ABSTRACT

Aiming to capitalize on a market niche in which mobile applications are gaining popularity, due to their effectiveness, a mobile platform will be created that allows restaurants to provide a faster, more efficient and convenient service, decreasing costs and improving customer loyalty. This will be done by managing reservations, orders, and payments through a mobile application, eliminating delays and helping management.

Palabras clave: *Mobile application, platform, restaurant, customer.*

1. Introduction

The smartphone and mobile apps market have experienced an unprecedented growth. In 2015, the total amount of smartphones worldwide exceeded 7.9 billion, more than the number of people in the planet [1]. Yearly app downloads reached 90 billion in 2016, and, during that same year, more than 900 billion hours were spent using this kind of apps [2]. During the last few years only industries like social media or transportation were perceived as the best fit for this technology. However, recently new niche opportunities are arising in which mobile applications are gaining popularity.

One of these niche markets is catering. In today's situation, the client demands fast service and convenience, at an ever-decreasing price. As a first solution, new applications were born as a facilitator for take-away and home delivery service (such as Just Eat, La Nevera Roja, etc.), or even others than allows to outsource them completely (Deliveroo, Glovo).

Nevertheless, these services only increase restaurant's operational costs, and in a context where profit margins are lower than ever, restaurants need to find a different solution, one that allows them to reach new customers while reducing operational costs to offer a more competitive service.

2. Project definition

As an answer to these needs, a platform will be built that allows the users to enjoy a fast and high-quality service. At the same time, this platform will serve the restaurants as an advertising space to reach new customer, distribute offers and sales, customer loyalty and as a managing tool, making the service faster and more efficient, reducing operational costs.

The goal for this project is to build a proof of concept. To do so, an Android mobile app will be implemented, through which the user can make a reservation and sit on tables, see the restaurant's menu and make and order, ask for the check and play, all of this online, which allows the user to do so anywhere, anytime. This way, restaurants will be able to increase

the number of tables each waiter can service. The number of clients per shift will increase as well, since the average time each customer spends will be reduced.

3. System description

The mobile platform will consist of two main components: a web server application and a client mobile app, which will serve as an interface between the user and the platform. These two components will communicate with each other through the Internet, using the HTTP protocol. The system's architecture is the following:

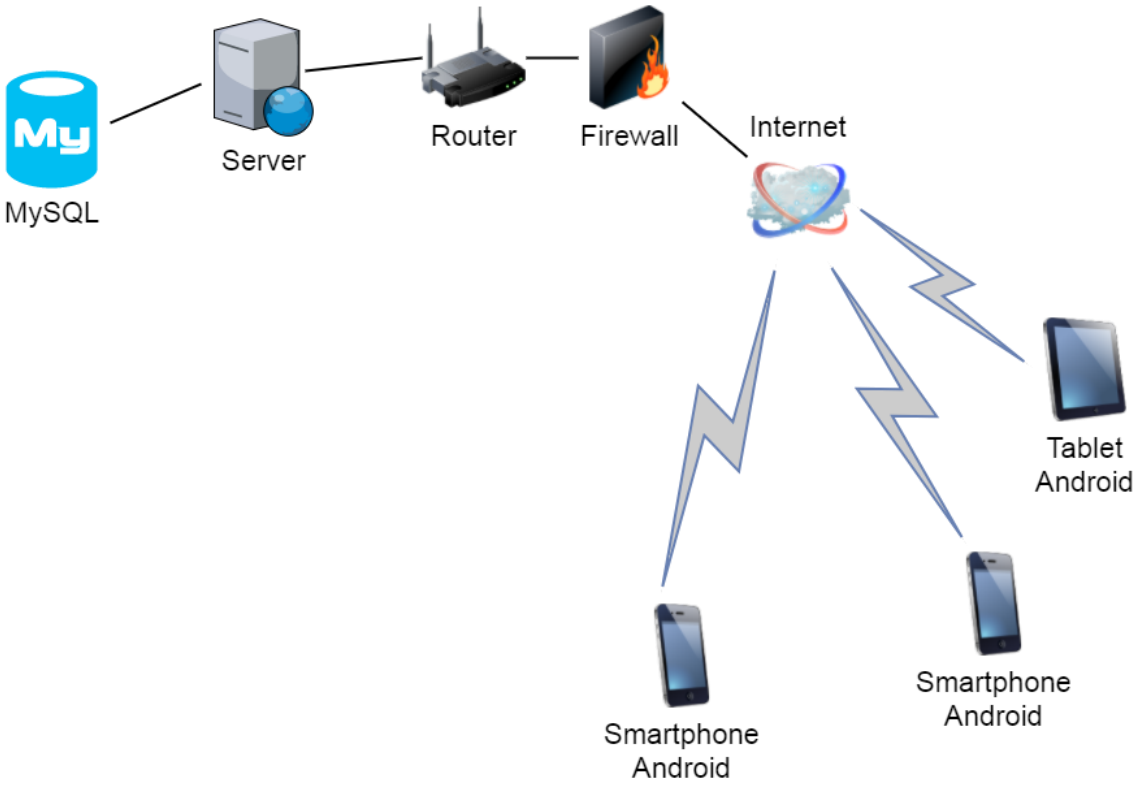

Figure 1. System's architecture diagram

The server will oversee accessing the database, and contains most the business logic, which will be implemented using Java. The server has multiple tasks:

- Managing platform's users: creating and authenticating the users, as well as login and logout functionalities.
- Managing reservations: creating, managing, and eliminating each restaurant's reservations.
- Managing orders: creating and eliminating orders generated within each reservation.
- Managing payment orders: keeping track of each user's check, allowing online payment.
- Requests and warnings: noticing restaurants with each reservation's request's and warnings.

The server must be connected to the Internet, which is why it will be connected to a main router, protected with a firewall, and connected to the Internet.

On the other side is the Android mobile app. This app will be installed in each customer's mobile devices and provides a graphic interface to interact with the platform, and generates HTTP requests directed to the server.

4. Results

The obtained results meet the goals set during the design and implementation phase. Through the mobile application, the user can list the restaurants and select one, make a reservation, make orders, and pay. An illustration of the app's use is shown in the next figure:

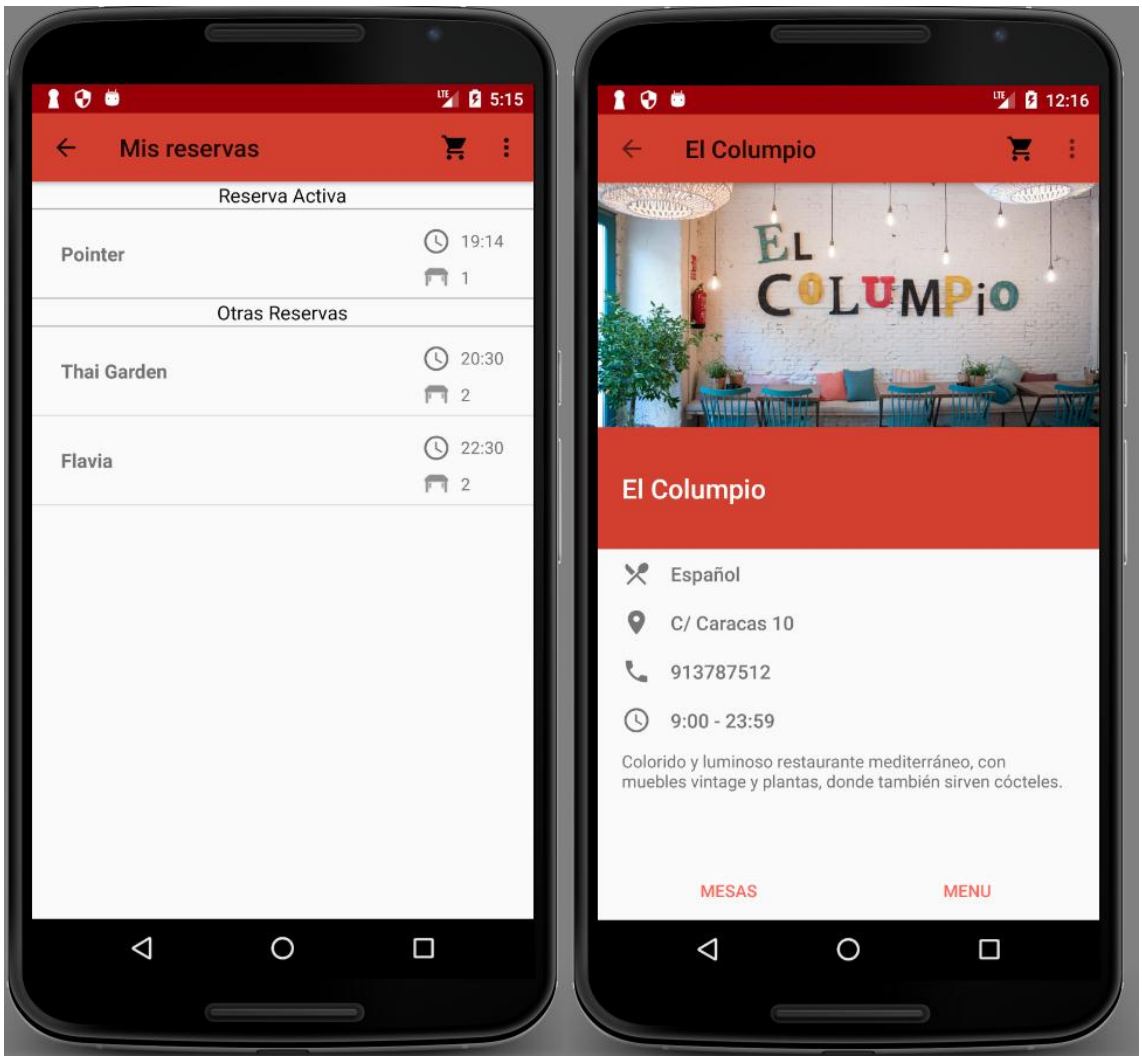

Figure 2. App screen examples

Furthermore, the online aspect works in a proper and fast manner, with lower than 3 second load times. This is due to low processing times and request sizes. On the client's side, information that may be reused (such as the restaurant list, menus, and images) is cached to avoid unnecessary requests.

However, this performance level may not be stable once the number of users grows, in which case more optimizations, and a server migration to cloud hosting services such as Bluemix or Amazon Web Services (AWS) will be necessary.

5. Conclusions

There is a niche in the restaurant-focused mobile app market that can still be exploited. Restaurants have a set of needs that can be met by the right service, and mobile apps are the ideal platform to do so. Moreover, customers demand a fast and convenient service, which can be provided using this technology.

The proposed solution meets these needs, and provides the average restaurant with a chance to compete with big restaurant chains and franchises, at a level that has not be possible to date.

Additionally, this platform makes the service provided to the client faster and more convenient, available anywhere. This can encourage consumption, increasing the average customer yearly spending in restaurants.

At last, it is worth stressing that the requirements set for this platform can be extrapolated to other applications. The needs wanted to be met are not exclusive to restaurants, and other hostelry (hotels, bars) as well as other kinds of commerce can take advantage of a platform with similar characteristics.

6. References

[3] Ditrendia: "Informe ditrendia 2016: Mobile en España y en el Mundo". Julio 2016. http://www.amic.media/media/files/file_352_1050.pdf

[4] Thompson, E: "App Annie 2016 Retrospective – Mobile's Continued Momentum". Enero 2017. https://www.appannie.com/en/insights/market-data/app-annie-2016-retrospective/

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

Los mercados de smartphones y de aplicaciones móviles han experimentado los últimos años un crecimiento sin precedentes. En 2015, el número de dispositivos móviles superó los 7,9 mil millones, más que personas hay en nuestro planeta [1]. Las descargas anuales de aplicaciones alcanzaron los 90 mil millones en 2016, y durante ese mismo año se emplearon cerca de 900 mil millones de horas utilizando este tipo aplicaciones [2]. En los últimos años se pensaba que las redes sociales, o los transportes serías las más beneficiadas por estas tecnologías. No obstante, recientemente se están encontrando nuevos nichos en los que su uso está ganando popularidad.

Uno de ellos es la restauración. En el contexto actual, el cliente exige al restaurante rapidez y comodidad, a un precio cada vez más bajo. Como primera solución, han surgido numerosas aplicaciones que facilitan el servicio a domicilio del restaurante (Just Eat, La Nevera Roja, etc.), u otras que incluso les permiten externalizar este servicio por completo (Deliveroo, Glovo).

Sin embargo, estos servicios sólo incrementan los costes operativos, y en una coyuntura en la que los márgenes de beneficio son cada vez más bajos, los restaurantes necesitan encontrar otra solución que les permita acceder a nuevos clientes, a la vez que conseguir reducir costes para ofrecer un servicio más competitivo.

Con este proyecto se pretende diseñar una plataforma que, aprovechando las características de las aplicaciones móviles, respondan las necesidades de los restaurantes, permitiéndoles atraer más clientes gracias a la base de usuarios potenciales de la plataforma, al mismo tiempo que se incrementa la eficiencia de su servicio, reduciendo los tiempos de espera.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

En este capítulo se van a exponer las tecnologías asociadas con este proyecto. En primer lugar, se van a mostrar las distintas opciones disponibles a la hora de elegir el tipo de aplicación móvil a desarrollar, después se explicarán los sistemas operativos objetivo y la influencia que tiene la elección de uno u otro, y por último se van a detallar el resto de tecnologías y aplicaciones utilizadas para el desarrollo de la plataforma.

2.1 ANDROID

Android es una plataforma móvil con un sistema operativo basado en el núcleo Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como *smartphones*, *tablets*, y recientemente utilizado también para televisores, relojes inteligentes e incluso automóviles.

La plataforma Android está compuesta por una base que es abierta y libre, conocida como Android Open Source Project (AOSP), que es *Open Source*, por lo que cualquier desarrollador puede utilizar el código y modificarlo como prefiera y publicar una distribución basada en este sistema. Sobre ello, Android incluye también otras aplicaciones y servicios, como Google Maps, Google Play o los Play Services (que incluyen las APIs y servicios del sistema), que son propietarios.

Las aplicaciones Android se distribuyen a través del canal oficial de Google: Google Play. Desde ahí se pueden instalar aplicaciones tanto gratuitas como de pago, y gestionar las actualizaciones. Además, el sistema permite (aunque no recomienda) descargar directamente las aplicaciones, en formato .apk (*Android Application Package*) desde internet o a través de memorias externas, para instalarlas manualmente.

2.1.1 ARQUITECTURA

El sistema operativo de Android está basado en un núcleo Linux, y depende de éste para los servicios base del sistema como la seguridad, gestión de procesos, pila de rede y modelo de controladores.

Sobre este núcleo, se encuentran un conjunto de librerías de C y C++ que son utilizadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de las aplicaciones (*Application Framwork)*. Algunas de estas librerías son: la implementación estándar de C (libc), biblioteca de medios, de gráficos, 3D y SQLite para bases de datos, entre otras.

Al mismo nivel, se encuentra la Runtime de Android, que es un conjunto de bibliotecas que proporcionan la mayoría de servicios base del lenguaje Java. Cada aplicación de Android corre su propio proceso, con una instancia propia de la máquina virtual Dalvik, que se ha diseñado para que el sistema pueda manejar esta situación de forma eficiente. Hasta la versión 5.0 (Lollipop) el sistema Dalvik ejecutaba archivos del formato “Dalvik Excutable” (.dex), que se compilaban en el momento de ejecución (“Just In Time” o JIT).

Sin embargo, en las versiones posteriores, la ART sustituyo a Dalvik. La ART introdujo el uso de “Ahead-of-Time” (AOT) que crea un archivo compilado en el momento de la instalación de la aplicación. Esto mejoró el rendimiento del dispositivo, al reducir considerablemente el número de compilaciones que hace el dispositivo, y permitió optimizar el consumo de batería. La ART utiliza el mismo código de bytes de entrada que Dalvik, manteniendo compatibilidad con las versiones anteriores.

En la siguiente figura (Figura 1) podemos ver una comparativa de la arquitectura de Dalvik y ART.

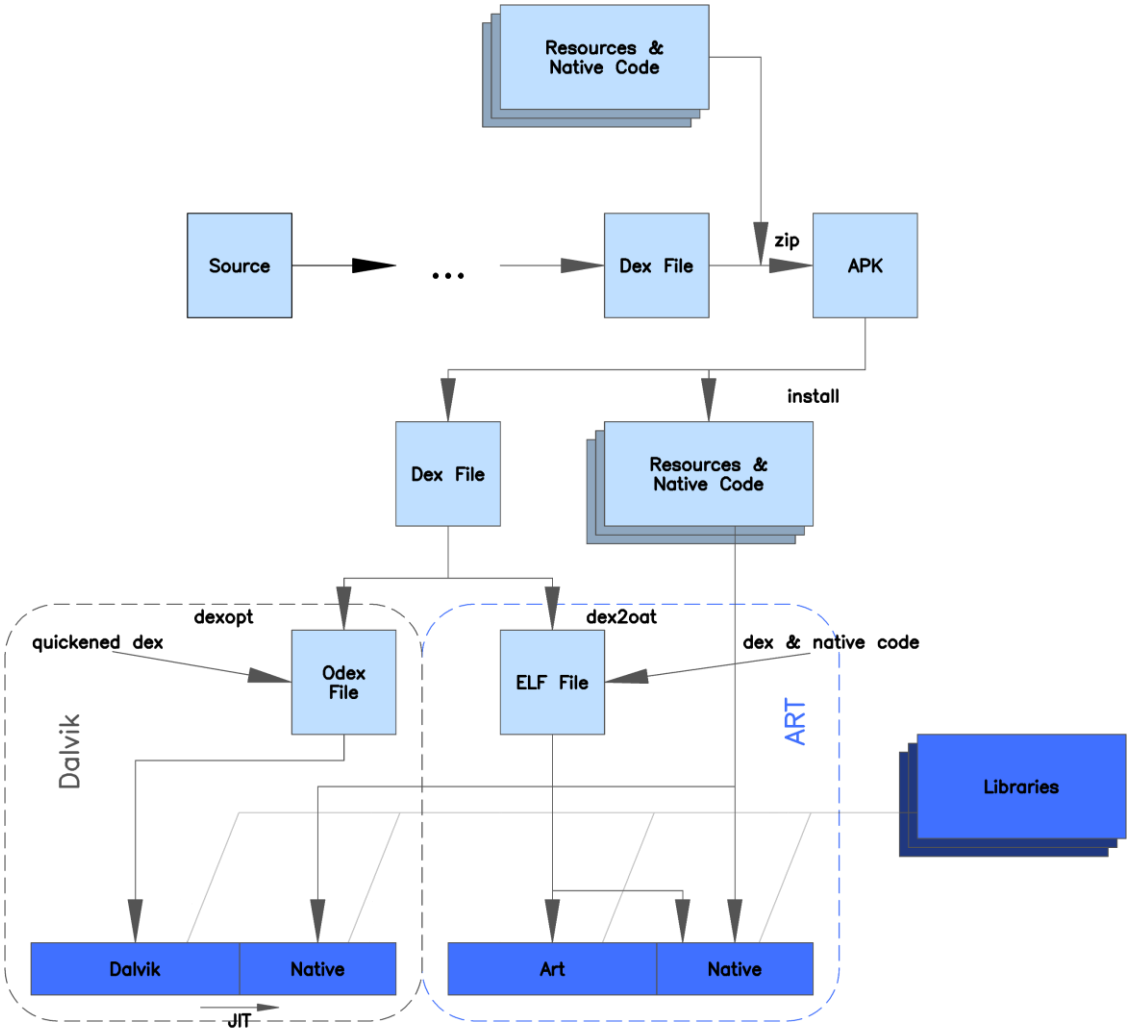

Figura 1. Comparativa de arquitectura de Dalvik y ART. Fuente: Wikipedia [3]

Sobre las bibliotecas y la ART se encuentra el marco de trabajo de aplicaciones (*Application Framework)*, desde el cual los desarrolladores acceden a las API utilizadas por las aplicaciones base. La arquitectura Android está diseñada para la reutilización de componentes. De esta forma, cualquier aplicación puede publicar sus capacidades para que más tarde otra aplicación pueda hacer uso de ellas, respetando unas normas de seguridad de este *framework*. Este mecanismo permite que los componentes puedan ser reemplazados por el usuario.

La capa superior es en la que se encuentran todas las aplicaciones. Aquí el sistema Android ofrece un número de aplicaciones base que ofrecer servicios específicos, como son Google Maps, la Galería, la Cámara, el Teléfono, un cliente de correo electrónico, y muchas más. El usuario puede instalar nuevas aplicaciones que añadan funcionalidad o sustituyan a las antes mencionadas.

Los elementos de la arquitectura antes mencionados, así como muchos otros, se pueden ver gráficamente en la siguiente figura (Figura 2):

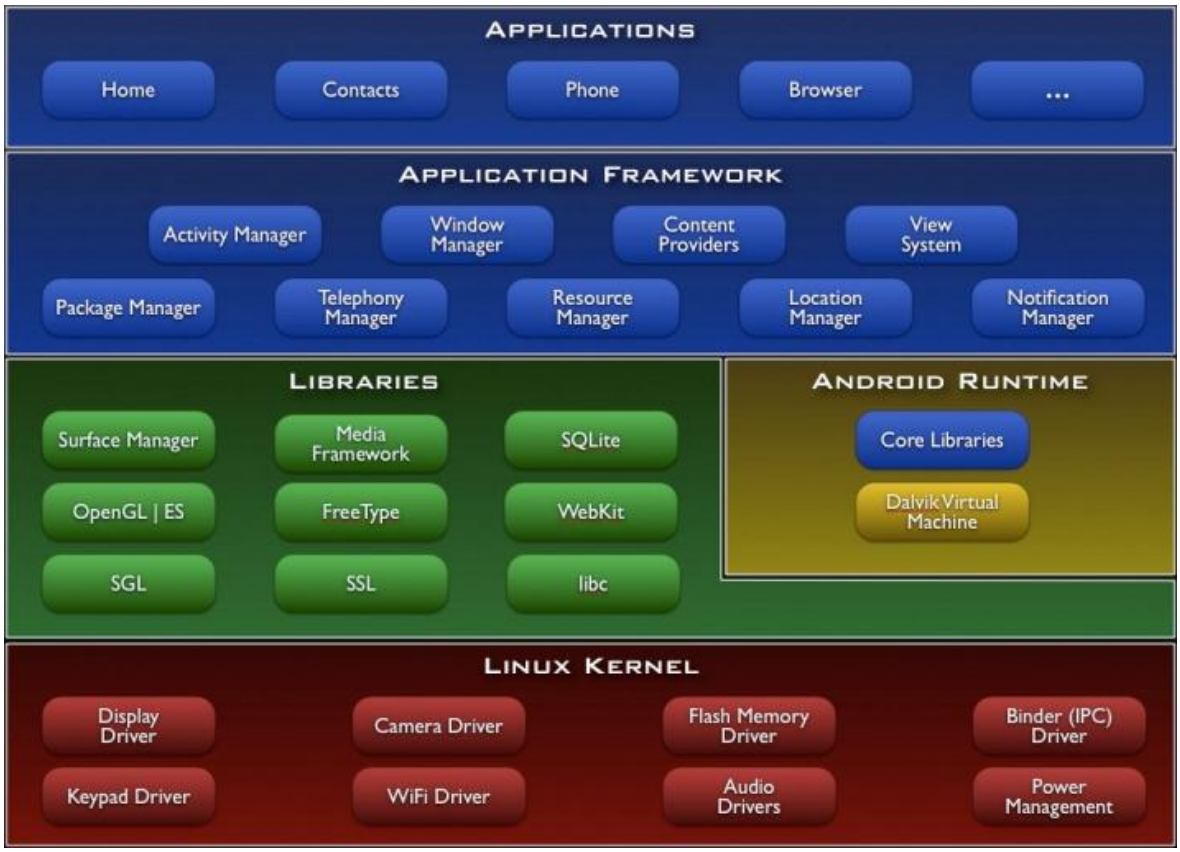

Figura 2. Arquitectura de Android. Fuente: Android Developers [4]

2.1.2 DESARROLLO

El desarrollo de aplicaciones para Android está basado en el lenguaje Java, utilizado el Android SDK (*Software Development Kit*). Sin embargo, existen librerías y *plugins* que permiten el desarrollo en C o C++. Para el desarrollo, Google ha publicado un entorno de

desarrollo integrado o IDE gratuito llamado Android Studio (del cual se hablará más adelante, en el apartado 2.2), el cual mantiene y actualiza regularmente. También existen otros entornos populares como Eclipse, que es *Open Source* y posee una gran comunidad con numerosos *plugins* y modificaciones.

2.2 ANDROID STUDIO

Android es el sistema operativo móvil que se ha elegido para llevar a cabo el desarrollo de la aplicación del cliente de este proyecto. Para tener acceso a las APIs y librerías estándar de Android, es necesario disponer del Android SDK, y de un entorno de desarrollo compatible con el mismo.

Además, el desarrollo Android tiene una gran componente de interfaz gráfica, por lo que se busca que el entorno de desarrollo ofrezca la mejor asistencia posible, con ayudas visuales y pre-renderizados que permitan observar los cambios en el interfaz sin necesidad de desplegar la aplicación completa en un terminal.

Una opción muy popular, sobre todo en los inicios del desarrollo Android es Eclipse. Eclipse es un IDE *Open Source* desarrollado y mantenido por Eclipse Foundation. Es el IDE más utilizado para desarrollo en Java, que posee una gran cantidad de librerías y *plugins* de una comunidad numerosa de usuarios, y su gran ventaja es que resulta familiar a la gran mayoría de desarrolladores Java.

Sin embargo, la otra opción es Android Studio, que es el entorno oficial recomendado, desarrollado y mantenido por Google. Está basado en el IntelliJ IDEA, uno de los IDE de primer nivel para Java. Este sistema ofrece una serie de ventajas que lo hacen la elección preferida sobre los competidores:

- Posee un potente editor de código que hace que codificar sea más rápido que con cualquier otra opción.
- Posee un emulador de dispositivos Android integrado muy completo, que permite simular dispositivos con distintas prestaciones, pantallas y densidades de pixeles.

- Un sistema de compilación Gradle que es muy flexible, facilita la reutilización de código y la gestión de dependencias (basándose en Maven), permite la compilación por línea de comandos y permite crear distintas versiones de la aplicación de forma muy sencilla, por ejemplo, con versiones destinadas a distintos dispositivos.
- Posee la función "Instant Run", que permite aplicar cambios y actualizaciones a las aplicaciones "en caliente", es decir, en tiempo de ejecución sin necesidad de compilar un nuevo código.
- Posee soporte integrado para la "Google Cloud Platform", que facilita la integración de Google Cloud Messaging y App Engine
- Compatible con C++ y NDK. El NDK (o *Native Development Kit*) permite instalar bibliotecas escritas en C, C++ y otros lenguajes.
- Una gran cantidad de herramientas y *frameworks* de prueba.
- Integración de plantillas de código y GitHub para compilar funciones comunes de las apps e importar ejemplos de código.

Por estas razones, para la consecución de los objetivos de este proyecto, ha utilizado el IDE Android Studio. En la siguiente figura (Figura 3) se muestra una captura del interfaz del entorno en un momento del desarrollo.

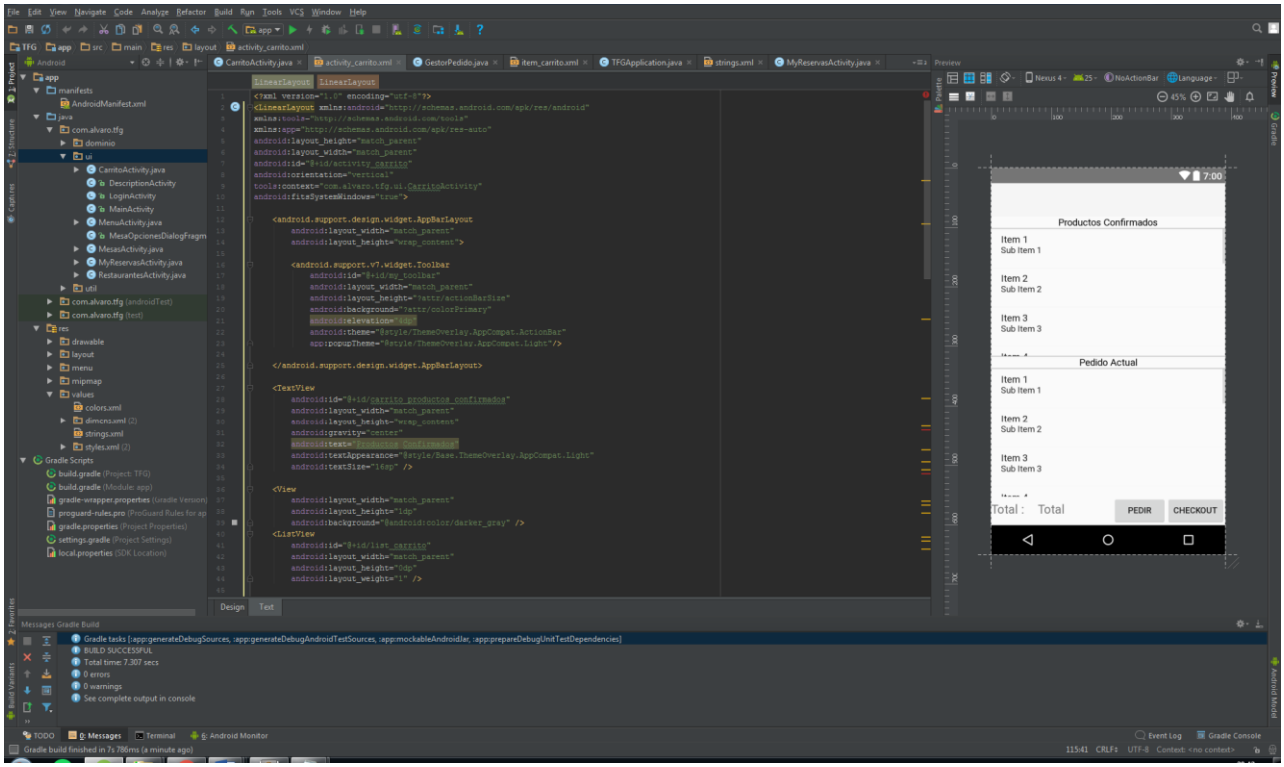

Figura 3. Ejemplo de proyecto en Android Studio

En la figura superior (Figura 3) se puede ver un ejemplo del interfaz del entorno de desarrollo. Al editar un archivo de formato XML que define un interfaz en Android, se puede ver en tiempo real, un ejemplo del resultado del código en un dispositivo con unas características de pantalla que son configurables.

2.3 JAVA EE

Java es el lenguaje elegido para el desarrollo de toda la lógica del software de este proyecto. Java es un lenguaje multiparadigma, orientado a objetos, que destaca porque permite a los desarrolladores escribir un único código sin importar las distintas plataformas en las que se vaya a ejecutar ("WORE" o *Write Once, Run Everywhere)*, es decir, el código es portable.

Esto es posible gracias a que el *bytecode* que genera el compilador Java (del código fuente en formato .java se genera un archivo .class) no es dependiente de la arquitectura del sistema en el que se encuentra, sino que este código es interpretado y ejecutado sobre una "*Java*

Virtual Machine" (JVM). Cada proceso Java funciona sobre una máquina virtual preinstalada en el dispositivo, que es la que ejecuta el programa y se adapta a la arquitectura que lo ejecuta. Este proceso se muestra en la siguiente figura:

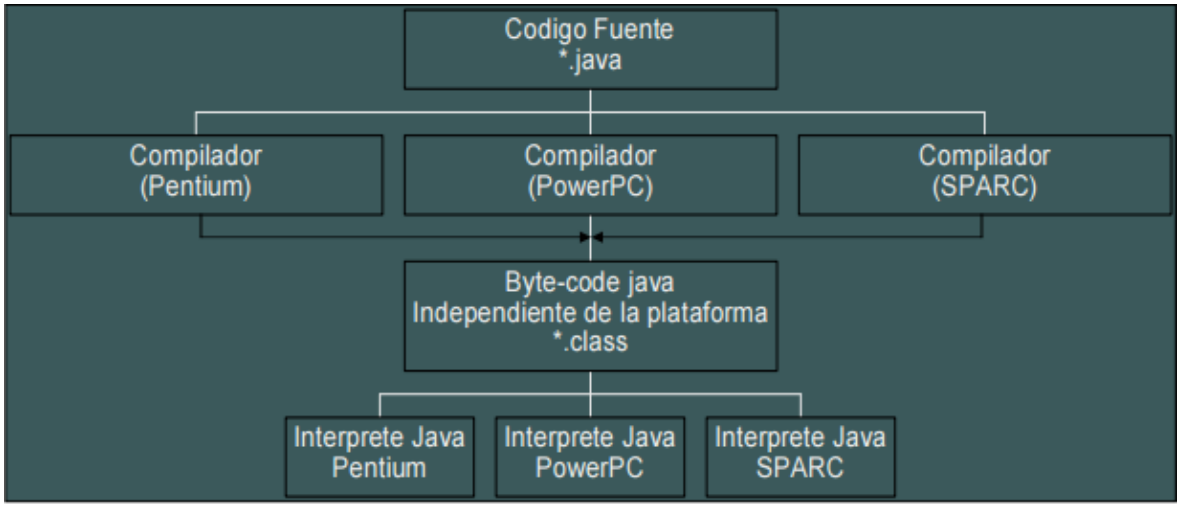

Figura 4. Programas compilados en Java

La plataforma Android posee a su vez una máquina virtual, sistema operativo elegido para el desarrollo de una de las piezas clave de este proyecto. De esta forma, parte de la lógica del servidor podrá ser reutilizada en el cliente y viceversa (por ejemplo, clases de dominio de aplicación).

Java es uno de los lenguajes más populares hoy en día para el desarrollo, en especial para el desarrollo de aplicaciones distribuidas basadas en web, como es el caso que a esta memoria ocupa. Esto es debido a una serie de ventajas debido a sus características:

- Simplicidad: la sintaxis es simple y muy similar a lenguajes populares como C++, lo que lo hace un lenguaje sencillo de aprender y manejar.
- Orientación a objetos: le ofrece al programador técnicas de diseño y recursos propios de este paradigma, como son la herencia o el polimorfismo.
- Abstracción: el lenguaje contiene numerosas librerías ofreciendo abstracciones que facilitan el uso de elementos de bajo nivel, como interfaces de entrada y salida o de red.

- Permite la concurrencia de forma muy sencilla, ya que cualquier objeto puede generar un nuevo proceso que se ejecuta sobre la JVM.
- Contiene numerosos elementos que facilitan la programación de aplicaciones distribuida, como las *Applets*, que son aplicaciones Java remotas que interactúan a través del protocolo HTTP.

En particular, estos servicios web no se incluyen en la versión estándar de Java, la Java SE (*Standard Edition*). Para acceder a ellos es necesario utilizar la *Enterprise Edition*, la Java EE, con una API que incluye servicios web como *Servlets* y *Applets.*

Java EE (en particular la versión 8) es la versión de Java que ha utilizado para desarrollar la lógica de negocio en el servidor y para gestionar la comunicación HTTP. También se accederá de forma programática a la base de datos, y para ello hará uso del *JDBC Connector* para MySQL, y se usará SQL a través de los *Prepared Statements*, que aportarán seguridad ante posibles ataques de *SQLInjection*.

2.4 NETBEANS

Como se ha comentado en el apartado anterior (2.3), se ha desarrollado el servidor y la lógica de negocio en Java. NetBeans es uno de los 3 entornos de desarrollos integrados más populares para el desarrollo en este lenguaje, los otros dos siendo los ya comentados Eclipse e IntelliJ.

El entorno de NetBeans ofrece una serie de ventajas que lo hacen la elección ideal para este proyecto:

- Tiene integrados todos los componentes necesarios: contiene el JDK de Java, el servidor web Apache Tomcat para Java.
- Es *Open Source*, por lo que es gratuito.
- Soporte para las tecnologías de Java EE, y fácil integración de librerías
- Incluye soporte para otros lenguajes además de Java, como son HTML o JavaScript, importantes en la programación web

Otro factor a favor de esta decisión es la experiencia ya adquirida en el uso de este entorno de desarrollo. En la siguiente figura (Figura 5) vemos un ejemplo de una pantalla de NetBeans durante el desarrollo del proyecto:

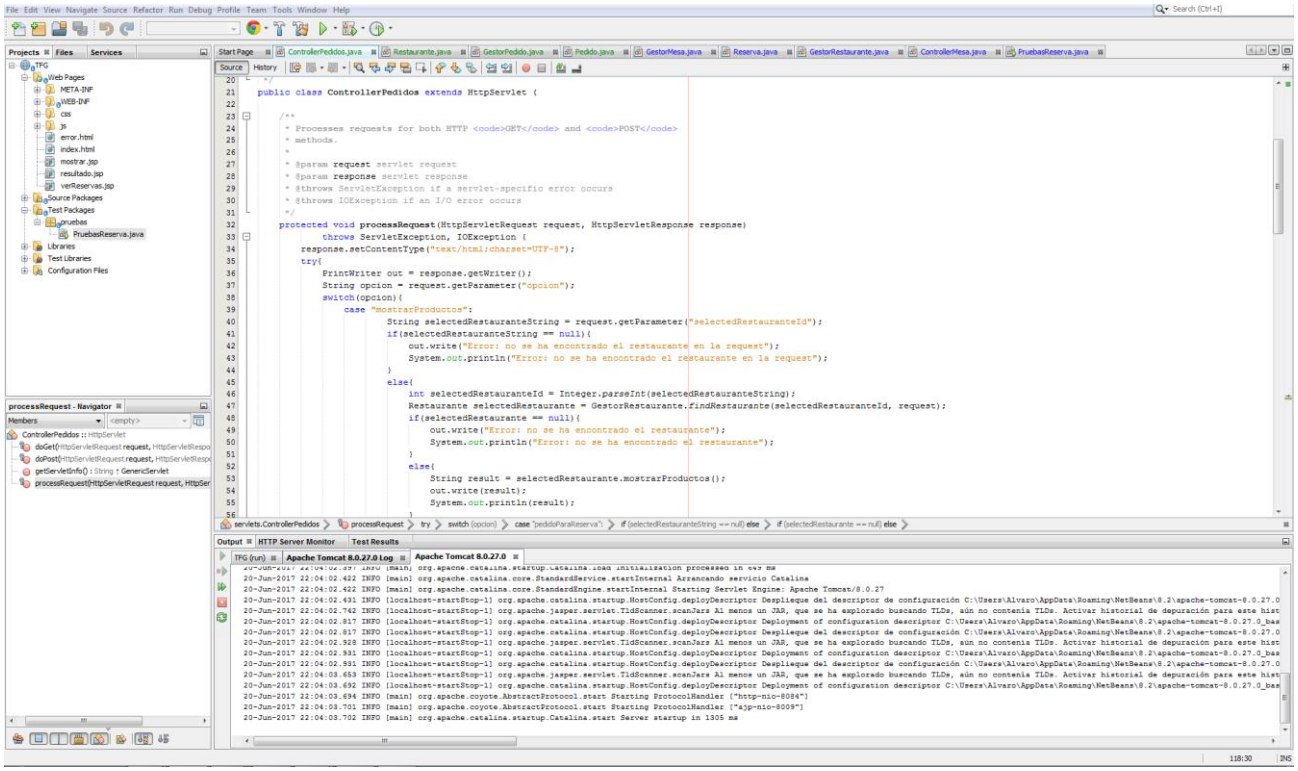

Figura 5. Ejemplo de proyecto en NetBeans

2.5 TOMCAT 8

Debido al lenguaje elegido, era necesario encontrar un servidor que fuera compatible con código Java. Una de las soluciones más populares es Apache Tomcat. Apache Tomcat es una implementación *Open Source* de los *Java Servlets*, *JavaServer Pages* (JSP), *Java Expression Language* (EL) y las *Java WebSocket Technologies*. De esta forma, el software hace las funciones de servidor, gestionando las comunicaciones HTTP y haciendo de interfaz con los *Servlets* y *JSP* de la aplicación.

2.6 SQL, MySQL y HeidiSQL

Para la capa de acceso a los datos, era necesario diseñar una base de datos. La primera cuestión por resolver es qué tipo de bases de datos utilizar. Para ello, analizamos la naturaleza de la información que se maneja:

- Información es sencilla y estructurada.
- La estructura no va a cambiar en el corto plazo.
- No se van a manejar grandes cantidades de información.

Por estas razones, la elección es una base de datos relacional. El lenguaje por excelencia para gestionar e interactuar con una base de datos relacional es SQL. SQL es un lenguaje diseñado por IBM en los años 70, y sobrevive desde entonces como el lenguaje más utilizado para trabajar con RDBMS (*Relational DataBase Management System*).

El lenguaje SQL está basado en el álgebra relacional, y la unidad básica de información son las tuplas. La sintaxis incluye:

- INSERT: inserción de datos en una base de datos
- SELECT: instrucción básica para extraer datos de una base de datos (*Query*)
- UPDATE: actualizar un dato en una base de datos
- DELETE: eliminar un dato de la base de datos
- Otras funciones de manipulación del esquema: CREATE TABLE, DROP TABLE, etc…
- Control de acceso a base datos
- Otras funciones para modificar los datos

A través de sentencias SQL y JOINS se accede a datos que se encuentran en bases de datos relacionales normalizadas.

Como sistema gestor de bases de datos relacionales (RDBMS) se ha elegido MySQL. Es un sistema *Open Source* ahora propiedad de Oracle, que está integrado en el IDE elegido

(NetBeans). El servidor accede a la base de datos a través de un interfaz para Java y MySQL, llamado *JDBC Connector* y el controlador (*driver*) correspondiente.

Para interactuar con la base de datos se ha utilizado un interfaz gráfico para SQL llamado HeidiSQL. Este programa *Open Source* ofrece una capa de abstracción sobre la base de datos, y da una vista de los datos de la base de datos, a la vez que permite al creación y modificación de las tablas mediante interfaz gráfico. Además, permite la ejecución de *Queries* desde el mismo interfaz, ofreciendo el resultado en forma de tabla a través del mismo.

En la siguiente figura (Figura 6) se observa una vista de la base de datos a través del interfaz gráfico de HeidiSQL:

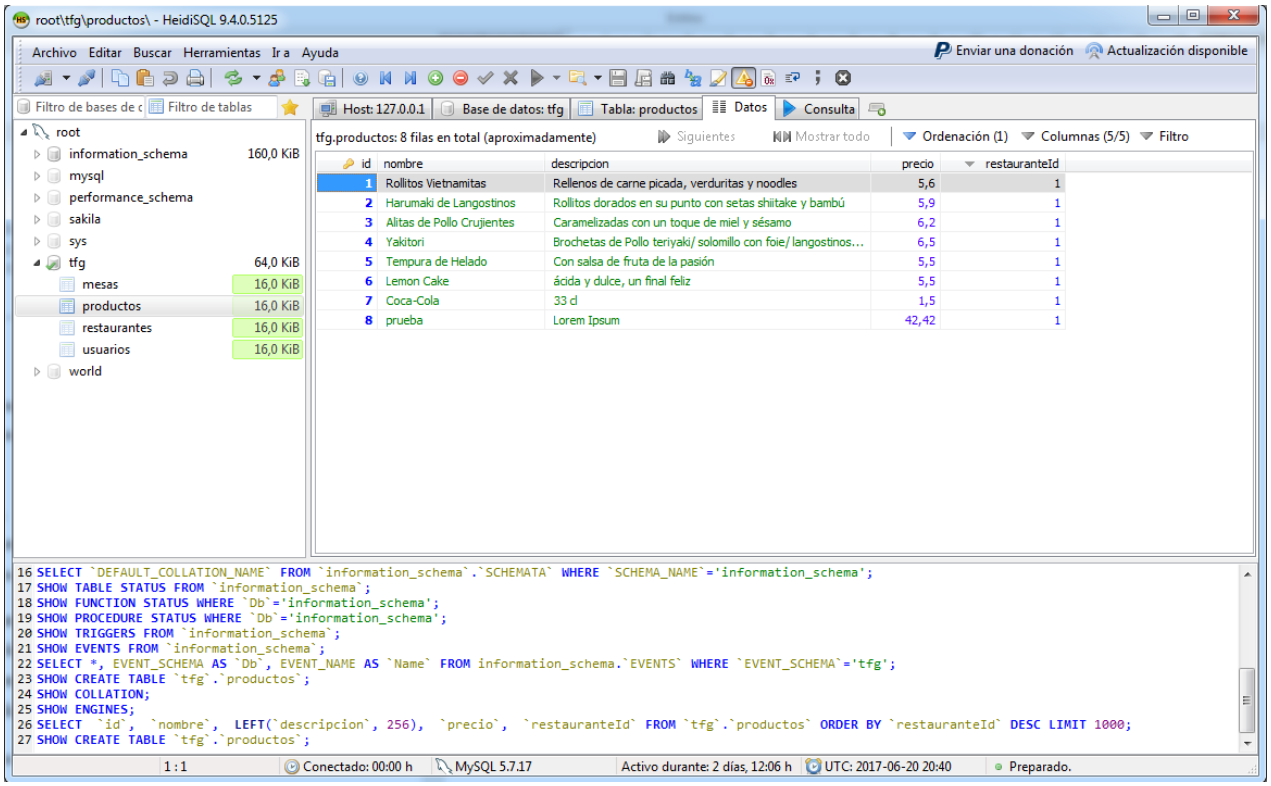

Figura 6. Base de datos vista a través de HeidiSQL

2.7 RASPBERRY PI 3

Uno de los requisitos del sistema es que pueda ser accedido desde cualquier lugar. Para ello es se ha decidido que el sistema debe estar conectado a Internet, y ser accesible desde una red externa (fuera de la red de área local, o red empresarial).

Para conseguir esto, es necesario instalar la lógica de negocio en algún equipo que sea compatible con Apache Tomcat, y que esté conectado a Internet. El equipo elegido ha sido una Raspberry Pi 3, que estará configurada para tener una IP estática, accesible desde el exterior de la red, con un dominio público gratuito registrado.

La Raspberry Pi 3 forma parte de la gama Rapsberry Pi, que es una serie de "*Single-Board Computers"* u ordenadores de una sola placa, que fueron desarrollados por la Raspberry Pi Foundation para promover la enseñanza de informática básica.

Estos ordenadores se caracterizan por tener todo el hardware necesario para su correcto funcionamiento en una misma placa, generalmente de pequeño tamaño. La gama de Raspberry no incluye periféricos de interfaz de usuario, como pantallas, teclado o ratón, pero sí que tienen conexiones (tanto USB y Micro USB como HDMI o Mini HDMI, dependiendo de la versión) preparadas para éstos en caso de que se desee utilizarlos, o para las configuraciones iniciales.

La gama Raspberry es compatible con sistemas operativos basados en Linux, aunque se recomienda una distribución *Open Source* desarrollada específicamente para su uso en esos dispositivos, llamada Raspbian.

Estos dispositivos destacan sobre todo por su reducido precio para las funcionalidades que ofrecen. La Raspberry Pi 3 es la versión más moderna de esta placa, e incluye un procesador de arquitectura ARM (ARM Cortex-A53 de 64-bits) de 1,2 GHz (el más potente de la gama), 1 GB de RAM, además de tarjetas de WiFi y Bluetooth integradas (por lo que no es necesario comprar periféricos para comunicaciones). Como memoria principal utiliza una tarjeta SD, que es un sistema de memoria de acceso más rápido y compacto que discos duros

tradicionales. En la siguiente figura () vemos una Raspberry Pi 3 similar a la utilizada, con las conexiones disponibles:

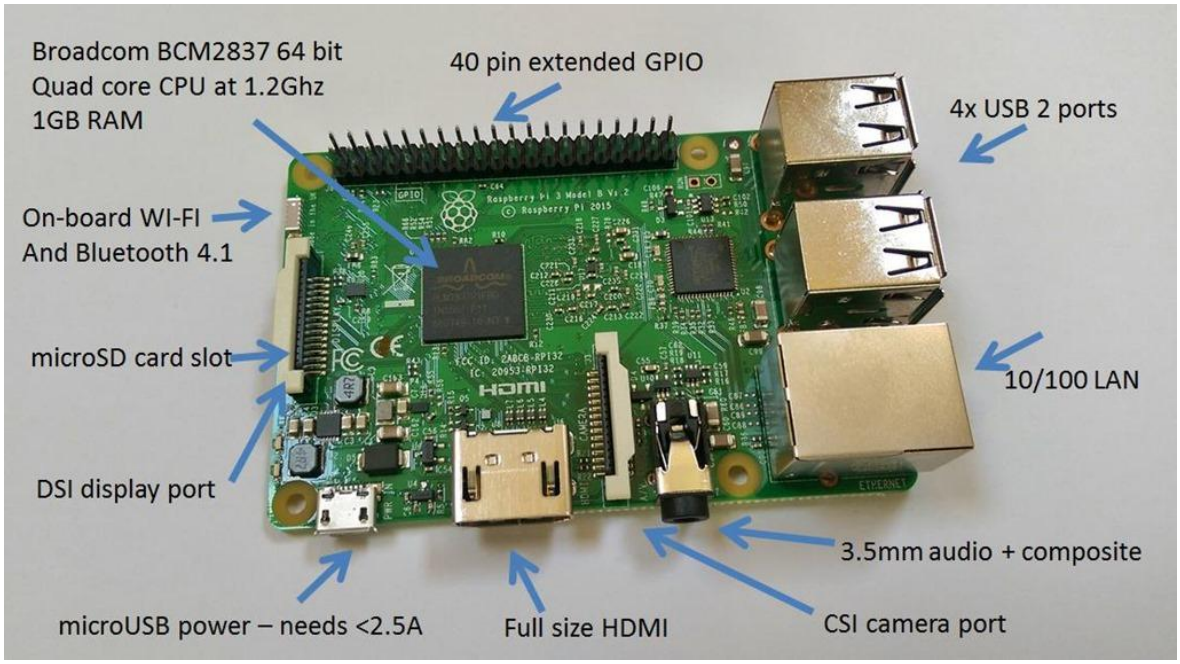

Figura 7. Raspberry Pi 3 y sus conexiones

Un sistema con estas características es más que suficiente para hacer las funciones de servidor de este sistema, al menos en la primera fase, ya que los recursos necesarios de este servidor son cubiertos por la Raspberry Pi 3. Además, este sistema destaca por tener un consumo eléctrico muy reducido, lo que reduce los costes de mantenimiento.

Capítulo 3. ESTADO DE LA CUESTIÓN

En este capítulo se procede a exponer el marco en el que se encuentra las industrias involucradas con este proyecto. En primer lugar, se va a hablar del mercado de los smartphones y las aplicaciones móviles, y más adelante se expondrá como este ha influido y la industria de la restauración.

3.1 PROLIFERACIÓN DE LAS APLICACIONES MÓVILES

3.1.1 CRECIMIENTO DEL *SMARTPHONE*

Hoy en día, los móviles han llegado a todos los rincones planeta, y han alcanzado a prácticamente la totalidad de la población. La penetración de los teléfonos móviles en el mundo alcanzó en 2015 el 97%, de hecho, únicamente cuatro regiones en el mundo poseen una penetración del teléfono móvil menor al 100%. El número de dispositivos móviles superó los 7,9 mil millones, más que personas en el mundo [1].

De estos teléfonos móviles, muchos son *smartphones*. Se considera *Smartphone* a un dispositivo de pequeño tamaño (que quepa en una mano) que además de las típicas funciones de teléfono (hacer y recibir llamadas, enviar y recibir mensajes) ofrece otros servicios, generalmente a través de aplicaciones o *apps*, como reproducción de música, navegación en Internet, ubicación GPS, cámara, etc.

En países desarrollados, la gran mayoría de estos dispositivos entran en la categoría de teléfonos inteligentes, en Europa un 78% de los teléfonos cumplen con estos requisitos. España es el líder europeo, con un 92% del total de teléfonos móviles. El *smartphone* es el dispositivo de elección para conectarse a Internet por la mayoría de los españoles.

Y estas tendencias no están cambiando, sino que se espera que se acentúen. Se espera que el número de dispositivos móviles en el mundo siga creciendo, y que el tráfico generado por estos se multiplique por 7 en 5 años.

3.1.2 MERCADO DE LOS SISTEMAS OPERATIVOS MÓVILES

A pesar de ser un mercado abierto, y de muchos de los productos estar basados en núcleos *Open Source*, el mercado de los sistemas operativos móviles ha quedado estructurado como un oligopolio de facto, en el cual 3 de los sistemas acaparan prácticamente la totalidad de los dispositivos actuales: Android, de Google; iOS, de Apple; Windows Phone, de Microsoft.

Sin embargo, la tendencia en los últimos años, y cada vez más acentuada, es que las marcas elijan el sistema Android para sus nuevos dispositivos, a excepción de Apple (en sus iPhone e iPad), mientras el resto de sistemas operativos van perdiendo importancia. En la tabla siguiente (Tabla 1) vemos las ventas de nuevos *Smartphones* en los últimos cuatrimestres de 2015 y 2016, agrupadas por el sistema operativo integrado.

Sistema Operativo	4Q16 Unidades	4Q16 Cuota de Mercado (%)	4Q15 Unidades	4Q15 Cuota de Mercado
Android	352 669,9	81,7	325 394,4	80,7
iOS	77 038,9	17,9	71 528,9	17,7
Windows	1 092,2	0,3	4 395,0	1,1
BlackBerry	207,9	0,0	906,9	0,2
Otros Sistemas	530,4	0,1	887,3	0,2
Total	431 539,3	100	403 109,4	100

Tabla 1. Ventas mundiales de Smartphones por Sistema Operativo. Fuente: Gartner [5]

La realidad es que el mercado está tendiendo a un duopolio, en el que Android e iOS acaparan el más del 99% de los dispositivos. En el mercado español, esta tendencia está más acentuada: el 93,9% de los nuevos dispositivos *smartphones* vendidos en España entre

febrero y abril de 2016 llevaban instalado un sistema operativo Android, un 5,5% iOS, y Windows Phone apenas un 0,5% (Figura 8).

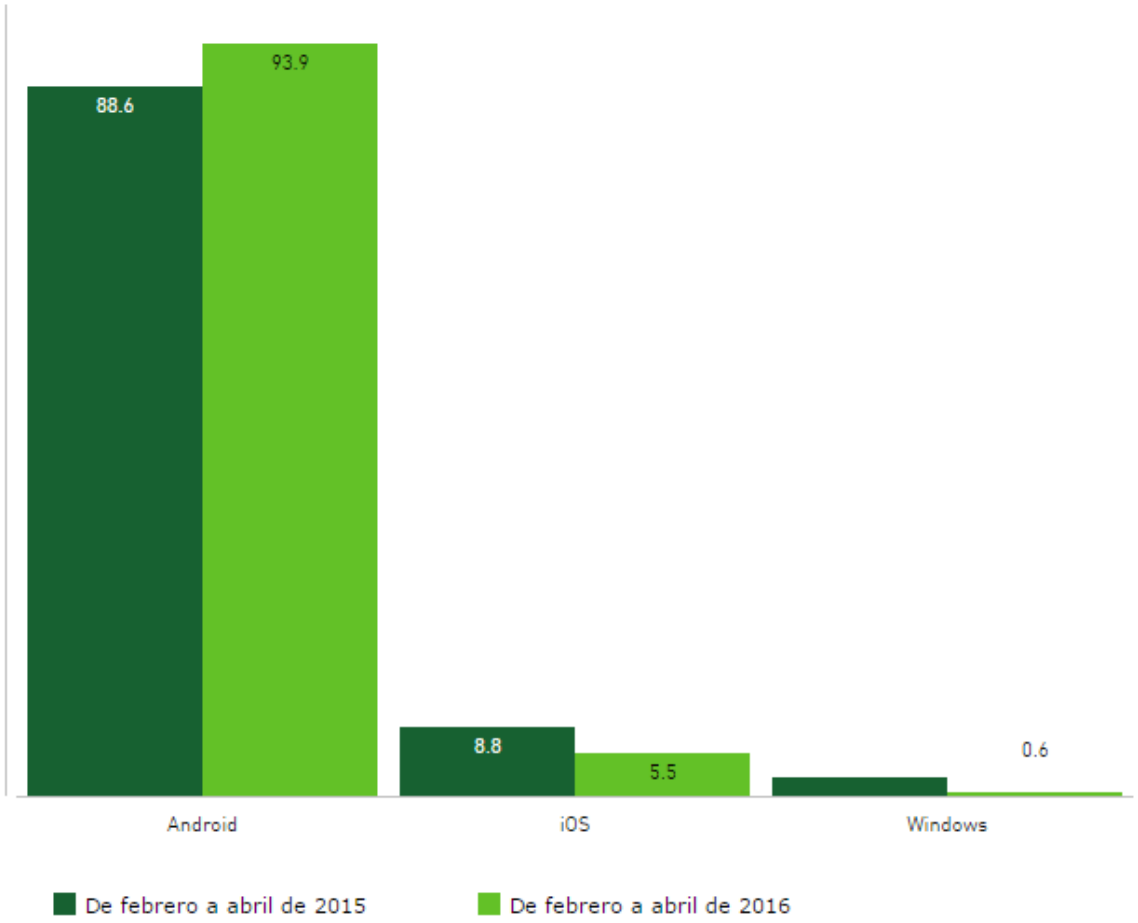

Figura 8. Cuota de mercado de nuevas ventas de smartphones. Fuente: Kantar [6]

La gran mayoría de fabricantes eligen Android como su sistema operativo, gracias al gran soporte que tiene por parte de Google con frecuentes actualizaciones que ofrecen mejoras visuales y técnicas. Además, es un sistema que fomenta mucho el desarrollo de aplicaciones independientes, al ser un sistema abierto y tener una gran comunidad de desarrolladores que comparten librerías y *plugins* que enriquecen las aplicaciones.

El otro gran competidor es iOS. iOS es el sistema desarrollado por Apple específicamente para sus dispositivos móviles. Al estar desarrollado únicamente para una pequeña gama de dispositivos, el sistema puede optimizarse al máximo al hardware. De esta forma el sistema destaca por una experiencia de usuario muy fluida y agradable.

3.1.3 APLICACIONES MÓVILES Y *ECOMMERCE*

La evolución de este mercado ha dependido del desarrollo de los *smartphones* y de los sistemas operativos. Sin embargo, con la aparición de las soluciones basadas en web las aplicaciones móviles evolucionan de forma más independiente.

El uso de aplicaciones móviles se ha normalizado entre la población. Las descargas anuales de aplicaciones alcanzaron los 90 mil millones en 2016, y durante ese año se emplearon cerca de 900 mil millones de horas utilizando aplicaciones móviles en el mundo (sin contar China, que es uno de los países que más contribuye a esta estadística), lo que supuso un crecimiento del 25% con respecto al año anterior. En la siguiente figura podemos observar esta tendencia (Figura 9):

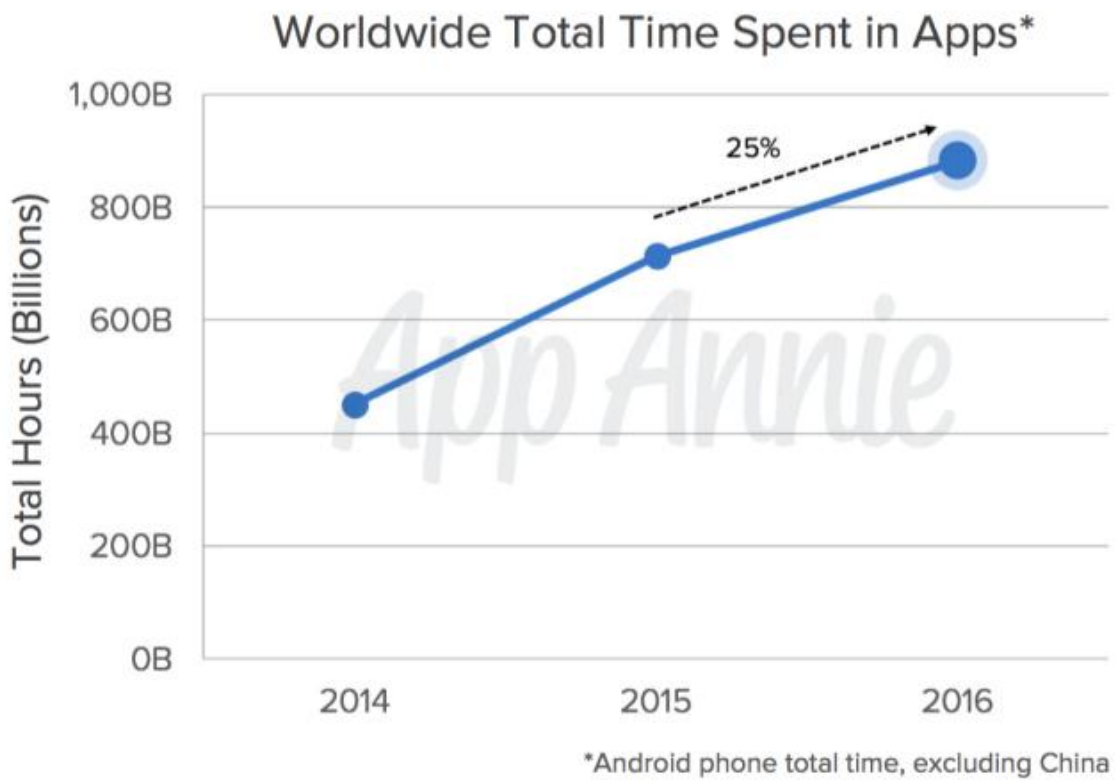

Figura 9. Horas empleadas dentro de aplicaciones móviles. Fuente: App Annie [2]

También se ha normalizado el pago mediante Internet. Esto ha provocado una explosión en la venta por internet o el *eCommerce*. El *eCommerce* supuso en 2016 el 8,7% en mercados como el de venta al por menor. Y este número se estima que crezca en los próximos años hasta llegar al 14,6%, como se observa en la siguiente figura (Figura 10):

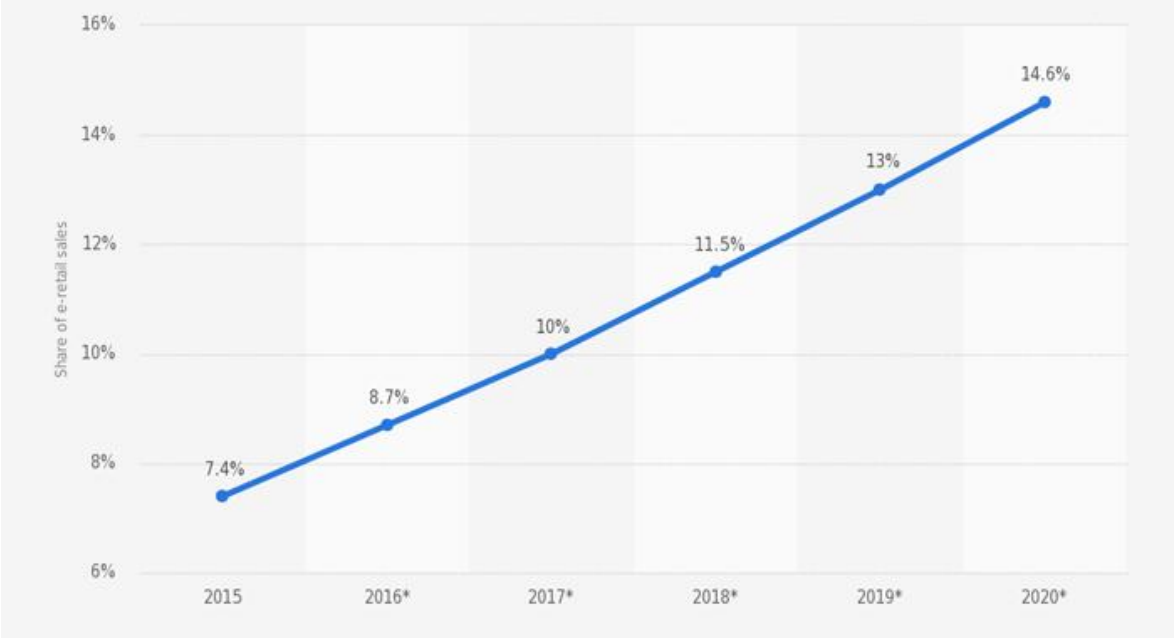

Figura 10. Porcentaje de ventas eCommerce sobre el total. Fuente: Statista [7]

Y recientemente, hay un sector del *eCommerce* que está creciendo más rápido que el resto: las ventas a través de dispositivos móviles, conocido como comercio móvil o *mCommerce.* Estas ventas constituían en 2015 el 30% de las ventas electrónicas, y cada año más.

Este tipo de comercio está triunfando porque es muy cómodo para el cliente. El cliente puede comprar en cualquier momento desde el sofá de su casa. Desde ahí puede observar una gran variedad de productos, comparar y elegir el que más le conviene. Además, ahorra tiempo eligiendo y en desplazamientos.

Estas ventajas se hacen mucho más evidentes desde dispositivos móviles. El cliente no tiene que esperar a llegar a su casa o a tener un ordenador con conexión a Internet para poder hacer la compra, ya que en cualquier momento tiene acceso desde su *smartphone*.

Desde el punto de vista del vendedor, esto es muy beneficioso. El cliente puede actuar de forma más rápida ante sus impulsos, se le pueden ofrecer más ofertas y de forma personalizada. De esta forma se consigue que el cliente no sólo compre una vez, sino que vuelva a por más. El comprador recurrente, es el cliente ideal que buscan las empresas, y esto es una de las grandes ventajas que ofrece la venta electrónica a las empresas. En una

encuesta el 55,3% de los encuestados afirmó que compraba todos los meses online (un 11,9% de ellos, con frecuencia semanal). De hecho, más de la mitad de los encuestados (54,5%) afirmó que prefería la venta online a la física [8].

3.2 INDUSTRIA DE LA RESTAURACIÓN Y LAS APLICACIONES MÓVILES

Aunque es en la industria del transporte, de la música y las redes sociales donde más se está viendo el potencial de las aplicaciones móviles, en la industria de la restauración también existen soluciones que son utilizadas por miles de personas a diario. El sector que más se ha aprovechado de las ventajas que ofrecen estas aplicaciones es el de la venta a domicilio.

3.2.1 SERVICIO A DOMICILIO

En España dominan este mercado Just Eat, que opera desde 2001 como un portal online que sirve de intermediario entre restaurantes que ofrecen comida a domicilio y clientes, no han dudado en apostar por esta plataforma, ideal para su modelo de negocio. Esta decisión que les ha ayudado a ofrecer sus servicios a más de 400.000 usuarios, solo en España. En la siguiente figura (Figura 11) vemos un ejemplo de una pantalla de su aplicación móvil en Android.

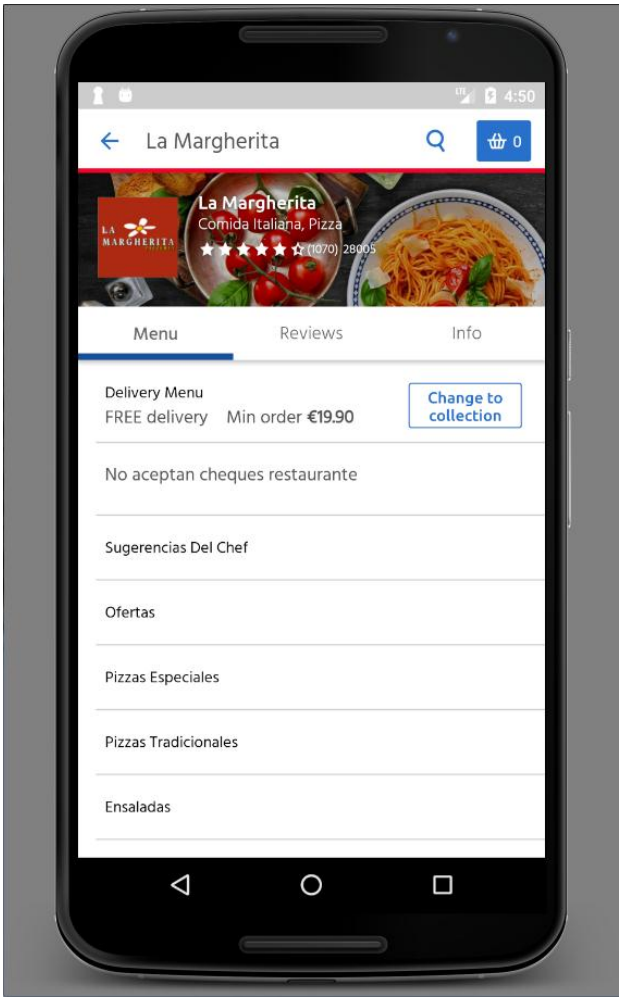

Figura 11. Ejemplo de pantalla de Just Eat en Android

También destaca La Nevera Roja (recientemente adquirida por Just Eat), que es una empresa más reciente que se centró exclusivamente en su aplicación móvil. La versatilidad de esta plataforma junto con su usabilidad hace que la mayoría de usuarios buscando un portal de restaurantes a domicilio apueste por la aplicación móvil como su plataforma de elección. De esta forma, La Nevera Roja llegó a dominar el 50% de la cuota de mercado en Madrid y Barcelona.

Este tipo de aplicaciones comparte unas características comunes, que persiguen un mismo objetivo: hacer la experiencia del usuario lo más cómoda y rápida posible. Éstas son:

- Basan sus servicios en la localización: debido a la naturaleza de la comida a domicilio, estas aplicaciones necesitan conocer la ubicación del usuario antes de ofrecer un listado de los restaurantes disponibles. Esto se hace accediendo a la funcionalidad GPS del dispositivo (si el usuario ha dado los permisos), o introduciendo manualmente una zona, mediante código postal o dirección.

- Sistema de usuarios: estas aplicaciones suelen disponer de un sistema de registro de usuarios, que hace que la aplicación pueda almacenar información sobre el usuario, como el nombre o la dirección o el medio de pago, que es información que suele ser solicitada por el restaurante y que no cambia con asiduidad, y evita que el usuario tenga que introducirla una y otra vez.

- Pago a través de la misma: estas aplicaciones suelen tener un sistema de pago online, ya sea por tarjeta de crédito o débito, u otro medio online como *PayPal*. Sin embargo, para aquellas personas que no desean utilizar este tipo de medios, suelen ofrecer también medios de pago tradicionales, tratando directamente con el repartidor del restaurante en cuestión.

- Dinamismo, transparente al usuario: la oferta de restaurantes, a la vez que la carta que ofrece cada uno de ellos, puede cambiar en cualquier momento, y gracias a que ésta se descarga cada vez que el usuario selecciona un restaurante, permite que ésta pueda cambiar sin necesidad de actualizar la aplicación o que el usuario tenga que hacer nada de forma activa.

- Ofertas y promociones: Estas aplicaciones suelen ofrecer promociones y ofertas, como códigos de descuento que cambian diariamente, mejorando la fidelidad del cliente y consiguiendo un incremento en el consumo.

A raíz del éxito de este modelo de negocio, han surgido otra serie de servicios a través de aplicaciones móviles, que ofrecen desde la empresa los repartidores, de forma que pueden incluir entre su oferta restaurantes que no poseen un servicio de reparto dedicado. Entre estas empresas destacan Deliveroo y Glovo, que han aplicado un modelo de negocio similar al de

Uber para el reparto. Así, cualquier persona puede ofrecerse para recoger pedidos en restaurante y llevarlos al destino, a cambio de una comisión.

3.2.2 APLICACIONES DE RESTAURANTES TRADICIONALES

Sin embargo, todas estas ventajas no son exclusivas del mercado de la restauración a domicilio, y algunos restaurantes tradicionales ya ofrecen aplicaciones móviles, aunque la mayoría, como Vips o Hollywood se centran en ofrecer promociones a sus clientes.

El mayor intento de introducir el potencial de este tipo de plataformas en servicio normal del mismo restaurante es el sistema de pedido de McDonald's (Figura 12). Este sistema se basa en unos grandes terminales táctiles (que no son realmente móviles), situados próximos a la barra donde se recoge la comida, en los que el usuario dispone de toda la carta que ofrece el restaurante, con todas las posibles modificaciones que hay en cada plato, con posibilidad de pago con tarjeta desde el mismo dispositivo. Esto agiliza el proceso de pedido y pago de los productos, y reduce el tiempo de espera medio del cliente.

Figura 12. Sistema McDonald's Easy Order. Fuente: Pinterest [9]

Este sistema, que lleva funcionando con éxito ya 2 años, no aprovecha al máximo la portabilidad que ofrecen aplicaciones móviles. El siguiente nivel consiste en ofrecer este mismo servicio en el teléfono de cada uno de los clientes por separado, sin necesidad de hacer colas para liberar los limitados terminales, e incluso sin necesidad de estar físicamente en el establecimiento. Esto se probó ya en el mismo McDonald's, pudiendo pedir sus famosos "Big King" a través de una aplicación como parte de una promoción temporal. En la actualidad, la cadena de cafeterías Starbucks, ofrece un servicio similar por el que toma pedidos desde teléfonos móviles.

Sin embargo, aún no hay ninguna solución que aúne todas las ventajas que ofrecen todos estos servicios por separado, en una única aplicación móvil, y que pueda ser aprovechada por una gran variedad de restaurantes (sin necesidad de tener una gran cadena o empresa detrás) para llegar al gran número de clientes potenciales que son los propietarios de un "Smartphone".

Capítulo 4. DEFINICIÓN DEL TRABAJO

En este capítulo se va a detallar cuáles son las motivaciones para la llevar a cabo este proyecto y cuáles serán los objetivos del mismo. Además, se va a exponer la planificación inicial con los distintos hitos que marcarán el desarrollo, y también cuáles serán los costes asociados a la consecución de los objetivos marcados.

4.1 JUSTIFICACIÓN

Tras años de recesión durante la crisis económica, las ventas en el sector de la restauración han retomado una tendencia positiva en los últimos años. En 2015, las ventas aumentaron un 3%, y se estimaba que para 2016, el crecimiento interanual fuera del 6%. Sin embargo, la mayor parte de este crecimiento (el 80% de las nuevas ventas) los canales de servicio rápido: *Fast Food*, servicio a domicilio, bares, cafeterías y heladerías [10].

No es casualidad que la mayoría de los restaurantes beneficiados son los que poseen aplicaciones móviles, o se ofertan a través de aplicaciones como Just East o Deliveroo.

El cliente cada vez dispone de menos tiempo para comer, y unido a que busca servicios más económicos, lleva a recurrir a las soluciones más rápidas y cómodas. Desgraciadamente, estas soluciones, por su naturaleza, suelen implicar menos calidad y pueden conducir a dietas desequilibradas y problemas asociados.

Los restaurantes necesitan nuevas formas de captar clientes, a la vez que buscan reducir los costes para poder ofrecer un producto más competitivo.

Hoy en día, una solución sencilla para captar nuevos clientes consiste en ofertarse en aplicaciones como Just Eat. Just Eat pone al alcance de los restaurantes la gran base de usuarios que posee, sirve como un buen punto para publicitarse. Sin embargo, esto solo está al alcance de restaurantes que pueden mantener un servicio de reparto a domicilio capaz de gestionar las solicitudes, y esto puede costar mucho dinero.

Como respuesta, han surgido aplicaciones como Deliveroo (y Just Eat recientemente lo ha añadido a su cartel), que incluyen el servicio de reparto, pero a cambio de una comisión sobre la cuenta final (que puede llegar al 30% del precio final). Esto es un coste que muchos restaurantes tampoco pueden permitirse, y sólo acentúa la otra necesidad: reducir costes operativos para ofrecer precios más competitivos, sin reducir la calidad del servicio ni del producto.

Con este proyecto, se busca ofrecer una plataforma que sirva como solución a estas dos cuestiones al mismo tiempo.

4.2 OBJETIVOS

El objetivo es crear una plataforma que permita a los usuarios disfrutar en un restaurante de un servicio rápido y de calidad. A su vez, esta plataforma servirá a las empresas como punto para publicitarse accediendo a nuevos clientes, distribuir ofertas, fidelización y como una herramienta que facilite la autogestión y que les permita ofrecer su servicio de forma más eficiente, reduciendo costes operativos.

Con este proyecto se busca implementar una prueba de concepto de dicha plataforma, que contenga una serie de funcionalidades esenciales para poder así confirmar la viabilidad del producto.

Las funcionalidades que se van a implementar en este proyecto son las siguientes (las funcionalidades que también son *Online* deben estar disponibles desde cualquier lugar, tanto desde el restaurante como desde cualquier otro sitio.

4.2.1 ELEGIR Y RESERVAR MESAS (ONLINE)

- El usuario debe poder ver la oferta de restaurantes disponibles a través de la plataforma.
- Para cada uno de estos restaurantes, se debe poder ver, la disponibilidad de las mesas de forma actualizada.

- El usuario debe poder elegir una mesa para "sentarse" (declararla como ocupada) o para reservarla en un futuro. También debe poder cancelar una reserva que ya haya hecho.

4.2.2 VER LA CARTA Y ELEGIR PRODUCTOS (ONLINE)

- El usuario debe poder ver la carta disponible de cada restaurante, elegir un producto de esa carta y añadirlo a su pedido.
- El usuario debe poder hacer el pedido para una hora determinada, para la que tenga la mesa reservada, y que se le sirva cuando se encuentre en el restaurante.
- El usuario debe poder ver ofertas y promociones activas en los distintos restaurantes.

4.2.3 SISTEMA DE USUARIOS REGISTRADOS (ONLINE)

- El usuario debe poder crear una cuenta personal, iniciar y cerrar la sesión en el dispositivo.
- La cuenta contendrá información básica sobre el usuario, como el correo electrónico, el nombre o su historial de compras.
- Parte de esa información podrá ser accedida por los restaurantes.

4.2.4 OPCIÓN DE USO TRADICIONAL

- Una vez se encuentre en el restaurante, sentado en una mesa, el usuario debe poder solicitar un camarero en cualquier momento, pudiendo así disfrutar del servicio de la forma tradicional.

4.3 METODOLOGÍA

Debido a las características del proyecto y del equipo (desarrollo de aplicación móvil de pequeño tamaño desarrollada por equipo de un único desarrollador), se ha seguido un modelo de desarrollo ágil, trabajando mediante incrementos y de forma modular.

Inicialmente, se pasará por una fase de especificación inicial. Se van a recopilar los requisitos de la aplicación, teniendo de antemano todas las funcionalidades deseadas y casos de uso que se darán en la aplicación, separados lo máximo posible en bloques claramente diferenciados y se creará un diseño lo más modular posible, acorde con dichos requisitos.

Una vez las funcionalidades estén claramente definidas, se elaborará un plan de trabajo de 3 meses, en el que se trabajarán de forma modular cada una de las funcionalidades, con estructuras similares al *sprint* de Scrum, con una planificación inicial del *sprint*, en base a la planificación del proyecto, y una duración del desarrollo de 3 semanas inicialmente.

Finalmente, al final de cada sprint se realizará un proceso de revisión del sprint, evaluando los resultados. A lo largo del desarrollo se irá midiendo la duración de las distintas tareas, y adaptando la planificación acorde con las variaciones sobre el plan original. Para ello se utiliza la plataforma online de temporizador y gestión de tiempos llamada Toggl. Un ejemplo de su uso se ve en la siguiente figura (Figura 13):

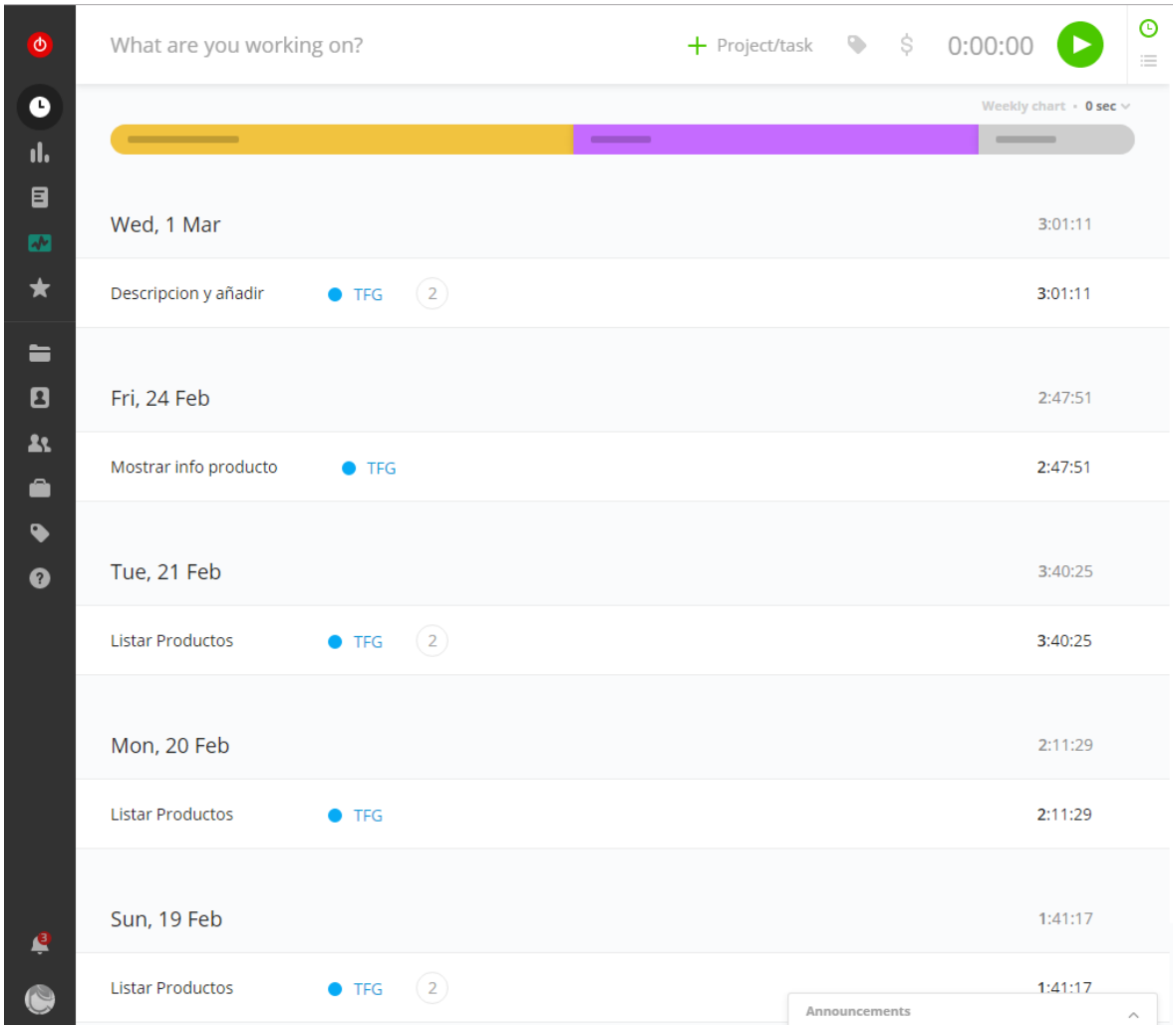

Figura 13. Ejemplo de uso de Toggl

En cualquier momento el diseño y planificación inicial pueden estar sujetos a cambios y modificaciones en función de la evolución del desarrollo.

4.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA

En la siguiente figura (Figura 14) se muestra un diagrama de Gantt con la planificación inicial de las distintas tareas de este proyecto:

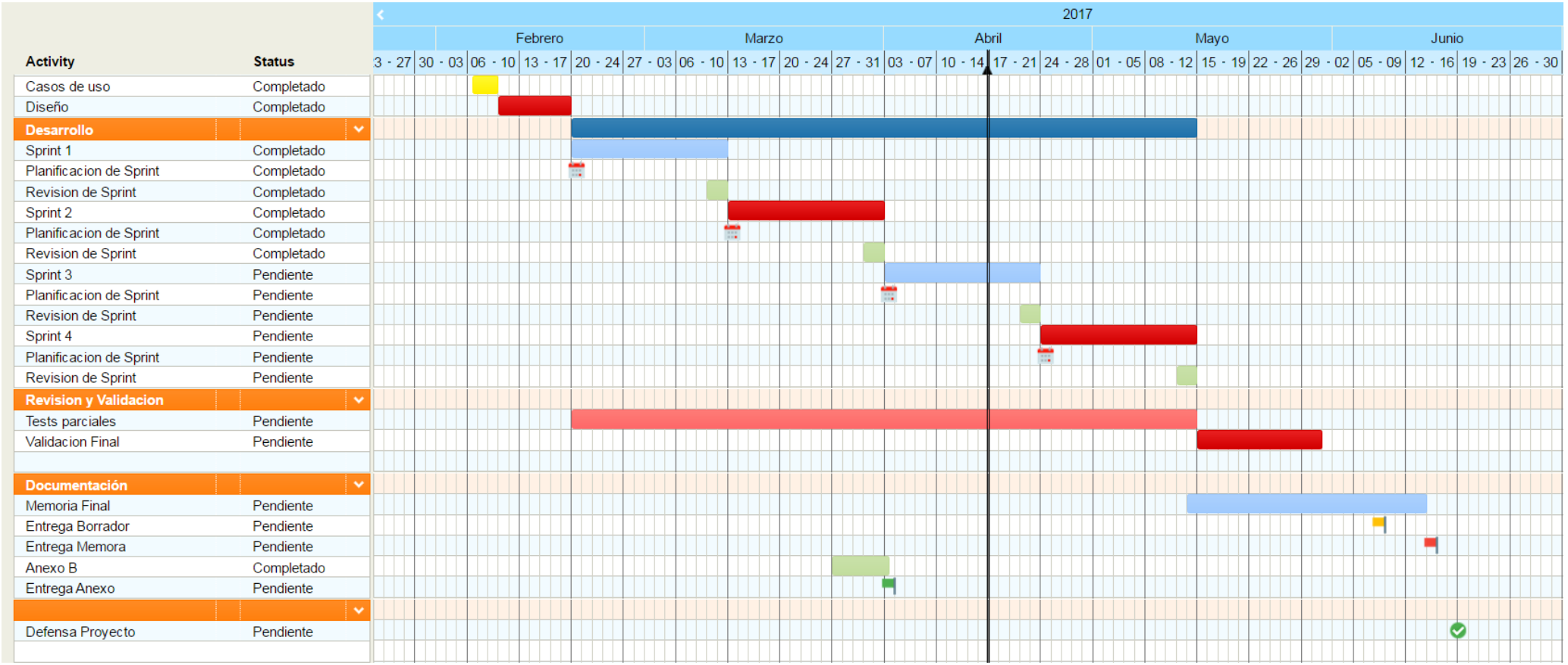

Figura 14. Diagrama de Gantt

Por otro lado, atendiendo a la planificación antes mencionada, los costes estimados de producción, operación y mantenimiento de la plataforma se dividen en el coste del trabajo y en el coste de la infraestructura:

- Trabajo: un ingeniero se ha encargado del diseño de la plataforma y de la implementación del servidor y de la aplicación Android. En el futuro será necesario un ingeniero de iOS para desarrollar la versión de la aplicación para dicha plataforma. Estos dos ingenieros son suficientes para mantener la plataforma durante las primeras fases de su crecimiento. El suelo estimado por ingeniero es de 1230€/mes.
- Infraestructura: los servidores se alojarán en un servicio de *hosting* en la nube, como AWS. En este el coste por día por servidor "t2.small" (AWS) con 2GB de memoria es de 0,609€, y se estima que se necesitarán 3 servidores en la primera fase de su desarrollo, lo que hace un coste mensual de 34,36€ en infraestructura. Al estar alojados en servicios en la nube, no serán necesarios más componentes como routers o switches.

De esta forma, el coste estimado es el siguiente:

	Coste Mensual (€)	Coste Anual (€)
Trabajo	2460	34440
Infraestructura	34,36	412,32
Total	2494,36	34852,32

Tabla 2. Estimación de costes

Capítulo 5. SISTEMA DESARROLLADO

En este capítulo se procede a describir el diseño de la plataforma en detalle, describiendo los casos de uso más relevantes, para más adelante explicar la implementación.

En el apartado 4.2 se explicaron los objetivos y se detalló una estructura que divide el sistema en grupos de funcionalidades. Estas divisiones se corresponderán con los módulos del sistema, y se utilizará la misma disposición a lo largo de este capítulo.

5.1 ANÁLISIS DEL SISTEMA

El sistema diseñado constará de un servidor de base de datos MySQL que junto con un servidor web y de aplicación Apache Tomcat irán instalados en una Raspberry Pi 3. Por último, estarán los clientes, que serán dispositivos móviles con sistema operativo Android instalado.

El servidor y los clientes se comunicarán a través de Internet, mediante el protocolo HTTP. Como los clientes no estarán necesariamente en la misma red local que el servidor, serán necesarios componentes externos que proporcionen conectividad con Internet al servidor web. Estos serán un router, al que se conectará la Raspberry Pi 3 (por WiFi o por conexión directa Ethernet), y un firewall que será el encargado de proteger la red contra ataques externos.

Esta estructura queda representada en el siguiente diagrama de arquitectura (Figura 15):

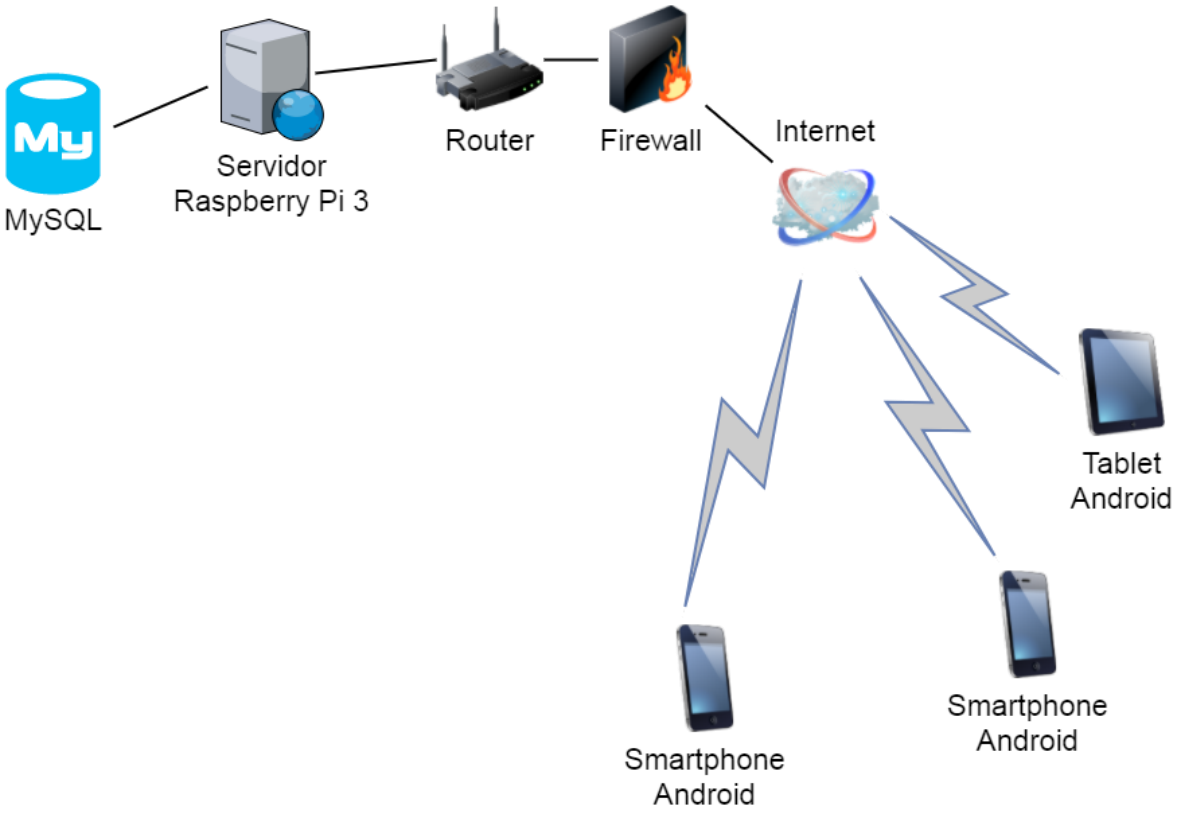

Figura 15. Diagrama de Arquitectura

Para el diseño software del sistema se ha elegido un patrón arquitectónico cliente-servidor. Este patrón se utiliza en plataformas que tienen una separación física o lógica entre el usuario del sistema (el cliente) y el servidor. De esta forma, se distribuyen las responsabilidades de las distintas capas de la aplicación. Estas capas son:

- Gestión de datos: esta es la capa encargada de dar persistencia a la información, guardándola en bases de datos, y de la recuperación de los mismos.
- Aplicación o lógica de negocio: esta es la capa encargada de implementar toda la lógica de negocio, tratando la información.
- Presentación: esta es la capa encargada de la adquisición de los datos y de la presentación de los mismos. Este es el conjunto de software que compone la interfaz entre la aplicación y el usuario.

Para este diseño, el patrón cliente-servidor elegido es función distribuida. De esta forma, el servidor será el encargado de todas las funcionalidades de la capa de gestión de datos, y de parte de la capa de negocio. Por su parte, parte del software encargado de la lógica de negocio se encontrará en la aplicación del cliente, y toda la capa de presentación. Esta división se muestra en la siguiente figura (Figura 16):

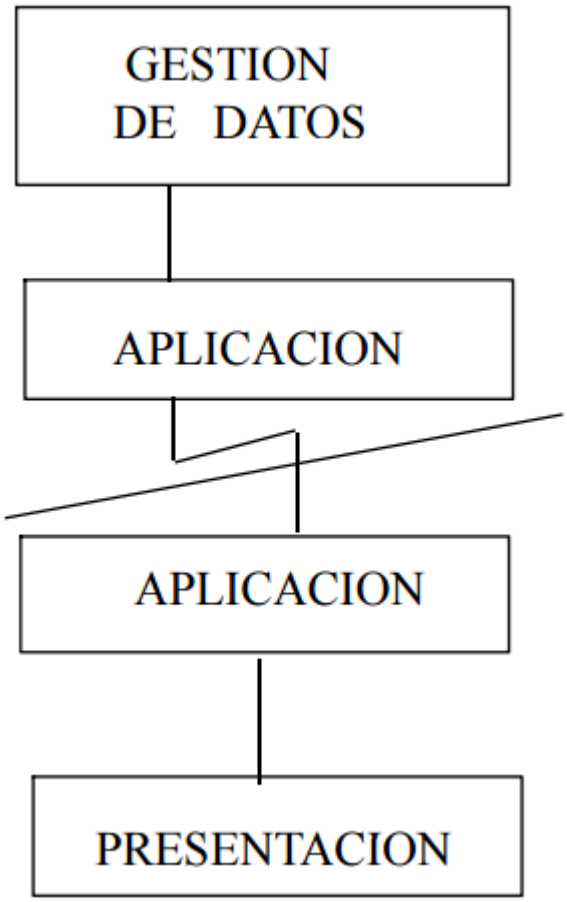

Figura 16. Arquitectura Cliente-Servidor con Función Distribuida

5.2 DISEÑO

La plataforma se ha dividido en una estructura modular. Esta estructura estará formada por módulos, compuestos de clases y librerías que gestionan funcionalidades de forma independiente entre sí, siguiendo una estructura que se corresponde con los objetivos

descritos en el apartado 4.2. Esta estructura se puede observar en el diagrama de bloques siguiente (Figura 17):

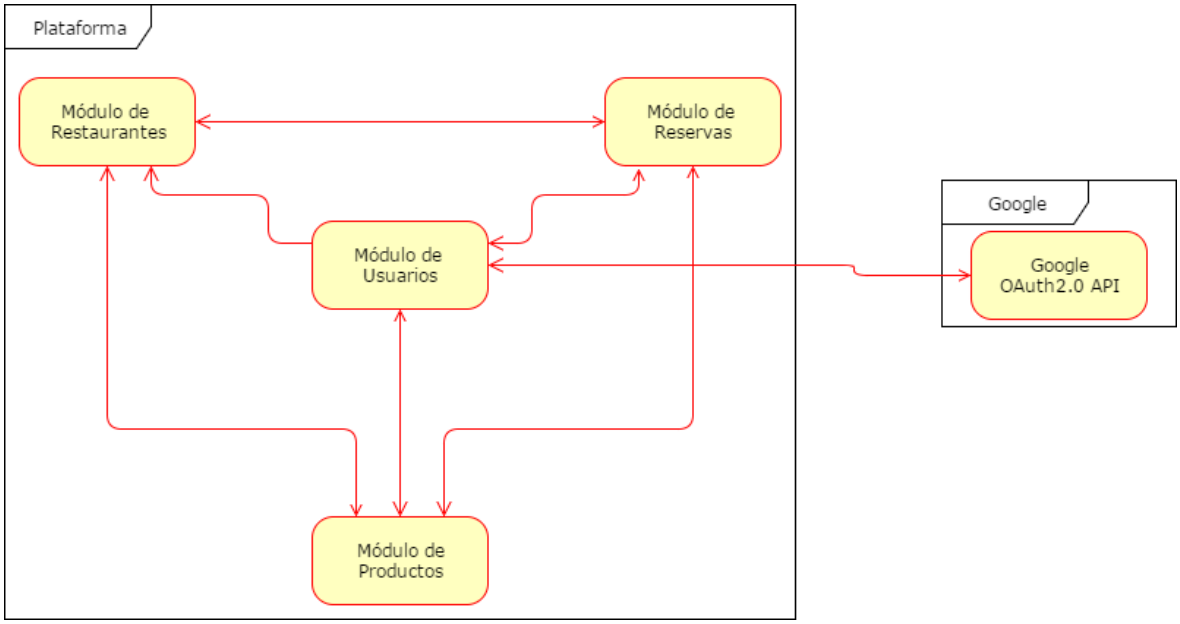

Figura 17. Diagrama de Bloques

En la figura podemos observar que existe un componente externo a la plataforma. Este es el servicio OAuth2.0 de Google, que permite externalizar la autenticación de los usuarios a través de un estándar con grandes niveles de seguridad como el ofrecido por Google.

Las aplicaciones del cliente y del servidor comparten esta estructura modular, aunque la implementación de los componentes varía en función de las necesidades de cada una de las partes.

A continuación, se va a detallar el diseño de estos módulos, explicando las funcionalidades que ofrecerán y su estructura.

5.2.1 MÓDULO DE USUARIOS

Este es el software encargado del sistema de gestión de los usuarios. Permite la creación de usuarios nuevos, asociándolos unívocamente a un cliente mediante la dirección de correo electrónico, aunque también contiene otra información personal del mismo. También es el encargado de mantener el historial de pedidos de un usuario.

El módulo de usuarios será el primer punto de interacción entre la plataforma y el cliente, ya que todos los servicios que esta ofrece requieren autenticación. En el diagrama siguiente (Figura 18) se muestran los casos de uso a los que este módulo dará servicio:

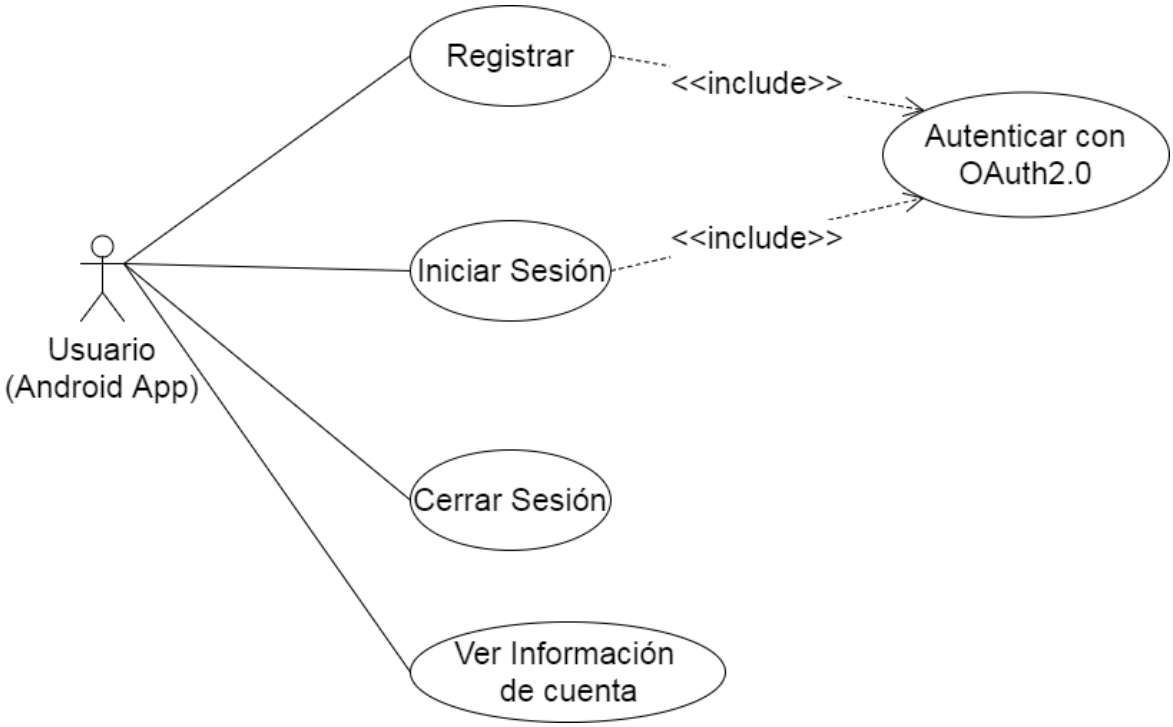

Figura 18. Diagrama de Casos de Uso del módulo de Usuarios

5.2.2 MÓDULO DE RESTAURANTES

Este es el software encargado de gestionar la información de los distintos restaurantes, y hacer de interfaz para el resto de módulos para identificar el restaurante sobre el cual cada usuario hace sus peticiones.

A través de este módulo el usuario podrá ver una lista con los restaurantes disponibles en la plataforma, pudiendo filtrar los restaurantes por categorías si se desea, y acceder a la información de un restaurante, como es la descripción, el teléfono, o el horario. Estas funcionalidades se resumen en el siguiente diagrama de casos de uso (Figura 19):

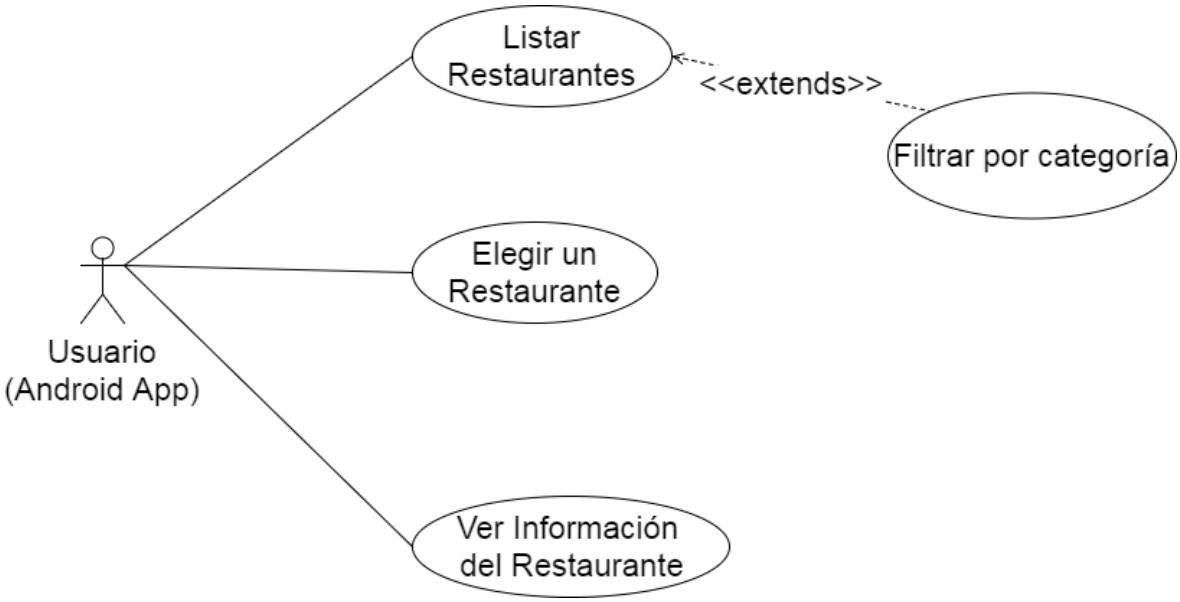

Figura 19. Diagrama de Casos de Uso de módulo de Restaurantes

5.2.3 MÓDULO DE RESERVAS

Este es el módulo encargado de gestionar las reservas en cada uno de los restaurantes. Una vez el cliente ha seleccionado un restaurante, el cliente puede listar las mesas que hay en el restaurante, pudiendo ocultar las que están ocupadas en el momento de la consulta. Además, el usuario podrá seleccionar una mesa para reservarla en función de su disponibilidad o para sentarse en el momento.

Adicionalmente, el cliente podrá ver un listado con las reservas que ya tiene, y podrá seleccionar una reserva no activa (en la que no esté sentado) y podrá cancelarla.

Todas estas funcionalidades se resumen en el diagrama de casos de uso siguiente (Figura 20):

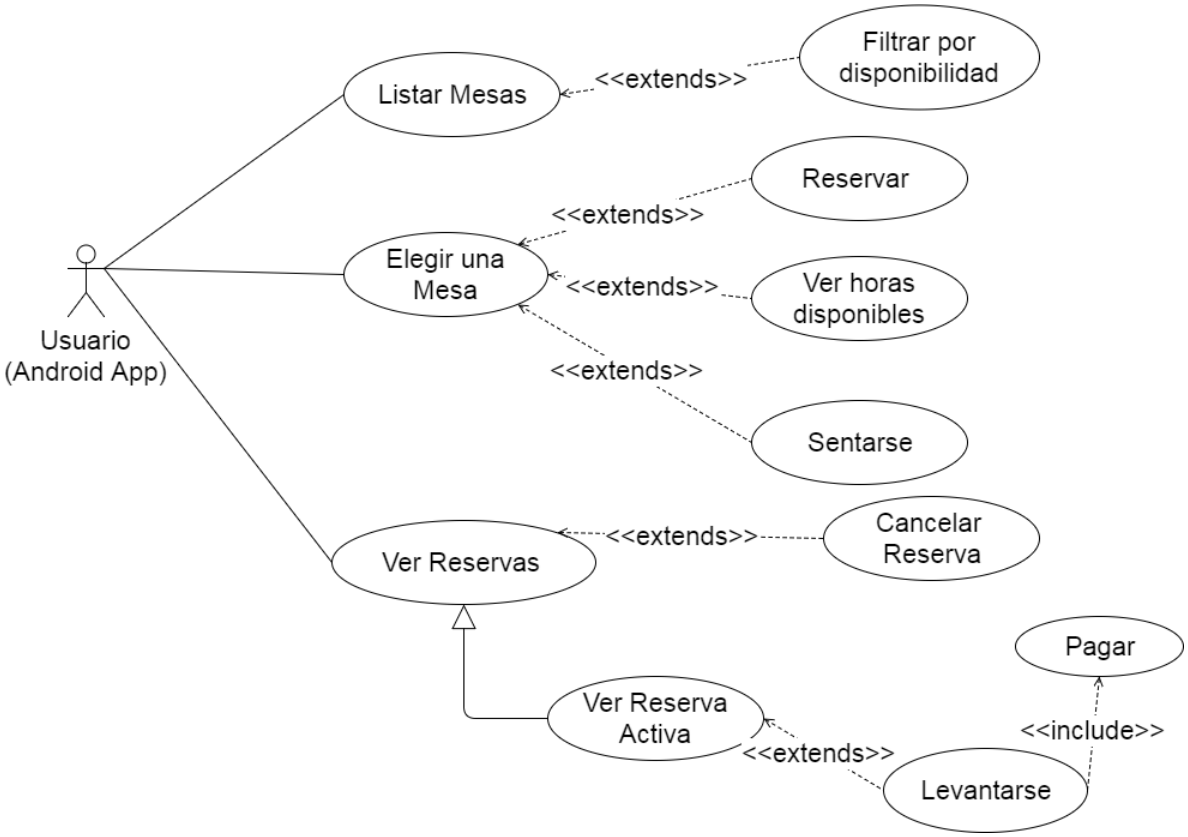

Figura 20. Diagrama de Casos de Uso del módulo de Reservas

5.2.4 MÓDULO DE PRODUCTOS

Este módulo está compuesto por el software encargado de gestionar la información de los productos del menú de cada restaurante, y las funcionalidades asociadas con ellos y los pedidos.

Este módulo permite al usuario listar los productos de cada restaurante, y, si está sentado en una mesa (tiene una reserva activa), puede añadir productos a un pedido para después confirmarlo y que el restaurante reciba la solicitud y pueda prepararlo. Adicionalmente, si el usuario tiene una reserva para una hora futura, podrá hacer un pedido asociado a esa reserva para que se le sirva a una hora determinada, en función de la hora de la reserva.

Estas funcionalidades se ilustran en el diagrama de casos de uso siguiente (Figura 21):

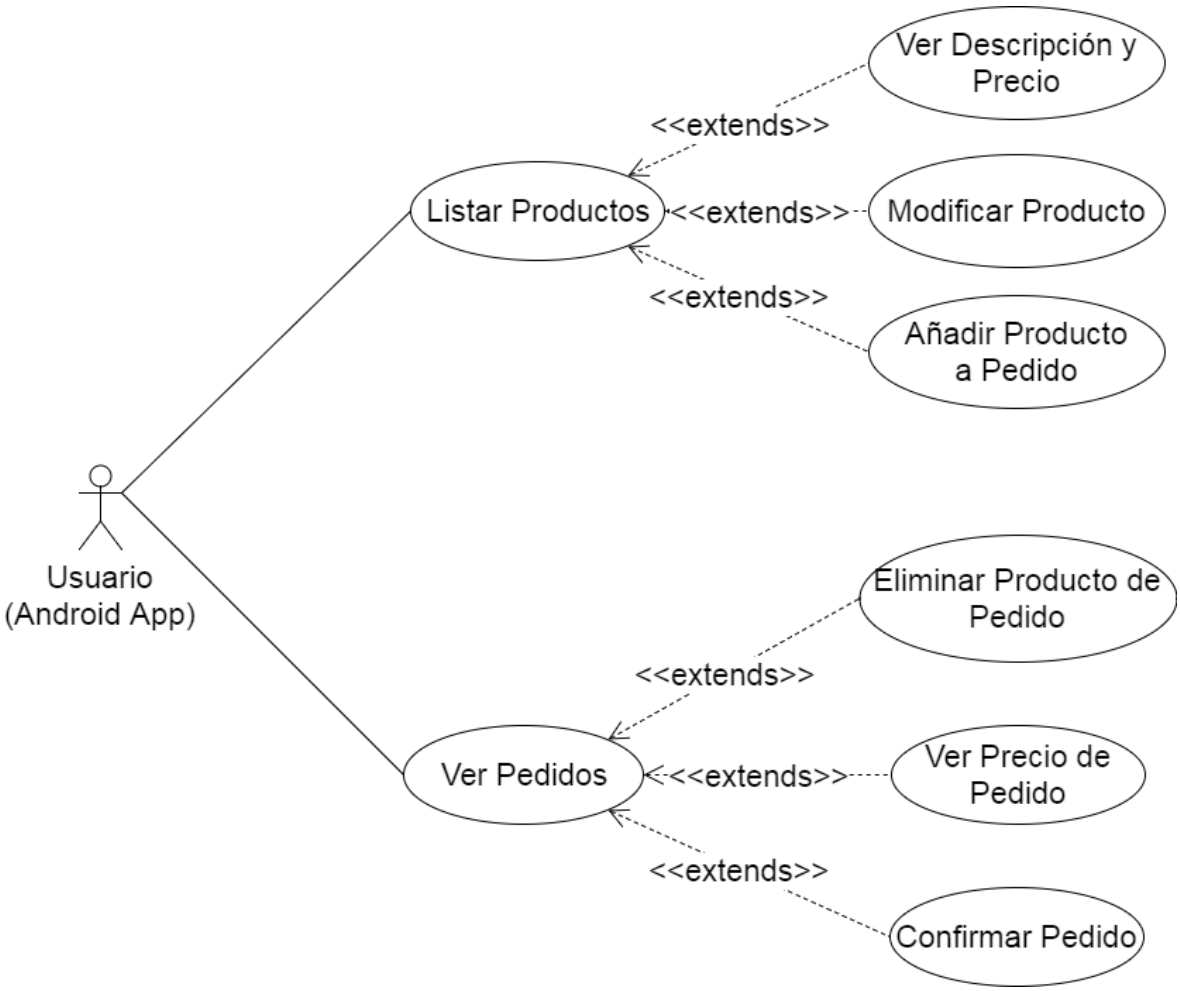

Figura 21. Diagrama de Casos de Uso de módulo de Productos

5.2.5 OPCIÓN DE USO TRADICIONAL

Adicionalmente, en cualquier momento el usuario dispondrá de una opción para notificar al restaurante de que se necesita de la presencia de un camarero en la mesa seleccionada, en caso de que haya cualquier problema o el cliente desee recibir el servicio de la forma tradicional a través del camarero.

5.3 IMPLEMENTACIÓN

5.3.1 MODELO VISTA CONTROLADOR

Para el diseño e implementación del servidor y de la lógica de negocio, se ha aplicado el patrón arquitectónico Modelo Vista Controlador (MVC). De esta forma, se separan el software en 3 componentes:

- Modelo: Representa la lógica de negocio y manipulación de la información (consultas, actualizaciones, etc.).
- Controlador: Es el encargado de responder a los eventos, e invoca al modelo cuando recibe solicitudes sobre la información. Es el *middleware* (software intermediario) entre el modelo y la vista.
- Vista: Es el encargado de presentar la información que recibe del modelo. Hace de interfaz entre el cliente y la plataforma.

Siguiendo este patrón, se han diseñado e implementado las clases. Estas clases se han dividido en paquetes que engloban los componentes antes mencionados:

- Servlets: Aquí se encuentran las clases que heredan de *Servlet*, y cumplen de Controlador. Gestionan las peticiones, las redirigen a los servicios necesarios y por último se encargan de la salida de los datos, ya sea para la aplicación o a un JSP.
- Dominio: En este paquete se engloban todas las clases que representan entidades con las que se van a trabajar en la lógica de negocio. Se considera parte del Modelo.
- DAO: Este paquete incluye clases que contienen funciones que hacen de interfaz con la base de datos. Se considera parte del Modelo.
- Servicios: En este paquete se incluyen los gestores que contienen la mayor parte de la lógica de negocio. Son los encargados de manipular los objetos de las clases de Dominio y acceder a la base de datos (a través de las clases DAO). Se considera parte del Modelo

En el siguiente diagrama (Figura 22) se explican las clases del paquete dominio y las relaciones que existen entre ellas:

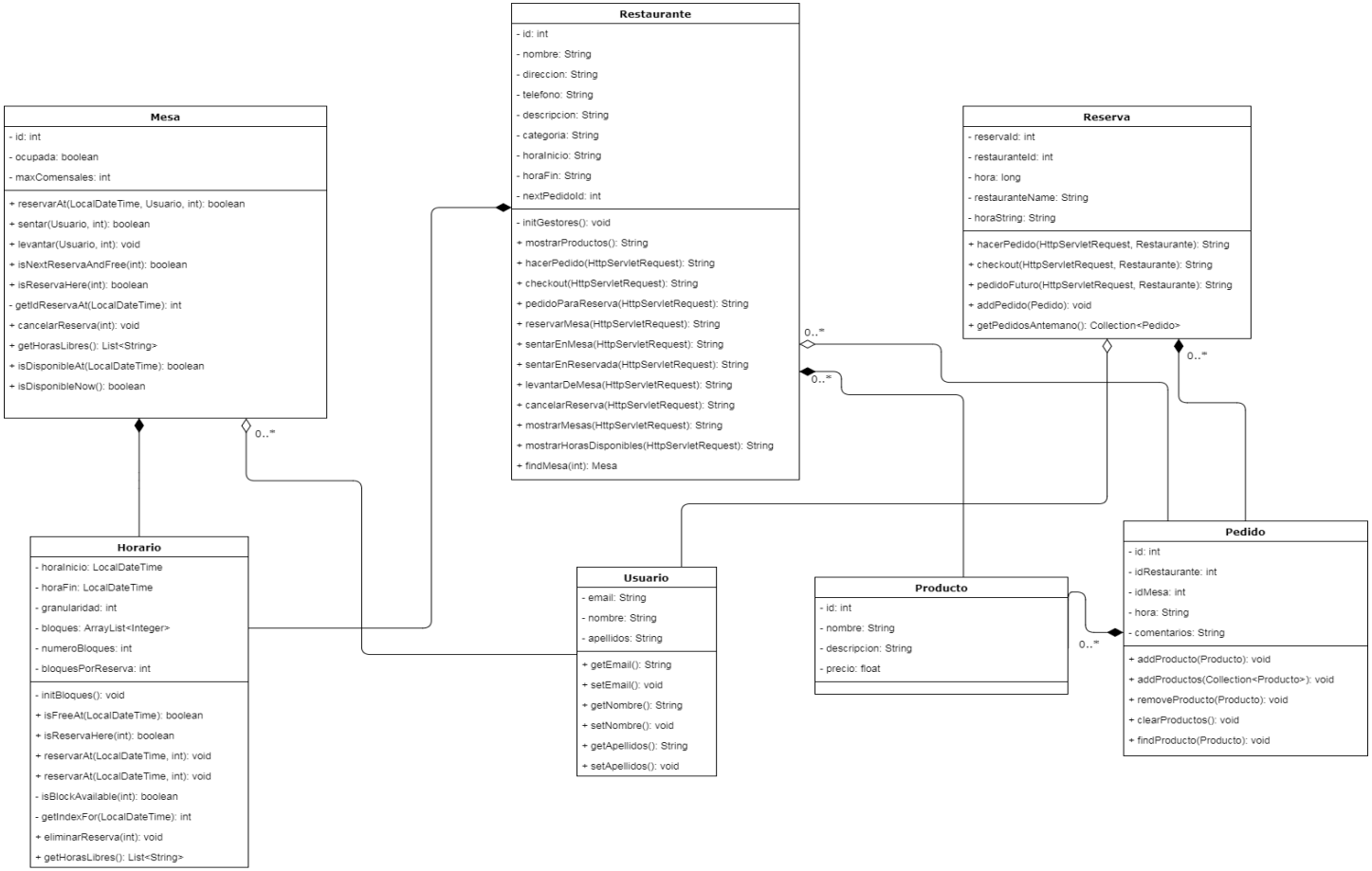

Figura 22.Diagrama de Clases del paquete Dominio

Además de las relaciones observables en la figura, la clase *Restaurante* contiene una instancia de la clase *GestorMesa* que contiene a su vez una lista de *Mesa* (las mesas del restaurante) y se encarga de gestionar las funcionalidades relacionadas con las mesas y sus reservas. La clase *Reserva* contiene una instancia de *GestorPedido* que se encarga de dar funcionalidades relacionadas con los *Pedidos* y el *checkout*.

En el lado del cliente, estas clases se han reutilizado, manteniendo sus relaciones, aunque se han omitido algunos atributos y métodos no utilizados, añadido métodos necesarios, e incluso se han eliminado clases enteras como *Horario* o *GestorMesa*, además del paquete de *servlets*.

Adicionalmente, existe un paquete dedicado al interfaz de usuario (*User Interface* o *UI*) que contiene las clases propias de Android dedicadas a mostrar la información. Estos se han agrupado en un paquete llamado *ui*.

5.3.2 CONSIDERACIONES DE ANDROID

Google ofrece una serie de interfaces de programación de aplicaciones (API) públicas para el desarrollo de aplicaciones en la plataforma Android. Para poder sacar el máximo partido de las ventajas que ofrece este sistema operativo, es necesario conocer cuál es la estructura interna, la organización de las clases, cómo diseñar una buena interfaz, interactuar con otras aplicaciones y servicios, etc. Toda esta información viene detallada en el portal oficial para desarrolladores de Android [11], y este ha sido el recurso básico de información durante la implementación.

En la siguiente figura se muestra un ejemplo del uso del portal (Figura 23). En este portal, la información se expone en forma de API con detalles sobre las clases, a la vez que hay tutoriales y códigos de ejemplo en los que uno puede basar sus clases.

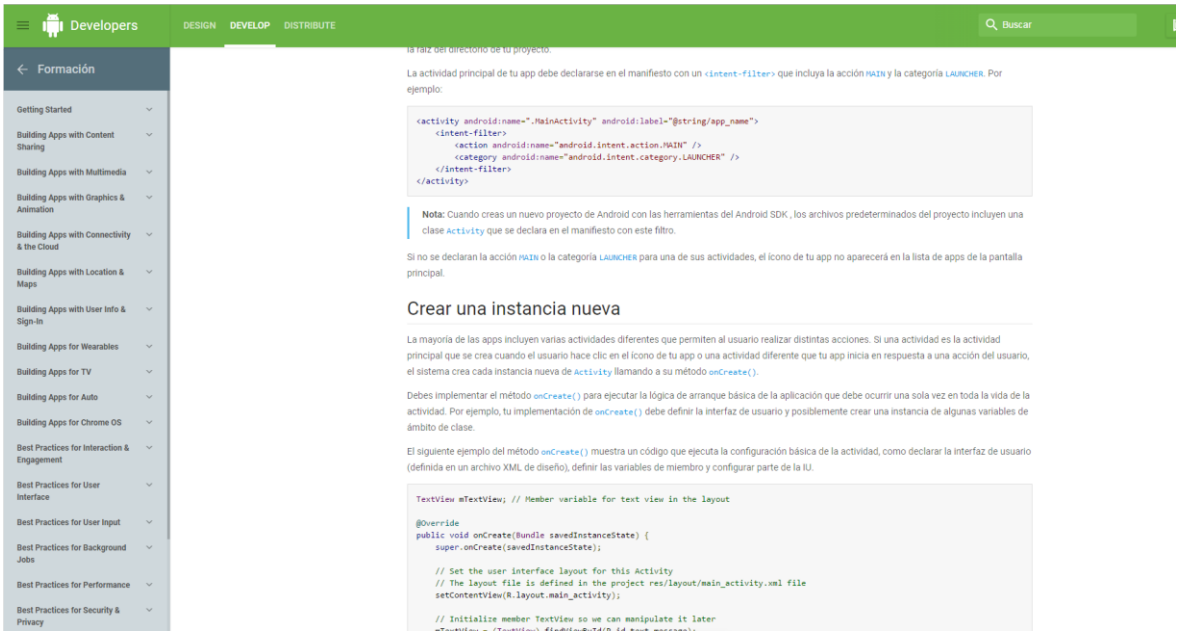

Figura 23. Ejemplo del Portal de Desarrolladores Android

5.3.2.1 Interfaz en Android: La clase Activity

En primer lugar, es necesario conocer la disposición básica de la interfaz de Android. En este sistema, la información se presenta en forma de "pantallas", representadas por la clase *Activity*. Esta es la unidad básica de interfaz y la base de cualquier aplicación Android. El sistema ofrece una gestión automática de creación, pausa, ocultación, muestra y destrucción de cada una de estas *Activities*. Este sistema viene explicado en el ciclo de vida de la *Activity*, mostrado en el siguiente esquema (Figura 24):

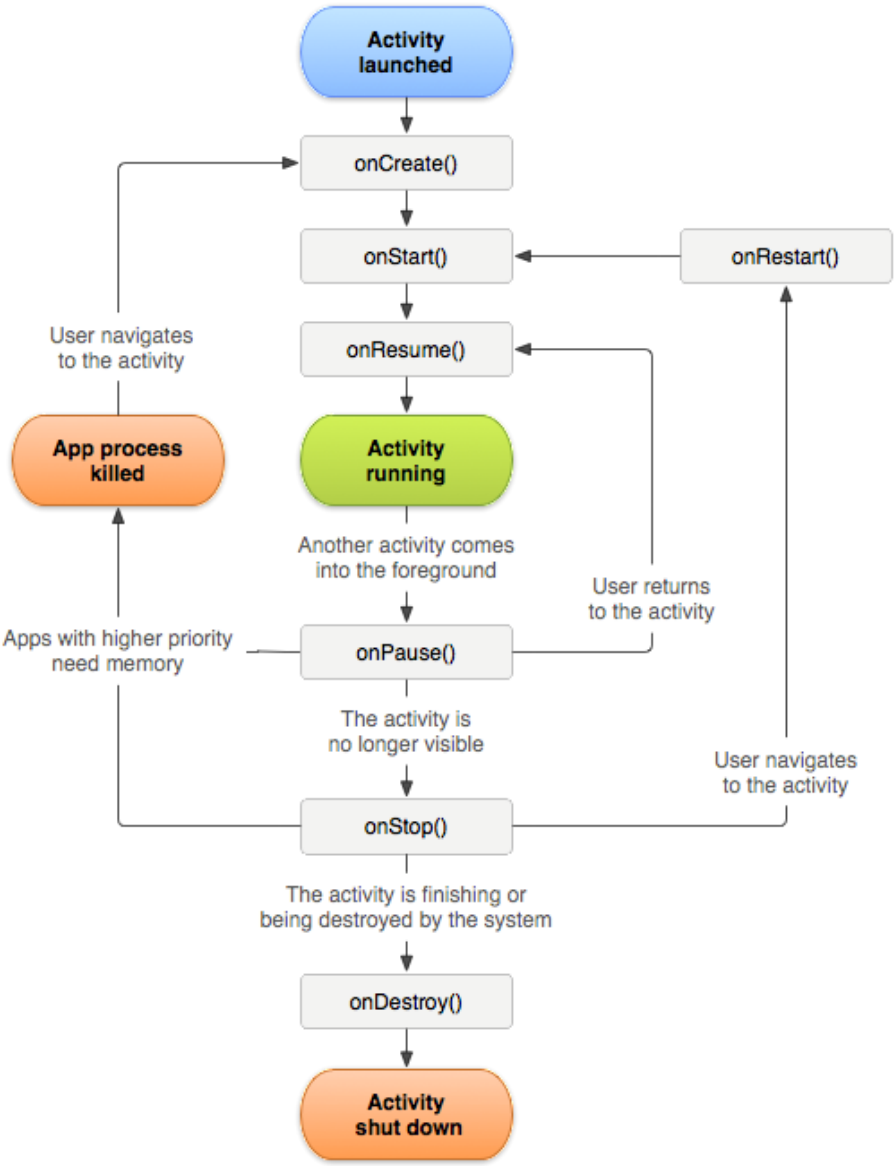

Figura 24. Ciclo de vida de una Activity

En función de las interacciones del usuario con la aplicación, nuevas instancias de *Activity* podrán ser creadas o eliminadas, y podrán pasar de un segundo plano a un primer plano. Cada una de estas acciones conlleva la llamada automática de una serie de métodos. En estos métodos es donde el desarrollador puede modificar e introducir código para conseguir el comportamiento deseado de la aplicación. Por ejemplo, el método *onCreate()* de la clase

"MainActivity", que es una clase que hereda de *Activity*, de la plataforma (que es el llamado automáticamente cuando la *Activity* se crea por primera vez) es el siguiente:

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //Asociar la barra superior
        Toolbar toolbar = (Toolbar) findViewById(R.id.my_toolbar);
        setSupportActionBar(toolbar);

        ActionBar actionBar = getSupportActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);

        TextView tvNombre = (TextView) findViewById(R.id.main_nombre);
        TextView tvDescripcion = (TextView) findViewById(R.id.main_descripcion);
        TextView tvDireccion= (TextView) findViewById(R.id.main_direccion);
        TextView tvTelefono = (TextView) findViewById(R.id.main_telefono);
        TextView tvHorario = (TextView) findViewById(R.id.main_horario);

        //Recuperar restaurante seleccionado
        Restaurante selectedRestaurante =
TFGApplication.getInstance().getSelectedRestaurante();
        if (selectedRestaurante == null){
            Log.e(TAG, "No hay un restaurante seleccionado");
            Toast.makeText(this, "No hay un restaurante seleccionado!",
Toast.LENGTH_LONG).show();
            Intent intent = new Intent(this, RestaurantesActivity.class);
            startActivity(intent);
        }else{
            setTitle(selectedRestaurante.getNombre());

            //Poblar con los datos del restaurante seleccionado
            tvNombre.setText(selectedRestaurante.getNombre());
            tvDescripcion.setText(selectedRestaurante.getDescripcion());
            tvDireccion.setText(selectedRestaurante.getDireccion());
            tvTelefono.setText("Teléfono : " +
selectedRestaurante.getTelefono());
            tvHorario.setText("Horario : " + selectedRestaurante.getHorario());

            //Ver si hay una mesa seleccionada
            Mesa selectedMesa = TFGApplication.getInstance().getSelectedMesa();
            if(selectedMesa != null){
                showSelectedMesa(selectedMesa);
            }
        }


    }
```

Éste es un código típico de *Activity* y la misma estructura se repite en la gran parte de las *Activities* que posee esta aplicación. En primer lugar, se llama al método de la superclase para asegurar que los métodos internos de Android se ejecutan, y se le asigna un "Layout". Este *Layout* se corresponde con un archivo XML que detalla los elementos de la interfaz que se tienen que mostrar y su disposición. Los siguientes métodos preparan y configuran la barra superior o "ActionBar", para darle uniformidad y funcionalidad extra a la aplicación.

El siguiente bloque se encarga de poblar los datos de la *Activity*. En este caso, esta pantalla es la vista principal de un restaurante, en la que se muestra su información asociada, así como una serie de botones de opciones para acceder a otras funcionalidades, como ver las mesas o ver el pedido. De esta forma, desde la misma *Activity*, se obtiene una referencia a los campos de texto que contendrán la información del restaurante, representados por la clase *TextView* y se modifica su contenido mediante el método *setText(String)*.

Además de la clase *Activity*, otras clases básicas utilizadas para el desarrollo de este proyecto son las anteriormente mencionadas *Layout* y *View* y la clase *Fragment*, entre otras. El *View* es el elemento básico de diseño. Cada botón, caja de texto, imagen, slider, o cualquier otro elemento del interfaz es una clase que hereda de *View*. Un ejemplo muy utilizado en esta aplicación es el *ListView*, que da un interfaz para mostrar una lista de elementos, con funcionalidad de deslizamiento y gestión de los elementos automatizada (mediante la clase *Adapter*).

La clase *Layout* permite agrupar una serie de *Views* u otros *Layouts*, pudiéndoles asignar una disposición por defecto, en función del *Layout* elegido. Clases de *Layout* muy utilizadas son:

- *LinearLayout*, que coloca los elementos uno encima de otro o uno al lado del otro (en función de la orientación elegida), y permite asignar pesos a los elementos para asignar el tamaño de forma dinámica.
- *RelativeLayout*, que permite colocar los elementos de forma relativa a otros elementos, como encima o debajo, alineados, etc.

La clase *Fragment* permite manipular conjuntos de *Layouts*, dándole un aspecto dinámico a la aplicación sin necesidad de cambiar de *Activiy*. Un ejemplo típico de uso es una vista de pestañas, en la que cada pestaña se corresponde con una instancia de *Fragment* distinta, que se ocultan y muestran en función de la acción del usuario. Esta clase posee su propio ciclo de vida, que permite su manipulación de forma análoga a *Activity*. En la aplicación se ha usado esta clase para mostrar cuadros de diálogo personalizados.

5.3.2.2 Comunicación entre elementos

Las aplicaciones Android no ofrecen por defecto una clase similar al contexto de aplicación en JavaEE. Para pasar información de una *Activity* a otra existen otros mecanismos, como por ejemplo los *Intent*. Al llamar a una nueva aplicación se crea una instancia de *Intent*, a la que se pueden asociar si se desea objetos en forma de *Extra* (que es un *Map*, con estructura clave-valor) que pueden ser recuperados por la clase de destino. En el siguiente *snippet* se muestra un ejemplo de este uso en la aplicación.

Por un lado, en la clase *MenuActivity*, encargada de listar los productos, cuando se selecciona un producto, se quiere mostrar la clase *DescriptionActivity*, que muestra una descripción del producto y da la opción de añadirlo al pedido. Para esto, la clase *DescriptionActivity* necesita conocer cuál ha sido el *Producto* seleccionado, y esto se consigue a través de un *Intent*:

```
Intent intent = new Intent(mContext, DescriptionActivity.class);
intent.putExtra(ProductoAdapter.ID, producto.getId());
if(TFGApplication.getInstance().getReservaFutura() != null){
    intent.putExtra("isTemporal", true);
}
mContext.startActivity(intent);
```

Por su parte, en *DescriptionActivity*, en el método *OnCreate()*, se encuentra el siguiente *snippet*:

```
isTemporal = getIntent().getBooleanExtra("isTemporal", false);
int id = getIntent().getExtras().getInt(ProductoAdapter.ID);
```

A partir de este momento, *DescriptionActivity* conoce cuál es el ID del producto seleccionado, y puede proceder a solicitar los datos del producto y mostrarlos.

Sin embargo, en muchas situaciones un *Intent* no es suficiente para compartir comunicar de forma efectiva a las *Activities*. En determinadas situaciones es necesario tener una zona de memoria compartida a la que puedan acceder las clases en cualquier momento para leer determinada información, similar a *ServletContext* en JavaEE.

En esta aplicación, elementos como el restaurante seleccionado, el con los productos seleccionados, las reservas son datos que se necesitan en numerosas ocasiones, en situaciones en las que la navegabilidad no siempre es la misma. Por ejemplo, el carrito está disponible en la mayoría de las *Activities* de la aplicación mediante un botón en la *ActionBar*.

Solucionar este problema, existe una clase que hace las veces de contexto de aplicación para Android: la clase *Application*. Esta clase por defecto no ofrece esta opción, pero es posible crear una clase que herede de la misma, en este caso *TFGApplication*, que se inicializa en el momento en el que se abre la aplicación. Esta clase es accesible desde cualquier elemento de la aplicación, se le pueden añadir atributos, y mediante *Getters&Setters* se puede acceder a la información. De la misma forma se pueden crear métodos de uso general que podrán ser accesibles en cualquier momento.

5.3.2.3 Persistencia de Datos: Las Preferencias

Uno de los objetivos de Android es el ahorro de batería. De esta forma, el sistema de forma automática manipula, esconde y puede incluso cerrar aplicaciones de forma automática, lo que puede llevar a resultados inesperados para el desarrollador. Uno de los efectos de este sistema es que la aplicación puede ser un cierre inesperado de la aplicación, lo que conllevaría la pérdida de los datos guardados en el contexto de aplicación antes de lo esperado.

Android ofrece dos opciones principales para dar persistencia a los datos que sean necesarios:

- Las bases de datos SQLite: Android tiene integrada una API para gestionar bases de datos SQL mediante SQLite. Esta es la opción elegida cuando es necesario persistir grandes cantidades de información estructurada.
- Archivos de preferencias: Estos son archivos de texto accesibles desde clases como *Activity* o *Application*, que permiten guardar datos en forma de clave-valor. Así, si es poca la información que es necesario persistir, se pueden utilizar las preferencias (que pueden ser públicas para que otras aplicaciones puedan acceder a los datos), pudiendo guardar objetos enteros de forma sencilla gracias al formato JSON.

La opción elegida para esta aplicación ha sido utilizar las preferencias, debido a que la información para persistir es reducida, además la rapidez y sencillez que estas ofrecen sobre las bases de datos SQLite.

En la aplicación se ha combinado el uso de un contexto de aplicación y de las preferencias, de forma que los datos que se guardan en el contexto y que no son temporales, como el restaurante que estaba seleccionado o la reserva activa, se guardan en preferencias al mismo tiempo que se guardan en el contexto, y cuando se terminan de usar se borran de las preferencias. En el siguiente *snippet* se muestra un ejemplo del uso de las preferencias en *TFGApplication*.

```
public void setSelectedRestaurante(Restaurante restaurante){
    selectedRestaurante = restaurante;
    menu.clear();

    //Guardar en preferencias
    SharedPreferences sharedPreferences =
getSharedPreferences(getString(R.string.preference_file_key),
Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    Gson gson = new Gson();
    editor.putString(getString(R.string.selected_restaurante),
gson.toJson(selectedRestaurante));
    editor.commit();
}

public Restaurante getSelectedRestaurante(){
    if(selectedRestaurante != null)
        return selectedRestaurante;

    //Si llegamos aqui es que no hay un restaurante inicializado, comprobar si
```

```
esta en las preferencias
    SharedPreferences sharedPreferences =
getSharedPreferences(getString(R.string.preference_file_key),
Context.MODE_PRIVATE);
    String toParse =
sharedPreferences.getString(getString(R.string.selected_restaurante), null);
    if(toParse == null){
        //No hay un restaurante seleccionado
        return null;
    }
    JsonObject jsonObject = new JsonParser().parse(toParse).getAsJsonObject();
    Gson gson = new Gson();
    selectedRestaurante = gson.fromJson(jsonObject, Restaurante.class);

    return selectedRestaurante;

}

public void clearSelectedRestaurante(){

    //Cuando se deseleccione o se termine el pago, se llama a esta funcion para
salir del restaurante
    SharedPreferences sharedPreferences =
getSharedPreferences(getString(R.string.preference_file_key),
Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.remove(getString(R.string.selected_restaurante));
    editor.commit();

    menu.clear();
    selectedRestaurante = null;
}
```

Aquí se observan que el método *setSelectedRestaurante()*, además de modificar el atributo, guarda en las preferencias el nuevo valor del restaurante. Por su parte, cuando se llama al método *getSelectedRestaurante()*, se comprueba si existe un valor en el contexto de aplicación, y si no, se comprueba si existe en las preferencias antes de devolver un *null*. Esta misma estructura se repite para otros atributos de la clase *TFGApplication*, como la mesa seleccionada, la reserva activa o el carrito.

5.3.2.4 La comunicación con el Servidor: HTTP y Volley

La comunicación entre el cliente y el servidor se realiza mediante el protocolo HTTP, utilizando los métodos GET y POST para las peticiones. Android ofrece un mecanismo para

gestionar estas peticiones, mediante la clase *HttpURLConnection*. Sin embargo, este método exige la gestión de las respuestas, de colas y el resto de elementos. Además, bloquea la ejecución del programa, algo que en Android está desaconsejado.

Como respuesta, Google hizo pública una librería para Android encargada de gestionar las peticiones HTTP de forma asíncrona, automática y transparente al desarrollador. El funcionamiento estándar de esta librería es el deseable en la mayoría de las ocasiones, y para casos específicos es posible modificarlo. Además, su utilización e integración en el proyecto es muy sencilla.

Para utilizarla, simplemente hay que añadir la librería a las dependencias, para que sea compilada con el proyecto. Esto se consigue añadiendo una línea en las dependencias del archivo *Gradle* del módulo *app* del proyecto Android:

```
dependencies {
        ...
        compile 'com.android.volley:volley:1.0.0'

}
```

Volley ofrece una serie de objetos que hacen de interfaz con las peticiones HTTP, y permite asignar una función de *callback* en el caso de éxito o de error. Así, puedes gestionar de forma asíncrona las peticiones sin necesidad de bloquear la ejecución del programa.

El objeto básico es la *RequestQueue*. Este objeto es capaz de gestionar múltiples peticiones al mismo tiempo. Cuando se desea hacer una petición, se prepara una *Request*, y cuando está lista se añade a la *RequestQueue*, y a partir de ese momento, el sistema operativo se encarga del resto de la comunicación de forma transparente hasta el momento de la *Response* (el proceso se detalla en la Figura 25). Para una sincronización correcta, se recomienda que únicamente se usa una *RequestQueue* en toda la aplicación. Para ello, se ha aprovechado la clase *TFGApplication*, que será la encargada de crearla y mantenerla. *TFGApplication* contiene los siguientes métodos para ello:

```
public RequestQueue getRequestQueue(){
    if(mRequestQueue == null){
        CookieManager manager = new CookieManager();
        CookieHandler.setDefault( manager  );
        mRequestQueue = Volley.newRequestQueue(getApplicationContext());
    }


    return mRequestQueue;
}

public <T> void addToRequestQueue(Request<T> request){
    getRequestQueue().add(request);
}
```

De esta forma, desde cualquier punto de la aplicación se puede añadir una *Request* a la cola para ser enviada.

Las peticiones se configuran y crean a través de la clase *Request*, que es el objeto que hace de *wrapper* para la petición HTTP. A la *Request* se le especifica la URL de destino, el método (GET, POST, etc) pudiendo añadir parámetros a través de un *Map* si se desea o si se está trabajando con una petición POST, gestionar las *Cookies*, etc.

Existen distintos tipos de *Request* que se pueden elegir en función del tipo de respuesta (*Response*) que se espera recibir. En el desarrollo de este proyecto se han utilizado 2:

- *StringRequest*: Esta es la *Request* que espera como resultado un *String* cualquiera, y una vez procesada la *Response* se recibe un objeto de este tipo.
- *JsonArrayRequest*: Esta es la *Request* que espera como resultado un *JsonArray*. En este proyecto, peticiones como listar productos, generan como respuesta un *Array* de objetos, transmitidos en formato JSON, y esta clase ofrece funciones incluidas que facilita su manipulación.

Cuando se crea una *Request*, además de especificarle las opciones, es necesario definir cuáles son las funciones *callback*, que son funciones que se ejecutarán automáticamente en caso de éxito de la *Request*, o en caso de *Error*. Un ejemplo del uso de la *Request* se muestra en el

siguiente *snippet*, que se encarga de hacer una petición HTTP para listar los productos de un restaurante seleccionado:

```
public void populateMenu(final ProductoAdapter adapter, final ArrayList<Producto>
menu){
    //Para probar la 10.0.2.2 redirige del emulador al host
    String serverAddress = getString(R.string.server_address);
    String url= serverAddress +
"Pedidos?opcion=mostrarProductos&selectedRestauranteId="+
selectedRestaurante.getId();
    JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(url, new
Response.Listener<JSONArray>() {
        @Override
        public void onResponse(JSONArray response) {
            TreeSet<Producto> savedMenu = (TreeSet<Producto>)
TFGApplication.getInstance().getMenu();
            for (int i = 0; i < response.length(); i++) {
                try {
                    JSONObject jsonObject = response.getJSONObject(i);
                    savedMenu.add(new Producto(jsonObject));
                } catch (JSONException e) {
                    e.printStackTrace();
                }
                TFGApplication.saveMenu(savedMenu);
                //Since we don't know what values are we getting from each
request (although it should be all of them, reload the whole menu)
                menu.clear();
                //Since everything is handled using a TreeSet, we can rest
assured there are no duplicates and they are ordered
                menu.addAll(savedMenu);
                adapter.notifyDataSetChanged();
                Log.d(TAG, "Poblando el list");

            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Log.e(TAG, "Error en la Response");
            Log.d(TAG, "Response : " + error);
        }
    });
    TFGApplication.getInstance().addToRequestQueue(jsonArrayRequest);
}
```

En este código se utiliza una *JsonArrayRequest*, y se espera una lista de *Producto* como resultado. En el constructor de la *Request* se encuentra la URL, seguida de una función de *callback* de éxito, y de otra de error. En la de éxito, se aprovechan las funciones de

JsonArrayRequest para transformar el *JsonArray* recibido en un *TreeSet* con el que manipular el resultado en la *Activity*.

Por último, se puede observar que la última línea de código tras crear la *Request* es añadirla a la cola, y esto es suficiente para que el sistema gestione la petición y respuesta automáticamente de forma transparente. El ciclo de vida de la *Request* se explica en el blog de desarrolladores con la siguiente figura (Figura 25):

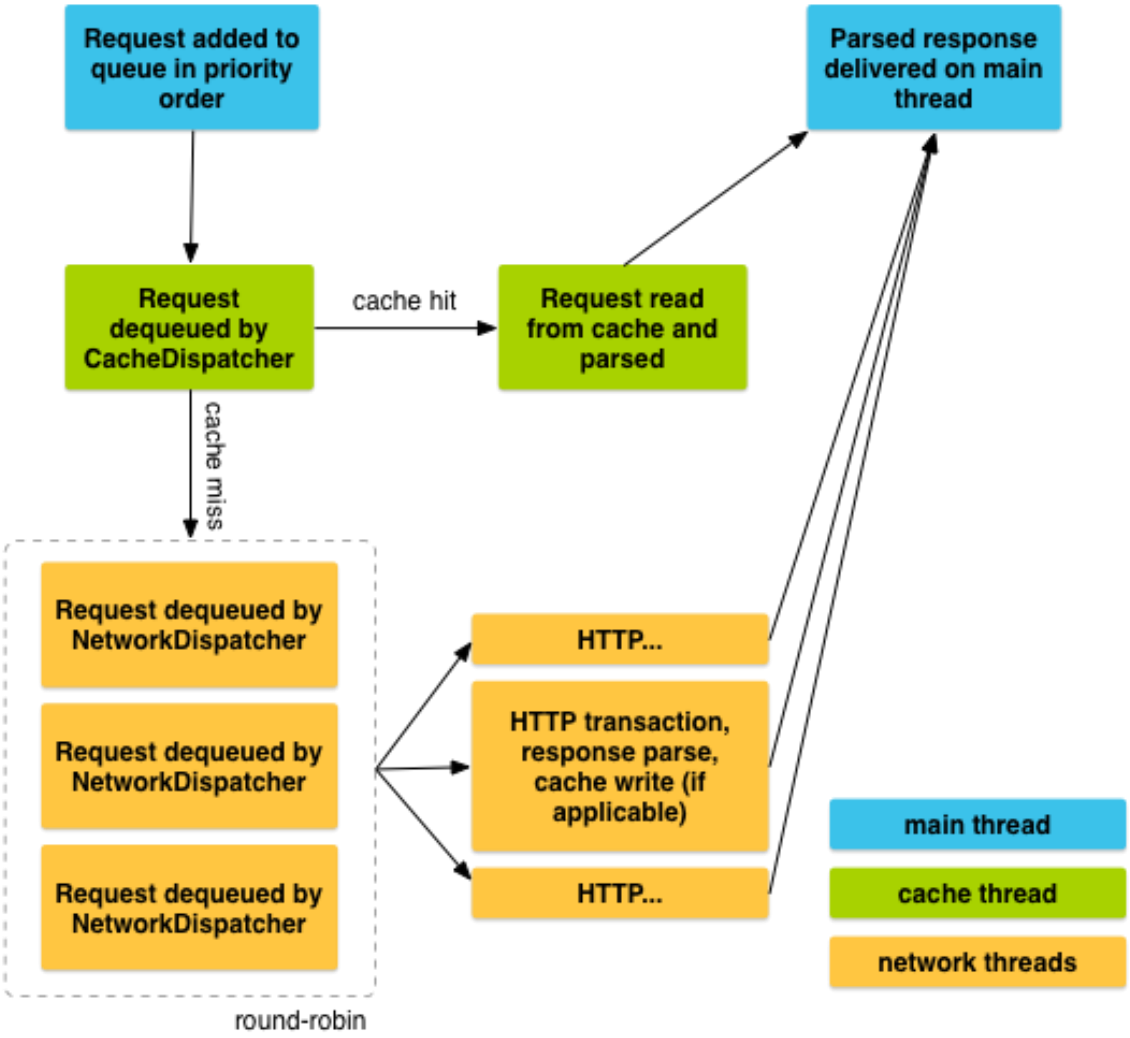

Figura 25. Ciclo de vida de la Request en Volley

Otra consideración a tener en cuenta para la correcta comunicación entre el servidor y el cliente es el mantenimiento de la sesión. Mantener la sesión entre el cliente y el servidor es necesario para distinguir un cliente con la sesión iniciada en un dispositivo, de otro, por ejemplo.

Esto en servidores basados en tecnología Java EE, como es el caso de Apache Tomcat 8, se consigue mediante una *Cookie* llamada JSESSIONID, que es un identificador de sesión asignado por el servidor a cada cliente, que este añade en todas sus futuras peticiones para poder identificarlo.

Sin embargo, *Volley* por defecto no guarda las *Cookies*, y es necesario configurarlas en el momento de la creación de la *RequestQueue*. Para ello se añade a la cola un *CookieManager*, que se encargará de añadir estas *Cookies* a las *Request* de forma automática. El siguiente *snippet* se encarga de asignar el *CookieManager* en el momento de la creación de la cola:

```
if(mRequestQueue == null){
        CookieManager manager = new CookieManager();
        CookieHandler.setDefault( manager  );
        mRequestQueue = Volley.newRequestQueue(getApplicationContext());
    }
```

A partir de este momento, la JSESSIONID será mantenida a lo largo de una conexión con el servidor.

5.3.2.5 JSON y GSON

La tecnología JSON permite describir objetos a través de texto de forma muy ligera y eficiente. Esta tecnología se ha utilizado para enviar datos entre el cliente y el servidor a través de peticiones HTTP.

JSON es un estándar muy extendido, y existen numerosas librerías que automatizan el proceso y lo hacen transparente al desarrollador. Una de las más utilizadas, y la elegida para este proyecto, es GSON, publicada por Google, e incluida en Android. Para utilizarla, hay que añadirla a las dependencias de *Gradle*, de la siguiente forma:

```
dependencies {
        ...
        compile 'com.google.code.gson:gson:2.8.0'

}
```

Una vez incluida, se puede utilizar desde cualquier clase del proyecto. Un ejemplo de su uso es el siguiente:

```
Gson gson = new Gson();
String result = gson.toJson(this.productos, TreeSet.class);
```

En este código se ve lo extremadamente sencillo y rápido que es convertir una lista entera de objetos (en este caso, "productos" es un *TreeSet* de la clase *Producto*), en un String con formato JSON. Deshacer el cambio en el receptor es igual de sencillo. Por ejemplo, para guardar y leer objetos en las preferencias se ha utilizado también el formato JSON, y en la clase *TFGApplication*, cuando se lee la reserva activa de las preferencias, se traduce el JSON en un objeto *Reserva* de la siguiente manera:

```
Gson gson = new Gson();
reservaActiva =  gson.fromJson(toParse, Reserva.class);
```

El objeto "reservaActiva" es un atributo del tipo *Reserva* listo para ser utilizado.

5.3.2.6 Android y la compatibilidad con versiones anteriores

Una de las realidades con las que un desarrollador de Android tiene que lidiar es que en el mercado conviven dispositivos Android de múltiples versiones. El sistema operativo Android ha evolucionado desde su lanzamiento en 2008, y a lo largo de las actualizaciones numerosas funcionalidades se han ido añadiendo, y no todos los dispositivos se han adaptado.

Por eso, en el momento de empezar a desarrollar, es necesario decidir dos parámetros:

- la versión de SDK objetivo: esta es la versión de Android para la cual la aplicación estará optimizada.
- la versión de SDK mínima: esta la versión mínima que será necesaria para la ejecución de la aplicación, aunque no se asegura la disponibilidad de todas las funciones.

Para este proyecto, se ha elegido la versión mínima 16 (Jelly Bean 4.1), y la versión 25 como versión objetivo (Nougat 7.1) (ver Tabla 3). Así, daremos soporte al 98,6% de los dispositivos Android en uso hoy en día. En la siguiente figura podemos ver la distribución de las versiones (Figura 26):

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.8%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.8%
4.1.x	Jelly Bean	16	3.1%
4.2.x		17	4.4%
4.3		18	1.3%
4.4	KitKat	19	18.1%
5.0	Lollipop	21	8.2%
5.1		22	22.6%
6.0	Marshmallow	23	31.2%
7.0	Nougat	24	8.9%
7.1		25	0.6%

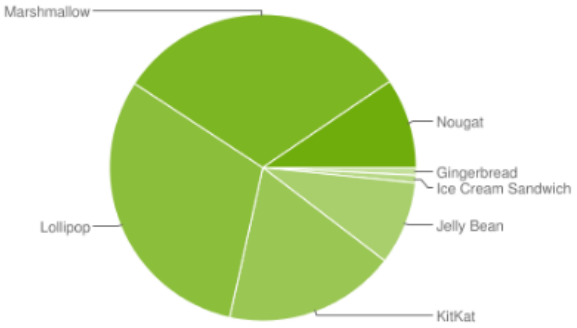

Figura 26. Distribución de versiones de Android. Fuente: Google Developers

Para dar soporte, primero se deben especificar las versiones en Android Studio, en el archivo *Gradle* de la *app*, de la siguiente forma:

```
android {
    ...

    compileSdkVersion 25
```

```
    buildToolsVersion "25.0.2"
    defaultConfig {
        applicationId "com.alvaro.tfg"
        minSdkVersion 16
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner
"android.support.test.runner.AndroidJUnitRunner"
    }
    ...
}
```

De esta forma, el IDE detecta automáticamente cuándo se utiliza una funcionalidad que no está disponible en las versiones especificadas, y ofrece alternativas compatibles.

Por otra parte, existen numerosas clases, como por ejemplo la clase *Activity*, que ha ido evolucionando con las versiones, y no son compatibles de forma automática. Para solucionar esto, Google ha publicado las bibliotecas de compatibilidad, con versiones compatibles de las clases más importantes. Un ejemplo de esto es la clase *AppCompatActivity*, que es la versión compatible de la clase *Activity*. Estas clases han sido las utilizadas en el desarrollo de este proyecto.

Para poder utilizar estas clases, es necesario añadir incluir las librerías en las dependencias, añadiéndolas al archivo *Gradle* de la siguiente forma:

```
dependencies {
    ...
    compile 'com.android.support:appcompat-v7:25.3.1'
    compile 'com.android.support:support-v4:25.3.1'
    compile 'com.android.support:design:25.3.1'
}
```

5.3.2.7 Material Design y Compatibilidad

En el diseño gráfico de aplicaciones es recomendable seguir unas pautas comunes, para conseguir una experiencia del usuario fluida y familiar. Así, se busca que la navegación y usabilidad del usuario sea sencilla, mediante elementos típicos como la barra superior de opciones (*AppBar o Toobar)*, estéticas similares, mismo diseño de botones y links, etc.

Para Android, Google ofrece el tema *Material Design*, que ofrece una serie de estilos, características, componentes, animaciones y otros recursos gráficos que se recomienda utilizar en el interfaz de la aplicación.

Sin embargo, este tema no es compatible con versiones anteriores a la versión de SDK 21 de Android (más detalles en la Tabla 3). Así, dos objetivos entran en conflicto:

- Por un lado, se busca compatibilidad, intentando llegar a dispositivos con versiones a partir de la versión 16.
- Por otro lado, se busca una experiencia de usuario fluida, por lo que interesa utilizar estilos procedentes del tema *Material Design*.

Para solucionar esto, existen numerosas librerías *Open Source* que implementan de forma manual los elementos gráficos que pueden ser compatibles, como son estilos de botones, algunas animaciones, etc.

En este caso, se ha utilizado la librería "Material Design Library" creada por el usuario *navasmdc*, y modificada por *vajro* [12], que introduce estos estilos en el proyecto Android. Se ha añadido de la siguiente forma, a través de *Gradle*:

```
repositories {
    maven { url "https://jitpack.io" }
}

dependencies {
    ...
    compile 'com.github.vajro:MaterialDesignLibrary:1.6'
}
```

5.3.3 MÓDULO DE USUARIOS

Para dar funcionalidad a este módulo se ha creado una clase perteneciente al paquete servicios, llamada *GestorUsuario*, que contiene los procesos de autenticación y desautenticación. A continuación, se detalla el funcionamiento de estos procesos.

5.3.3.1 Crear usuario (registrarse)

El usuario debe poder crear un usuario que lo identifique dentro de la plataforma, dando como datos una dirección de correo electrónico que no esté ya registrada en la base de datos de la plataforma, junto con un nombre y apellidos.

Para registrarse, el usuario tiene dos opciones:

- Registro a través del sistema de autenticación interno de la plataforma, que necesita de un usuario y además de una contraseña que será la tupla de datos necesaria para identificarle
- Registro a través del sistema de autenticación OAuth2.0 de Google. Este sistema es más seguro, ya que utiliza sistemas de encriptado para la comunicación entre el cliente y el servidor de autenticación, y evita compartir contraseñas con la plataforma. Con este sistema, únicamente la cuenta de Google del usuario es suficiente para identificarle, junto con el token de verificación que genera OAuth2.0 que se puede procesar utilizando la API de Google para extraer información de la cuenta en función de los permisos otorgados por el cliente.

En el siguiente diagrama de secuencia (Figura 27) se muestra cómo es el flujo de datos en un registro utilizando el método de OAuth2.0 en el que todo funciona correctamente (datos de inicio de sesión de Google correctos y la cuenta no está ya registrada):

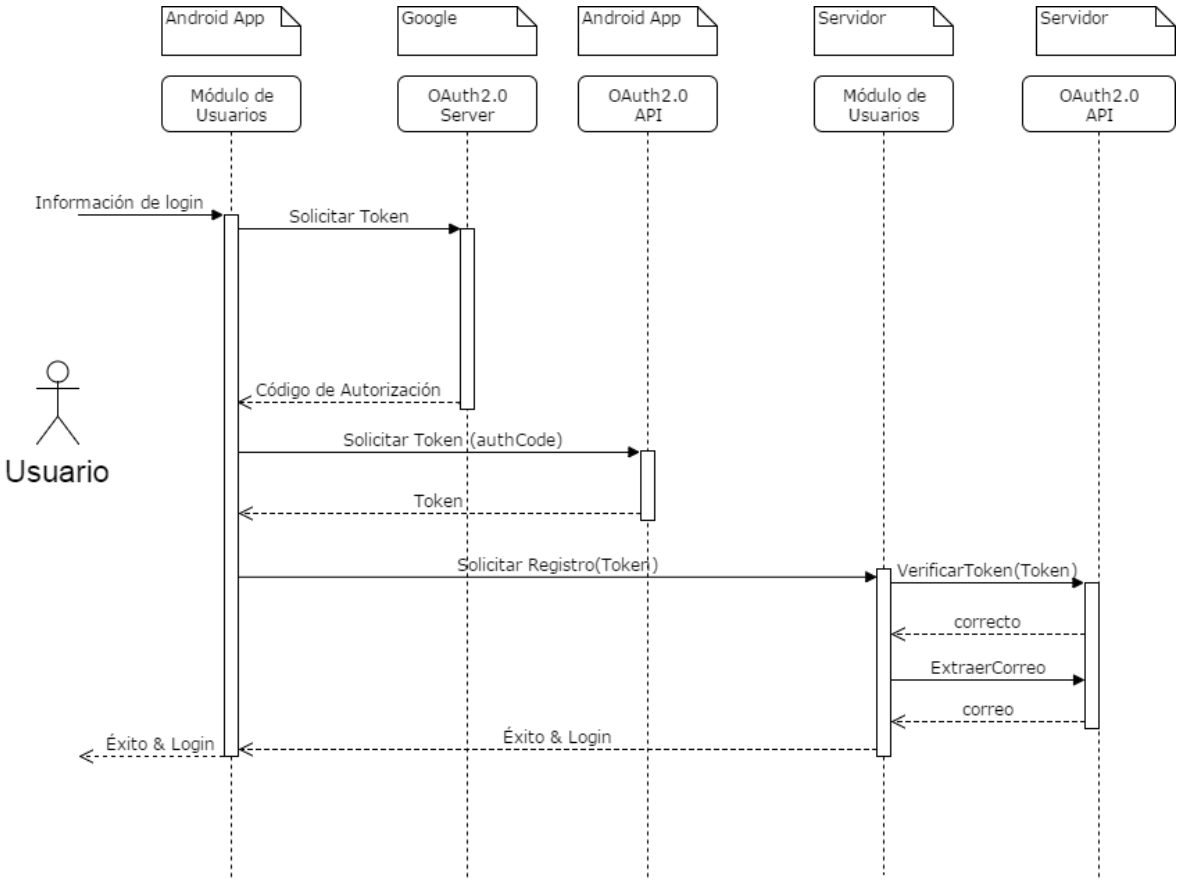

Figura 27. Diagrama de Secuencia de Registro mediante OAuth2.0

5.3.3.2 Iniciar Sesión

El usuario debe poder identificarse utilizando la dirección de correo elegida en el proceso de registro, y su contraseña correspondiente. Al igual que en el apartado anterior, es posible utilizar el método de Auth2.0 para el inicio de sesión, en cuyo caso el correo será la cuenta de Google elegida durante el registro y la contraseña la correspondiente a dicha cuenta.

El proceso de identificación a través de OAuth2.0 es similar al explicado en la Figura 27, y el flujo de datos se muestra en el diagrama siguiente (Figura 28):

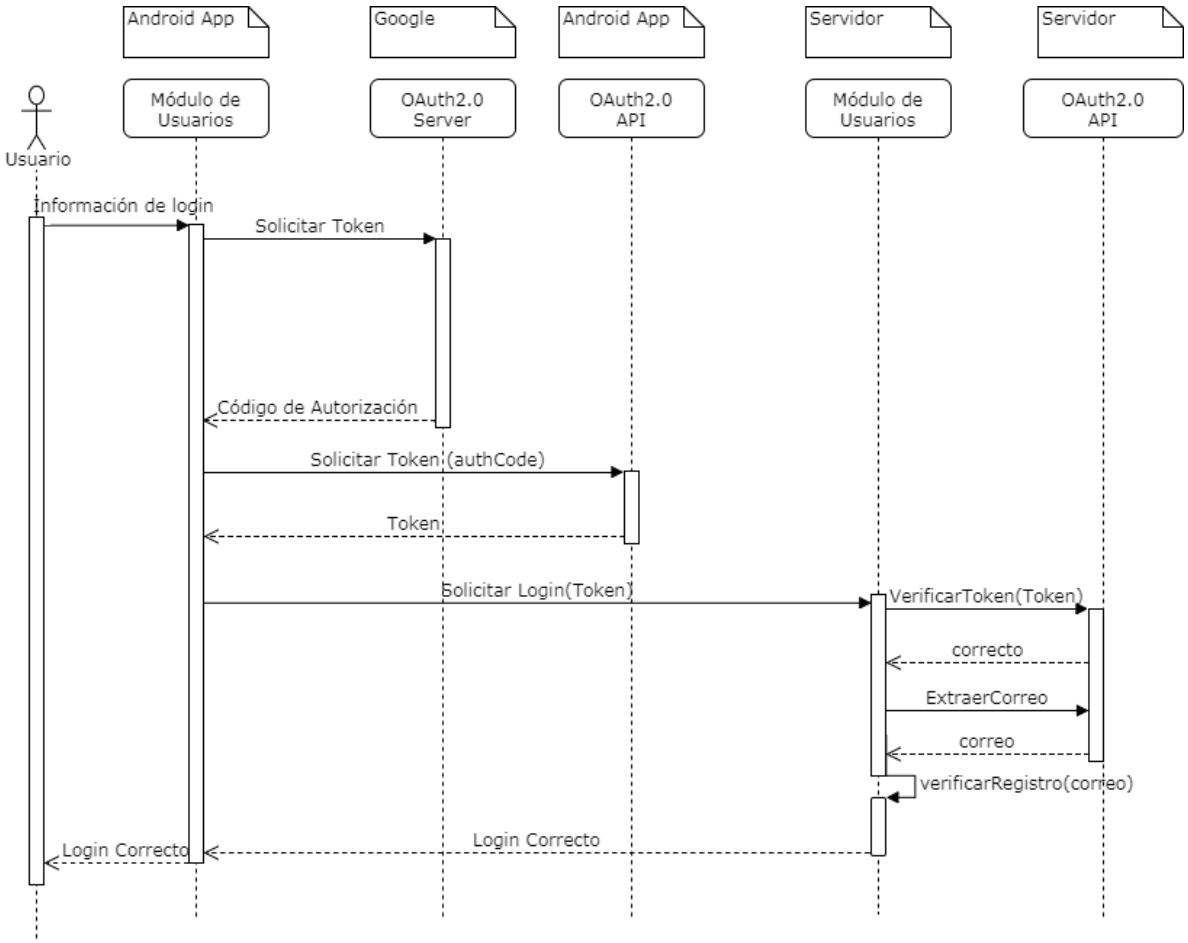

Figura 28. Diagrama de Secuencia de Login mediante OAuth2.0

Una vez iniciada la sesión, el usuario podrá acceder al resto de servicios ofrecidos por la plataforma.

5.3.3.3 Cerrar Sesión

El usuario podrá cerrar sesión en cualquier momento. Para ello se le muestra una opción en el menú superior de opciones desplegables, y ésta manda una solicitud al servidor de cierre de sesión. En caso de éxito, la sesión se cierra y se reinician los datos locales en la aplicación.

5.3.3.4 La clase EstadoLogin

Uno de los objetivos del sistema es ofrecer una experiencia fluida, incluso si se cambia de dispositivo. Para conseguir esto, el sistema mantiene el estado de las reservas del cliente conforme las va confirmando con el restaurante.

Cuando el cliente inicia sesión correctamente, el sistema devuelve como resultado una instancia de la clase *EstadoLogin*, cuyos detalles se explican en la siguiente figura (Figura 29. Diagrama de clase de EstadoLogin):

Figura 29. Diagrama de clase de EstadoLogin

Con este objeto, la aplicación del cliente puede conocer el estado inicial de la aplicación, y, si existe una reserva activa, puede cargar directamente la pantalla principal del restaurante

(*MainActivity)*, con la información del restaurante y mesa seleccionados, productos que se hayan confirmado ya, junto con otras reservas que haya hecho el usuario.

5.3.4 MÓDULO DE RESTAURANTES

Esta es el primer módulo al que accede el usuario una vez ha iniciado sesión. Además, el resto de módulos se apoyarán en este, ya que es el encargado de recuperar el restaurante mediante una id. Para ello utiliza el método estático de la clase abstracta *GestorRestaurante*:

```
public static Restaurante findRestaurante(int id, HttpServletRequest request)
```

Este método devuelve una instancia de la clase *Restaurante*¸ y lo consigue accediendo a la base de datos mediante la clase *RestauranteDAO* si no existe, o usando un *HashMap<Integer, Restaurante>* que se utiliza para cachear los restaurantes extraídos de la base de datos.

Además, es el encargado de servir las funcionalidades de listar restaurantes, ver la información y seleccionarlos. Estos se explican más en detalle a continuación.

5.3.4.1 Listar Restaurantes

La primera pantalla a la que el cliente accede tras el inicio de sesión muestra una lista con todos los restaurantes disponibles en la aplicación. Esto se consigue mediante una *Request* usando *Volley*, y en el servidor la recibe el *ControllerRestaurante*, que lee la opción y llama al método estático *mostrarRestaurantes()* del *GestorRestaurante.* Este método se puede ver a continuación:

```
public static String mostrarRestaurantes(ServletContext contextoApp){
        Gson gson = new Gson();
        String res = "No se han encontrado restaurantes";
        //Lista de Restaurantes deberia estar cacheada
        HashMap<Integer, Restaurante> restaurantesMap = (HashMap)
contextoApp.getAttribute("restaurantesMap");
        if(restaurantesMap == null){
            restaurantesMap = initRestauranteMap(contextoApp);
        }
        Collection<Restaurante> values = restaurantesMap.values();
        if(values.isEmpty()){
```

```
            System.out.println("Error: No se ha encontrado ningun restaurante");
        }
        else{
            res = gson.toJson(values);
        }

        return res;
    }
```

Al igual que el método *findRestaurante()*, este método busca primero los restaurantes cacheados en memoria antes de acceder a la base de datos. De esta forma se reducen las solicitudes a la base de datos, evitando búsquedas repetidas.

Adicionalmente, en el lado del cliente se muestra una opción en la barra de opciones superior (opción exclusiva de esta *Activity*) que muestra un desplegable con las categorías de restaurantes y permite filtrar los restaurantes mostrados por su categoría. Esto se consigue con el siguiente método:

```
private void filtrarByCategoria(List<Integer> categoriasIndex){
    String[] categorias =
getResources().getStringArray(R.array.categorias_restaurante);
    restaurantes.clear();

    if(categoriasIndex.isEmpty()){
        //Mostrar Todos
        Log.d(TAG, "Quitando Filtro");
        restaurantes.addAll(TFGApplication.getRestaurantes());
    }
    else{
        Log.d(TAG, "Filtrando por categorias " + categoriasIndex);
        TreeSet<Restaurante> restaurantesFiltrados = new TreeSet();

        for(Restaurante restaurante : TFGApplication.getRestaurantes()){
            for(Integer i : categoriasIndex){
                if(restaurante.getCategoria().equals(categorias[i])){
                    restaurantesFiltrados.add(restaurante);
                    break;
                }
            }
        }

        restaurantes.addAll(restaurantesFiltrados);
    }

    //Aplicar cambios al adapter
```

```
    adapter.notifyDataSetChanged();
}
```

Este método limpia la lista, recupera la lista de restaurantes completa que había sido guardada previamente en contexto de aplicación, y recorre la lista comparando con las categorías seleccionadas, y si coinciden, se añadirán a la lista de restaurantes a mostrar.

5.3.4.2 Seleccionar Restaurante

Esta función ocurre exclusivamente en el lado del cliente. Una vez se muestra la lista de restaurantes, el usuario puede seleccionar un restaurante. Esta acción inicia dos procedimientos que se ejecutan secuencialmente.

```
convertView.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View view){

        Toast.makeText(getContext(), "Has pulsado el restaurante : " +
restaurante.getNombre(), Toast.LENGTH_SHORT).show();
        TFGApplication.getInstance().setSelectedRestaurante(restaurante);

        Intent intent = new Intent(mContext, MainActivity.class);
        mContext.startActivity(intent);

    }
});
```

Este método asigna una función *Listener* que se ejecuta cuando un restaurante de la lista es pulsado.En primer lugar, se guarda en el contexto de la aplicación (*TFGApplication)* el restaurante seleccionado. En segundo lugar, se inicia una nueva *Activity*, *MainActivity*, que nos lleva a la siguiente funcionalidad.

5.3.4.3 Ver información del Restaurante

Una vez seleccionado el restaurante deseado, se abre una instancia de *MainActivity*. Esto muestra una pantalla que contiene toda la información relevante del restaurante seleccionado (leído del contexto de aplicación, *TFGApplication)*: nombre, descripción, categoría, horarios, etc.

Además, esta *Activity* contiene unos botones que permiten navegación hacia otras pantallas y opciones, como son un botón para mostrar el menú, un botón para mostrar las mesas y uno para ver el pedido.

5.3.5 MÓDULO DE RESERVAS

Este módulo es el encargado de gestionar las funcionalidades relacionadas con las reservas y las mesas. En primer lugar, permite mostrar las mesas del restaurante, pudiendo seleccionar una mesa para sentarse o reservarla, en función de las horas disponibles de la misma. También permite mostrar las reservas que hay pendientes, pudiendo cancelarlas si se desea, además de ver la reserva que hay activa en ese momento (en la que el usuario está sentado).

Estas funcionalidades se ofrecen en el cliente a través de una serie de *Activities* encargadas de mostrar la información y enseñar las opciones disponibles, y una vez el cliente ha elegido, la aplicación hará una petición dirigida al servlet *ControllerMesa*, que a su vez redirige la comunicación hacia el restaurante correspondiente (mediante *GestorRestaurante*). El restaurante tiene a su vez una instancia de *GestorMesa*. Esta es la clase que contiene una lista de las mesas del restaurante, y se encargará de gestionarlas.

También existe una clase *Reserva*, que representa a una reserva en un restaurante determinado, en una mesa, a una hora. Esta reserva contiene un ID de reserva único para el restaurante, e información adicional (mesa, hora, comensales, etc).

Estas serán las clases encargadas de dar soporte a este módulo. La implementación de cada una de las funcionalidades que este ofrece se explica a continuación. Pero en primer lugar se va exponer la clase *Horario*, que contiene y mantiene la estructura de datos sobre la que se basa el sistema de *Reservas*.

5.3.5.1 Horario

Esta clase, además de guardar el horario apertura y cierre de los *Restaurantes*, ofrece una abstracción sobre el horario de cada una de las mesas que permite saber qué mesa está reservada a qué hora. Las relaciones que tiene esta clase con otras de Dominio se pueden ver

en la Figura 22. Además, se muestra en la siguiente figura () una vista de la clase *Horario* por separado:

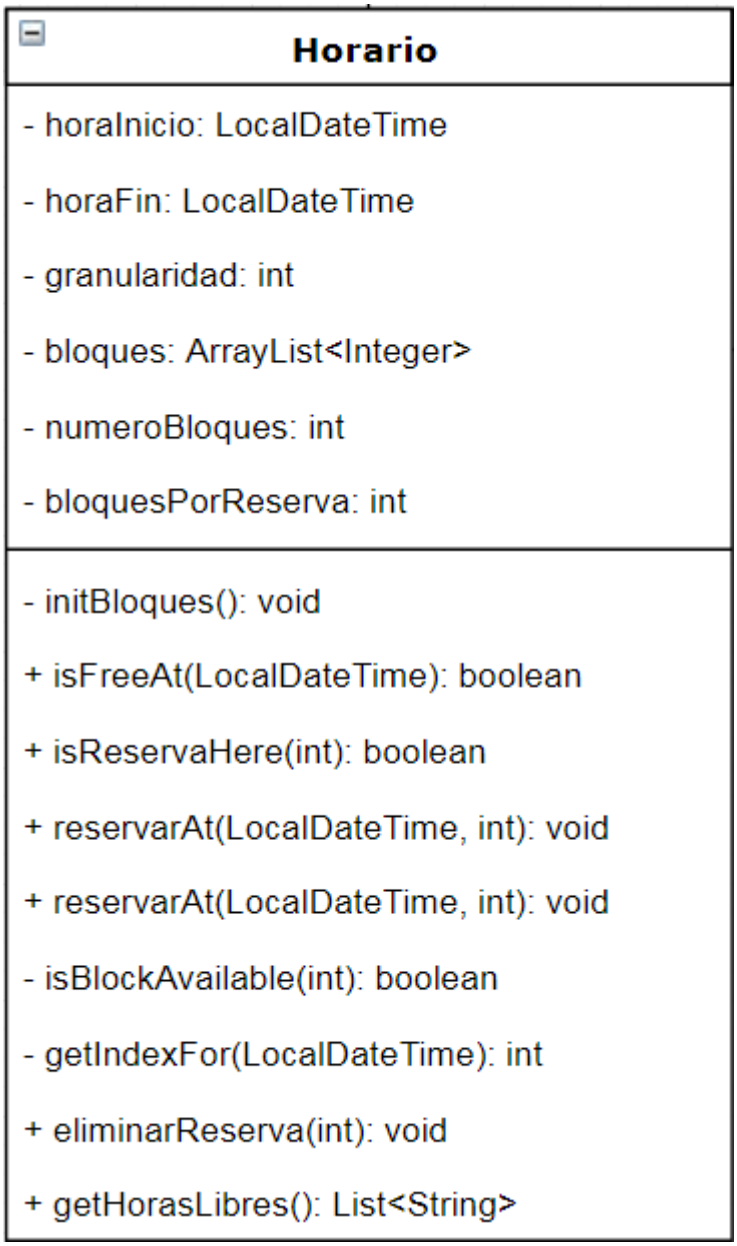

Figura 30. Diagrama de Clase de Horario

Ya que esta clase contiene la información del horario de apertura y cierre del *Restaurante*, y se asigna un *Horario* al *Restaurante.* El hecho de asignar un *Horario* en un restaurante hace que todas las mesas obtengan una copia de esa instancia de *Horario*. Así, cada *Mesa* tiene una estructura de datos que guarda qué hora está ocupada y por qué reserva.

Para esto, se ha elegido una lista de "bloques". Un bloque representa una unidad de tiempo, que puede variar de un restaurante a otro, y está definida por la variable *granularidad*. Esta variable contiene un entero (por defecto, 10) con el número de minutos que representa un bloque. Por su parte, cada uno de estos "bloques" está representado por un entero (*Integer*). Un bloque puede tomar dos categorías de valores:

- 0: este valor indica que en esta hora no se hay una reserva. Es el valor por defecto.
- *reservaId*: este es un valor entero estrictamente positivo (> 0) que representa que este bloque de tiempo está reservado, y con qué *reservaId*, que puede ser utilizado después para identificar la instancia de *Reserva* correspondiente a ese bloque de tiempo.

Los bloques se ordenan de forma que el bloque con índice 0 representa la hora de apertura del restaurante (guardada en *horaInicio*), y el último (bloque con índice *numeroBloques* -1) representa la hora de cierre (*horaFin*).

El método *initBloques()* se encarga de crear esta estructura. Esta calcula el número de minutos que existen entre *horaInicio* y *horaFin*, y los divide en bloques de tamaño *granularidad*, inicializándolos a 0. Para esto se aprovecha de la API introducida en la versión 1.8 de Java: *Java.time*. Estos paquetes introducen las clases *LocalDateTime y ZonedDateTime*(con sus variantes), que permiten manipular horas de forma sencilla. La implementación de este método es la siguiente:

```
private void initBloques(){
        long minutes = ChronoUnit.MINUTES.between(horaInicio, horaFin);
        this.numeroBloques = (int) Math.ceil(minutes/granularidad);

        bloquesPorReserva = (int) Math.ceil(TIEMPO_RESERVA/granularidad);
}
```

Con esta lógica, se implementan métodos privados para el uso de la clase que permiten conocer el índice al que corresponde cualquier hora (dentro del rango) con *getIndexFor(LocalDateTime hora)*.

Por otra parte, se ha implementado el método *isBlockAvailable(int bloque)*, que permite conocer si un bloque está disponible para ser reservado. Disponible en este contexto quiere decir que no hay otra reserva en ese momento o en un tiempo menor a la duración esperada de reserva (estimada en 180 minutos).

Sin embargo, estos métodos y esta estructura son transparentes para la clase *Mesa* (encargada de interactuar directamente con *Horario*), y su interfaz está formada por los métodos públicos *isReservaHere(int reservaId)*, *isFreeAt(LocalDateTime hora)*, *reservarAt(LocalDateTime hora)*, *eliminarReserva(int reservaId)*.

El método más relevante de estos es *reservarAt(LocalDateTime hora).* Este es el método que se llama tanto cuando se desea reservar a una hora como cuando se desea sentarse en el momento:

- Reservar: se llama a este método pasado una instancia de *LocalDateTime* con la hora elegida para la reserva. Debido a que Android no tiene implementado la versión 1.8 de Java, no se ha podido seleccionar la hora desde en cliente usando *LocalDateTime*. En cambio, la hora llega en formato *String (HH:mm)* y hay que *parsearla* y crear una instancia de *LocalDateTime*. Para hacer esto se ha creado un método estático en la clase *Horario* que tiene la siguiente forma:

```
public static LocalDateTime stringToLocalDateTime(String hora){
        /**
         * parsea una hora en formato HH:mm a LocalDateTime, añadiendo la fecha
de hoy
         */
        LocalDate hoy = LocalDate.now();
        int[] horaInt = horaToInt(hora);
        if(horaInt[0] == -1 || horaInt[1] == -1)
            return null;

        LocalTime ahora = LocalTime.of(horaInt[0], horaInt[1]);
        return LocalDateTime.of(hoy, ahora);
    }
```

- Sentarse: se llama a este método pasando como parámetro la hora local en ese instante. Esto se consigue gracias al método proporcionado por la clase *LocalDateTime: LocalDateTime.now();*

El funcionamiento del método *reservarAt(LocalDateTime hora)* se detalla en el siguiente diagrama (Figura 31):

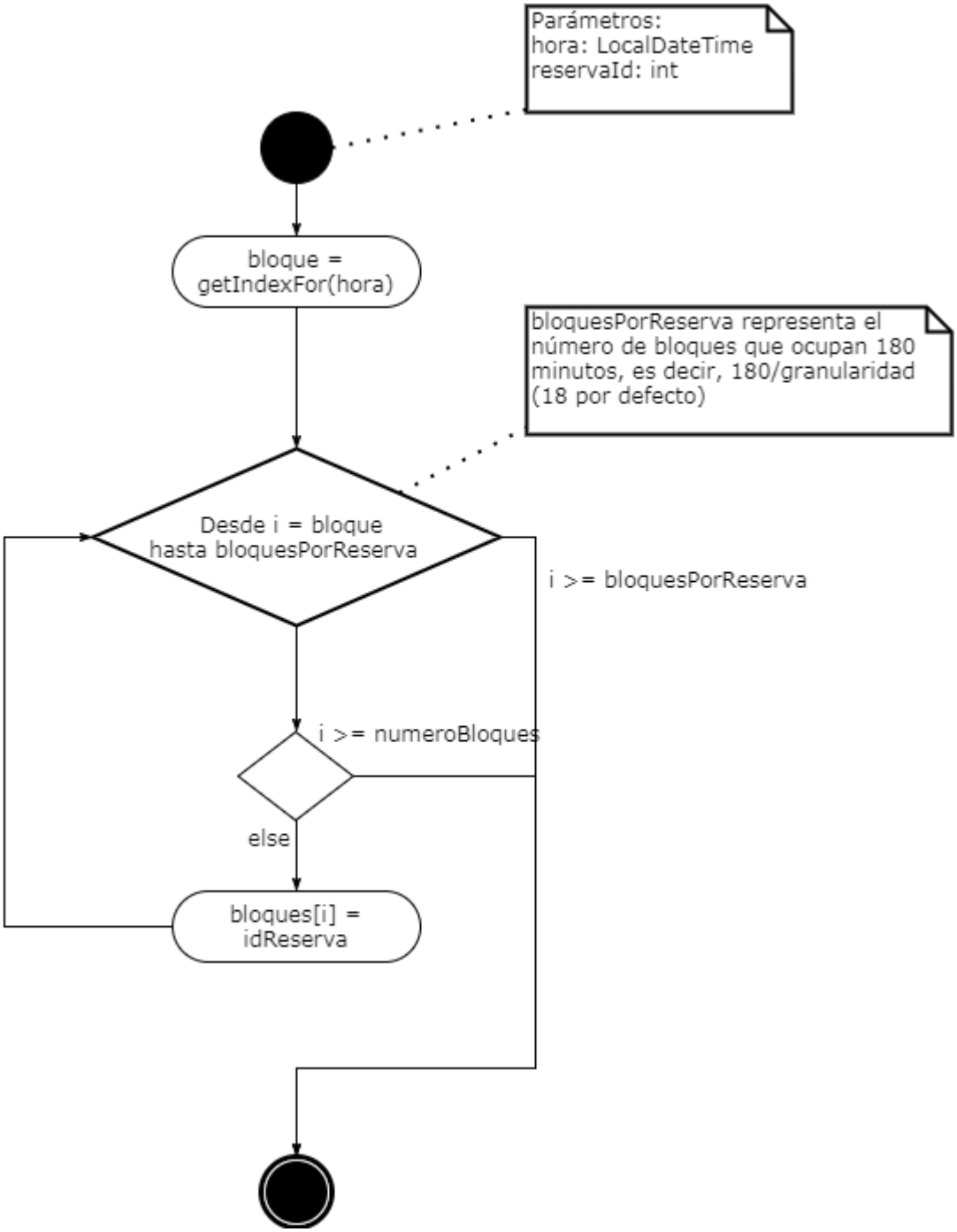

Figura 31. Diagrama de Actividad del método reservarAt()

Adicionalmente, el *Horario* también tiene un método que devuelve una lista con las horas disponibles (en formato HH:mm, de tipo String) desde el momento en el que se llama, utilizado para mostrar las horas disponibles para reservar cuando el cliente desea reservar una mesa. Este método se implementa de la siguiente manera:

```
public List<String> getHorasLibres(){
        /**
         * Devuelve un string con las horas libres desde el momento que se
pregunta
         */
        LinkedList<String> result = new LinkedList();

        //Asegurarte que estas en la franja correcta
        LocalDateTime now = LocalDateTime.now();
        if(now.isBefore(horaInicio)){
            System.out.println("La hora es anterior al inicio del turno");
            System.out.println("horaInicio= " + horaInicio.toString() + " hora =
" + now.toString());
            return result;
        }
        else if(now.isAfter(horaFin)){
            System.out.println("La hora es posterior al fin del turno");
            return result;
        }
        int nowIndex = getIndexFor(now);
        int horaDeInicio = this.horaInicio.getHour();
        int minutoDeInicio = this.horaInicio.getMinute();
        int minutoDeBloque = minutoDeInicio + nowIndex*this.granularidad;
        for(int i = nowIndex; i< numeroBloques; i++){
            if(isBlockAvailable(i)){
                //Este bloque esta disponible, lo traducimos a hora en String y
lo añadimos
                //FIXME: Esto por ahora solo devuelve la hora local del servidor
                int horaDeBloque = horaDeInicio + minutoDeBloque/60;
                int minutoAdd = minutoDeBloque % 60;
                String horaParsed = Integer.toString(horaDeBloque);
                if(horaDeBloque < 10){
                    horaParsed = "0" + horaParsed;
                }
                String minutoParsed = Integer.toString(minutoAdd);
                if(minutoAdd < 10){
                    minutoParsed = "0" + minutoParsed;
                }
                String toAdd = horaParsed + ":" + minutoParsed;
                result.add(toAdd);
            }
            minutoDeBloque += granularidad;
        }
        return result;
    }
```

La estrategia de diseño elegida para la clase *Horario* permite abstraer al resto de clases de la implementación del sistema de reservas, pudiendo ser modificado y mejorado en cualquier momento sin afectar al resto de clases.

5.3.5.2 Listar Mesas

Cuando el cliente pulsa en el botón de ver mesas desde la pantalla principal del restaurante, la aplicación crea una instancia de *MesasActivity*, que a su vez genera una petición HTTP para el *ControllerMesa*, pidiendo una lista de mesas en formato JSON.

El C*ontroller*, redirige la petición al restaurante seleccionado, y este a su vez lo redirige al *GestorMesa*. El gestor será el encargado de generar la respuesta en el formato deseado (Array JSON). Esta respuesta se procesa en la aplicación cliente, que traduce el JSON a objetos de tipo *Mesa*. Este proceso se consigue con el siguiente *snippet*:

```
for (int i = 0; i < response.length(); i++) {
    try {
        JSONObject jsonObject = response.getJSONObject(i);
        Log.d(TAG, "Se ha recibido: " + jsonObject);
        mesas.add(new Mesa(jsonObject));
    } catch (JSONException e) {
        e.printStackTrace();
    }
    adapter.notifyDataSetChanged();
    Log.d(TAG, "Poblando el list");
}
TFGApplication.getInstance().saveMesas(mesas);
```

Este código se encuentra en el *Listener* de la *Response* que se asigna en el momento de crear una *JsonArrayRequest* en *Volley*. Esta clase permite recorrer la respuesta como si fuera un *array*, y extraer cada uno de los objetos por separado como *JSONObject*. Por último, es necesario añadir un constructor a la clase *Mesa* que tome como atributo un *JSONObject* ,y este será el encargado de extraer los elementos del objeto. Esta técnica se ha repetido en distintas áreas del proyecto:

```
public Mesa(JSONObject jsonObject){
    try {
        this.id = jsonObject.getInt("id");
        this.maxComensales= jsonObject.getInt("maxComensales");
        this.ocupada = jsonObject.getBoolean("ocupada");
        this.comensales = new HashSet();
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

Se ha escogido esta técnica sobre la traducción usando GSON (explicada en el apartado 5.3.2.5) debido a que atributos del JSON creados en el servidor no se quieren traducir del lado del cliente (el *Horario* de la mesa no es necesario, es interno del servidor). Otra solución para este problema consiste en añadir una etiqueta en la declaración de la clase que excluye el atributo de la serialización GSON, por ejemplo:

```
//Gestores no serializables para GSON
    @Expose(serialize = false)
    private GestorMesa gestorMesa;
```

En la clase *Restaurante*, el *GestorMesa* no se quiere serializar cuando se listan los restaurantes, y con esta etiqueta se excluye.

Adicionalmente, en el cliente es posible filtrar la lista de mesas mediante un método análogo al explicado en el apartado 5.3.4.1.

5.3.5.3 Elegir una Mesa: Reservar y Sentarse

Una vez se han listado las mesas en el cliente Android, en *MesasActivity*, se puede seleccionar un elemento de la lista. Esto a su vez, muestra dos opciones

- Sentarse: Esta es la opción que elige un usuario que se encuentra físicamente en el restaurante y quiere el servicio inmediatamente. Sentarse manda una petición al servidor indicando el restaurante y la mesa y, si está disponible, registra la reserva para ese momento en el *Horario de la mesa*, y responde con una instancia de *Reserva*, que el cliente añade al contexto como *reservaActiva*. Esta *Reserva* tiene un

GestorPedido que se encarga de gestionar los pedidos en el cliente (explicado en detalle en 5.3.6). Existe una opción para sentarse, explicada en el apartado 5.3.5.3, que permite al usuario sentarse en una mesa que ya ha reservado. Este proceso se ilustra en la Figura 32.

- Reservar: Esta es la opción que elige un usuario que no se encuentra en el restaurante, y desea hacer una reserva para una hora determinada (en el futuro). Al pulsar esta opción, al usuario se le muestra una lista con las horas disponibles para hacer la reserva, teniendo en cuenta otras posibles reservas que existan sobre esta mesa. Después el usuario elige la hora que desea reservar y la aplicación genera una petición al servidor para hacer la reserva, indicando el restaurante, la mesa y la hora. El restaurante comprueba la reserva, y si está disponible, registra la reserva en el *Horario* de la mesa y devuelve una instancia de *Reserva* que el cliente guarda para su uso futuro. Este proceso se ilustra en la Figura 33.

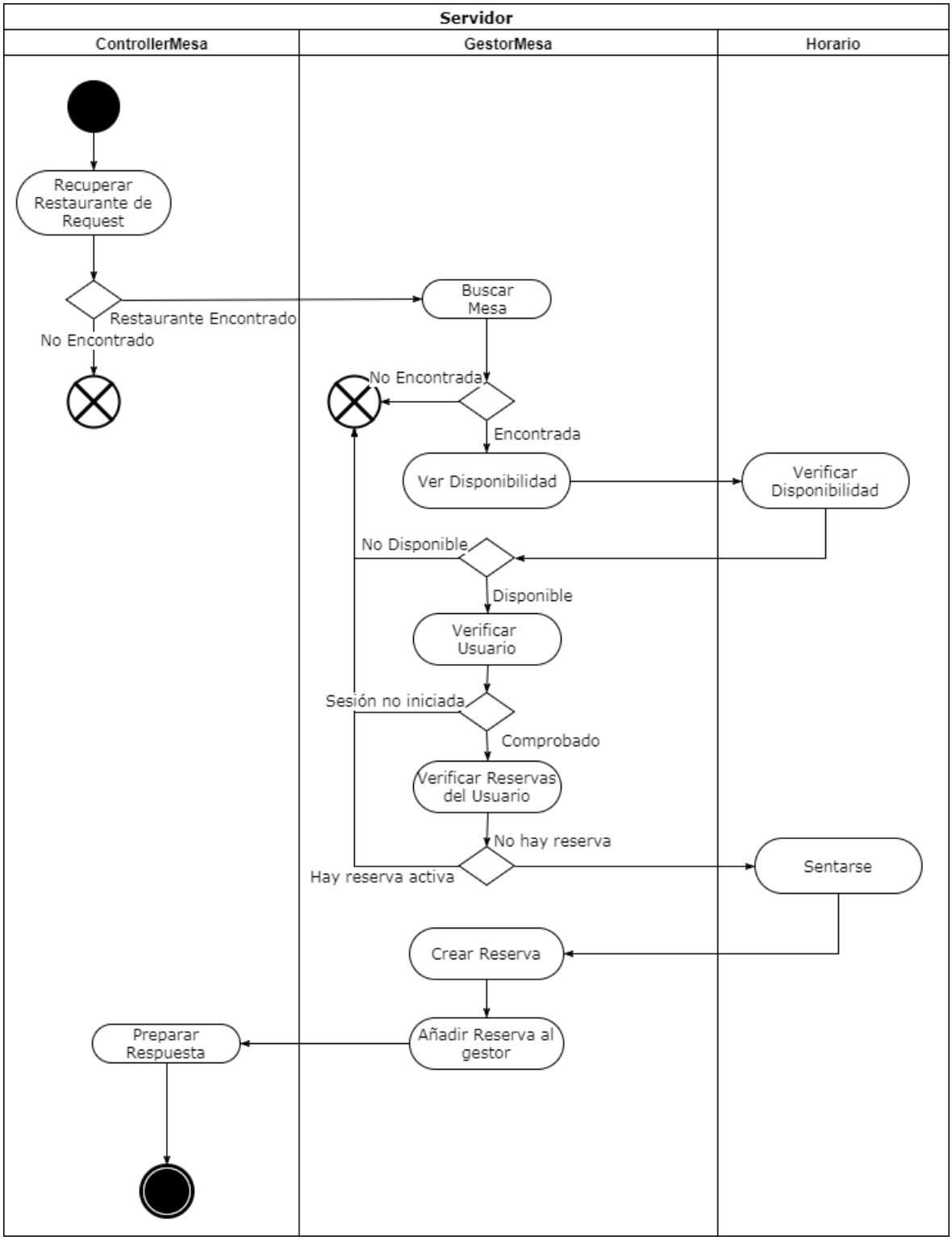

Figura 32. Diagrama de Actividad del proceso Sentarse

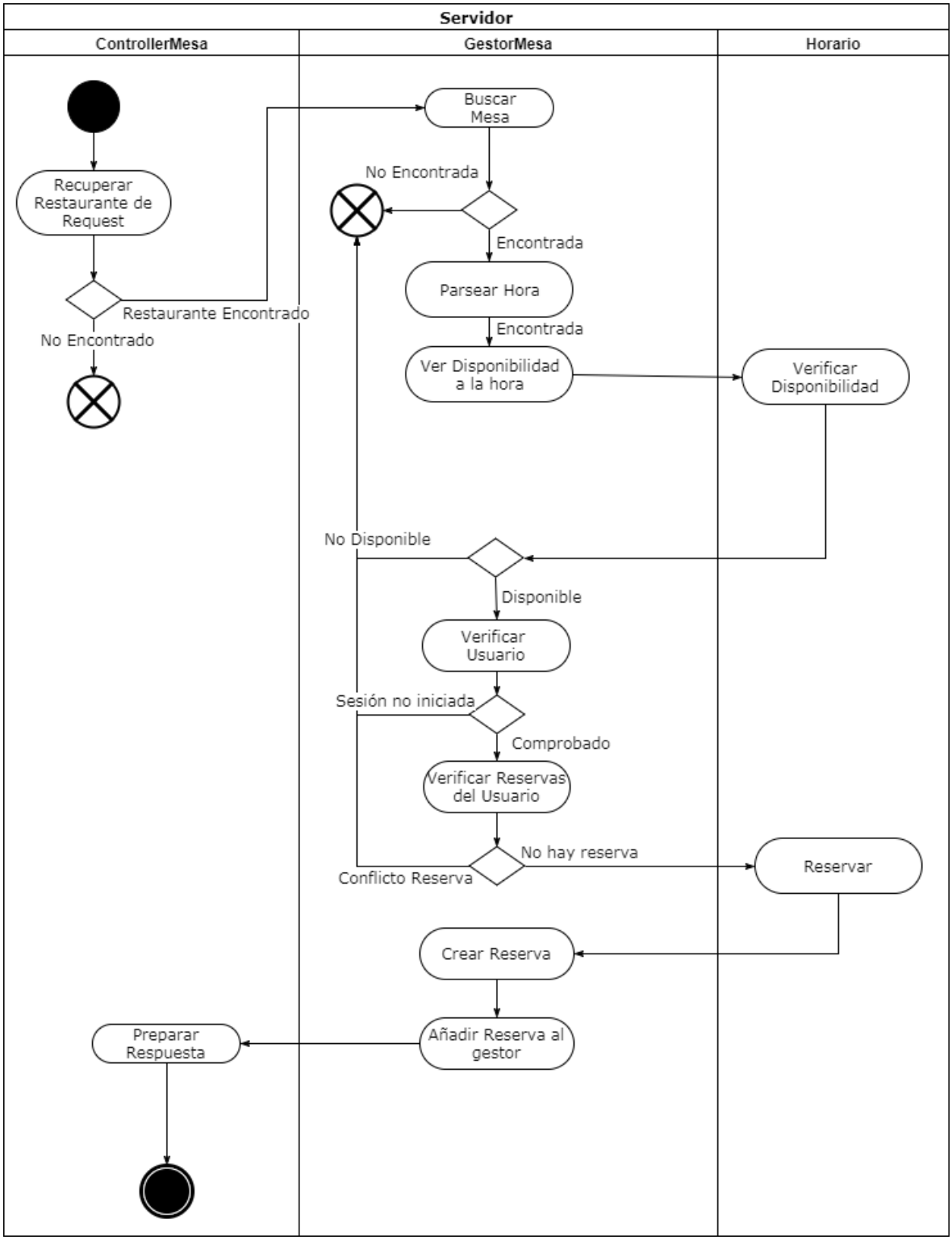

Figura 33. Diagrama de Actividad del proceso Reservar

5.3.5.4 Sentarse mediante código QR

La aplicación pone a disposición del cliente la opción de sentarse en la mesa, además del mecanismo manual (explicado en la sección 5.3.5.3), sentarse a leyendo un código QR que se encuentra en la mesa. Esta opción no sólo es más sencilla que buscar el identificador de la mesa deseada y seleccionarla de la lista, sino que añade una capa de seguridad al sistema, ya que el restaurante se asegura que el cliente se encuentra físicamente en el establecimiento.

Para ofrecer esta funcionalidad, se ha aprovechado una librería Android: *ZXing* [13]. Esta librería permite integrar de forma nativa en la aplicación una *Activity* que accede a la cámara y escanea automáticamente una variedad de códigos 2D y 3D, entre ellos, códigos QR. Una vez lo ha leído, la librería también traduce el código y extrae la información codificada. Un ejemplo del uso de este sistema es el siguiente:

```
public void scanQR(View view){
    //Abrir escáner
    Intent intent = new Intent(this, QRCodeScannerActivity.class);
    startActivityForResult(intent, RC_ESCANER);

}
```

Esta función se ejecuta cuando se pulsa el botón de escanear desde *MesasActivity*. Esto inicializa una nueva *Activity* que será la encargada de leer el código accediendo a la cámara. Una vez lee un código, esta devuelve un resultado a *MesasActivity*. Este resultado permite identificar la mesa en la que se quiere sentar el usuario, y una vez ahí se puede generar la petición al servidor de la misma forma que si se hubiera seleccionado manualmente.

5.3.5.5 Ver Reservas: Cancelar y Levantarse

El cliente tiene a su disposición en todo momento, desde la barra superior de opciones (*ActionBar*), una opción en el menú desplegable para ver sus reservas. Esta opción lleva a la pantalla de reservas: My*ReservasActivity*. En esta pantalla se le muestran una lista con las reservas que tiene pendiente, pudiendo ver el restaurante, la mesa y la hora de la reserva, y en la parte superior la reserva activa (si existe).

Desde estas reservas, el usuario puede seleccionar una reserva pendiente. Esto abrirá un diálogo con 3 opciones:

- Sentarse: La primera opción que se muestra permite al usuario sentarse en la mesa correspondiente a la reserva. Esta es la opción que elige un usuario que se encuentra en físicamente en el restaurante, ha hecho una reserva previamente, y desea activarla. Esta opción genera un proceso similar al descrito en la Figura 32, con la diferencia que al servidor se envía una id de reserva que se utiliza para identificar la *Reserva* correspondiente, para activarla. Esta opción sólo está permitida como máximo 30 minutos antes de la hora de la *Reserva*, siempre que la mesa no esté aun ocupada.
- Productos: Esta opción permite a un usuario con una reserva futura hacer un pedido para una hora determinada (dentro del rango de su *Reserva*). Esta opción abre una pantalla de *MenuActivity* y permite realizar un pedido de la misma forma que se explica en la sección 5.3.6.1. Esto añade un *Pedido* pendiente a la sección de pedidos, que se podrá confirmar cuando se desee (el proceso se describe más en detalle en la sección 5.3.6.2).
- Cancelar: Esta es la opción que elige un usuario que tiene una reserva previa y desea cancelarla. Esto genera una petición al servidor, con una id de reserva, y, si la reserva es del usuario, aun no se ha activado y no contiene un pedido, esto elimina la reserva del cliente y del servidor.

5.3.6 MÓDULO DE PRODUCTOS

Este módulo es el encargado de gestionar las funcionalidades relacionadas con los productos y con los pedidos. De esta forma, se encargará de listar los productos, mostrar su información y añadirlos al pedido, a la vez que permite ver el pedido con sus productos, su precio total y eliminar productos del mismo.

La mayor parte de esta funcionalidad ocurre en el lado del cliente, en la aplicación Android, y en el lado del servidor, las peticiones se dirigen al *ControllerPedido*, que gestiona el listado de productos y la confirmación de pedidos para el restaurante. Esto último se consigue

mediante un *GestorPedidos* que se encuentra en cada instancia de *Reserva*, que mantiene los pedidos confirmados y los pendientes por pagar.

5.3.6.1 Listar los Productos

Una vez se ha seleccionado un restaurante, desde la pantalla principal (*MainActivity*) se puede acceder pulsando el botón de "Menú" al listado de productos disponibles (*MenuActivity*). Esto se consigue mediante una petición HTTP al *ControllerPedido* que se transmite al *Restaurante* seleccionado, que será el encargado de devolver un listado de los productos que ahí se ofertan.

Desde el listado, el usuario podrá acceder a la descripción del producto, pulsando en cualquier elemento de la lista, lo que abrirá una instancia de *DescriptionActivity*. Desde aquí, se podrá ver la información del producto, así como añadirlo para el pedido pendiente.

5.3.6.2 Ver Pedidos

Desde cualquier punto de la aplicación existe la opción en la barra superior (*ActionBar*) de ir a la pantalla de pedidos (*CarritoActivity*). Desde ella se muestran un listado con todos los productos que se han confirmado ya para el restaurante, con su precio correspondiente, y también se muestra un listado con los productos pertenecientes al pedido pendiente. Desde esta pantalla, se pueden eliminar productos de este pedido que aún no ha sido confirmado, ver su precio, y confirmar el pedido.

Todas estas funciones se gestionan localmente en el cliente, excepto la confirmación de pedido. Esta opción genera una petición HTTP dirigida al *ControllerPedido*, que le solicitará añadir el pedido incluido en la petición en forma de JSON a la *Reserva* activa del usuario, al que se identificará mediante la sesión. Este proceso de detalla en el siguiente diagrama de actividad (Figura 34):

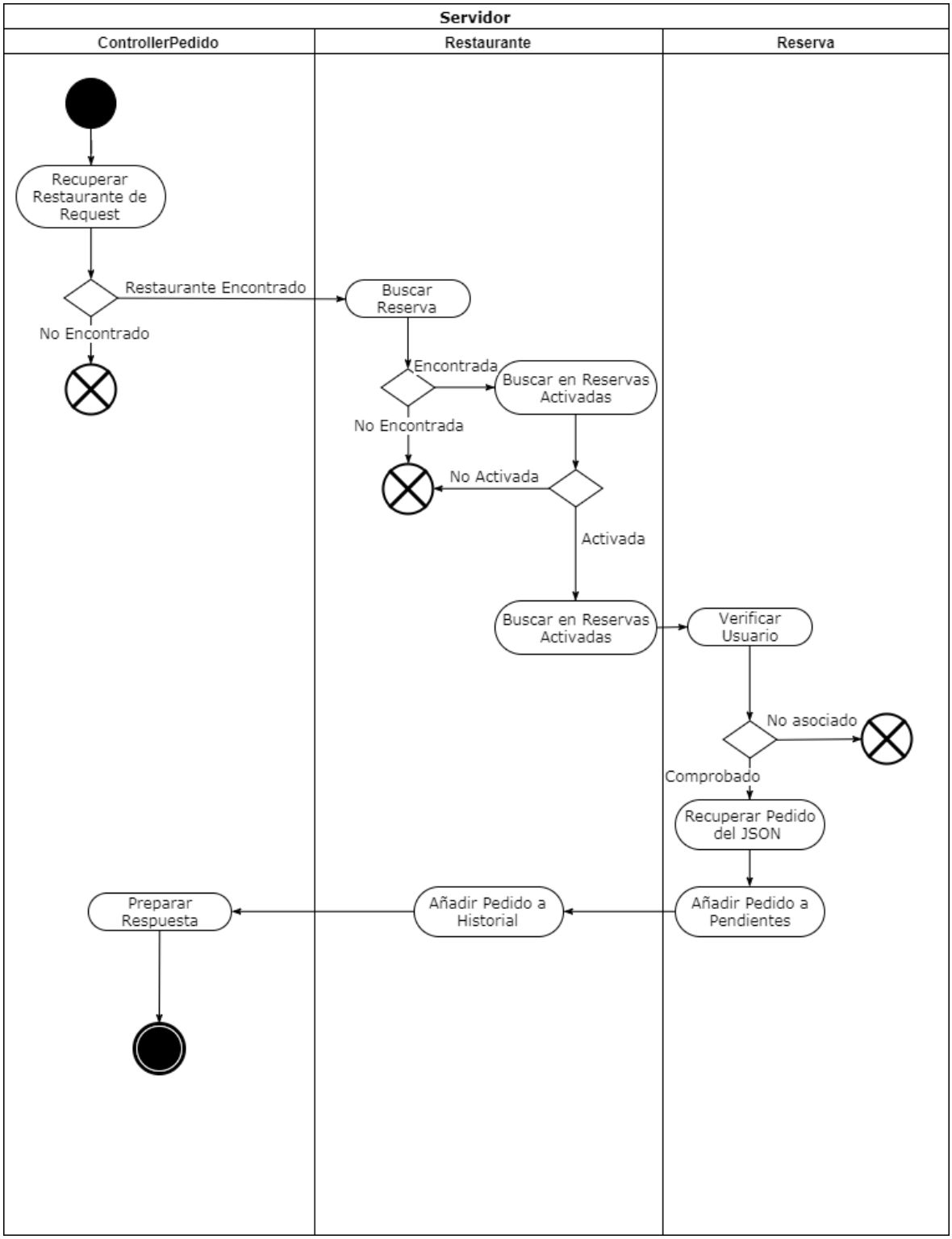

Figura 34. Diagrama de Actividad de la gestión del Pedido

Capítulo 6. ANÁLISIS DE RESULTADOS

En este capítulo se van a mostrar y analizar los resultados obtenidos de la implementación de las funcionalidades diseñadas y explicadas en los capítulos Capítulo 4. y Capítulo 5. También se van a mostrar capturas de la aplicación para mostrar el interfaz con el que usuario interactua.

6.1 PANTALLA DE INICIO DE SESIÓN

La primera pantalla con la que el usuario se encuentra una vez inicia la aplicación es la pantalla de inicio de sesión, *LoginAcitvity*. En esta pantalla se le muestra al usuario unas cajas de texto, para introducir su información de inicio de sesión (si dispone de una cuenta creada). El interfaz es el siguiente (Figura 35) :

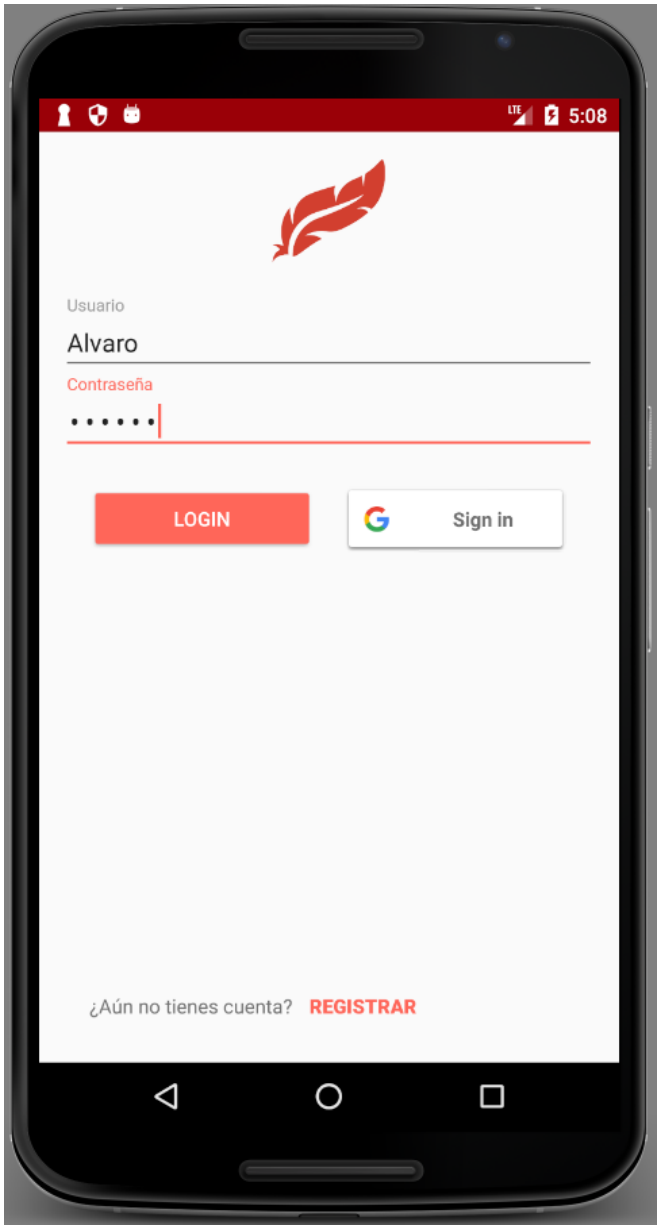

Figura 35. Pantalla de inicio de sesión

En el caso de que el usuario no disponga de una cuenta de usuario creada, puede pulsar el botón "Registrar" situado en la parte inferior de la pantalla, que le muestra un formulario en una *BottomSheet* (recurso gráfico recomendado en *Material Design*), desde el cual el usuario puede introducir su información personal, un nombre de usuario y una contraseña que le servirán para identificarse. Este formulario se muestra de la siguiente manera (Figura 36):

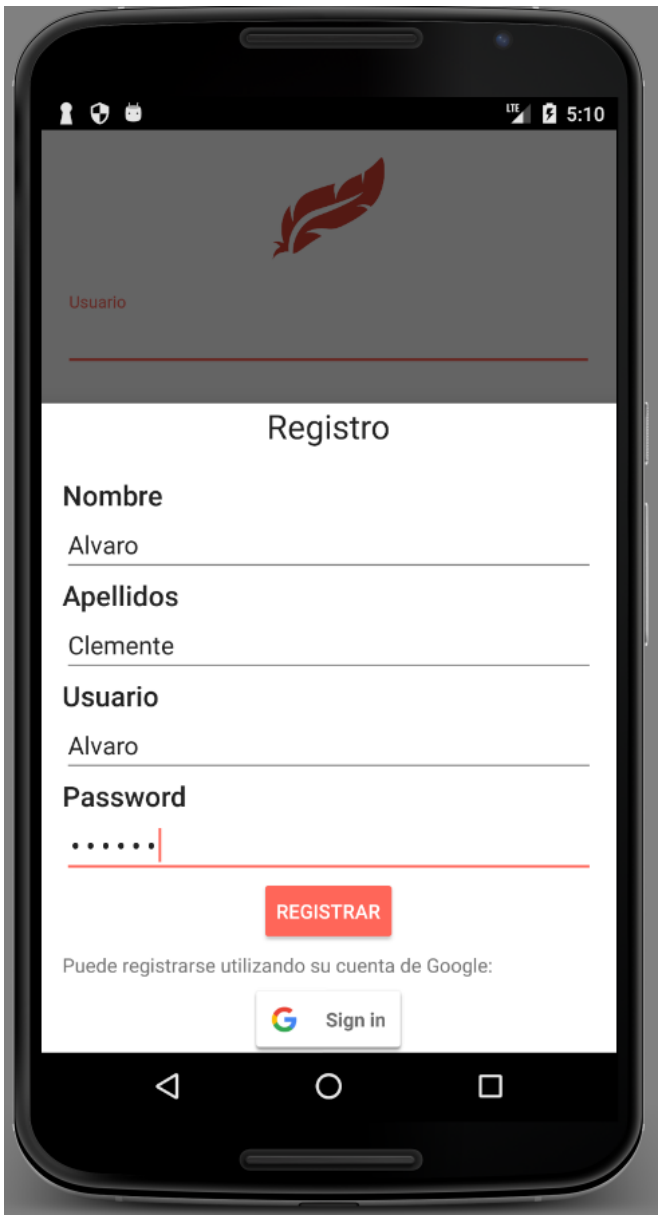

Figura 36. Pantalla de registro

Al registrarse, el sistema iniciará la sesión del usuario de forma automática. Una vez el usuario ha iniciado sesión, se le redirige a la pantalla de restaurantes.

6.2 PANTALLA DE RESTAURANTES

Esta es la primera pantalla que ve el usuario una vez ha iniciado sesión en la plataforma. Desde esta pantalla el usuario puede ver un listado con los restaurantes que se encuentran dados de alta en la plataforma. Un ejemplo es la siguiente figura (Figura 37):

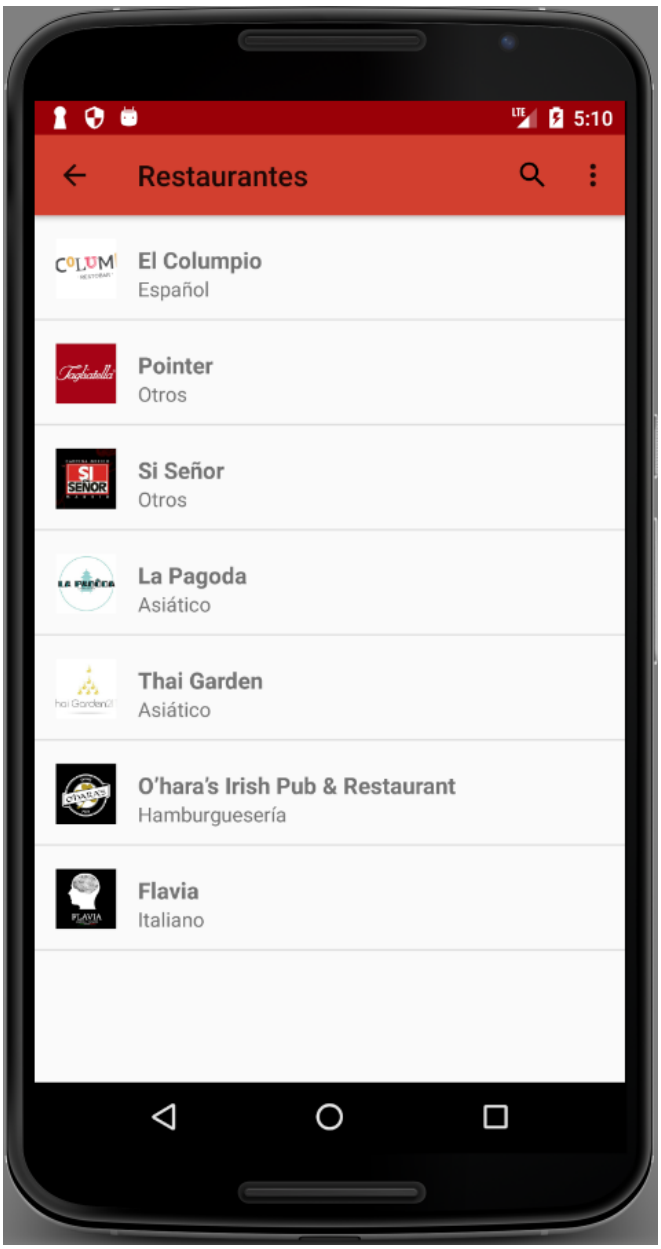

Figura 37. Pantalla de restaurantes

Si lo desea, el usuario puede filtrar la lista de restaurantes por una o varias categorías. Desde esta pantalla el usuario puede elegir el restaurante que desee, y al pulsar en el elemento de la lista correspondiente, se le redirige a la pantalla principal del restaurante.

6.3 PANTALLA PRINCIPAL

Esta es la pantalla principal del restaurante (*MainActivity*). Desde ella, el usuario puede ver información relevante sobre el restaurante: horario, teléfono, dirección o una descripción sobre el establecimiento y la comida. Adicionalmente, si se encuentra disponible se muestra una fotografía relevante sobre el restaurante. Un ejemplo de esta pantalla es el siguiente (Figura 38):

Figura 38. Pantalla principal del restaurante

Desde esta pantalla, el usuario puede elegir qué es lo que desea ver sobre el restaurante. Las dos opciones principales son "Mesas" y "Menu". El botón de mesas le redirigirá a la pantalla de mesas (*MesasActivity*), mientras que el botón menú le redirigirá a la pantalla de productos (*MenuActivity*).

6.4 PANTALLA DE MESAS

En esta pantalla se le muestra al usuario una lista con las mesas que tiene el restaurante que ha seleccionado. Si lo desea, el usuario puede filtrar la lista de mesas por su estado, dejando a la vista únicamente las que están disponibles para reservar en ese momento. Un ejemplo se muestra en la siguiente figura (Figura 39):

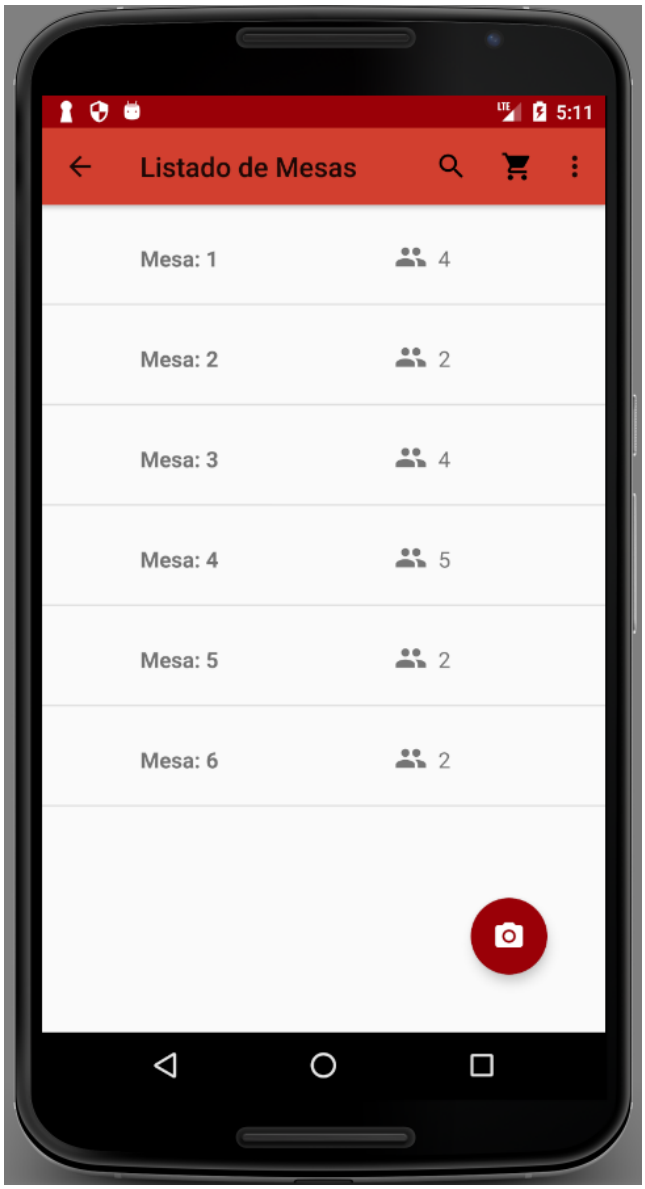

Figura 39. Pantalla de mesas

El usuario puede seleccionar una mesa de la lista, y al hacerlo se muestra un cuadro de diálogo que le da dos opciones:

- Sentarse: esta es la opción que elige si se encuentra físicamente en el restaurante y desea sentarse a la mesa.
- Reservar: esta es la opción que elige si el usuario no se encuentra físicamente en el restaurante, y lo que desea es reservar una mesa en el futuro. Si elige esta opción, se le muestra un diálogo como el mostrado en la Figura 40, que le permite elegir la hora a la que quiere hacer la reserva. En esta lista sólo se encuentran las horas que están disponibles.

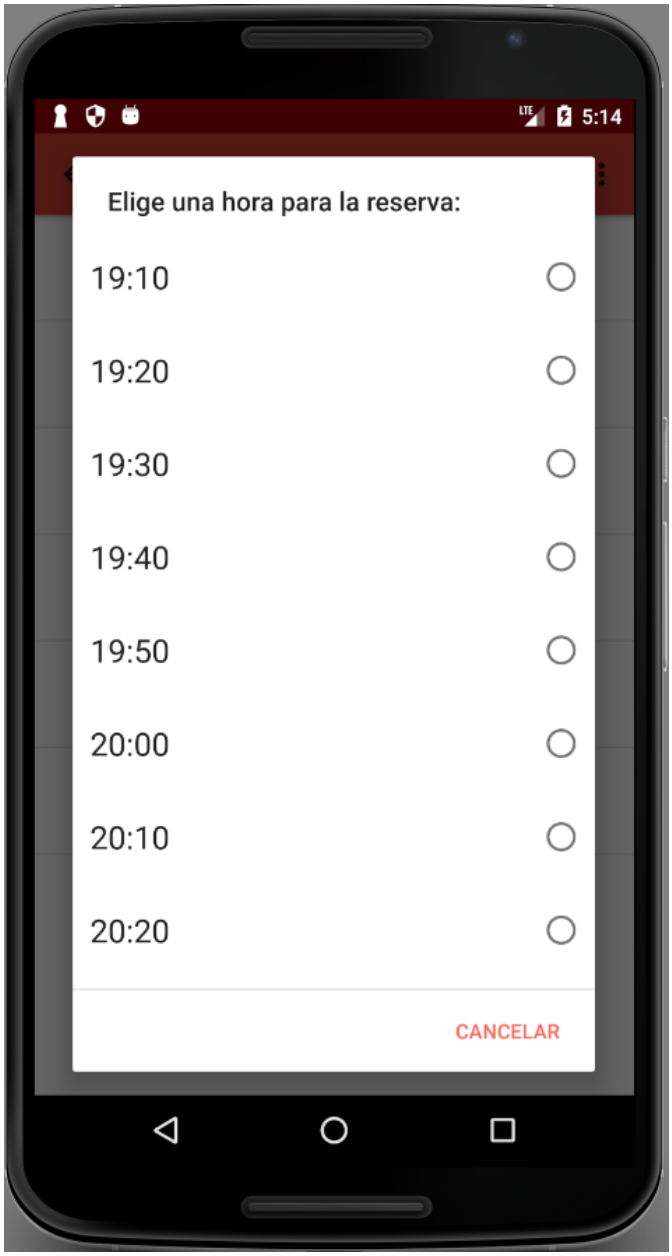

Figura 40. Pantalla de productos: reservar

Adicionalmente, se encuentra un botón flotante (*FloatingButton*) que permite al usuario escanear el código QR de una mesa en la que desee sentarse. Desde esta pantalla no hay más navegación, más que volver a la pantalla principal del restaurante.

6.5 PANTALLA DE PRODUCTOS

Esta es la otra pantalla a la que se puede llegar desde *MainActivity*. Esta pantalla muestra un listado con los productos que ofrece el restaurante. Un ejemplo de su uso se muestra en la siguiente imagen (Figura 41):

Figura 41. Pantalla de productos

Desde esta pantalla el usuario puede seleccionar un producto para ver más información sobre el mismo y, si se encuentra sentado en una mesa (con una reserva activa), puede añadir los productos a su pedido activo para después confirmarlos al restaurante.

6.6 PANTALLA DE DESCRIPCIÓN

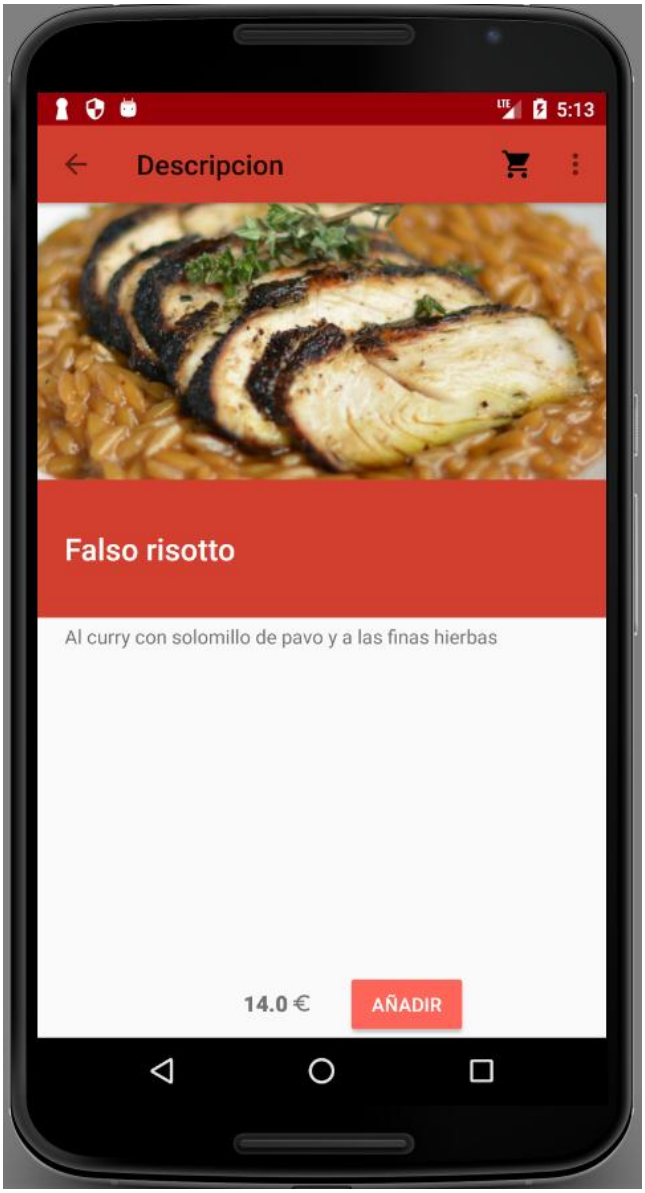

Figura 42. Pantalla de descripción

En esta pantalla (un ejemplo se muestra en la Figura 42) el usuario ve información sobre el producto que ha elegido, y puede añadirlo al carrito si lo desea y tiene una reserva activa. Desde aquí el usuario sólo puede navegar de vuelta a la pantalla de productos.

6.7 BARRA SUPERIOR DE OPCIONES

Este es un elemento común de muchas aplicaciones Android. Esta barra muestre el nombre de la pantalla y ofrece una serie de opciones que puede ir variando dependiendo del estado en el que se encuentre la aplicación. Este es un elemento que es familiar al usuario, y que le ofrece opciones importantes en las distintas pantallas.

6.7.1 OPCIÓN DE "MIS RESERVAS"

Esta opción se muestra desde todas las pantallas, exceptuando ella misma. Esta opción se encuentra oculta y se accede desde el menú expandible (3 puntos situados en el extremo derecho de la barra). Esta pantalla muestra una lista con las reservas del usuario, separando la reserva activa, si existe (en la que se encuentra sentado). En la Figura 43 se muestra un ejemplo de esta pantalla.

Desde aquí, el usuario ver la mesa en la que se encuentra sentado, y las reservas que tiene el futuro. Además, puede seleccionar una reserva de la sección "Otras Reservas", y esto le muestra dos 3 opciones:

- Sentarse: si el usuario no se tiene una reserva activa, quedan menos de 30 minutos para su reserva, y la mesa está libre, el usuario puede sentarse en la mesa.
- Productos: el usuario puede hacer un pedido para una hora en la que tenga la mesa reservada. Esta opción le lleva a la pantalla de productos, desde la que podrá añadir productos para un pedido.
- Cancelar: esta opción permite al usuario cancelar una reserva que no está activa aún.

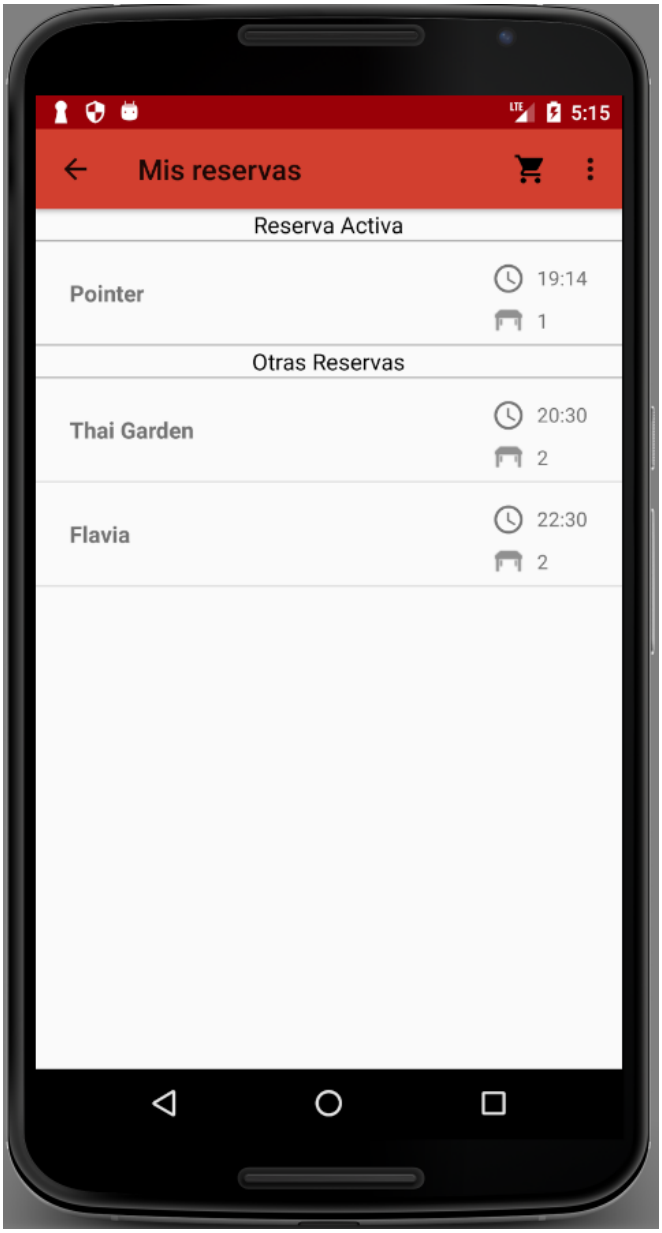

Figura 43. Pantalla de reservas

6.7.2 PANTALLA DE CHECK OUT

Esta opción se muestra visible (si hay hueco) en la barra mediante un icono de un carrito de la compra, y está disponible desde cualquier pantalla, excepto ella misma. Al pulsar esta opción, se muestra la pantalla de *"Check Out"*, desde la que el usuario puede ver los

productos que ha confirmado y el pedido que tiene pendiente, así como el precio. Un ejemplo de esta pantalla se muestra en la imagen siguiente (Figura 44):

Figura 44. Pantalla de check out

Desde esta pantalla, el usuario puede:

- Eliminar productos del pedido actual, que aún no han sido confirmados al restaurante.
- Pedir: esta opción confirma el pedido actual al restaurante, para que sea procesado.
- Check out: esta opción permite pedir la cuenta o pagar para levantarse de la mesa.

Además, el usuario puede ver el precio total de los productos que tiene confirmados, y el precio que suman todos los productos que forman parte del pedido actual.

6.7.3 SOLICITAR CAMARERO

Esta opción está disponible desde cualquier pantalla, y se accede a ella desde el menú desplegable. Esta opción es la que elige el usuario si desea solicitar que un camarero vaya a la mesa en la que se encuentra.

Esta opción no muestra ninguna pantalla, y sólo si el usuario se encuentra con una reserva activa (sentado en una mesa), genera una petición que es procesada por el restaurante, que se encargará de avisar a un camarero que se encuentre disponible para atender al cliente. Esta opción se muestra en la siguiente captura (Figura 45):

Figura 45. Solicitar camarero

6.8 ANÁLISIS

Este proyecto se centraba en diseñar e implementar una versión de prueba que ofreciera las funcionalidades básicas que tendrá esta plataforma en el lado del cliente. Con esta versión se busca probar el interés que puede tener el cliente medio de un restaurante en una aplicación de estas características, para servir como una prueba de concepto. Es importante

conocer el interés del usuario ya que ser de gran utilidad (tanto para el restaurante como para el propio usuario), es importante que esta aplicación tenga una base extensa de usuarios, que la utilicen regularmente.

Los objetivos básicos que se han cumplido son:

- El sistema permite la creación de cuentas de usuario, y permite a los clientes iniciar sesión con su cuenta personal, incluso su cuenta de Google, así como cerrar la sesión cuando lo desee.
- El sistema ofrece una lista con los distintos restaurantes, muestra información básica sobre ellos, como el tipo de comida que ofrecen, horario, dirección, teléfono, etc.
- El sistema permite hacer reservas, ya sea en el momento o para el futuro, y en este caso, cancelarlas si se desea.
- El usuario puede ver los productos que ofrece el restaurante, y permite hacer pedidos.
- El usuario puede hacer un pedido para que se le tenga listo a una hora determinada. Así, puede pedir comida antes de encontrarse físicamente en el restaurante para que esté lista cuando llegue, reduciendo el tiempo de esperas.

El objetivo inicial que no se ha podido cumplir es el de ofrecer las ofertas y promociones de forma dinámica. Esto se ha debido a la necesidad de diseñar un sistema que permita a los administradores de los restaurantes interactuar de forma sencilla con la plataforma.

Por otra parte, tras la implementación del prototipo se ha concluido que pronto, conforme evolucione y crezca el número de usuarios de la aplicación, será necesario utilizar un servidor dedicado para el sistema con mayores prestaciones la Raspberry Pi 3. A pesar de ser una buena plataforma de pruebas, la Raspberry Pi 3 tiene unas características limitadas que no son suficientes para dar servicio a un sistema cuyo objetivo es llegar a miles, y potencialmente millones de clientes.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

La popularidad de las aplicaciones móviles ha crecido exponencialmente en los últimos años, y esta tendencia no muestra signos de cambio en los años venideros. Los dispositivos móviles se han convertido en un punto más de interacción entre empresas y clientes. A pesar de que tradicionalmente se creía que estas aplicaciones sólo beneficiaban a mercados muy específicos, como redes sociales o transporte, recientemente se está descubriendo que hay otras industrias que pueden sacar provecho de la movilidad y conexión permanente que estos dispositivos ofrecen.

Uno de estos mercados es el de la restauración. Los restaurantes tienen una necesidad de atraer a nuevos clientes y de reducir sus costes operativos, que pueden ser cubiertas por el servicio adecuado, y las aplicaciones móviles son unas plataformas ideales para ello. Por otra parte, los usuarios buscan comodidad y rapidez, y las funcionalidades online móviles cubren estas dos necesidades.

La solución propuesta responde a estas necesidades y ofrece al restaurante medio una oportunidad de competir con grandes cadenas y franquicias, a un nivel que no ha sido posible hasta ahora. Por otra parte, esta aplicación hace que el servicio que recibe el cliente sea más rápido y cómodo. Esto puede fomentar el consumo, incrementando el consumo anual por cliente.

Por último, es conveniente destacar que existen otras aplicaciones y otras industrias que pueden ser beneficiados por un sistema de características similares. Otros sectores de la hostelería, como la industria hotelera o bares tienen necesidades similares. La gran mayoría de los comercios pueden beneficiarse de un sistema que permita ofrecer sus servicios de forma tanto remota como física, y que haga la experiencia del cliente más rápida y cómoda. Los smartphones y aplicaciones móviles ofrecen un nuevo punto de interacción con el cliente cuyas posibilidades están aún por descubrir y explotar.

7.1 TRABAJOS FUTUROS

Una vez se dispone ya de un prototipo de la plataforma que puede ser presentada a los primeros clientes, los siguientes pasos son:

- Encontrar restaurantes que estén interesados en participar en este servicio: encontrar restaurantes que estén dispuestos a incluir este sistema dentro de su sistema de gestión a pesar de estar en desarrollo aún, es importante para poder probar el potencial real de la plataforma.
- Buscar usuarios que prueben la aplicación para, encontrar fallos para poder mejorarla y ampliarla.
- Realizar cambios, corregir fallos e implementar mejoras y nuevas funcionalidades.
- Desarrollar una versión de la plataforma que sirva de interfaz para los restaurantes, que les permita ver los datos sobre las ventas, los clientes, los productos, etc. También les permite actualizar su información, gestionar los productos y publicar ofertas y promociones.

Capítulo 8. BIBLIOGRAFÍA

[1] Ditrendia: "Informe ditrendia 2016: Mobile en España y en el Mundo". Julio 2016. http://www.amic.media/media/files/file_352_1050.pdf

[2] Thompson, E: "App Annie 2016 Retrospective – Mobile's Continued Momentum". Enero 2017. https://www.appannie.com/en/insights/market-data/app-annie-2016-retrospective/

[3] Wikipedia: "Android Runtime". https://es.wikipedia.org/wiki/Android_Runtime

[4] Android Developers: "System Architecture". https://developer.android.com/images/system_architecture.jpg

[5] Goasduff, L; Forni, Amy A.: "Market Share: Final PCs, Ultramobiles and Mobile Phones, All Countries, 4Q16." Febrero 2017. http://www.gartner.com/document/3606031

[6] Guenveur, L; "Venta de smartphones: Android alcanza su máximo histórico en España", Kantar Worldpanel. Junio 2016. http://es.kantar.com/tech/m%C3%B3vil/2016/junio-2016-cuota-de-mercado-de-smartphones-en-espa%C3%B1a-abril-2016/

[7] Statista. 2015. https://www.statista.com/statistics/534123/e-commerce-share-of-retail-sales-worldwide/

[8] Nominalia. "Estudio sobre mCommerce y hábitos de consumo online". Junio 2014. https://www.dada.eu/wp-content/uploads/2016/06/140616-Los-espa%C3%B1oles-preferimos-comprar-online-desde-casa.pdf

[9] ZIVELO. "McDonald's Self-Service Ordering Kiosk by ZIVELO". https://es.pinterest.com/pin/390335492689260034/

[10] Mapal Software. "Estadísticas sobre el sector de la restauración en España". Enero 2016. http://mapalsoftware.com/estadisticas-sobre-sector-de-la-restauracion-espana/

[11] Android Developers. https://developer.android.com/index.html

[12] *Navasmdc, vajro*. Material Design Library. https://github.com/vajro/MaterialDesignLibrary

[13] Zebra Crossing "ZXing", https://github.com/zxing/zxing

[14] Historial de versiones de Android, Wikipedia, https://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android

ANEXO A

HISTORIAL DE VERSIONES DE ANDROID

El sistema Android ha sido actualizado frecuentemente desde su presentación en 2008. Es conocido además por asignar un nombre de un dulce en orden alfabético. En estas nuevas versiones, se suelen corregir errores e introducir nuevas funcionalidades, mejoras de rendimiento y nuevos elementos gráficos que mejoran la experiencia del usuario. En la siguiente tabla (Tabla 3) se ven las distintas versiones del sistema desde su lanzamiento:

Nombre	Número de versión	Lanzamiento	Nivel de API
Android 1.0	1.0	Septiembre 2008	1
Android 1.1	1.1	Febrero 2009	2
Cupcake	1.5	Abril 2009	3
Donut	1.6	Septiembre 2009	4
Eclair	2.0-2.1	Octubre 2009	5-7
Froyo	2.2-2.2.3	Mayo 2010	8
Gingerbread	2.3-2.3.7	Diciembre 2010	9-10
HoneyComb	3.0-3.2.6	Febrero 2011	11-13
Ice Cream Sandwich	4.0-4.0.4	Octubre 2011	14-15
Jelly Bean	4.1-4.3.1	Julio 2012	16-18

KitKat	4.4-4.4.4	Octubre 2013	19-20
Lollipop	5.0-5.1.1	Noviembre 2014	21-22
Marshmallow	6.0-6.0.1	Octubre 2015	23
Nougat	7.0-7.1.2	Agosto 2016	26

Tabla 3. Historial de versiones Android. Fuente: Wikipedia [14]