Desarrollo de Aplicación Móvil para la gestión de comandas de un bar/restaurante por parte del cliente

Trabajo de Fin de Grado

INGENIERÍA INFORMÁTICA

VNiVERSiDAD D SALAMANCA

Julio de 2022

Autor

Pablo Díaz Rodríguez

Tutor

Juan Francisco de Paz Santana

D. Juan Francisco de Paz Santana, personal del departamento de Informática y Automática de la Universidad de Salamanca.

CERTIFICA:

Que el trabajo titulado "**Desarrollo de Aplicación Móvil para la gestión de comandas de un bar/restaurante por parte del cliente**" ha sido realizado bajo su dirección por **Pablo Díaz Rodríguez.**

Y para que así conste a todos los efectos oportunos.

En Salamanca, 6 de Julio de 2022

DE PAZ SANTANA, JUAN FRANCISCO (AUTENTICACIÓN)

Firmado digitalmente por DE PAZ SANTANA, JUAN FRANCISCO (AUTENTICACIÓN)
Nombre de reconocimiento (DN): c=ES, serialNumber=76125754D, sn=DE PAZ, givenName=JUAN FRANCISCO, cn=DE PAZ SANTANA, JUAN FRANCISCO (AUTENTICACIÓN)
Fecha: 2022.07.07 09:16:10 +02'00'

Agradecimientos

Transmitir mi más sincero agradecimiento a todos aquellos que me han ayudado a lo largo de esta etapa y han colaborado en el desarrollo del proyecto.

En primer lugar, a mi tutor, Juan Francisco de Paz Santana, por su ayuda a la hora de establecer la planificación del proyecto y por las correcciones realizadas sobre el mismo.

En segundo lugar, a mi familia, mi madre María Isabel, mi padre Luis, por todas las lecciones que me han dado a lo largo de mi vida para ser la clase de persona que quiero ser, y por animarme a aspirar siempre a más.

A mis amigos por apoyarme no sólo en el desarrollo del trabajo, sino también en el resto de las etapas a lo largo de mi vida.

También agradecer al conjunto de profesores del I.E.S. Venancio Blanco, que despertaron mi pasión por el mundo de la informática, y a los profesores de la USAL, por enseñarme a querer perfeccionar siempre mi trabajo.

A todos ellos, mil gracias.

Resumen

El trabajo de fin de grado presentado a continuación se centra en la creación de un sistema, a partir del cual se crean y gestionan comandas en un establecimiento mediante el uso de un dispositivo móvil.

El objetivo principal del proyecto es facilitar la organización de un restaurante o bar mediante la digitalización de sus servicios. Esto afecta tanto a clientes como a los empleados del establecimiento, simplificando la interacción entre ambos usuarios.

Por un lado, los clientes serán capaces de pedir una serie de platos y bebidas sin necesidad de atención por parte de alguno de los camareros del establecimiento. Para poder realizar comandas, el usuario (previamente autenticado mediante el servicio de Google SignIn) deberá escanear el código QR asociado a la mesa. Este código QR contendrá el identificador de la mesa en base de datos, mediante el cual se obtendrán los datos del establecimiento para poder empezar a pedir platos y bebidas.

Los empleados podrán recibir información sobre las comandas realizadas en las distintas mesas del local, así como comprobar el estado de cada mesa del establecimiento. Para mayor control de los datos ofrecidos por el negocio, todo usuario que sea empleado de un determinado restaurante o bar podrá modificar información en respecta a los platos y mesas que se encuentran dados de alta.

El proyecto se basará en un sistema de consultas realizadas a una base de datos remota. Mediante un conjunto de colecciones establecido en el servicio Google Firestore y una serie de objetos de escucha, se podrá obtener información actualizada sobre las comandas realizadas e información sobre las mesas y establecimientos, y así cumplir con el objetivo establecido para este proyecto, el cual se trata de ofrecer datos en tiempo real. El coste de mantenimiento del proyecto dependerá de la cantidad de consultas realizadas a la base de datos Firestore.

Por último, se ofrecerá al cliente opción de pago con tarjeta mediante la plataforma Stripe, la cual solicita los datos del cliente y realiza la transferencia a la cuenta bancaria establecida.

Palabras clave: Código QR, Flutter, FireStore, Authentication, Storage, Android, Provider, MVVM, Stripe, base de datos en tiempo real, objeto de escucha, orden, comanda, platos, bebidas, establecimiento.

Abstract

The dissertation presented below focuses on the creation of a system, from which orders are created and managed in an establishment through the use of a mobile device.

The main objective of the project is to make the organization of a restaurant or bar easier by digitizing its services. This affects both customers and employees of the establishment, simplifying the interaction between both users.

On the one hand, customers will be able to order a series of dishes and drinks without the need for attention from any of the waiters in the establishment. To be able to place orders, the user (previously authenticated through the Google SignIn service) must scan the QR code associated with the table. This QR code will contain the table's identifier in the database, through which the establishment's data will be obtained so the user could start ordering dishes and drinks.

Staff will be able to receive information about the orders placed at the different tables in the establishment, as well as being able to check the status of each table in the establishment. For greater control of the data offered by the business, any user who is an employee of a particular restaurant or bar can modify information regarding the dishes and tables that are registered.

The project will be based on a system of queries made to a remote database. Through a set of collections established in the Google Firestore service and a series of listening objects, it will be possible to obtain updated information about the orders made and information about the tables and establishments, and thus meet the objective established for this project, which is to provide data in real time. The maintenance cost of the project will depend on the number of queries made to the Firestore database.

Finally, the customer will be offered the option of paying by credit card through the Stripe platform, which requests the customer's data and makes the transfer to the established bank account.

Keywords: QR Code, Flutter, FireStore, Authentication, Storage, Android, Provider, MVVM, Stripe, Realtime Database, Listener, order, order line, dishes, drinks, establishment.

Tabla de contenido

Tabla de figuras

Listado de tablas

1. Introducción

Digitalización del sector servicios

El sector de hostelería es uno de los grandes afectados durante los 2 últimos años. Tanto ha sido que la digitalización de éstos se ha agilizado debido a la necesidad de supervivencia. Como se puede ver en la Figura 1. Gráfico de Facturación en sector de la Hostelería, el año 2020 fue nefasto para la hostelería, registrando una caída de más del 50% en facturación y la clausura de al menos 8.000 establecimientos.

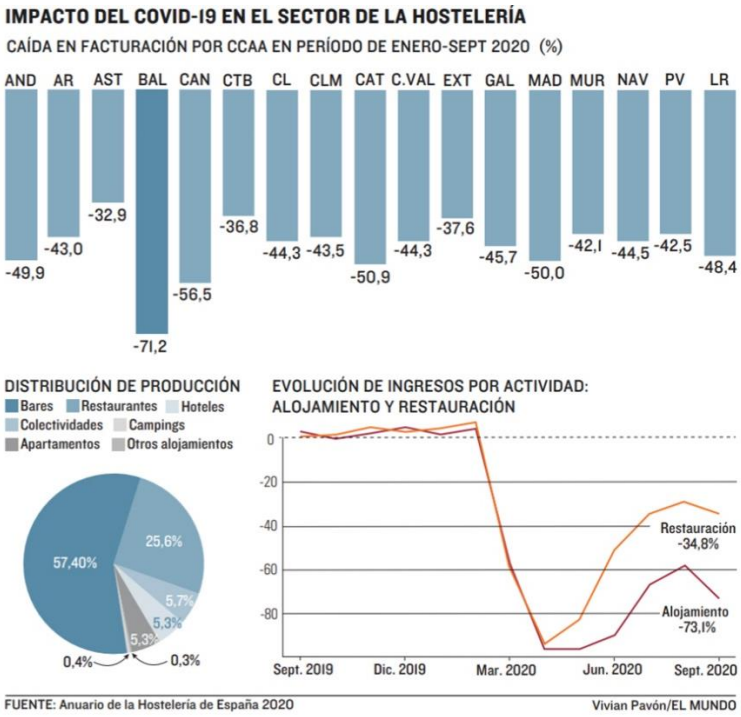

Figura 1. Gráfico de Facturación en sector de la Hostelería

A finales de 2021 el sector consiguió aumentar en un 33% su facturación, pese a la paralización que hubo en navidades. Esto indicaba el inicio de la recuperación en la hostelería [1]. Muchos expertos son los que indican la recuperación del sector durante el año presente, calculando la esperada "normalidad" en 2023.

Según fuentes consultadas por *Last.app,* "La adaptación del sector a los nuevos métodos de **servicio digitalizados**, explotados a raíz de la pandemia, ha sido una de las claves de su **supervivencia** y posible **recuperación**" [2]. En 2021 la hostelería incrementó su digitalización en un 58%, y actualmente, entre todos los negocios españoles, el 38% de ellos facturan más de 25% de sus ventas digitalmente, según un estudio de *Meta*.

Dentro del conjunto de soluciones digitales, una de las primeras en llegar fue el código QR. Este código representa una cadena de texto que permite ser escaneada mediante la cámara de cualquier dispositivo móvil actual. Debido a las normas de higiene y salud, se desaconsejó el uso de elementos como servilleteros, barra, carta… lo que supuso un desafío para los establecimientos de hostelería a la hora de presentar servicios básicos, entre ellos la necesidad de consultar los platos que se ofrecen en carta mediante códigos como en la *Figura 2. Mesa con código QR.*

Figura 2. Mesa con código QR

Otra opción que muchos establecimientos contemplaron ha sido la implementación de nuevas alternativas, como puede ser el reparto a domicilio. Esto no es algo nuevo, ya que existía antes de la pandemia. Debido al descenso de ingresos, los restaurantes y bares han decidido ampliar su red de clientes mediante opciones de venta por entrega mediante aplicaciones como *JustEat*, *UberEats*, *Glovo*… Parte de este proceso de digitalización permite al sector de hostelería recuperarse del duro golpe que ha supuesto la pandemia.

Efecto en la población

Debido a esta necesidad de digitalizar los negocios, la población se ha ido acostumbrando al uso de estas nuevas tecnologías. No sólo supone un avance para el propio establecimiento, sino que ofrece facilidad a los usuarios para la obtención de datos del establecimiento.

Cada vez son más personas las que aprenden a escanear el código QR, navegar mediante su dispositivo móvil para obtener la información que busca sobre el restaurante/bar, utilizar aplicaciones propias del negocio, como se puede ver en la *Figura 3. Carta Digital.* Gracias a este aprendizaje se pueden evaluar nuevas opciones de digitalización para cualquier establecimiento, de manera que no sólo se les facilite la vida a los clientes, sino que también pueda servir de gran ayuda para el empresario y trabajadores.

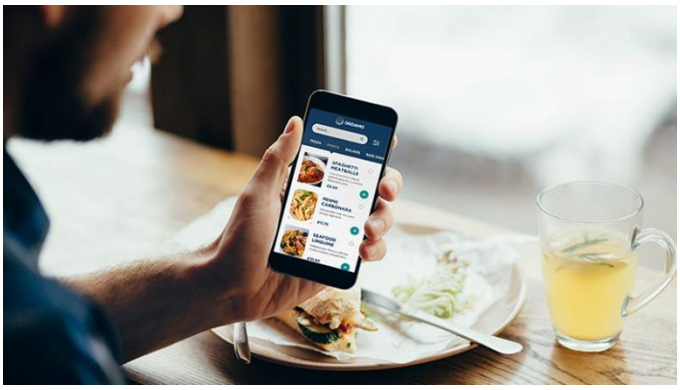

Figura 3. Carta Digital

Sistema propuesto

Una de las mejoras que se plantean es la posibilidad, por parte de los clientes y sin intermediarios (camareros), pedir una serie de platos y bebidas al restaurante o bar. El planteamiento inicial es el que tiene cualquier aplicación de entrega de comidas, añadiendo la posibilidad de gestionar todas las comandas que se han realizado en el establecimiento.

No obstante, la gestión de comandas mediante dispositivos tecnológicos ya existía antes del incremento de digitalización en el sector hostelería. Mediante aparatos como la *PDA* (Figura 4. Gestión con PDA) se realizaban las comandas, permitiendo al personal del establecimiento conocer en todo momento los platos y bebidas asignados a cada mesa. Estos dispositivos no son de uso público, ya que son asignados a los camareros para la gestión correctamente del establecimiento.

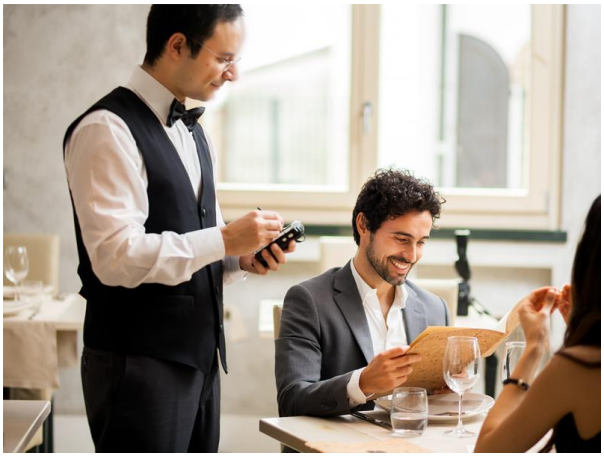

Figura 4. Gestión con PDA

El proyecto que se plantea procesa la interacción de cliente y personal del restaurante en una misma aplicación que sirva para ambos, separándose por funcionalidades:

- El cliente será capaz de pedir platos y bebidas mediante la aplicación, sin necesidad de interactuar con uno de los camareros del establecimiento.

- El personal del establecimiento podrá visualizar la información enviada por los clientes para su preparación.

Esta aplicación será diseñada inicialmente para dispositivos Android permitiendo la escalabilidad hacia Apple, ya que está desarrollada en un lenguaje común para ambos sistemas.

Dentro de este documento se presentarán los aspectos importantes del mismo, indicando las fases de desarrollo por las que pasa. En los Anexos I, II, III, IV, V y VI adjuntos se encuentran estas fases con un mayor grado de detalle.

Seguidamente se detallarán los capítulos a desarrollar en la memoria:

Capítulo 2. Objetivos: Se explicarán y describirán los principales objetivos del sistema a desarrollar, así como los objetivos personales del alumno.

Capítulo 3. Conceptos teóricos: En este apartado se detallarán los conocimientos previos necesarios a la hora de desarrollar el proyecto.

Capítulo 4. Técnicas y herramientas utilizadas: Se recopilará información sobre todas las herramientas, aplicaciones y técnicas que se hayan utilizado en la realización del proyecto.

Capítulo 5. Aspectos del desarrollo: Se indicarán los puntos más relevantes del desarrollo, así como propuestas iniciales y limitaciones asociadas a dichas propuestas, creando un modelo final que difiere al diseño original.

Capítulo 6. Resumen de funcionalidad: Se explicará la interacción existente por parte del cliente y del administrador, describiendo los cambios que se producen en el sistema para cada acción.

Capítulo 7. Conclusiones y líneas de trabajo futuras: Se recopilará la información obtenida durante el desarrollo del proyecto y se realizará un análisis sobre el sistema final propuesto y las conclusiones obtenidas. Adicionalmente se propondrán una serie de implementaciones con el objetivo de mejorar el producto.

Glosario y bibliografía: Se explicarán los conceptos técnicos que se han utilizado en el desarrollo de la memoria y las fuentes utilizadas para el proyecto.

2. Objetivos

En este apartado se propondrán los objetivos establecidos para el proyecto y las funcionalidades a desarrollar.

2.1. Objetivos del sistema

El objetivo principal del proyecto es ofrecer un sistema que sea capaz de almacenar y gestionar información relativa a los pedidos que realiza un cliente en un restaurante o bar.

2.1.1. Obtener información a partir del código QR

Un cliente que se haya sentado en la mesa podrá escanear el código QR adjunto a ella. Este código de mesa tendrá un restaurante o bar asociado, del cual se obtendrán los datos asociados a dicho establecimiento. La mesa será marcada como "Ocupada" para que cualquier cliente pueda comprobar el número de mesas disponibles en el establecimiento.

2.1.2. Realizar comandas

El objetivo principal de la aplicación es ofrecer al cliente una nueva experiencia a la hora de pedir en un restaurante o bar. El usuario se encargará de añadir los platos y bebidas del menú mediante la aplicación, y el sistema se encargará de enviar estos datos para el conocimiento del personal. Estas comandas guardarán información sobre la cantidad de platos y bebidas que se han pedido y comentarios sobre la preparación de alguno de los platos.

2.1.3. Administrar comandas

El personal del establecimiento podrá, en cualquier momento, revisar el listado de comandas activas. Estas comandas están vinculadas a la mesa que el cliente haya escaneado. Una vez se hayan finalizado o cancelado, el encargado de revisar dichas comandas indicará su finalización

2.1.4. Administrar establecimiento

Las personas pertenecientes a un negocio podrán modificar la información relativa al restaurante o bar que tienen asociado, incluyendo cambios en los platos (crear, modificar y borrar) y configuración de mesas (si una mesa debiera aparecer como utilizada o no). Una vez modificada la información, el resto de los usuarios tendrán disponibles las modificaciones al momento.

2.1.5. Ofrecer establecimientos cercanos

El sistema guarda información relativa a los restaurantes o bares que se han registrado en la base de datos. Esta información se utilizará para ofrecer al usuario un listado de los establecimientos cercanos a él, y la cantidad de mesas disponibles en cada uno de ellos.

Se podrá establecer una ruta hacia cualquiera de los establecimientos registrados mediante Google Maps.

2.1.6. Informar sobre cambios en tiempo real

Una de las principales características del sistema desarrollado es la capacidad de actualización de los datos. Cuando los datos son guardados en una BBDD en la nube, el sistema deberá notificar a todos los usuarios pertinentes sobre dichos cambios con rapidez. Esto permite que cualquiera de los trabajadores del establecimiento sea notificado en el momento en el que algún cliente haya realizado una comanda.

De igual forma, si durante el servicio se debe corregir el precio de un plato o información sobre alguno de sus componentes, se deberá notificar a los usuarios de dichos cambios, ofreciendo en todo momento una carta del establecimiento acorde a las últimas modificaciones que se hayan realizado en ella.

Esto permite la modificación simultánea por parte del cliente y camarero de una comanda. El usuario podrá ser ayudado por cualquier personal del establecimiento a la hora de realizar una comanda sin perder el progreso de ella, permitiendo al camarero realizar acciones que el usuario no sea capaz de hacer.

Al permitir la modificación simultánea de una comanda también se incluye la posibilidad de que varios usuarios escaneen el mismo código, de manera que cada persona añade a la comanda los platos y bebidas que quiera, sin necesidad de centralizar la acción de realizar comanda a una única persona encargada de preguntar a todos los usuarios sentados en la mesa por sus platos.

2.1.7 Realizar pagos desde la aplicación

Mediante alguna de las plataformas de pago que sean compatibles con el proyecto, se ofrecerá la opción de pagar mediante la aplicación, sin necesidad de intermediarios (camareros) para la realización del pago.

2.2 Objetivos personales

El desarrollo ha sido, desde que empecé a programar, una de las cosas que más me ha llegado a interesar. Durante años he estado trabajando con aplicaciones móviles, lo cual me ha llevado a generar una afición dentro de todas las ramas existentes en la Informática.

Dentro de los objetivos que se pueden plantear, uno de los más importantes ha sido el aprendizaje de lenguajes de programación flexibles como Flutter, permitiendo crear aplicaciones funcionales tanto para Android como para Apple.

Otra de las cosas con las que quería trabajar eran servicios ofrecidos por Google. Dentro de todos, FireStore es el que más atención me llamó, siendo catalogado por un amplio grupo de personas como el mejor a la hora de establecer aplicaciones en tiempo real.

Sobre el proyecto, siempre he pensado en la mejora digital de ciertos servicios cotidianos. Entre ellos me interesé en la gestión de un bar, ya que creo que muchos de los errores que se cometen podrían ser evitados con un control adecuado del ambiente. Esto no solo afecta a los trabajadores, sino también a los clientes. ¿Cuántas veces se siente un cliente invisible, buscando como loco a un camarero?

Por parte de cualquier cliente, la posibilidad de llegar a un establecimiento, realizar una comanda y quedarse tranquilo es el principal objetivo.

Asimismo, se facilita el trabajo de los camareros de un restaurante o bar. No es necesario estar pendiente de todas y cada una de las mesas, sino únicamente de aquellas que necesiten de su ayuda.

3. Conceptos teóricos

Dentro de este apartado se hará una breve descripción de los conceptos necesarios para entender el desarrollo del proyecto, y el porqué de la toma de decisiones a la hora de escoger entre varios sistemas disponibles.

3.1. Código QR

Un código QR (Quick Response code) es una matriz de puntos que guarda información. Es la evolución del código de barras, impulsada por la necesidad de guardar una cantidad mayor de información. Cada código QR generado permite almacenar datos de hasta 4.293 caracteres alfanuméricos.

La matriz es leída desde cualquier dispositivo móvil que disponga de un lector de QR. Mediante los 3 cuadros de las esquinas permite detectar la posición del código, la cual se indicará al lector QR. En la Figura 5. Estructura de un código QR se pueden ver los distintos elementos de un código QR. El formato del dato devuelto será una cadena alfanumérica con la información.

Una de las principales características de los códigos QR y uno de los objetivos de su creación es la alta velocidad de lectura del contenido de cada código.

Su origen se centra en Japón, fueron creados en el 1994 y en el año 2000 se aprobó como estándar internacional ISO, siendo el código 2D más utilizado en la actualidad de entre los existentes.

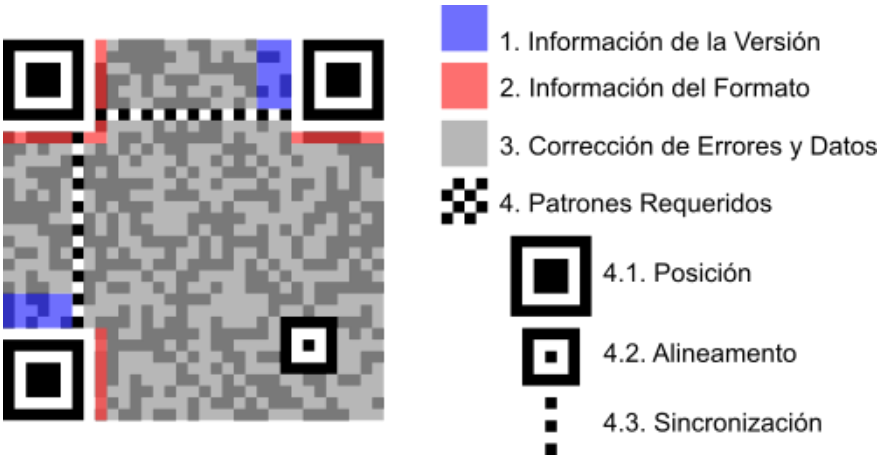

Figura 5. Estructura de un código QR

3.2. Base de datos relacional y no relacional

Una base de datos relacional se compone de varias tablas relacionadas entre sí. Se basan en el modelo relacional, en el que cada fila tiene un registro con ID único llamado "primary key", atributos de la misma tabla y registros con ID externo llamados "foreign-key" relacionando dicha fila de la tabla con otra fila de una tabla distinta, como se puede ver en la Figura 6. Relacional.

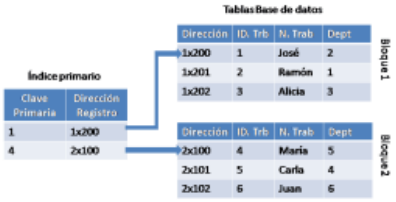

Figura 6. Relacional

Una base de datos no relacional guarda la información de manera no estructurada, por lo que no requieren estructuras fijas como tablas. También son llamadas NoSQL, debido a que no utilizan el lenguaje SQL como herramienta de consulta (únicamente como herramienta de apoyo). Al no estar estructuradas son más flexibles al crear nuevos esquemas de información y ofrecen un mayor grado de rendimiento, pero se pierden las propiedades del modelo relacional (atomicidad, consistencia, integridad y durabilidad). Se puede ver su estructura en la Figura 7. No Relacional.

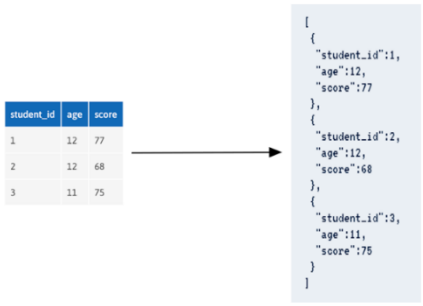

Figura 7. No Relacional

Más adelante en el desarrollo del proyecto valoraremos cuál de los 2 modelos se adapta mejor a nuestro sistema.

3.3. Base de datos en tiempo real

Una BBDD en tiempo real es un sistema de bases de datos que utiliza procesamiento en tiempo real para manejar datos en constante cambio, a diferencia de las bases de datos con datos persistentes. El procesamiento en tiempo real garantiza que la información procesada es devuelta de inmediato para ser tratada por los distintos consumidores que, en nuestro caso, serán clientes y trabajadores del establecimiento, como se puede ver en la Figura 8. Base de datos en tiempo real.

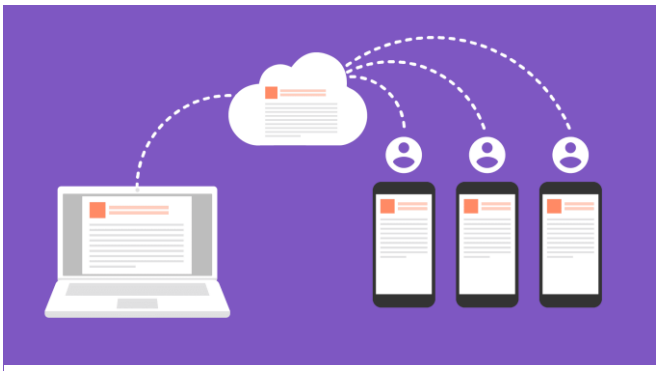

Figura 8. Base de datos en tiempo real

En casos como Google Firebase los datos son guardados como JSON. De esta forma se permite un modelado de datos más simple y flexible.

4. Técnicas y herramientas utilizadas

En este apartado hablaremos del conjunto de herramientas utilizadas para el correcto desarrollo del proyecto.

4.1. VSCode

Visual Studio Code es un editor de código desarrollado por Microsoft. Es totalmente gratuito y de código abierto, lo que permite a los usuarios desarrollar distintas herramientas. En la Figura 9. Visual Studio Code se puede ver el entorno de trabajo y emulador que ofrece.

Incluye soporte para realizar tareas de depuración y permite su extensión mediante complementos, disponibles en un repositorio central. Está basado en Electron, un framework utilizado para Chromium y Node.js.

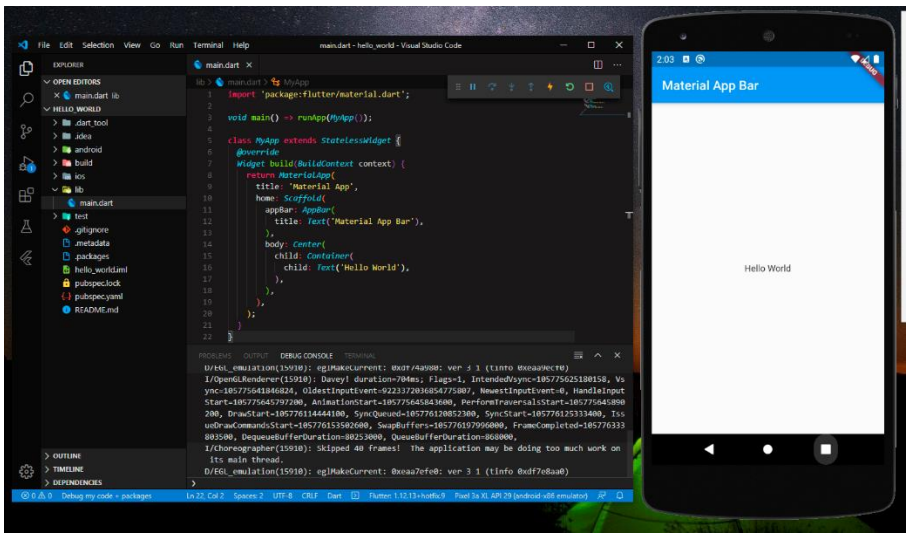

Figura 9. Visual Studio Code

4.2. Paradigma de programación

El desarrollo del proyecto se ha realizado mediante la Programación Orientada a Objetos (POO). Parte del concepto “objetos” como base, quienes almacenan la información en atributos. Introduce conceptos nuevos para solucionar problemas existentes, de los cuales destacan:

- **Clase**: Plantilla para la creación de objetos con determinados atributos y métodos definidos. Crear nuevos objetos se denomina “instanciación de la clase”.

- **Herencia**: Es una cualidad de las clases, en la que una clase B hereda los atributos y métodos de la clase A como si fuesen definidos por esa clase misma. Los métodos privados de la clase A no serán accesibles por la clase B a menos que dispongan de un método público para su acceso.

- **Objeto**: Representa una instancia de una clase. Posee unos atributos y métodos que interaccionan con eventos. Simulan los objetos del mundo real.

- **Métodos**: Funcionalidad añadida a un objeto, cuya ejecución se indica mediante una llamada al mismo. Un método puede generar cambios en las propiedades del mismo objeto o generar eventos.

4.3. Flutter

Flutter es un conjunto de herramientas diseñadas para el desarrollo de aplicaciones hardware o Software, también denominado SDK (*Software Development Kit*, Kit de Desarrollo Software). Es un código fuente abierto para el desarrollo de aplicaciones móvil. Ha sido desarrollado por Google con el objetivo de crear interfaces para aplicaciones en Android, iOS y Web. Su logo se puede ver en la Figura 10. Logotipo Flutter.

Fue presentado por Google en 2015 y lanzado en 2018. Una de las características que ha popularizado Flutter ha sido su velocidad de desarrollo, experiencia nativa y renderización de la interfaz, permitiendo realizar cambios en el código sin necesidad de reiniciar el dispositivo. Utiliza lenguaje dart para el desarrollo de aplicaciones multiplataforma.

Figura 10. Logotipo Flutter

Flutter utiliza el motor gráfico Skia, el cual se encarga de renderizar en 2D el conjunto de elementos gráficos de la aplicación. La capa del motor propio está escrita en C y C++ y la parte de Widgets (conjunto de elementos visuales de la interfaz) principalmente en Dart.

La estrategia de Flutter, “Todo es un widget”, está basada en la programación orientada a objetos. La interfaz de la aplicación cuenta con distintos widgets anidados entre sí, los cuales tienen propiedades modificables para ajustar la apariencia deseada.

La ventaja de esta estrategia es que interactúa entre sí y reacciona a cambios de estado, tanto internos como externos, mediante mecanismos integrados. Todos los elementos de la interfaz incorporan widgets, los cuales permiten la ampliación de funcionalidad, o la opción de crear nuevos mediante el concepto de Herencia.

La principal desventaja de Flutter es que los widgets forman parte del código fuente de la aplicación. Esto provoca que el código resultante sea complicado de entender y fuertemente anidado.

4.4. Lenguajes y formatos

4.4.1. Dart

Dart es un lenguaje de programación desarrollado por Google. Se trata de un lenguaje de código abierto y fue presentado en 2011. Ofrece una alternativa más moderna frente a otros lenguajes como JavaScript, denominándose a sí mismo como "lenguaje estructurado pero flexible".

La sintaxis de Google Dart se asemeja más al lenguaje humano, por lo que lo hace más fácil de leer frente a otros lenguajes. Algunas de las características más importantes son:

- Programación flexible y estructurada: Dart admite proyectos simples desarrollados por una sola persona hasta proyectos complejos y con un mayor grado de desarrollo.

- Lenguaje fácil de aprendizaje: Se trata de un lenguaje que, mediante la propia documentación ofrecida por Google, se considera sencillo y rápido de aprender debido a las facilidades que ofrece.

- Lenguaje basado en POO: Debido al uso de clases, se facilita la reutilización de código en distintos módulos del proyecto y la encapsulación de la información contenida en las clases.

Una de las desventajas de utilizar Dart es que, en la actualidad, no es un lenguaje tan utilizado comparado con lenguajes como Swift, Kotlin, Java… No obstante, se asemeja a los lenguajes Java y C#, por lo que es muy intuitivo si se tiene conocimiento sobre dichos lenguajes. Google continúa mejorando Dart para que incluya características presentes en los lenguajes mencionados.

4.4.2. JSON

JSON (*JavaScript Object Notation*) representa un formato de texto más sencillo de interpretar, con el objetivo de intercambiar datos a través de la red. Se propone como alternativa al ya existente formato XML.

Una de las ventajas de JSON es la sencillez a la hora de escribir un parser (analizador sintáctico) para él. Este formato se puede ver representado en la Figura 11. Formato JSON y es utilizado comúnmente en entornos en los que el intercambio de datos entre cliente y servidor es vital.

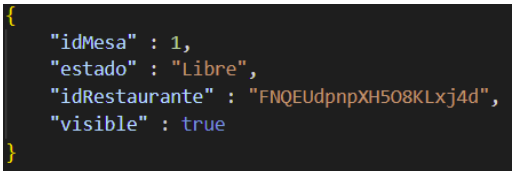

Figura 11. Formato JSON

La sintaxis JSON se basa en 2 elementos: claves (Keys) y valores (Values):

- Las *Keys* son cadenas alfanuméricas que contienen el índice para acceder a un determinado valor.
- Los *Values* son tipos de datos JSON. Estos datos pueden ser alfanuméricos, numéricos, boolean, arrays…

Todos los objetos JSON comienzan y terminan con llaves {}. Cada par de clave/valor es separado mediante una coma, y cada valor es precedido por el carácter dos puntos después de la clave.

4.4.3. XML

Algunos de los archivos de configuración de Flutter están escritos en XML (*eXtensible Markup Language*, Lenguaje de Marcas Extensible). Este lenguaje de marcado es similar a HTML, aunque se diferencia en que no está predefinido, es decir, las etiquetas son definidas por el usuario. Forma un estándar para el almacenamiento e intercambio de información entre plataformas, aunque se puede usar en ficheros de configuración como el de la Figura 12. Formato XML.

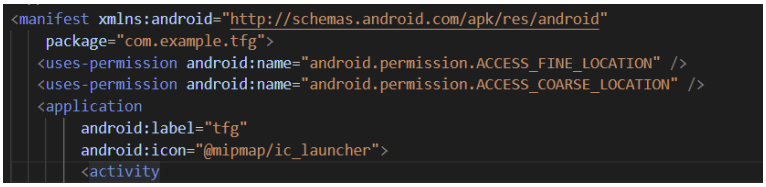

Figura 12. Formato XML

4.5. Bibliotecas

Las bibliotecas utilizadas en Flutter se indican en el archivo pubspec.yaml del proyecto. Éstas son tratadas como dependencias necesarias para el funcionamiento del sistema. Se comentarán las más importantes, ignorando aquellas cuya funcionalidad es meramente visual:

- **firebase_core** y **firebase_cloudstore**: establecen las bases necesarias para conectar la aplicación con una base de datos Google Firestore.

- **mobile_scanner**: permite el escaneo de códigos QR utilizando la propia cámara del dispositivo móvil.

- **geolocator**, **permission_handler**, **maps_launcher**: estos paquetes permiten gestionar los elementos principales para la localización del usuario y restaurantes. Geolocator provee de las coordenadas del usuario y calcula la distancia entre 2 coordenadas, permission_handler

gestiona los permisos necesarios para localizar al usuario y maps_launcher ejecuta Google Maps con una dirección indicada.

- **firebase_auth** y **google_sign_in**: se encargan de establecer las conexiones necesarias para verificar la identidad del usuario. Esta autenticación se realizará mediante una de las herramientas de Firebase utilizando la cuenta Google del usuario.

- **pay**, **flutter_stripe**, **get**: contienen la funcionalidad necesaria para realizar pagos desde la aplicación. Stripe funciona como pasarela de pago y como formulario de petición de datos de tarjeta.

- **provider**: permite la utilización del patrón Provider, enviando notificaciones a todas las vistas que utilicen dicho Provider. Facilita opciones para manejar varios archivos a la vez, de manera que cada Widget utilice el más adecuado para sus datos.

4.6. Herramientas de Firebase

4.6.1. Firebase Cloud Firestore

Firestore es una herramienta ofrecida por Google Firebase. Se trata de un servicio de almacenamiento de datos, en el cual la información se guarda en una base de datos no relacional (NoSQL). Organiza la información en forma de documentos y colecciones, los cuales contienen campos de distintos tipos (String, numéricos, maps, boolean…). Todo ello se gestiona mediante un *Dashboard* ilustrado en la Figura 13. Firestore.

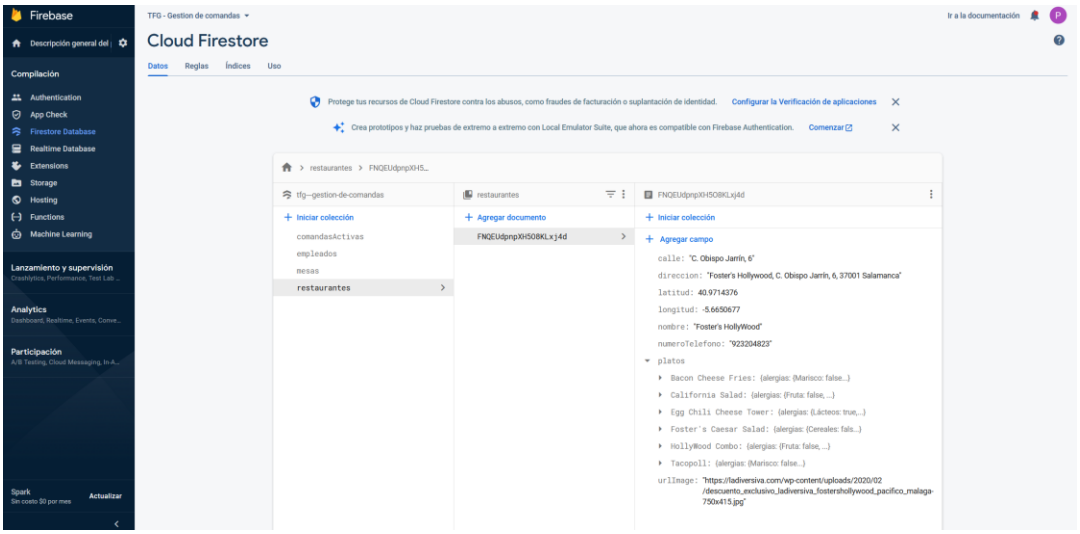

Figura 13. Firestore

La estructura de datos se guarda en formato JSON, pudiendo diferenciar de forma sencilla los elementos que forman la base de datos.

Al tratarse de una base de datos no relacional tiene limitaciones a la hora de realizar consultas complejas como, por ejemplo, permitir un único valor de campo para filtrar las búsquedas. Aun así, nuestro proyecto no requiere de

4.6.2. Firebase Authentication

Firebase permite la autenticación de usuarios mediante unos datos de login. Guarda un registro de las cuentas utilizadas para autenticarse y una cadena alfanumérica que representa el UID del usuario como se muestra en la Figura 14. Firebase Authentication.

Provee de servicios backend fáciles de utilizar y bibliotecas con interfaces ya predefinidas para identificar a los usuarios dentro de la aplicación. Una vez se conozca la identidad de un usuario se podrá guardar información relativa a la interacción con dicha persona.

Esta herramienta permite la autenticación de los usuarios mediante proveedores de inicio de sesión como Google, Facebook, Microsoft… En nuestra aplicación se permitirá el inicio de sesión mediante la cuenta Google.

Una vez obtenido el token de inicio de sesión, se utilizará para relacionar un empleado con un establecimiento, lo cual permitirá al usuario autenticado realizar acciones de administrador sobre el negocio.

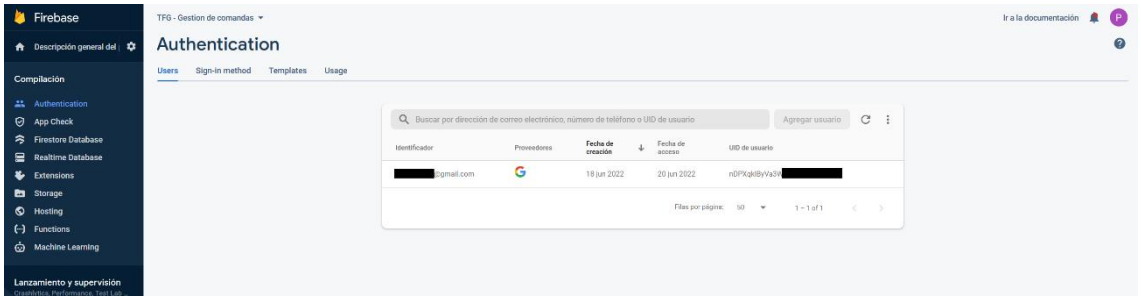

Figura 14. Firebase Authentication

4.6.3. Firebase Storage

Esta herramienta de Firebase proporciona métodos de carga y descarga de archivos para un proyecto creado. Permite la gestión de los archivos desde el Dashboard que ofrece y desde librerías compatibles con Flutter.

En nuestro proyecto se utilizará para almacenar las imágenes que contiene el establecimiento sobre los distintos platos.

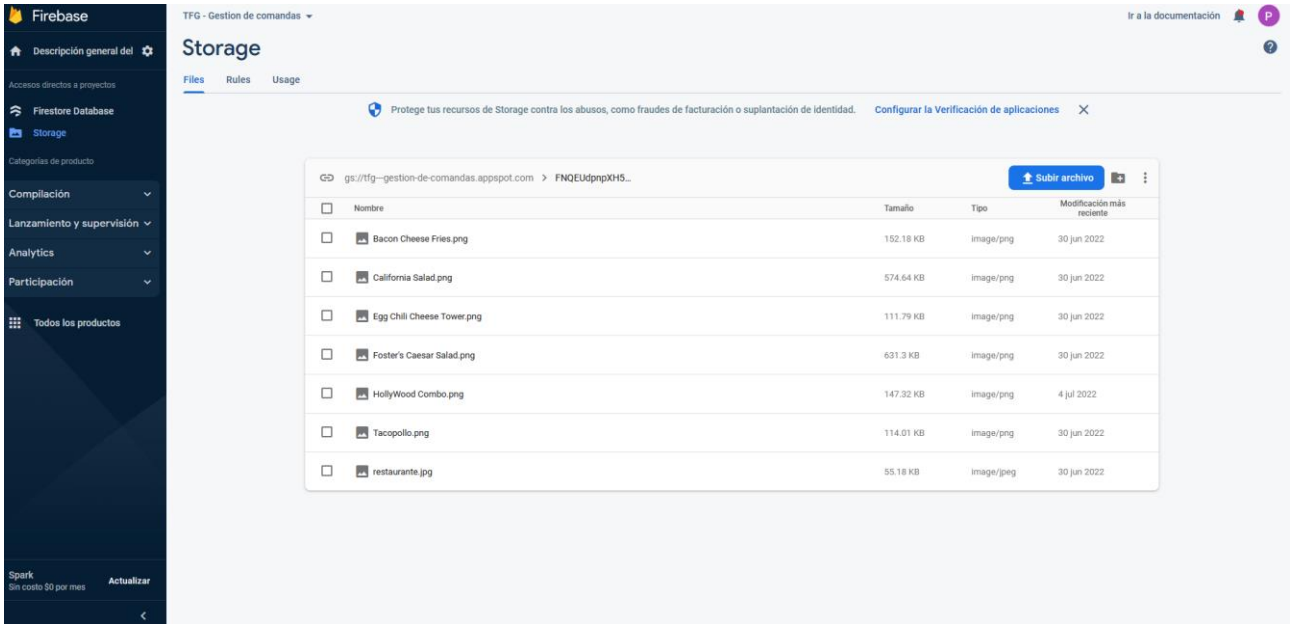

Figura 15. Firebase Storage

4.7. Ingeniería del Software

A continuación, se enunciarán y explicarán brevemente las herramientas utilizadas para las tareas llevadas a cabo en los anexos.

4.7.1. Microsoft Project

Microsoft Project es una herramienta para diseñar planificaciones temporales. Se desarrollan una serie de tareas, a las cuales se les pueden asignar recursos (personal) para estimar su finalización.

Permite establecer rutas críticas en función de la planificación que se ha establecido, así como visualizaciones en el calendario para comprobar el uso y disponibilidad de recursos. El objetivo es destinar el 100% de los recursos durante el desarrollo software completo de una aplicación. En la Figura 16. Microsoft Project se puede ver la estructura para organizar las tareas en un periodo de tiempo.

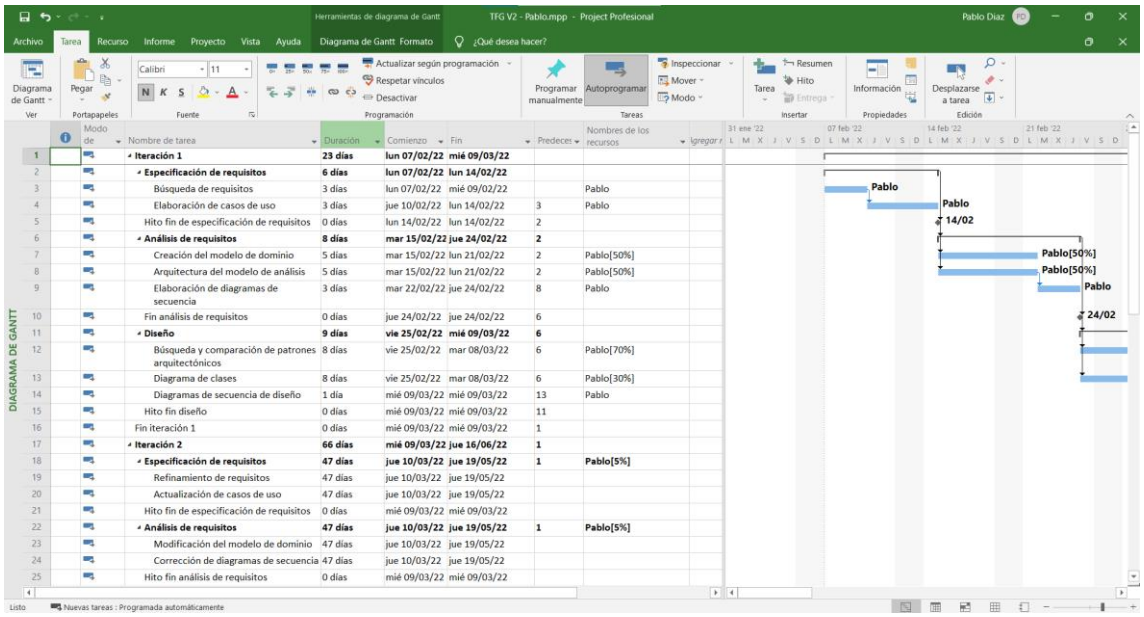

Figura 16. Microsoft Project

4.7.2. REM

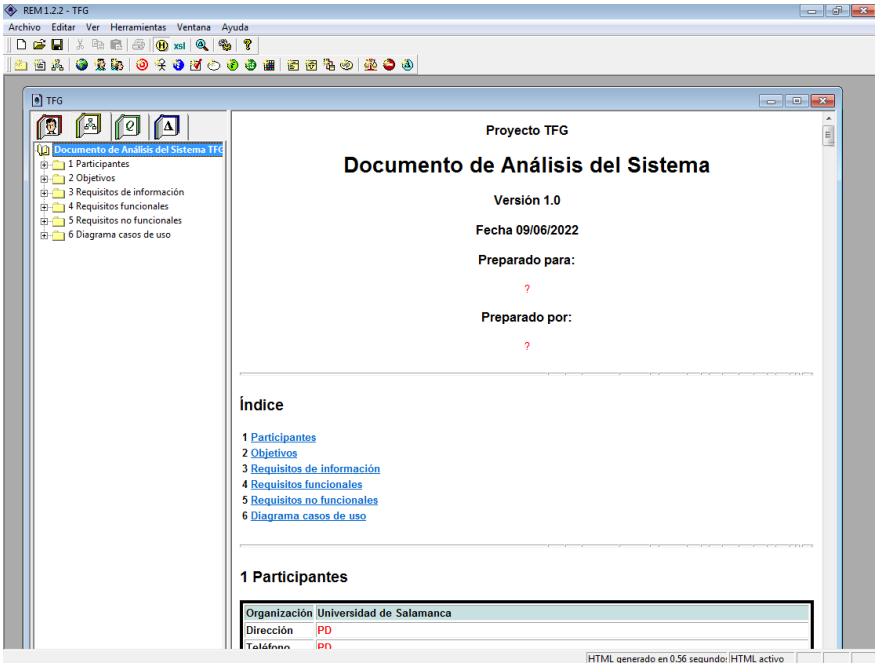

Figura 17. REM

REM (Requirements Management) es una herramienta para la gestión de requisitos totalmente gratuita. Está orientada en la fase de búsqueda de requisitos. Permite establecer categorías y crear documentación con todos los requisitos obtenidos en la fase. En la Figura 17. REM podemos ver su apariencia visual.

4.7.3. EzEstimate

EzEstimate es una herramienta para el cálculo de valores de la estimación de esfuerzo. Permite introducir manualmente los actores que participan en el sistema, los casos de uso y su complejidad y los factores de complejidad técnica y ambientales. A partir de los datos ofrecidos se calculan los puntos de casos de uso para el sistema que hemos establecido, como se puede ver en la Figura 18. EZEstimate.

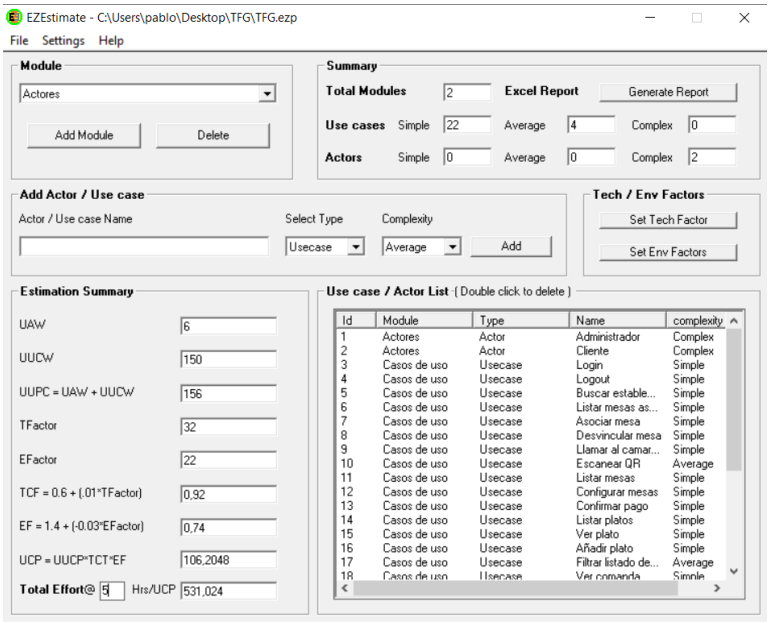

Figura 18. EZEstimate

4.7.4. Draw.io

Draw.io es una herramienta multiplataforma para el diseño de gráficos. Permite la creación de diagramas UML. En concreto, se utilizará para indicar el modelo de dominio y los diagramas de casos de uso y secuencia.

Presenta una barra lateral izquierda para la elección de los elementos a añadir y un cuadro de dibujo en el que se distribuyen todos los elementos. Se puede ver su apariencia en la

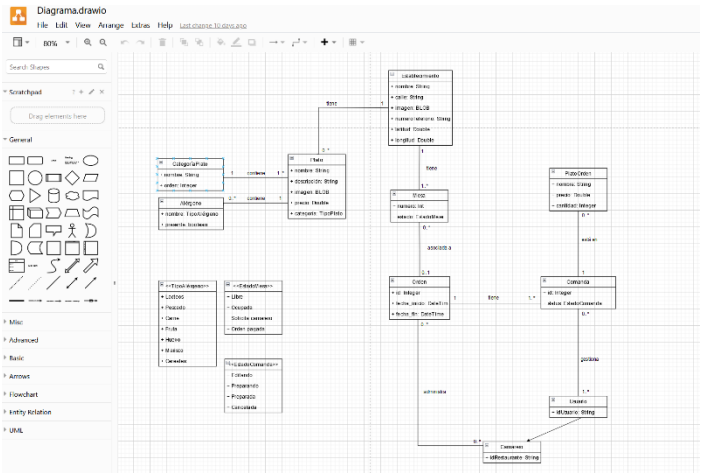

Figura 19. Draw.io.

Figura 19. Draw.io

4.8. Adobe XD

Adobe XD es una herramienta para el diseño de interfaces, enfocada principalmente en la experiencia de usuario. Permite compartir prototipos e interactuar con los elementos del prototipo diseñado, como se puede ver en la Figura 20. Adobe XD.

Se creará un diseño inicial en Adobe XD para las interfaces a desarrollar en el proyecto, y se utilizarán como plantilla a la hora de realizar la implementación en Flutter mediante lenguaje Dart.

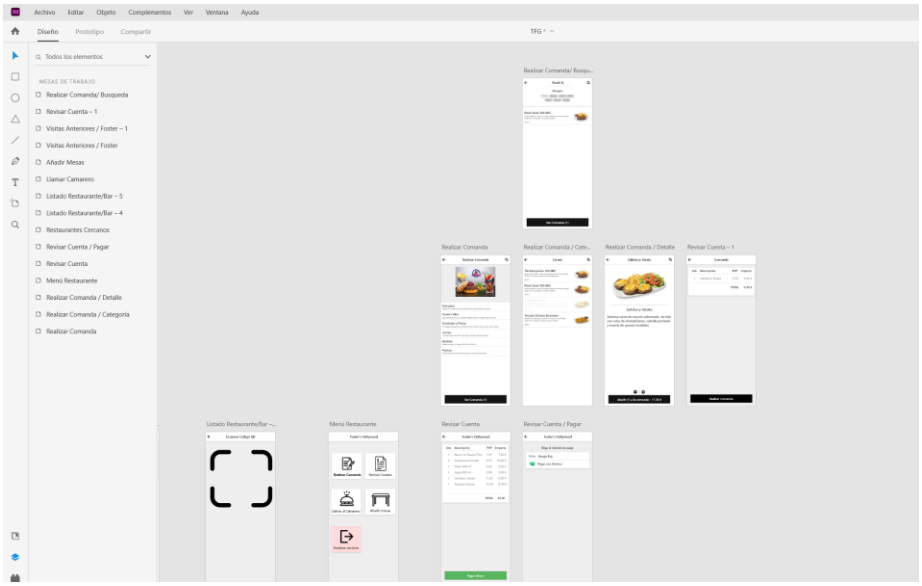

Figura 20. Adobe XD

4.9. GitHub

GitHub es un repositorio remoto para alojar proyectos software. Utiliza el sistema de control de versiones ofrecido por Git. Un repositorio es un lugar virtual localizado y alojado en la nube donde los desarrolladores almacenan cualquier tipo de información, generalmente sobre el proyecto actual. Como todo control de versiones, da soporte para las principales operaciones de commit, update, push…

5. Aspectos del desarrollo

5.1 Introducción

Dentro de este apartado se enunciarán los estados iniciales del proyecto, así como las decisiones que se tomaron en los cambios de tecnología y modelo de negocio a utilizar.

5.2. Ingeniería del software

Mediante el Proceso Unificado (Figura 21. Proceso Unificado), se han ido separando las distintas etapas del proceso de desarrollo de software. Al ser una metodología iterativa e incremental, los procesos de refinamiento modificarán las necesidades del sistema utilizando iteraciones y evaluando el estado actual del software producido.

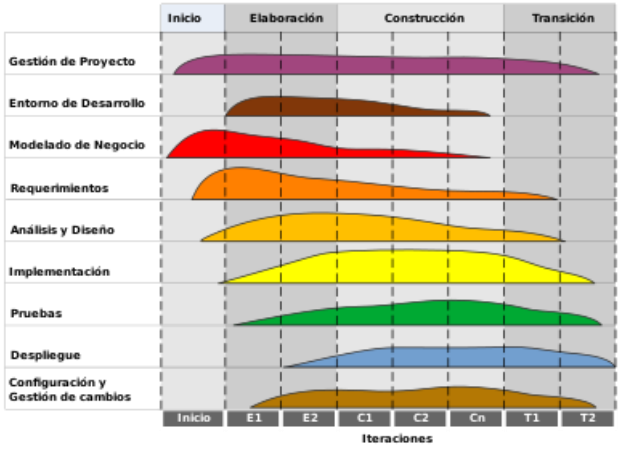

Figura 21. Proceso Unificado

- **Fase de Inicio**: Se define el alcance del proyecto a desarrollar. Se plantea un sistema capaz de llevar la gestión de comandas de un establecimiento tanto por parte del cliente como del empleado.

- **Fase de Elaboración**: Se establecen los requisitos y estimaciones para el proyecto y se obtiene una visión con mayor grado de refinamiento del proyecto software.

- **Fase de construcción**: Representa el conjunto de acciones que se realizan hasta obtener un producto con unos requisitos mínimos.

- **Fase de Transición**: Una vez el proyecto esté preparado, debe de presentarse funcionalmente correcto y habiendo pasado unas pruebas. En esta fase se plantea el mantenimiento del producto y posibles mejoras.

Todas estas fases se repetirán durante un número determinado de iteraciones necesarias para obtener un producto mínimo.

5.2.1. Estimación del esfuerzo

Mediante el cálculo de factores de complejidad relativos a los actores y a los casos de uso del sistema, se realizará una estimación del esfuerzo que nos permitirá obtener el tiempo esperado para la finalización del proyecto.

Para realizar el cálculo de estimación de esfuerzo se utilizará el análisis de puntos de casos de uso (UCP). Estos puntos se utilizarán en el programa EzEstimate para obtener el número de horas calculadas para el proyecto.

Para ver en detalle estos cálculos, se recomienda la lectura del documento *Anexo I: Estimación de costes y planificación temporal*.

A continuación, se mostrará un resumen de los datos obtenidos en el proceso de estimación de esfuerzo para **UAW** (*Unadjusted Actor Weight*, Factor de peso de los actores sin ajustar, Tabla 1. Cálculo de UAW), **UUCW** (*Unadjusted Use Case Weight*, Factor de peso de los casos de uso sin ajustar, Tabla 2. Cálculo de UCCW), **TCF** (*Technical Complexity Factor*, Factores de complejidad técnica, Tabla 3. Cálculo de TCF) y **EF** (Environmental Factor, Factores de entorno, Tabla 4. Cálculo de EF)

UAW

Actor	Descripción	Peso	Cantidad
Simple	Si el actor es un sistema y la aplicación se comunica con él mediante una API	1	0
Medio	Si el actor es un sistema y la aplicación se comunica con él mediante un protocolo (Internet)	2	0
Complejo	Persona con una interfaz gráfica	3	2
UAW: 6			

Tabla 1. Cálculo de UAW

UUCW

Caso de uso	Descripción	Peso	Cantidad
Simple	3 transacciones o menos	5	22
Medio	De 4 a 7 transacciones	10	5
Complejo	Más de 7 transacciones	15	0
UCCW: 150			

Tabla 2. Cálculo de UCCW

TCF

Factor	Descripción	Peso	Valor	Descripción
T1	Sistemas distribuidos	2	0	No es necesario distribuir tareas entre varios servidores, ya que los envíos se realizan a la base de datos remota
T2	Rendimiento	2	3	Es necesaria un buen tiempo de respuesta para no percibir lentitud
T3	Eficiencia del usuario final	1	5	Dado que nuestra aplicación será usada por un grupo amplio de personas, debemos facilitar el entorno para evitar dificultades
T4	Procesamiento interno complejo	1	1	Mayormente la funcionalidad de la aplicación se resume en el envío de información a una base de datos
T5	Reutilización de código	1	3	El código debe ser reutilizable por los distintos elementos que forman la aplicación
T6	Facilidad de instalación	0.5	1	La aplicación se instalará mediante Google Play
T7	Facilidad de uso	0.5	5	Se debe conseguir una satisfacción de uso por parte de los clientes
T8	Portabilidad	2	1	La aplicación deberá funcionar en sus inicios únicamente para Android
T9	Facilidad de cambio	1	4	Se debe permitir realizar cambios en el sistema de forma relativamente sencilla

T10	Concurrencia	1	5	El sistema deberá soportar llamadas realizadas desde múltiples clientes
T11	Características especiales de seguridad	1	1	Únicamente se deberá cubrir datos de tarjeta. En este caso, como la seguridad se ofrece mediante otra plataforma, no debemos destinar recursos
T12	Acceso directo a terceras partes	1	2	Se deberá conectar a la plataforma de pago para finalizar transacciones
T13	Se requiere entrenamiento especial del usuario	1	0	Al ser una aplicación intuitiva no necesitará de entrenamiento especial para sus usuarios
TOTAL		32		
TCF = 0.6 + (0.01 * TFactor) = 0.6 + (0.01 * 32) = 0.92				

Tabla 3. Cálculo de TCF

EF

Factor	Descripción	Peso	Valor	Descripción
E1	Familiaridad con el modelo del proyecto	1.5	4	El desarrollador ha trabajado en varios proyectos.
E2	Experiencia en la aplicación	0.5	4	El desarrollador realizado varias aplicaciones Android.
E3	Experiencia en orientación a objetos	1	5	El desarrollador ha estado trabajando con POO durante toda su carrera de programación.
E4	Capacidad del analista líder	0.5	3	Las tareas que se realicen por parte del programador no serán complicadas
E5	Motivación	1	4	Debido a la falta de alternativas del mismo tipo y ver una oportunidad de negocio en ello, el desarrollador se encontrará muy motivado.
E6	Estabilidad de los requerimientos	2	4	Se realizarán cambios menores en los requisitos
E7	Trabajadores a tiempo parcial	-1	2	El desarrollador realizará el proyecto durante el desarrollo del curso.
E8	Dificultad del lenguaje de programación	-1	2	El desarrollador no ha trabajado tanto con Flutter.
TOTAL		22		

EF = 1.4+(0.03 * EFactor) = 1.4 + (0.03 + 22) = 0.74

Tabla 4. Cálculo de EF

Una vez obtenidos los datos necesarios, calculamos el **UCP** (Use Case Points, Puntos de Casos de Uso)

UCP = UUCP * TCF * EF = 156 * 0.92 * 0.74 = **106.2048**

Calculamos el esfuerzo a desempeñar. En este caso se ha realizado una división exhaustiva de los casos de uso, por lo que resultan en casos bastante simples. Por ello, se ha establecido un esfuerzo de 5 horas por cada UCP, ya que no suponen un trabajo excesivo para el desarrollador. Si calculamos el tiempo de proyecto:

E = UCP * CF = 100.659 * 5 = **531.024** horas por persona

Dado que el desarrollador tiene prácticas por la mañana en el período indicado, se dispone de la tarde para trabajar en el proyecto. Por tanto, si establecemos una labor semanal de 6 horas semanales (horario 15:00 – 21:00) obtenemos una estimación de 531.024 / 6= **89** días para realizar el proyecto.

5.2.2. Planificación temporal

En el apartado 5.2.1. Estimación del esfuerzo se concluyó que el proyecto tenía una duración estimada de 89 días. Por tanto, se establecerá un plan de proyecto para su desarrollo en el período de 89 días.

Dentro del proceso unificado, en el que se realizan varias iteraciones para refinar el producto final, se han encontrado y descrito 2 iteraciones en concreto:

- **Iteración 1**: Representa la fase Inicio y parte de la fase de Elaboración. En esta iteración se hace enfoque en la recogida de requisitos, análisis y diseño, sin entrar en la parte de implementación del sistema. Está representada en la Figura 22. Iteración 1

- **Iteración 2**: Se centra en la fase de Construcción del software. Gran parte del trabajo que se realiza recae en la implementación (80%), mientras que se realiza una labor de análisis y diseño durante el desarrollo (20%) para refinar los resultados obtenidos. Una vez acabada el período de implementación se comienza a probar el producto. Está representada en la Figura 23. Iteración 2.

Iteración 1

Id		Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predecesor	Nombres de los recursos
1			**Iteración 1**	**23 días**	**lun 07/02/22**	**mié 09/03/22**		
2			**Especificación de requisitos**	**6 días**	**lun 07/02/22**	**lun 14/02/22**		
3			Búsqueda de requisitos	3 días	lun 07/02/22	mié 09/02/22		Pablo
4			Elaboración de casos de uso	3 días	jue 10/02/22	lun 14/02/22	3	Pablo
5			Hito fin de especificación de requisito	0 días	lun 14/02/22	lun 14/02/22	2	
6			**Análisis de requisitos**	**8 días**	**mar 15/02/2**	**jue 24/02/22**	**2**	
7			Creación del modelo de dominio	5 días	mar 15/02/2	lun 21/02/22	2	Pablo[50%]
8			Arquitectura del modelo de análisis	5 días	mar 15/02/2	lun 21/02/22	2	Pablo[50%]
9			Elaboración de diagramas de secuencia	3 días	mar 22/02/22	jue 24/02/22	8	Pablo
10			Fin análisis de requisitos	0 días	jue 24/02/22	jue 24/02/22	6	
11			**Diseño**	**9 días**	**vie 25/02/22**	**mié 09/03/22**	**6**	
12			Búsqueda y comparación de patrones arquitectónicos	8 días	vie 25/02/22	mar 08/03/22	6	Pablo[70%]
13			Diagrama de clases	8 días	vie 25/02/22	mar 08/03/22	6	Pablo[30%]
14			Diagramas de secuencia de diseño	1 día	mié 09/03/22	mié 09/03/22	13	Pablo
15			Hito fin diseño	0 días	mié 09/03/22	mié 09/03/22	11	
16			Fin iteración 1	0 días	mié 09/03/22	mié 09/03/22	1	

Figura 22. Iteración 1

Iteración 2

Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predecesor	Nombres de los recursos
17		**Iteración 2**	**66 días**	**mié 09/03/2**	**jue 16/06/22**	**1**	
18		**Especificación de requisitos**	**47 días**	**jue 10/03/22**	**jue 19/05/22**	**1**	**Pablo[5%]**
19		Refinamiento de requisitos	47 días	jue 10/03/22	jue 19/05/22		
20		Actualización de casos de uso	47 días	jue 10/03/22	jue 19/05/22		
21		Hito fin de especificación de requisito	0 días	mié 09/03/22	mié 09/03/22		
22		**Análisis de requisitos**	**47 días**	**jue 10/03/22**	**jue 19/05/22**	**1**	**Pablo[5%]**
23		Modificación del modelo de domin	47 días	jue 10/03/22	jue 19/05/22		
24		Corrección de diagramas de secuer	47 días	jue 10/03/22	jue 19/05/22		
25		Hito fin análisis de requisitos	0 días	mié 09/03/22	mié 09/03/22		
26		**Diseño**	**47 días**	**jue 10/03/22**	**jue 19/05/22**	**1**	**Pablo[10%]**
27		Comparación del funcionamiento de patrones arquitectónicos	47 días	jue 10/03/22	jue 19/05/22		
28		Añadir métodos y atributos en diagrama de clases	47 días	jue 10/03/22	jue 19/05/22		
29		Modificación de diagramas de secuencia de diseño	47 días	jue 10/03/22	jue 19/05/22		
30		Hito fin diseño	0 días	mié 09/03/22	mié 09/03/22		
31		**Implementación**	**47 días**	**jue 10/03/22**	**jue 19/05/22**	**11**	**Pablo[80%]**
32		Configurar Firestore en Flutter	1 día	jue 10/03/22	jue 10/03/22	11	
33		Crear e inicializar datos en Firestor	2 días	vie 11/03/22	lun 14/03/22	32	
34		Sistema de login y logout	1 día	mar 15/03/2	mar 15/03/22	33	
35		Sistema para leer y procesar ubicaciones	3 días	mié 16/03/22	vie 18/03/22	34	
36		**Módulo Cliente**	**20 días**	**lun 21/03/22**	**mar 19/04/22**	**35**	
37		Diseño de interfaces	3 días	lun 21/03/22	mié 23/03/22	35	
38		Conexión de interfaces con Providers	4 días	jue 24/03/22	mar 29/03/22	37	
39		Funcionalidad lectura código QR	2 días	mié 30/03/22	jue 31/03/22	38	
40		Establecer llamadas a Firestore	5 días	vie 01/04/22	jue 07/04/22	39	
41		Establecer listener a colecciones de Firestore	1 día	vie 08/04/22	vie 08/04/22	40	
42		Sistema de pago	5 días	lun 11/04/22	mar 19/04/22	41	
43		**Módulo administrador**	**20 días**	**mié 20/04/2**	**jue 19/05/22**		
44		Diseño de interfaces versión administrador	5 días	mié 20/04/22	mié 27/04/22	36	
45		Conexión de interfaces con Providers	6 días	jue 28/04/22	vie 06/05/22	44	
46		Establecer llamadas a Firestore	5 días	lun 09/05/22	vie 13/05/22	45	
47		Establecer listener a colecciones de Firestore	4 días	lun 16/05/22	jue 19/05/22	46	
48		Hito fin de implementación	0 días	mié 09/03/22	mié 09/03/22		
49		**Pruebas**	**19 días**	**vie 20/05/22**	**jue 16/06/22**	**31**	
50		**Pruebas unitarias**	**6 días**	**vie 20/05/22**	**vie 27/05/22**	**31**	
51		Módulo cliente	6 días	vie 20/05/22	vie 27/05/22	31	Pablo[50%]
52		Módulo administrador	6 días	vie 20/05/22	vie 27/05/22	31	Pablo[50%]
53		**Pruebas de integración**	**13 días**	**lun 30/05/22**	**jue 16/06/22**	**50**	
54		Envío y recibo de información mediante Firestore	13 días	lun 30/05/22	jue 16/06/22	50	Pablo[30%]
55		Integración del sistema complet	13 días	lun 30/05/22	jue 16/06/22	50	Pablo[70%]
56		Fin iteración 2	0 días	jue 16/06/22	jue 16/06/22	17	

Figura 23. Iteración 2

Diagrama Gantt del proceso completo

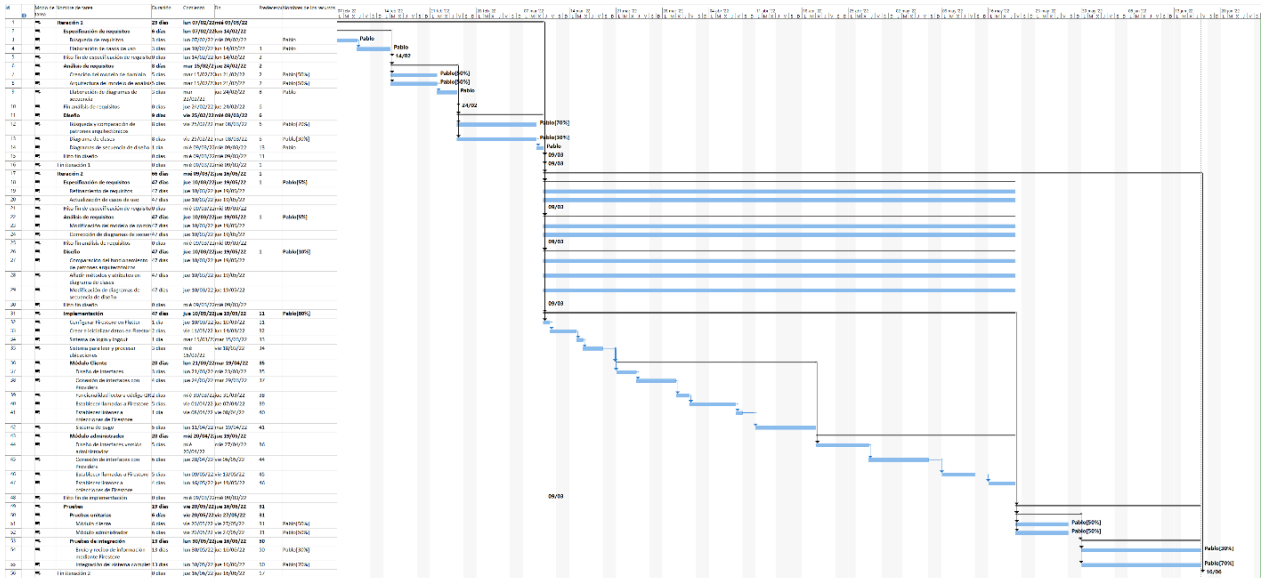

Figura 24. Diagrama Gantt

Uso de recursos

Como podemos comprobar en la Figura 25. Uso de recursos, la utilización de recursos ha permanecido al 100% en todo momento de la fase de desarrollo del proceso, evitando también sobreasignaciones al empleado.

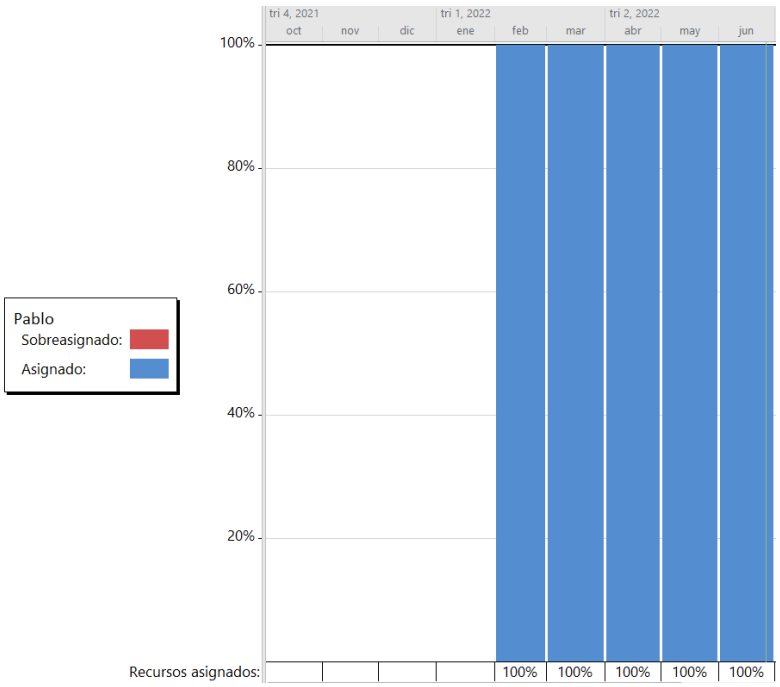

Figura 25. Uso de recursos

5.2.3. Especificación de requisitos

Mediante la metodología Durán y Bernárdez se han obtenido los requisitos necesarios para el dominio del problema, incluyendo la realización de los diagramas de casos de uso para representarlos. Estos diagramas representan el comportamiento del sistema frente a la interacción del usuario. También se recopilarán los requisitos no funcionales presentes en el diseño.

Todas las especificaciones de requisitos se recogen en el Anexo II. En la Figura 26. Diagrama de Casos de Uso se puede ver el diagrama de casos de uso completo del sistema.

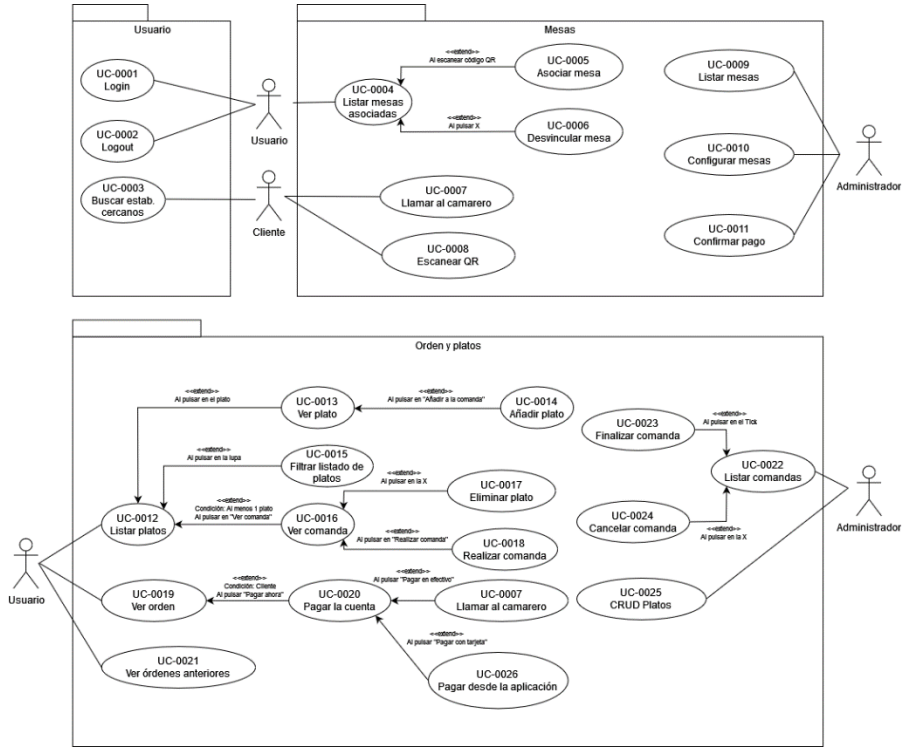

Figura 26. Diagrama de Casos de Uso

5.2.4. Análisis de requisitos

En este proceso de análisis de requisitos se determinarán las funciones que el sistema debe realizar a partir de los resultados obtenidos en la búsqueda de requisitos realizada.

Se definirán las restricciones aplicadas al sistema y se evaluarán las necesidades del usuario para definir el modelo del sistema. Dentro de este proceso se establecerán varios puntos:

- Construcción del modelo de dominio
- Organización de paquetes
- Organización de la vista de arquitectura
- Definición de vista de interacción

Esta información se encuentra en el Anexo III. En la Figura 27. Modelo de Dominio podemos ver el planteamiento inicial para el modelo de datos.

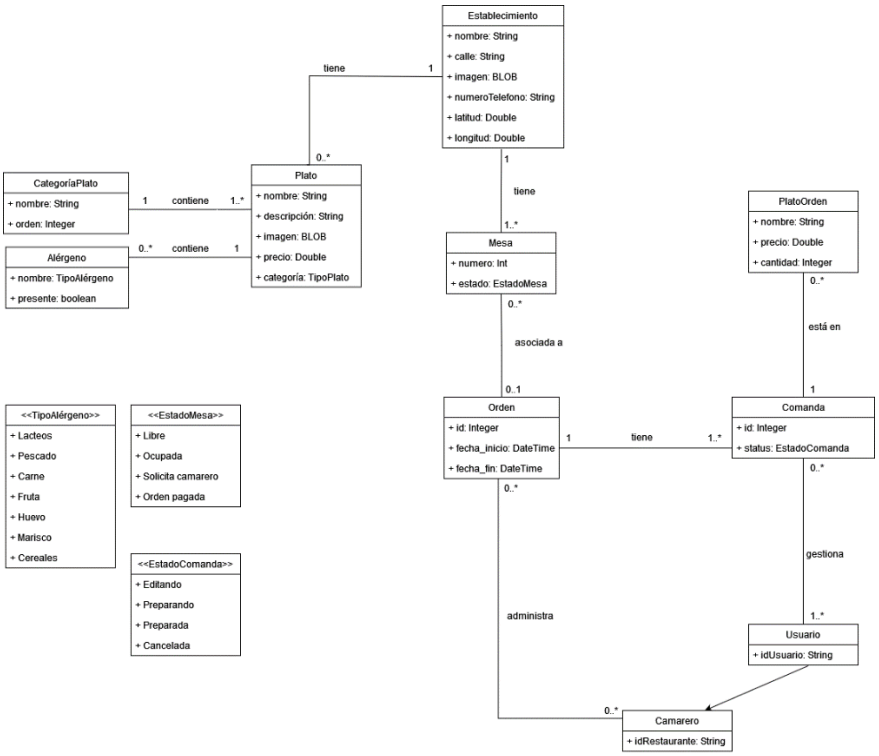

Figura 27. Modelo de Dominio

5.2.5. Diseño y arquitectura del sistema

En este proceso de diseño se especificarán los componentes identificados en nuestro sistema, sus relaciones y los subsistemas que se crean a partir del conjunto de todos ellos. Se identificarán los patrones de diseño a utilizar para la fase de implementación, dentro del Proceso Unificado.

En el Anexo IV se puede comprobar toda esta información en detalle. La Figura 28. Diagrama de paquetes representa la estructura establecida para la fase de diseño.

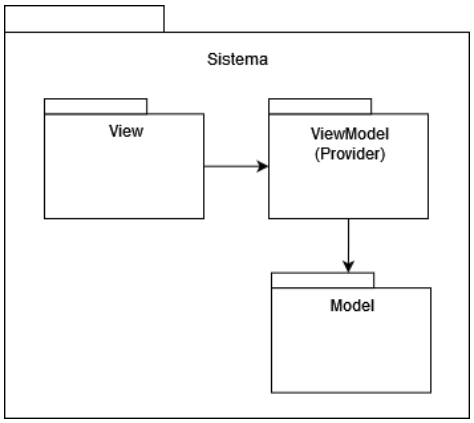

Figura 28. Diagrama de paquetes

Patrones arquitectónicos planteados

En la fase inicial del proyecto se valoró en utilizar 2 patrones de arquitectura: Modelo Vista Controlador [3] y MVVM [4] (*Model-View-ViewModel*, Modelo Vista Vista-Modelo). A continuación, se explicarán las ventajas e inconvenientes sobre cada modelo.

Modelo Vista Controlador

El MVC es uno de los patrones más utilizados, debido a su simplicidad. Separa los datos de la aplicación en 3 componentes:

- Modelo: Representación de los datos
- Vista: Información visible por el usuario
- Controlador: Actúa como intermediario de Modelo y Vista, gestionando el flujo de información.

A diferencia de los siguientes patrones arquitectónicos, este modelo no permite una actualización de datos eficiente en función de eventos, por lo que se descartó en la fase inicial del proceso de diseño de la arquitectura del sistema.

MVVM

Es un patrón que intenta desacoplar la interfaz ofrecida por el sistema de la lógica de un proyecto. Se puede ver la representación de sus elementos en la Figura 29. Patrón MVVM.

- **Vista**: Representa la parte visual de la aplicación. Mediante eventos generados por el usuario la interfaz informa al *ViewModel* sobre cambios que se tienen que realizar.

- **Vista-Modelo**: Actúa como intermediario entre la vista y el modelo. Establece la comunicación entre ambos componentes y proporciona los datos necesarios a los distintos View.

- **Modelo**: Guarda los datos necesarios para el sistema. Contiene la información necesaria de negocio, pero no contiene acciones para manipular dicha información. Este modelo puede ser remoto (Google Firebase) y local. El modelo local se proveerá de los datos guardados en el modelo remoto, asemejándose todo lo posible a él.

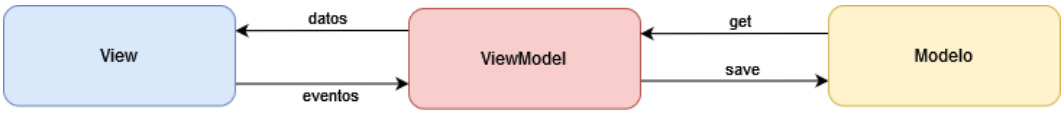

Figura 29. Patrón MVVM

Mediante el patrón MVVM definimos el flujo de información y la forma en la que se recibe en la vista. Sin embargo, tenemos otros problemas que resolver:

- Los cambios que se hagan en un determinado View se harán únicamente en dicho elemento. Si nuestra aplicación tiene un número considerable de Views la lógica e interfaz visual se complican, ya que los cambios en un View se tendrían que gestionar manualmente en cualquier otro View.

- Cuando se produce un cambio en uno de los View, el árbol entero de Widgets vuelve a ser pintado sea necesario o no, por lo que produce una ralentización del sistema.

Para ello aparecen patrones como BloC y Provider.

Patrones de funcionalidad planteados

BLoC

El patrón BLoC (*Business Logic Component*, Figura 30. Funcionamiento del patrón BLoC) actúa como intermediario entre la vista y el modelo. Fue diseñado por Google, y su objetivo es la reutilización de código entre sus aplicaciones, utilizando Flutter con Dart [5].

Los objetivos que se pretenden conseguir utilizando el patrón son los siguientes:

- Centralizar la lógica de negocio
- Centralizar los cambios de estado
- Transformar al formato que necesite la vista

Por cada vista que haga uso de alguna de las estructuras de datos sujetas a cambios tendrá su correspondiente BLoC.

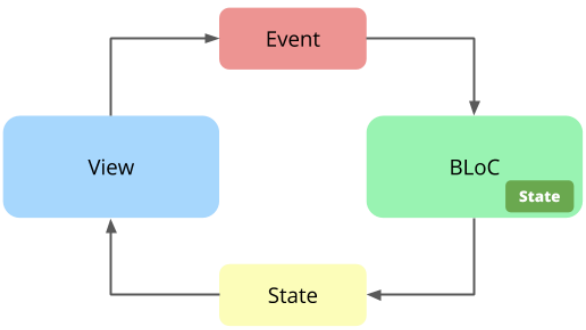

Figura 30. Funcionamiento del patrón BLoC

Como también indicamos en el Anexo IV, BLoC proporciona una robusta integridad de los datos mediante la utilización de Eventos y Estados, los cuales se encargan de informar a todos los View disponibles de dichos cambios. BLoC está pensado para aplicaciones que tienen un grado de complejidad alto con muchas interfaces. Dado que nuestra aplicación tiene una complejidad baja-media no es del todo adecuado utilizar este patrón.

Provider

Provider (Figura 31. Patrón Provider) establece una metodología en la que cada Vista o *Widget* que quiera hacer uso de un dato que puede cambiar con el tiempo, utiliza el dato guardado en dicho *Provider*.

Los que hacen uso del *Provider* se denominan "Consumidores", los cuales obtienen los datos necesarios para su visualización mediante el *Provider*.

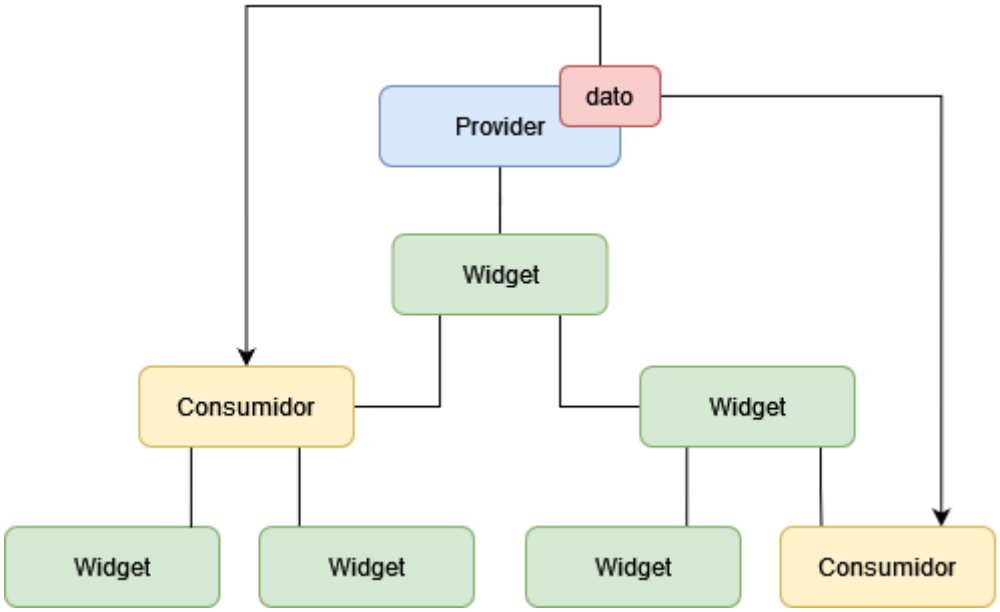

Figura 31. Patrón Provider

Dado que nuestra aplicación está basada en consultas en tiempo real, es necesario algún patrón como el Provider para informar de cambios en el modelo, ya que estos cambios pueden ser realizados local o remotamente. Para la implementación del patrón *Provider* se siguió la guía [6].

5.2.6. Implementación

Sistema de bases de datos

Iteración 1

En un modelo inicial del proyecto se planteó establecer un servidor Node.js, al cual le llegarían peticiones de distintos clientes (dispositivos móviles). Éste se encargaría de establecer las conexiones necesarias con la BBDD para guardar, leer y modificar los datos contenidos en las distintas tablas del sistema. Esto nos permitiría tener una base de datos con un modelo relacional, el cual se asemejaría en mayor medida al modelo que hemos establecido en el *Análisis de Requisitos*. Sería una arquitectura cliente-servidor representada en la Figura 32. Servidor con Node.js.

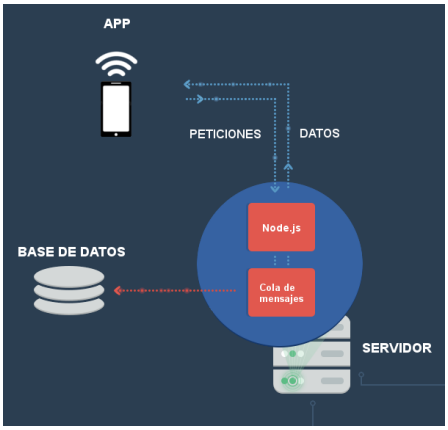

Figura 32. Servidor con Node.js

El sistema nos valdría para cualquier sistema de bases de datos que tuviese información persistente. El problema que se planteó fue que, en realidad, nuestra base de datos debía asemejarse a una base de datos en tiempo real, ya que se producen envíos y peticiones constantes.

Node.js permite la creación de sockets de escucha para informar sobre cambios en cualquiera de las filas de una tabla que hayamos indicado. Aun así, supone una mayor carga de trabajo estar creando tantos sockets como clientes.

Iteración 2

Debido a estas desventajas, en las fases iniciales de desarrollo del proyecto se concluyó en que utilizar Node.js para una base de datos en tiempo real no sería del todo óptimo.

Por ello, de entre las opciones de bases de datos en tiempo real existentes, se escogieron Google Firestore y Google Realtime Database. Se hizo un estudio de ventajas e inconvenientes [7] considerando ambas plataformas (Tabla 5. Comparativa Firestore y Realtime Database).

	Firestore	Realtime Database
Estructura de datos	Estructura visible, sencilla y fácil de escalar	Formato JSON, más compleja
Queries	Consultas compuestas, permitiendo filtros de ordenamiento	Filtrar y ordenar por un campo
Escalabilidad	Escala de forma automática	No
Precios	Por operaciones de lectura, escritura y eliminado	Por ancho de banda.

Tabla 5. Comparativa Firestore y Realtime Database

Una vez realizada la comparación, se decidió optar por Google Firestore, principalmente porque la estructura de datos nos interesa que sea sencilla para poder explicarla a los usuarios, necesitamos algo de escalabilidad debido al gran número de usuarios que pueden utilizar la aplicación y porque el precio de una base Firestore suele ser más barato que en Realtime Database.

Flutter dispone de soporte para realizar conexiones con Firestore siguiendo la guía [8]. Una vez se importen los archivos de configuración, disponibles en la configuración de proyecto de Firestore, se podrá establecer la conexión. El sistema de conexión se explica en la Figura 33. Google Firestore.

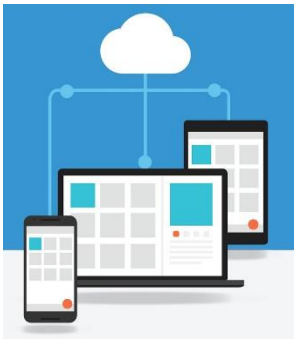

Figura 33. Google Firestore

Modelo de datos

Iteración 1

El modelo inicial que se plantea es un modelo de datos relacional, en el que existen múltiples tablas conectadas entre sí. La relación y la multiplicidad de cada tabla es la que decide en cuál de los lados se almacenará la información correspondiente a la relación. Este modelo está representado en la Figura 27. Modelo de Dominio.

Este modelo presenta un problema debido a los cambios realizados en la base de datos. Google Firestore es una base de datos no relacional. Esto significa que no se pueden hacer operaciones como JOIN para obtener todos los datos relativos a un determinado identificador de una fila. Existen métodos para tratar una BBDD en Firestore como relacional, pero supone un mayor número de lecturas y, en consecuencia, un mayor coste.

Iteración 2

Por ello la base de datos se debe denormalizar. En este proceso se optimiza la eficiencia de las bases no relacionales añadiendo datos redundantes en distintos sitios. Los datos que se guardan en una tabla pasan a estar en la que tiene la clave foránea. En las figuras Figura 34. Modelo Relacional y Figura 35. Modelo denormalizado se puede ver el proceso de transformación de unos datos normalizados a denormalizados, siguiendo la guía [9].

Figura 34. Modelo Relacional

Figura 35. Modelo denormalizado

Si se compara la lectura de los valores del restaurante en los 2 modelos:

- **Relacional**: Se obtiene el ID del restaurante asociado a la mesa, y después se buscan los platos asociados a dicho restaurante. Como en Firestore no existen las operaciones JOIN, se realizan 2 operaciones de lectura (Restaurante y Platos).

- **No relacional**: Se obtiene el ID del restaurante y se recuperan todos los datos del restaurante. 1 única operación de lectura.

Dado que en Firestore se cobra en función del número de lecturas y escrituras realizadas, debemos tener un modelo lo suficientemente eficiente como para devolver los datos necesarios a la consulta realizada, y a la vez no generar problemas de seguridad, ya que al denormalizar las tablas es posible que guardemos información confidencial en la nueva fila.

También hay que tener en cuenta las modificaciones que se hagan en cada elemento. Por ejemplo, si cada mesa contiene el restaurante y platos (duplicados), en el momento en el que se modifique alguno de los platos se deberá modificar en el mismo restaurante de todas las mesas. Por ello, hay ocasiones en las que beneficia mantener un modelo relacional.

Tras haber realizado los cambios en el modelo, tenemos las siguientes colecciones resultantes:

- **Mesas**: Listado de mesas que se han creado. Cada mesa tiene un restaurante asociado, una orden actual (vacía si no la tiene), un estado (Libre, Ocupada, Solicita camarero, Orden pagada, Asociada), un número de mesa y un boolean indicando si está siendo utilizada por el establecimiento o no.

- **Establecimientos**: Listado de restaurantes y bares que se han añadido al sistema. Un establecimiento tiene nombre, calle, dirección (utilizada por Google Maps), latitud, longitud, número de teléfono, una lista de platos que ofrece y una imagen. Cada plato tiene nombre, precio, categoría, alergias, descripción y una imagen.

- **Órdenes**: Listado de órdenes (visitas de usuarios) que se han creado. Una orden tiene una fecha de llegada, fecha de fin de orden, restaurante asociado, usuario asociado, lista de mesas utilizadas y un listado de comandas.

- **Comandas activas**: Listado de comandas en estado "Preparando". Una comanda activa tiene estado, fecha de emisión de la comanda, mesa asociada, orden asociada y una lista de platos a preparar.

- **Empleados**: Listado de usuarios que tienen acceso de administrador para un determinado establecimiento. Guarda el identificador del usuario con el que se ha iniciado sesión y el restaurante o bar al que tiene permisos.

Actualización de datos

La aplicación tiene que actualizarse en función de los datos que se encuentren en la base de datos Firestore. Dado que las filas van a sufrir cambios constantemente, el sistema deberá informar de dichos cambios a cada una de las aplicaciones que tengan relación con dicha fila. Esto se puede hacer creando objetos de escucha (listeners) dentro de la aplicación. Siguiendo la guía [10] crearemos 4 tipos principales de listeners:

- **Listener de Mesa**: Está pendiente de los cambios que se producen en la mesa. Estos cambios pueden ser sobre el estado de la mesa: Ocupada, Libre y Solicita camarero.

- **Listener de Orden**: Se establece un objeto de escucha para la orden actual. Una vez el usuario escanea el código QR, se recibe una actualización de datos cada vez que la orden es modificada. Principalmente esta actualización se genera al añadir platos a la comanda, eliminar platos y realizar la comanda.

- **Listener de Establecimiento**: Un empleado puede modificar los datos del restaurante o bar en cualquier momento. Si un usuario se encuentra en la pantalla principal de orden, no recibirá dichos cambios. Por tanto, para actualizar los datos de platos y mesas se deberá escuchar los cambios del restaurante que esté asociada la mesa.

- **Listener de Comandas**: Cada vez que el usuario finaliza una comanda, ésta se envía a una colección distinta llamada "comandasActivas", que contiene las emitidas en un restaurante o bar. Una vez finalizadas o canceladas, el empleado del establecimiento podrá eliminarlas de la lista.

Hay que tener en cuenta que, si se establece un listener para escuchar los cambios en órdenes, se aplicaría a todas ellas, por lo que sería un número elevado de actualizaciones que no nos interesan.

Por ello, los listener se establecen en relación a un identificador de documento (Figura 36. Estructura Firestore). Al hacerlo, sólo se recibirá notificaciones de cambios cuando los datos de la colección con un determinado ID sean modificados.

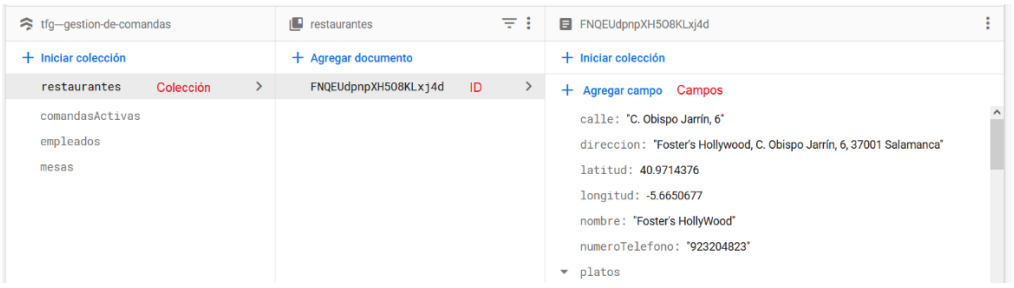

Figura 36. Estructura Firestore

Una vez se obtiene la notificación de cambio de datos del documento, se añade la información recibida al Provider, el cual se encargará de notificar a todos los Widget sobre los cambios. En la Figura 37. Estructura Listener se pueden ver los elementos del objeto de escucha creado mediante código.

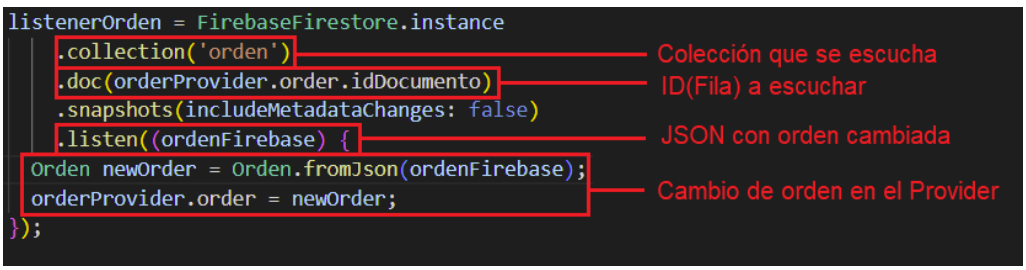

Figura 37. Estructura Listener

Plataforma de pago

Stripe proporciona la infraestructura necesaria para realizar pagos en línea. Esta infraestructura provee sistemas de prevención de fraude y gestión bancaria. Gracias a sus funcionalidades, permite el envío y recibo de pagos por internet. Existen guías [11] para la creación de una plataforma de pago en Flutter.

Lo único que se necesita para utilizar esta pasarela de pago es crear una cuenta en Stripe. Al crear dicha cuenta se vinculará una cuenta bancaria, la cual recibirá el dinero enviado por los clientes.

Se proporcionan APIs para su utilización por parte de los desarrolladores. En concreto, Flutter utiliza una de las bibliotecas para implementar su funcionalidad.

Ofrece un Dashboard (Figura 38. Dashboard de Stripe) para la visualización de pagos a lo largo del mes, indicando las cantidades ingresadas.

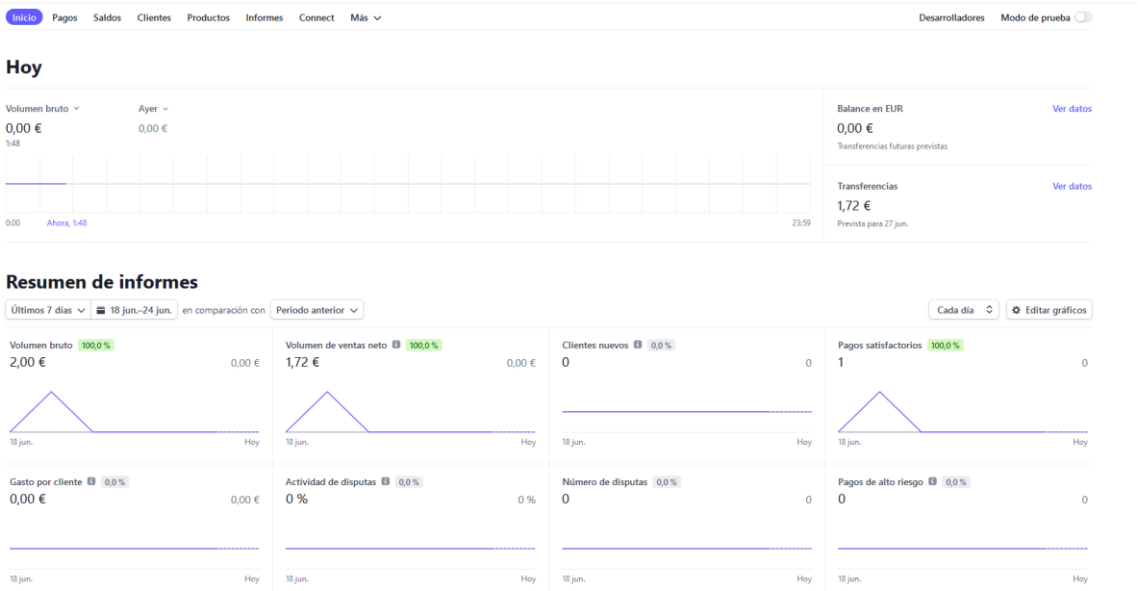

Figura 38. Dashboard de Stripe

En la misma aplicación, una vez que el usuario pulse sobre “Pagar con tarjeta” se desplegará un menú, en el que se indicará la cantidad a pagar, y el cliente indicará los datos de su tarjeta de crédito.

Una vez realizada la transacción cabe mencionar que Stripe se lleva una comisión del 1,4% +0,25 € sobre la transacción. Una vez sea validada dicha transacción (en un plazo de 3 días en Europa) se realizará el ingreso en la cuenta bancaria indicada al crear el usuario.

6. Resumen de funcionalidad

En este apartado veremos las funcionalidades más importantes de la aplicación, y cómo se resuelven mediante una serie de métodos aplicados. Para la creación de todos los elementos visuales se sigue la documentación [12].

6.1. Login y logout

La aplicación debe de realizar labores de autenticación por 2 motivos principales:

- Para asociar las órdenes a un determinado usuario, y de esta manera permitir recuperar el listado de órdenes anteriores que ha realizado un determinado usuario.
- Para comprobar si un usuario está registrado en algún establecimiento. Si lo estuviese, tendría permisos de administración del restaurante o bar.

Esta autenticación se hace mediante Google Sign In y Firebase Authentication [13]. Al iniciar sesión con una cuenta Google, se obtiene un token único para el usuario que acaba de identificarse. Utilizaremos ese token como ID de documento dentro de la colección “Empleados” de Google Firestore para guardar la posibilidad de que un usuario pueda administrar un establecimiento.

Para poder acceder a cualquiera de las opciones del menú principal se debe iniciar sesión previamente. Las siguientes pantallas (Figura 39. Sesión iniciada, Figura 40. Formulario de Google Sign In, Figura 41. Sesión sin iniciar) representan la gestión de cuenta realizada desde el menú lateral:

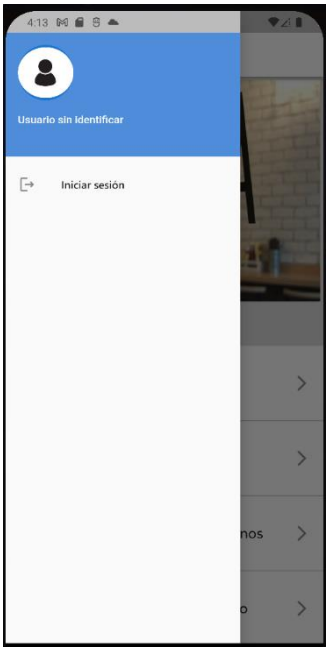

Figura 41. Sesión sin iniciar

Figura 40. Formulario de Google Sign In

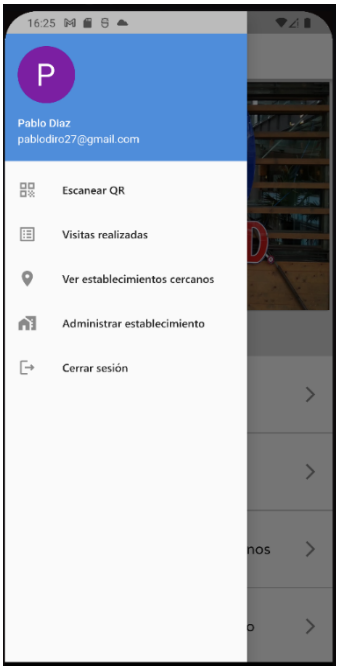

Figura 39. Sesión iniciada

6.2. Buscar establecimientos cercanos

Si el usuario permite la utilización de datos de ubicación dentro de la aplicación, se podrá obtener las coordenadas de dicho usuario para localizar los establecimientos que se encuentren en un radio de 1 kilómetro.

Dentro de la base de datos Firestore, se guardará la información para cada restaurante o bar sobre su latitud y longitud. Esta información puede ser obtenida desde la herramienta Google Maps. Una vez obtenidas las coordenadas, se utiliza la biblioteca Geolocator de Flutter para calcular una distancia en metros entre las coordenadas del usuario y las del establecimiento, y filtrar aquellos que cumplan con la condición de cercanía (Figura 43. Listado de establecimientos cercanos).

Adicionalmente se implementa funcionalidad [14] para podrá abrir cada ubicación mediante Google Maps para establecer una ruta hacia el restaurante o bar (Figura 42. Establecimiento en Google Maps).

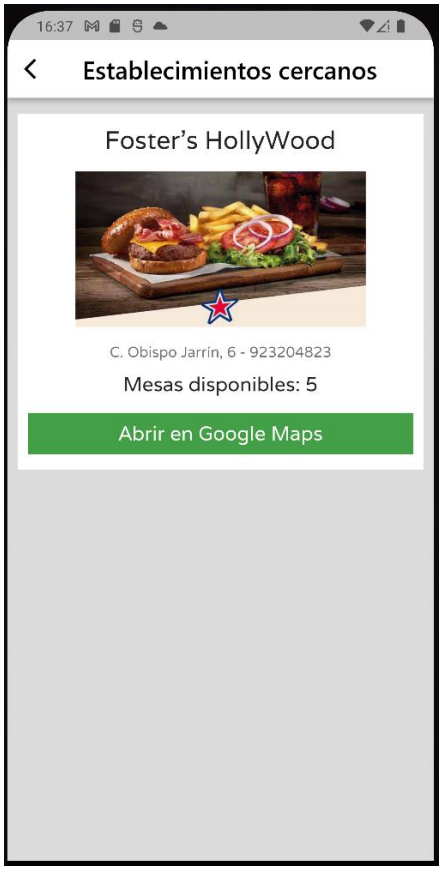

Figura 43. Listado de establecimientos cercanos

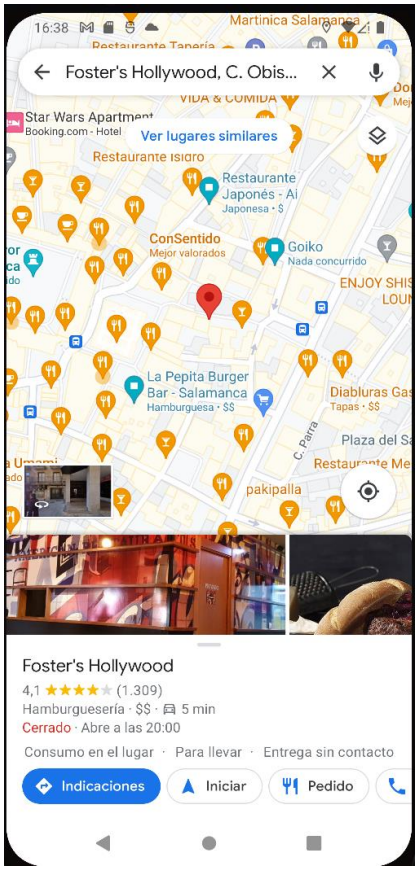

Figura 42. Establecimiento en Google Maps

6.3. Interacción Cliente

En este apartado se especificará en detalle las acciones que puede realizar un cliente al visitar un establecimiento, y su efecto en el sistema.

6.3.1. Escanear QR

Se seguirá una guía [15] para implementar la función de escanear códigos QR. Cada mesa tendrá un código QR que contiene el ID de la mesa. Este ID hace referencia a la información de la mesa que se acaba de escanear. Una vez se escanee el código QR (como se muestra en la Figura 44. Escanear QR), la aplicación realizará los siguientes pasos:

Figura 44. Escanear QR

- Si la mesa tiene una orden ya asociada:
    - Se recuperan los datos de dicha orden y del restaurante asociado a la mesa.

- Si la mesa no tiene una orden asociada todavía:
    - Se crea una nueva orden, vinculando al cliente con dicha orden y se recuperan los datos del restaurante asociado a dicha mesa.

A continuación, el cliente podría empezar a añadir platos dentro de una comanda. Se establecen 2 listeners:

- **Listener al establecimiento**: Si los datos de platos del restaurante o bar cambian (por ejemplo, precio) cada cliente que esté asociado a la mesa deberá recibir un listado actualizado de los platos.

- **Listener a la orden**: Si se modifica la orden (por ejemplo, se añade un plato a la comanda) la lista de platos de comanda cambiará en todos los clientes que hayan escaneado dicho código QR de mesa. De esta forma, se permite que cada cliente añada sus propios platos a la comanda desde su propio dispositivo móvil en tiempo real.

6.3.2. Añadir plato

Dentro de la ficha de cada plato, existe la opción de añadir a la comanda. El cliente indica la cantidad de unidades del plato que se quieren agregar a la comanda, así como un comentario opcional sobre la preparación de dicho plato (Figura 45. Detalle de plato, Figura 46. Listado de platos de la categoría, Figura 47. Listado de categorías)

Este plato se guardará en la comanda que se encuentre en estado “Editando”. Si no existe ninguna comanda creada con dicho estado (no hay ningún plato agregado) se creará añadiendo dicho plato.

Si este plato ya se encuentra en la comanda, el sistema se encargará de añadir las unidades indicadas por el usuario, en vez de crear un nuevo plato.

Una vez añadido, se envían los datos al documento con el id de orden asociado dentro de la colección de órdenes, para que todos los clientes tengan una versión actualizada de la comanda.

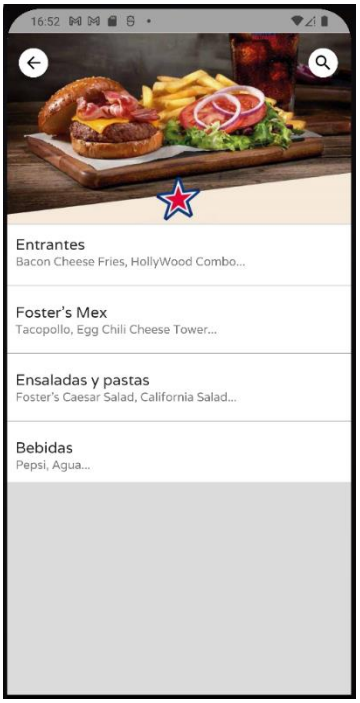

Figura 47. Listado de categorías

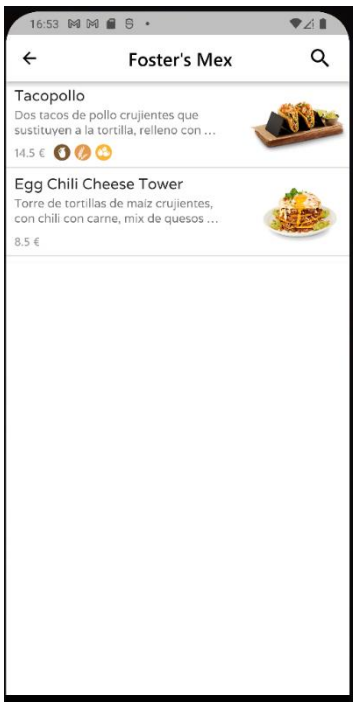

Figura 46. Listado de platos de la categoría

Figura 45. Detalle de plato

6.3.3. Ver comanda

Cuando se ha añadido al menos 1 plato a la comanda, se mostrará un botón “Ver comanda” en la pantalla de listado de platos y categorías. El sistema devolverá el listado de platos que se han establecido para una comanda, así como datos relevantes sobre el precio total de los platos a pedir y la opción de eliminar platos (Figura 49. Listado de platos de la comanda).

Tras haber añadido todos los platos a la comanda, el usuario podrá finalizar la comanda pulsando sobre “Pedir”. Esta comanda pasa a tener el estado “Preparando” dentro de la orden, y se crea una nueva comanda dentro de la colección “comandasActivas”.

Eliminar plato

Dentro del listado de comanda, al pulsar sobre la X en la fila del plato a eliminar se mostrará un diálogo para indicar los platos a eliminar de la comanda (Figura 48. Eliminar plato).

Una vez especificada la cantidad, se enviará la orden a la base de datos para informar a todos los clientes asociados a dicha mesa sobre la eliminación del plato.

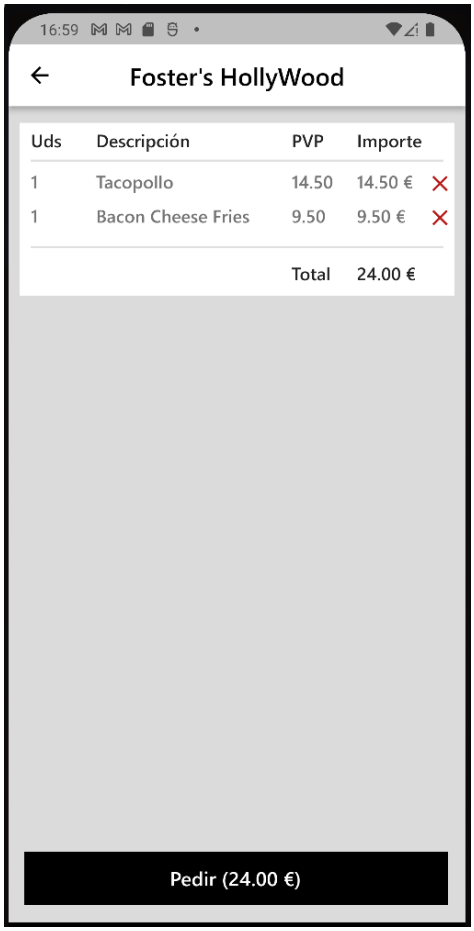

Figura 49. Listado de platos de la comanda

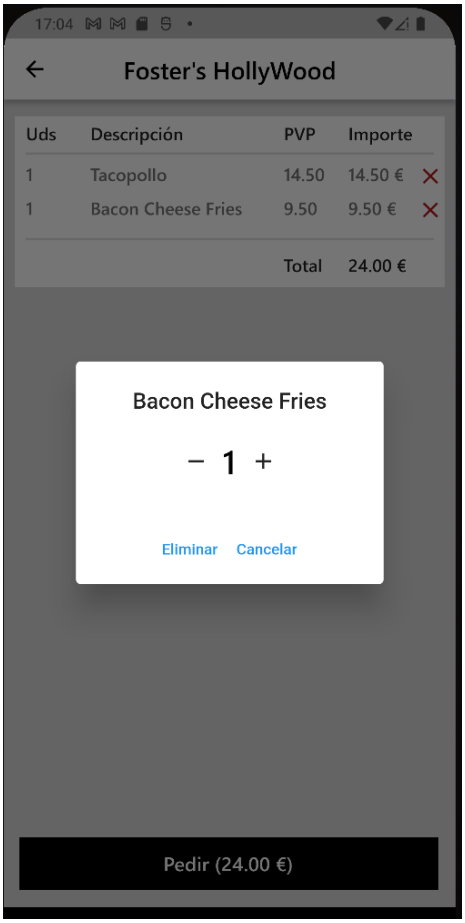

Figura 48. Eliminar plato

6.3.4. Avisar al camarero

Dentro del menú de orden, si el usuario pulsa sobre “camarero”, se cambiará de color el botón para indicar que se ha llamado al camarero (Figura 50. Botón camarero sin pulsar y Figura 51. Botón camarero pulsado) y se modificará el estado de la mesa. Si, por algún motivo, ya no se necesita al camarero, se podrá volver a pulsar el botón para borrar la solicitud. En el apartado 6.4.1. Administrar mesas se podrá ver el efecto de dicho cambio.

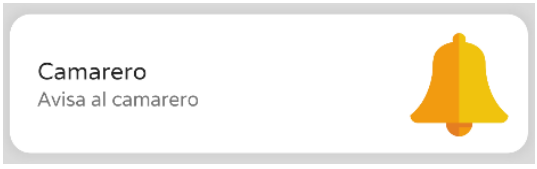

Figura 50. Botón camarero sin pulsar

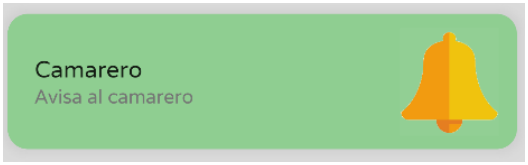

Figura 51. Botón camarero pulsado

6.3.5. Asociar mesas

Dentro de una visita del usuario se puede dar la situación de juntar varias mesas. El cliente deberá mantener un listado actualizado de las mesas que se están utilizando en un determinado momento (Figura 52. Listado de mesas asociadas).

Para ello, se muestra un listado de las mesas que se encuentran asociadas a la orden. La primera de ellas está vinculada con la orden, por lo que no se podrá borrar. Si se quiere añadir una mesa, basta con escanear el código QR de la nueva mesa a añadir.

En caso de querer borrar una mesa, el cliente o el administrador podrá pulsar sobre el botón “X” de la fila que corresponde la mesa a borrar. La primera mesa en la que se inició la orden no podrá ser eliminada.

Figura 52. Listado de mesas asociadas

6.3.6. Pagar cuenta

El cliente podrá pagar la cuenta una vez haya finalizado su visita (Figura 53. Pantalla cuenta, Figura 55. Métodos de pago). Para ello, se le ofrecen 2 opciones:

- **Pagar en efectivo**: La mesa cambiará el estado a "Solicita camarero" para que reciba atención de uno de los trabajadores del establecimiento.

- **Pagar con tarjeta**: La aplicación ofrecerá un formulario para rellenar con los datos de tarjeta de crédito del usuario (Figura 54. Pagar con tarjeta). Este pago se realizará mediante la plataforma Stripe, por lo que se asegura la privacidad de los datos ofrecidos por el cliente. Una vez se confirme el pago, la mesa cambiará el estado a "Orden pagada".

Tras realizar el pago, bien sea mediante efectivo o con tarjeta, alguno de los administradores del local deberá confirmar el pago de dicha mesa, lo cual eliminará la orden asociada a dicha mesa.

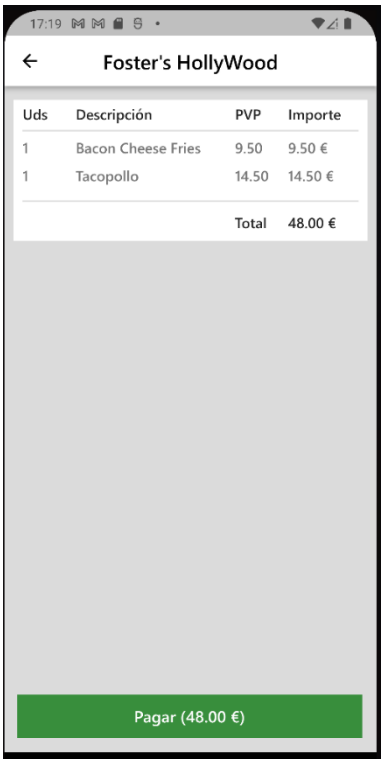

Figura 53. Pantalla cuenta

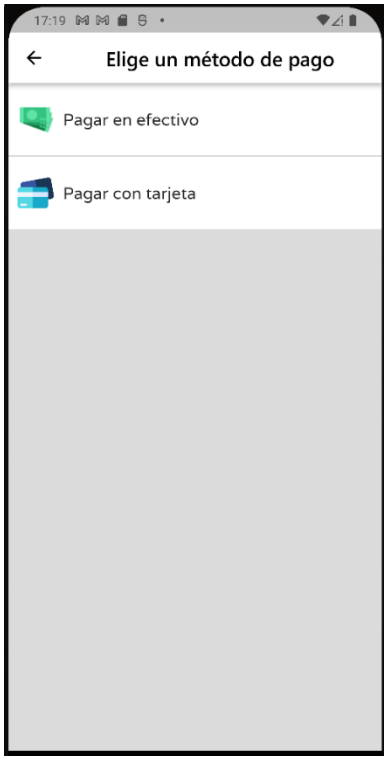

Figura 55. Métodos de pago

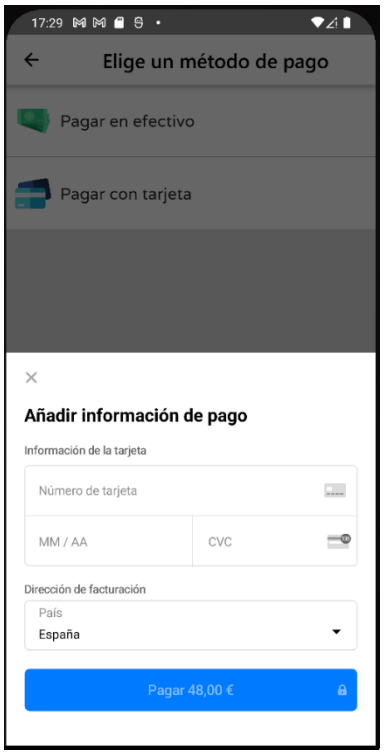

Figura 54. Pagar con tarjeta

6.4. Interacción Administrador

Este bloque recoge todas las acciones que un usuario con permisos de modificación en alguno de los restaurantes puede realizar.

6.4.1. Administrar mesas

En función del restaurante asociado al empleado se obtendrán las mesas vinculadas a dicho restaurante. Se ofrecerá un listado de mesas visibles, indicando mediante colores el estado de la mesa:

- **Rojo**: La mesa se encuentra ocupada por algún cliente.
- **Verde**: La mesa se encuentra disponible.
- **Gris**: La mesa está asociada a alguna otra mesa (juntar mesas).
- **Amarillo**: La mesa solicita ayuda del camarero
- **Naranja**: La mesa ha pagado la orden y está esperando ser liberada.

Se establece un Listener por cada mesa del restaurante para recibir los cambios realizados en base de datos en tiempo real.

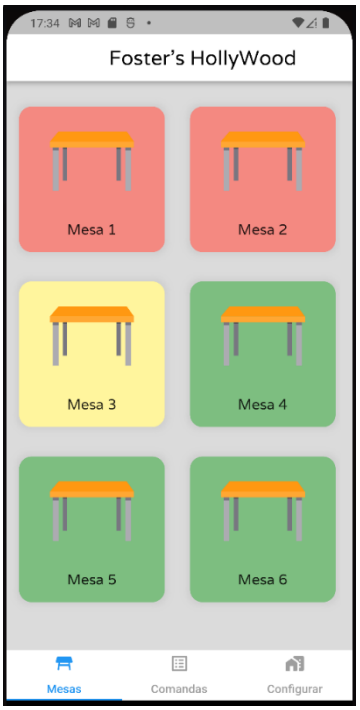

Figura 56. Administrar mesas

6.4.2. Administrar comandas

Los administradores podrán mostrar un listado de las comandas activas del restaurante. Estas comandas se mostrarán en función de la fecha de emisión, para mantener un orden de llegada.

Adicionalmente se podrá filtrar, mediante 2 botones situados en la barra superior de la aplicación para visualizar comida o bebidas, ya que en los establecimientos las bebidas se sirven en barra y las comidas se preparan en cocina, por lo que nos interesa tener los datos por separado para su visualización en cocina y en barra.

Una vez finalizada o cancelada la comanda se podrá indicar mediante los botones en cada comanda. Esto borrará la comanda de la colección de "comandas activas" y modificará el estado de la comanda en la orden asociada a "preparada" o "cancelada".

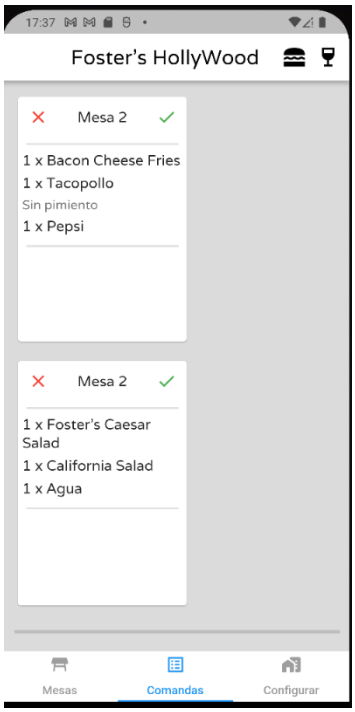

Figura 57. Administrar comandas

6.4.3. Modificar mesas

Un establecimiento puede disponer de 10 mesas, pero tener sacadas en un momento del día sólo 6. No sería óptimo mostrar todas las mesas que no se están usando en el listado de mesas del restaurante, ya que daría lugar a confusión para los trabajadores del establecimiento. En el apartado 6.4.2. Administrar comandas deberán mostrarse únicamente las mesas que se están utilizando actualmente.

Por ello, se ofrece un listado de las mesas que tiene el restaurante, y si queremos mostrarla dentro de la administración de mesas. Al pulsar sobre el switch de la fila de mesa que queramos alternar, modificaremos su visibilidad entre mostrar y no mostrar.

Esta información se enviará a la base de datos Firestore para que el resto de los administradores tengan constancia del cambio de visibilidad de la mesa.

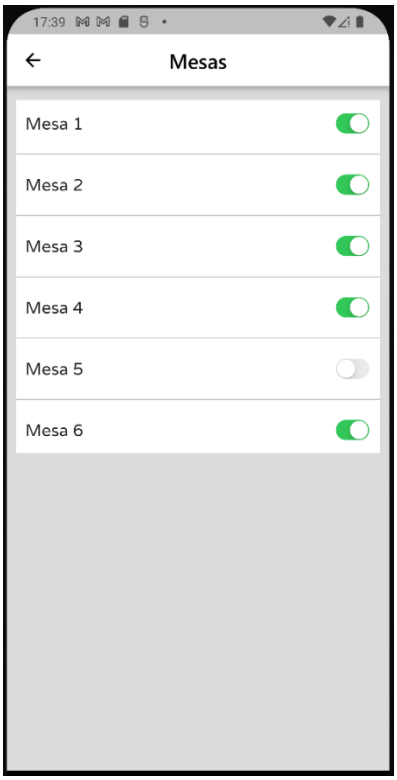

Figura 58. Configurar mesas

6.4.4. Modificar platos

Los platos del restaurante pueden variar en el transcurso de un servicio. Por tanto, se debe permitir a los administradores modificar la información de dichos platos, así como eliminarlos o añadir nuevos platos.

Se establece un Listener al restaurante para estar pendiente de los cambios que se realicen en el establecimiento. En caso de que un administrador modifique un plato se deberá notificar al resto de administradores para evitar errores.

El sistema muestra un listado de los platos que se encuentran añadidos en un restaurante (Figura 59. Listado de platos a configurar). En caso de que un usuario quiera modificar o eliminar alguno de los platos ofrecidos podrá pulsar los iconos “X” y “Lápiz” dentro de la fila del plato a modificar. Asimismo, se permite la creación de nuevos platos al pulsar sobre el botón “+” (Figura 60. Modificar plato).

Se ofrecerá un formulario al administrador para rellenar con los datos del nuevo plato o del plato a modificar. El usuario podrá subir imágenes que se encuentren en su galería gracias a Firebase Cloud Storage, siguiendo la guía [16] .Una vez se finalicen las modificaciones, se enviará la información a la base de datos Firestore.

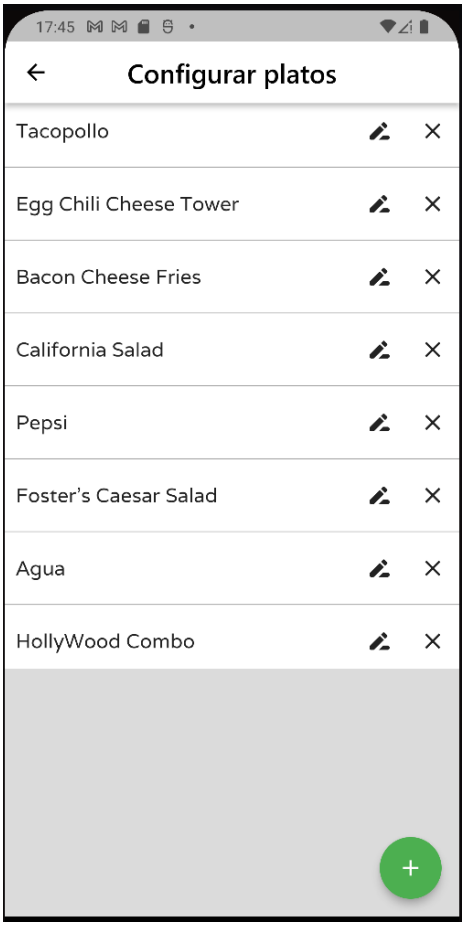

Figura 60. Listado de platos a configurar

Figura 59. Modificar plato

7. Conclusiones y líneas de trabajo futuras

La digitalización es un proceso en auge con el paso de los años. Se ofrecen nuevas soluciones informáticas para problemas de la vida cotidiana en los que, con ayuda de la tecnología, se plantean nuevas metodologías para la realización de procesos corrientes.

A lo largo del proyecto se ha buscado simplificar al máximo los procesos realizados en el negocio de la hostelería, facilitando al cliente y al empleado la interacción con el establecimiento.

Gracias a tecnologías ofrecidas por Google Firebase y la utilización del patrón Provider se ha logrado establecer un modelo eficiente a la hora de obtener datos remotos.

En respecta a los objetivos del proyecto planteados:

- **Obtener información a partir del código QR:** Se ha logrado establecer una serie de conexiones con la base de datos para la obtención de información relativa al establecimiento asociado a la mesa con el código QR.

- **Realizar comandas:** La principal funcionalidad de la aplicación era la de añadir platos a una comanda para su posterior envío al establecimiento. El cliente es capaz de realizar todas las acciones necesarias para notificar a los empleados de sus exigencias.

- **Administrar comandas:** Los empleados del bar o restaurante son capaces de gestionar las comandas enviadas por los clientes de manera óptima, indicando su finalización o cancelación.

- **Administrar establecimiento:** Los datos del local pueden ser modificados por cualquier empleado del negocio, informando de dicho cambio a todos los clientes y resto de empleados gracias a la utilización de listeners.

- **Ofrecer establecimientos cercanos:** Gracias a los datos de negocio almacenados en Firestore se puede determinar la distancia entre el usuario y el establecimiento mediante la librería "*geolocator*", permitiendo también la apertura de Google Maps con la dirección del local.

- **Informar sobre cambios en tiempo real:** Como ya indicamos en apartados anteriores, gracias al uso de la base de datos Google Firestore y listeners en la aplicación, se ha logrado que todos los usuarios que comparten información tengan la última actualización de datos, evitando incongruencia de datos entre los distintos usuarios.

- **Realizar pagos desde la aplicación:** En un inicio se pensó que con Google Pay se podría cumplir este objetivo. A medida que se investigó un poco más, se dedujo que era necesario otro método para realizar el pago. Mediante la plataforma Stripe se permiten los pagos enviados desde la aplicación.

En general existe un alto grado de cumplimiento con los objetivos establecidos al inicio del proyecto. Se han tenido que modificar diseños iniciales, pero al final se ha mantenido la funcionalidad esperada.

En cuanto a los objetivos personales del alumno, el aprendizaje de nuevas tecnologías existentes en la plataforma Firebase, lenguajes nuevos como Flutter y la creación de un sistema de digitalización casi nuevo se han cumplido satisfactoriamente. Este desarrollo ha sido de gran utilidad, permitiendo al alumno obtener conocimientos que pueden ser usados a la hora de crear un nuevo sistema que se asemeje al que se ha desarrollado.

Del proceso de aprendizaje hay que destacar la capacidad de adaptación a las distintas herramientas tecnológicas ofrecidas, siendo capaz de utilizar distintas alternativas en cuanto se ha comprobado de la ineficiencia de alguna de las planteadas inicialmente.

Líneas de trabajo futuras

A continuación, se indicarán una serie de posibles mejoras planteadas para el proyecto que no han sido implementadas en el sistema actual.

Doble QR: Es posible que un usuario, utilizando el código QR de una mesa, introduzca platos en una mesa que no es suya. Para asegurar de que no se genera envío de datos no deseados (platos y bebidas a una mesa que no ha pedido nada) se podría cambiar el sistema de escaneo de QR por una interacción en 2 pasos:

- El usuario escanea el QR. A efectos del sistema notificaría la necesidad de un camarero en la mesa asociada.
- El camarero ofrece un segundo código QR con el verdadero identificador de la mesa, desde el cual el usuario o grupo de usuarios podrán realizar comandas para su preparación por parte del establecimiento.

De esta forma se evitan problemas de seguridad con los códigos QR.

Sistema de notificaciones: Es posible que los camareros no se den cuenta del cambio de estado en una mesa. Para solucionar estos problemas se puede recurrir al envío de notificaciones push al dispositivo de los camareros. Los camareros recibirían un mensaje en la zona superior del móvil indicando la mesa y el estado en el que se encuentran, lo cual agilizaría el tiempo de respuesta en

casos de llamar al camarero, o de haber realizado el pago y encontrarse a la espera de confirmar el pago.

Esta idea no ha sido implementada debido a que se podrían saturar los dispositivos de los camareros con mensajes sobre alertas.

Google Pay: Inicialmente se pensó en utilizar Google Pay como proveedor de pagos. Una vez se buscó solución a dicha propuesta, se encontró que el sistema ofrecido por Google no funciona como pasarela de pagos, sino que provee de los datos de tarjeta de crédito a las distintas pasarelas de pago compatibles.

Validar una aplicación para poder utilizar Google Pay en ella como proveedor de datos de la tarjeta es una tarea costosa, ya que Google tiene que comprobar la validez de la organización de la aplicación que se ha diseñado para evitar fraudes y problemas de seguridad.

Por tanto, se recurrió a una plataforma de como Stripe. Su implementación en Flutter incluye un simple formulario para indicar los datos de la tarjeta de crédito del usuario. Esto puede dar lugar a dudas sobre la seguridad y confidencialidad de los datos en la aplicación, por lo que lo más conveniente sería que un sistema más fiable como Google Pay ofrezca los datos de la tarjeta, y finalmente Stripe realice el cargo.

Más proveedores de Login: Actualmente la aplicación acepta autenticaciones mediante el servicio ofrecido por Google Sign In. Añadir más métodos de inicio de sesión (Facebook, Microsoft…) puede ser útil para los usuarios que no dispongan de cuenta de Google, pero sí de alguno de estos otros proveedores.

8. Glosario

Flutter

Flutter es un conjunto de herramientas para el desarrollo de aplicaciones multiplataforma. Fue creado por Google con el objetivo de crear aplicaciones Android, iOS y web. Utiliza el lenguaje de programación Dart.

Dart

Lenguaje de programación de código abierto desarrollado por Google. Su objetivo es permitir a los desarrolladores utilizar un lenguaje orientado a objetos con análisis estático.

Análisis estático de software

Es un tipo de análisis que permite ser realizado sin la necesidad de ejecutar el programa, sino en el código fuente y en el código objeto.

JSON

Formato de texto simplificado con el objetivo de intercambiar datos entre sistemas. Es más sencillo de analizar que otros formatos.

XML

Metalenguaje que permite definir lenguajes de marcas utilizando elementos “tags”, los cuales encapsulan la información entre símbolos “< >”.

Firebase

Plataforma para el desarrollo de aplicaciones web y móvil. Fue creada por Google y ofrece una serie de servicios a los usuarios.

Storage

Servicio ofrecido por Firebase para la carga y descarga de archivos en aplicaciones Firebase. Se utilizará para almacenar las imágenes utilizadas en la aplicación.

Auth

Servicio ofrecido por Firebase para la autenticación de usuarios. Permite la autenticación mediante proveedores de inicio de sesión como Google, Facebook, Microsoft…

Firestore

Servicio de almacenamiento ofrecido por Firebase. Se trata de una base NoSQL que almacena la información en colecciones.

Colección

Datos almacenados en una base de datos Firestore. Cada documento presente en la colección contiene campos con su información. Equivalente a tabla en modelo relacional.

Listener

Objeto de escucha, el cual es notificado de los cambios realizados en el documento a la escucha, localizado en una base de datos remota.

Denormalizar

Denormalizar una base de datos es el proceso de optimización de bases de datos agregando datos redundantes.

Widget

Elemento visual utilizado en Flutter. Describen la vista de una pantalla según su configuración y estado.

Orden

Conjunto de comandas que se han realizado en la visita de un usuario a un establecimiento determinado.

Comanda

Conjunto de platos y bebidas que el usuario ha solicitado para su preparación.

9. Bibliografía

[1] F. R. &. Service, «foodretail.es - La hostelería incrementa sus ventas el 100,7% respecto a 2021,» 28 Marzo 2022. [En línea]. Available: https://www.foodretail.es/horeca/hosteleria-incrementa-ventas-respecto_0_1642335759.html. [Último acceso: 2 Abril 2022].

[2] hostelpro, «revistahostelpro.com - Digitalización y terrazas claves en el incremento en ventas del 100’7% en hostelería,» 28 Marzo 2022. [En línea]. Available: https://revistahostelpro.com/comunicados/digitalizacion-y-terrazas-claves-en-el-incremento-en-ventas-del-1007-en-hosteleria. [Último acceso: 2 Abril 2022].

[3] «medium.com - MVC Design Pattern in Flutter,» 20 Noviembre 2020. [En línea]. Available: https://medium.com/flutter-community/mvc-design-pattern-in-flutter-4ad6641b7863. [Último acceso: 13 Abril 2022].

[4] J. Mohite, «medium.com - Flutter: MVVM Architecture,» 13 Diciembre 2020. [En línea]. Available: https://medium.com/flutterworld/flutter-mvvm-architecture-f8bed2521958. [Último acceso: 13 Abril 2022].

[5] J. Sande, «raywenderlich - State Management With Provider,» 24 Marzo 2020. [En línea]. Available: https://www.raywenderlich.com/6373413-state-management-with-provider. [Último acceso: 13 Abril 2022].

[6] J. G. Gómez Torres, «medium.com - Diferencias entre Realtime Database vs Firestore,» 22 Noviembre 2018. [En línea]. Available: https://jggomezt.medium.com/diferencias-entre-realtime-database-vs-firestore-94364a2cc09e. [Último acceso: 13 Abril 2022].

[7] Google, «firebase.google.com/docs/firestore - Get realtime updates with Cloud Firestore,» 30 Abril 2022. [En línea]. Available: https://firebase.google.com/docs/firestore/query-data/listen. [Último acceso: 30 Abril 2022].

[8] «github - Payment implementation on flutter with Stripe,» 9 Marzo 2022. [En línea]. Available: https://github.com/jaswinda/stripe_payment_integration. [Último acceso: 23 Mayo 2022].

[9] S. Biswas, «blog.codemagic.io - Google Sign-In & Firebase Authentication Using Flutter,» 30 Septiembre 2020. [En línea]. Available: https://blog.codemagic.io/firebase-authentication-google-sign-in-using-flutter/. [Último acceso: 5 Mayo 2022].

[10] Anónimo, «pub.dev/packages - Maps launcher for Flutter,» 29 Julio 2021. [En línea]. Available: https://pub.dev/packages/maps_launcher. [Último acceso: 13 Mayo 2022].

[11] P. Haddad, «medium.com - How To Use Firebase Storage in Flutter,» 6 Mayo 2020. [En línea]. Available: https://medium.com/firebase-tips-tricks/how-to-use-firebase-storage-in-flutter-9f23b04291e5. [Último acceso: 5 Mayo 2022].

[12] J. Rodkey, «itnext.io - To normalize or to denormalize, that is the question -,» 23 Abril 2021. [En línea]. Available: https://itnext.io/to-normalize-or-to-denormalize-that-is-the-question-4cc3968439a8. [Último acceso: 2022 Abril 13].

[13] M. Reyes Seiffert, «leanmind.es - BLoC Pattern con Flutter,» 19 Junio 2020. [En línea]. Available: https://leanmind.es/es/blog/bloc-pattern/. [Último acceso: 13 Abril 2022].

[14] Y. Choudhary, «raywenderlich.com - Firestore Tutorial for Flutter: Getting Started,» 16 Septiembre 2021. [En línea]. Available: https://www.raywenderlich.com/26435435-firestore-tutorial-for-flutter-getting-started. [Último acceso: 23 Abril 2022].

[15] C. Farley, «github.com - Demo of flutter qr_code_scanner,» 17 Enero 2022. [En línea]. Available: https://github.com/chelseafarley/flutter_qr_code_reader. [Último acceso: 27 Abril 2022].

[16] Google, «docs.flutter.dev - Flutter documentation,» 4 Diciembre 2018. [En línea]. Available: https://docs.flutter.dev/. [Último acceso: 4 Abril 2022].