UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Aplicación móvil Android para la búsqueda y valoración de platos de las cartas de los restaurantes

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Tena Gimeno, Juan

Tutor/a: Carsí Cubel, José Ángel

CURSO ACADÉMICO: 2021/2022

Resumen

En este proyecto se ha llevado a cabo el desarrollo de una aplicación para Android con el fin de poseer un lugar dónde buscar y consultar la carta que ofrece un restaurante y poder valorar de forma individual cada uno de sus platos. También ofrece otras funcionalidades como un sistema de reserva de mesas y un sistema de pedidos en mesa una vez se está físicamente en el restaurante. Además se han implementado métodos para formar una base de datos inicial de restaurantes y cartas y para insertar una carta en la base de datos mediante únicamente una imágen de ella.

Palabras clave: aplicación móvil, ocr, restaurante, menú, plato.

Abstract

In this project, the development of an Android application has been carried out in order to have a place to search and consult the menu offered by a restaurant and to be able to rate each of its dishes individually. It also offers other functionalities such as a table reservation system and a table ordering system once you are physically in the restaurant. In addition, methods have been implemented to form an initial database of restaurants and menus and to insert a menu into the database using only an image of it.

Keywords : mobile app, ocr, restaurant, menu, dish.

Tabla de contenidos

Tabla de ilustraciones

Lista de tablas

1. Introducción

Introducción y motivación

Existe el problema de que a la hora de elegir restaurante al que ir, no había una forma eficiente y directa de encontrar la carta/menú digital de este. El proceso pasa por una serie de estrategias arcaicas, tales como: acudir a la página web de cada restaurante o buscar una foto en internet, muy pixelada y de mala calidad en la mayoría de los casos, pero nada realmente directo y claro.

Se puede observar que también existe el hecho de no tener una herramienta o aplicación en la que la valoración fuese a cada plato, dando la oportunidad de buscar los mejores platos de una ciudad/zona (ej. la mejor paella de valencia) y la información sobre qué platos son mejores en cada restaurante gracias a las opiniones de otros usuarios, ayudando al cliente facilitando su elección.

Para resolver estos problemas, se plantea una aplicación centrada en la consulta de la carta de los restaurantes junto con la valoración de forma individual de los platos de esta. Además, se plantean un conjunto de funcionalidades para facilitar o complementar el propósito principal de la aplicación, como son: un sistema de búsqueda por nombre de platos o restaurantes junto a un conjunto de filtros, un mapa dónde visualmente buscar restaurantes cercanos a una ubicación, un sistema de seguimiento de usuarios para ver las valoraciones que realicen, un sistema de reserva de mesas y un sistema de pedidos en mesa.

El objetivo final de la aplicación pues, sería poseer una gran base de datos con cartas de los restaurantes, dónde poder consultarla siempre que se quiera encontrar una carta de un restaurante, al que por ejemplo, un usuario quiera ir. Además facilita encontrar nuevos restaurantes en base a la comida que un usuario prefiere. Para conseguir esta base de datos, además de que un hostelero pueda registrar su restaurante con su carta, se plantea un mecanismo medianamente automatizado para crear perfiles a los restaurantes de una zona sin necesidad de que los hosteleros lo hagan.

Estructura del proyecto

El proyecto consta de la parte principal que es el cliente, la app para Android, junto al servidor. Debido a la naturaleza de la aplicación, es necesario poseer una base de datos de restaurantes y cartas inicial relativamente grande para que esta pueda ser utilizada por los usuarios que lo deseen, por ello, el proyecto consta de una segunda parte que consiste en un conjunto de programas para conseguir este objetivo. Estos programas consistirán en el scraping de dos portales de comida a domicilio (JustEat y UberEats), para conseguir cartas de los restaurantes y el portal Tripadvisor para conseguir información de estos. Además, otro programa que analizará el resultado obtenido de un sistema de OCR para digitalizar las cartas gracias a una imagen de estas.

2. Estado del arte

La aplicación gira en torno a unas funcionalidades principales, que serán las que se comparará con otros servicios que poseen algunas de estas. Actualmente no existe ninguna aplicación móvil o web que posea todas las funcionalidades que se plantean al mismo tiempo, las cuáles son:

1) Base de datos de restaurantes con información básica e imágenes.
2) Base de datos de cartas y menús.
3) Sistema de búsqueda de restaurantes.
4) Sistema de búsqueda y valoración de platos.
5) Sistema de reserva de mesa.
6) Sistema de pedidos online en mesa.

Los servicios que más se aproximan a este proyecto, y poseen alguna de estas funcionalidades, son:

Servicios	Ventajas	Inconvenientes
Tripadvisor / Google Maps	Grandes bases de datos de restaurantes con información básica e imágenes y búsqueda de estos.	Carecen del resto de funcionalidades. La carta se puede encontrar en ocasiones en fotografía.
El Tenedor [1]	Posee las cartas y menús de los restaurantes. Centrada en el sistema de reserva de mesas.	Carece de una gran base de datos de restaurantes y de la búsqueda y valoración de platos, además del sistema de pedidos en mesa.
Foodyt [2]	Posee sistemas de búsqueda por restaurantes y platos. Posee valoración de platos.	Carece de una gran base de datos en general, además del resto de sistemas.
QRCarta [3]	Carta digitalizada para ver en el restaurante mediante un código QR.	No está centrada en poseer una base de datos de restaurantes con búsquedas y valoraciones, ni el resto de sistemas.

Ilustración 1. Panorama competitivo

3. Modelo de negocio

Monetización

La monetización de la aplicación consiste principalmente en dos partes, la primera, que sería con una **suscripción premium** que dispondrá cada restaurante mediante la cual podrá utilizar ciertas funciones exclusivas:

- Sistema de reservas: Un sistema de reserva de mesas directamente desde la aplicación.
- Sistema de pedidos en mesa: Un sistema de pedidos en mesa desde la aplicación, al cual se puede acceder simplemente escaneando un código *QR*, único para cada mesa, y todos los usuarios de la mesa podrán acceder a un pedido en conjunto en tiempo real.

Esta suscripción tendrá un coste de 15€ mensuales.

La otra forma de monetización consiste en un sistema de **patrocinados**, donde los restaurantes podrán salir en los primeros lugares cuando un usuario hace una búsqueda cualquiera, o en las listas de restaurantes de la pantalla principal. Cada restaurante podrá comprar distintos *packs* con cierta cantidad de visitas, estas visitas se irán reduciendo cada vez que un usuario entre al perfil del restaurante, por supuesto, sólo si ha accedido cuando este se mostraba como patrocinado.

Los precios de cada pack serán los siguientes:

Ilustración 2. Precios visitas

Lean Canvas

El *Lean Canvas* permite documentar el modelo de negocio mediante una plantilla que recolecta toda la información respecto a la gestión estratégica.

<table>
<tr>
<td rowspan="2">PROBLEMAS

- Falta de disposición de los menús y cartas en los restaurantes vía online.

- No hay una forma real de encontrar restaurantes filtrando por platos, junto con el precio de estos.

- No se encuentra disponible ninguna herramienta que permita saber qué platos son los mejor valorados o simplemente los mejores de cada restaurante (sirviendo estos de referencia para el cliente).

-</td>
<td>SOLUCIÓN

- Una aplicación móvil que permita a los restaurantes tener un perfil donde puedan subir su menú/carta, junto con una foto de cada plato.
Por lo que respecta a los usuarios: podrán saber la carta/menú de cada restaurante; podrán filtrar por platos, combinación de estos, alérgenos, precio real (no una estimación), valoración de cada plato, etc.</td>
<td rowspan="2">PROPOSICIÓN DE VALOR ÚNICA

- Un portal de búsqueda de restaurantes único en dar la oportunidad de conocer, valorar y filtrar por platos.</td>
<td>VENTAJA ESPECIAL

- Primeros en el mercado.

- Banco de datos (menús y cartas) en crecimiento.

- La situación actual por el Covid, obliga a que sea necesario tener el menú digital, lo que haría que más usuarios utilizasen nuestra aplicación.</td>
<td rowspan="2">SEGMENTOS DE CLIENTES

- Nuestro nicho de mercado es el de usuarios adultos, habituados a utilizar el móvil (en especial aplicaciones “similares”) y que acostumbren a comer fuera.</td>
</tr>
<tr>
<td>MÉTRICAS CLAVE

- Número de usuarios registrados.

- Número de usuarios activos.

- Número de administradores de restaurantes que promocionan su restaurante.</td>
<td>CANALES

- Campañas de publicidad en redes sociales mediante anuncios y con promociones de personalidades del sector.</td>
</tr>
<tr>
<td colspan="3">ESTRUCTURA DE COSTES

- Marketing online.
- Servidores.
- Posicionamiento SEO.
- Personal.</td>
<td colspan="3">FLUJO DE INGRESOS
- Sistema de promoción de restaurantes dentro de la aplicación.
- Sistema de suscripción premium de los restaurantes.
- Financiación inicial.</td>
</tr>
</table>

Ilustración 3. Lean Canvas

4. Metodología y planificación

Metodología

A continuación, se detalla la metodología utilizada en el desarrollo del proyecto, que en este caso, ha sido una metodología clásica **en cascada**[4].

Esta metodología plantea la construcción del proyecto en distintas etapas dónde cada una depende de la anterior. Estas etapas son: análisis, diseño, implementación, pruebas y mantenimiento:

- Análisis: Proceso de recopilación de los requisitos.
- Diseño: El proceso de diseño traduce los requisitos en una representación del software con la calidad requerida antes de que comience la implementación.
- Implementación: Fase de creación del código fuente en base al diseño.
- Pruebas: Comprobación del correcto funcionamiento de los elementos implementados.
- Mantenimiento: Actualización de cambios o arreglo de errores del software ya entregado.

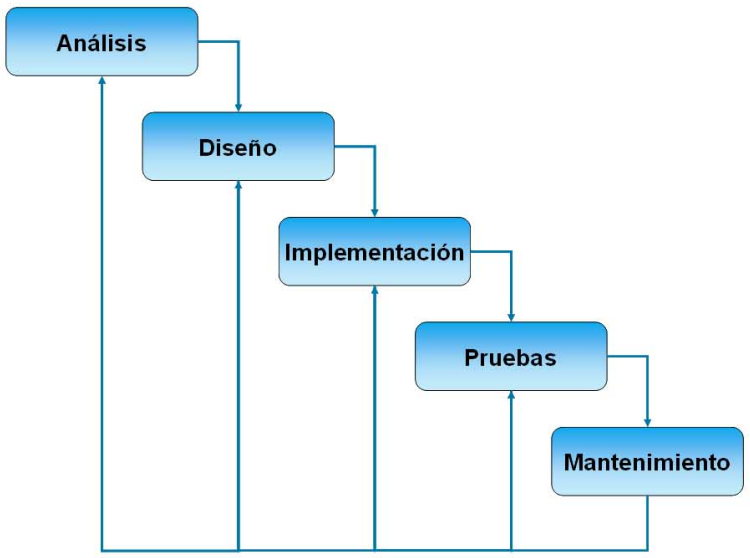

Ilustración 4. Metodología

Planificación

En este apartado, se observarán las distintas etapas del proyecto y el tiempo que se le ha dedicado a cada de una de ellas. Está planificado para una jornada de 6 horas, de lunes a viernes.

Cada una de estas etapas, se descompondrá en partes más concretas.

	Empieza	Termina	Días
Análisis	01-01-2021	09-01-2021	7
Investigación de tecnologías	01-01-2021	05-01-2021	4
Especificación de requisitos	06-01-2021	09-01-2021	3
Diseño	10-01-2021	10-02-2021	29
Arquitectura	10-01-2021	15-01-2021	5
Base de datos	16-01-2021	22-01-2021	6
Interfaz gráfica	23-01-2021	10-02-2021	18
Implementación	11-02-2021	31-05-2021	108
Instalación entorno	11-02-2021	13-02-2021	2
Desarrollo completo	14-02-2021	31-05-2021	106

Ilustración 5. Planificación del proyecto

5. Especificación de requisitos

Requisitos funcionales

A continuación, se muestran la lista de funcionalidades que ha de ofrecer la aplicación:

REQ01 - Inicio de sesión: El usuario podrá iniciar sesión mediante el método clásico, con email y contraseña, o a través de los servicios de *Google* o *Facebook*.

REQ02 - Búsqueda de restaurantes: El usuario podrá buscar restaurantes por su nombre, filtrando por el tipo de comida, la distancia máxima a la que se encuentra y ordenando el resultado por diferentes criterios (relevancia o distancia).

REQ03 - Búsqueda de platos: El usuario podrá buscar platos y su respectivo restaurante por su nombre, filtrando por la nota que tengan, un precio máximo o los alérgenos y ordenando el resultado por diferentes criterios (precio o relevancia). Además, podrá buscar hasta 3 platos distintos y saldrán los restaurantes que coincida que tengan esos platos.

REQ04 - Mapa: Existirá un mapa dónde se pueda buscar por ubicación exacta cualquier restaurante.

REQ05 - Pantalla inicial: Existirá una pantalla inicial donde el usuario verá restaurantes y platos cercanos, populares o vistos recientemente.

REQ06 - Ubicación: El usuario podrá seleccionar una ubicación en el mapa para poder buscar los restaurantes como si estuviera en esa ubicación.

REQ07 - Pantalla de mejores: Existirá una pantalla donde el usuario verá los mejores restaurantes y platos cercanos a su ubicación.

REQ08 - Favoritos: Se podrán añadir los restaurantes o los platos a una lista de favoritos.

REQ9 - Historial de valoraciones: Se podrá consultar un historial de las valoraciones realizadas.

REQ10 - Información del restaurante: Se podrá consultar la información de un restaurante si existiese, como tipo de comida, teléfono, ubicación en el mapa, email, página web, horario de apertura y una galería de imágenes.

REQ11 - Carta y platos del restaurante: Se podrá consultar una carta y un menú diario del restaurante si existiesen. En cada plato se mostrará la cantidad de votos, su nota media y se podrá votar con una puntuación de 0 a 5 estrellas junto a un comentario. Además de otra información como su descripción, una imagen o sus alérgenos, junto a los comentarios realizados por otros usuarios.

REQ12 - Propietario: Si un usuario es propietario de un restaurante, tanto si ya existe en la base de datos como si no, podrá solicitar su control del perfil o su creación de forma gratuita mediante una verificación de identidad.

REQ13 - Perfil de restaurante propietario: Si un usuario es propietario de un restaurante podrá editar la información, su carta, su menú del día, dar acceso a otros propietarios o administradores y la creación de etiquetas con un código QR para las mesas, una vez tenga el control del perfil de este. Mediante una tarifa de pago, podrá optar a un servicio de reserva de mesas, un servicio de 7pedidos en mesa para los comensales mediante la aplicación y un servicio de consulta de estadísticas.

REQ14 - Edición de perfil: Se podrá editar el perfil del usuario.

REQ15 - Traducción de carta: Se podrá traducir la carta a distintos idiomas.

REQ16 - Código QR: Si un usuario es propietario de un restaurante podrá generar códigos QR para poner en las mesas, los cuáles llevarán a los comensales al sistema de pedidos del restaurante si este tiene la tarifa premium, o directamente al perfil de restaurante en caso contrario.

REQ17 - Escaneo de QR: Existirá un botón dónde se abrirá un escáner para los códigos QR, una vez se detecte uno válido se activará el sistema de pedidos de su respectiva mesa.

REQ18 - Tickets: Se podrán enviar *tickets* de ayuda o consulta sobre la aplicación.

REQ19 - Notificaciones: Existirá un sistema de notificaciones las cuales se podrán consultar dentro de la app.

REQ20 - Restaurante sin carta: Si un restaurante no posee una carta subida, un usuario puede colaborar subiendo distintas fotos de esta para que en un futuro pueda ser incluida.

REQ21 - Pantalla social: Se podrá acceder a una pantalla en la que buscar otros usuarios y poder ver las últimas valoraciones de los usuarios que ya sigues.

Requisitos no funcionales

Usabilidad

- La aplicación contará con un manual de usuario disponible para los administradores de locales.
- La aplicación proporcionará mensajes de error que sean informativos y orientados para los usuarios normales.
- La aplicación se adaptará a distintos tamaños de pantalla.
- La aplicación funcionará en *Android* a partir de una versión mínima de *Android 10.0*.

Eficiencia y rendimiento

- El servidor deberá funcionar adecuadamente con hasta 2000 usuarios con sesiones concurrentes.
- En el sistema de pedidos, si un usuario añade, quita o hace un cambio a un plato de la lista el resto de usuarios de la mesa lo podrán ver en tiempo real.

Seguridad

- La edición de datos de un local sólo podrá llevarse a cabo por sus administradores.
- Se realizará una copia de seguridad de la base de datos cada poco tiempo.

Casos de uso

Los diagramas de casos de uso están divididos en dos, uno para el usuario **estándar**, que puede ser un comensal o simplemente utiliza la aplicación para consultar información, (una carta, las valoraciones de un plato, etc.) y otro para el usuario que es **propietario** de un local.

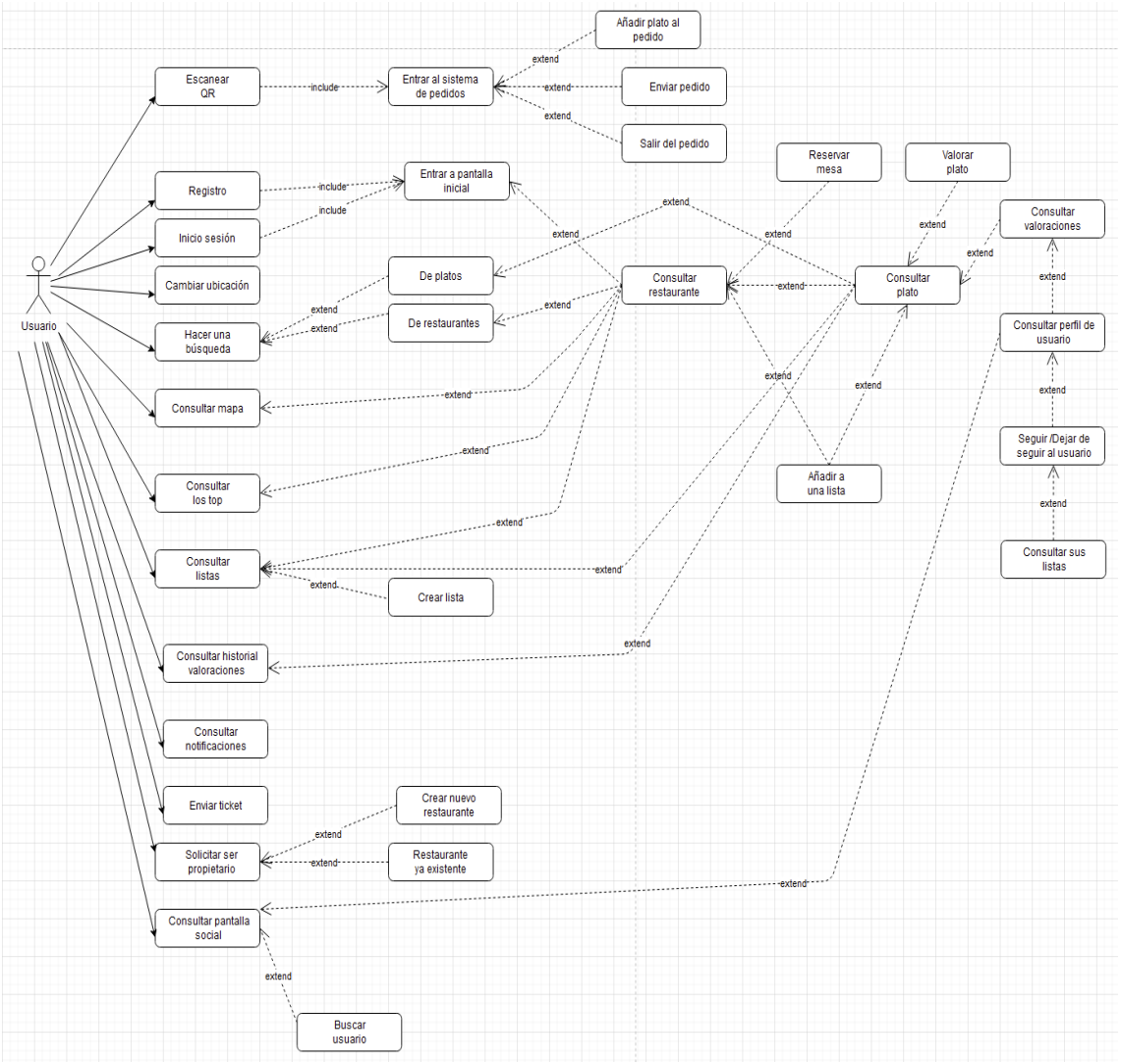

Ilustración 6. Casos de uso Usuario estándar

A continuación se puede ver definición textual de los casos de uso del usuario estándar:

CU001 Registro
Descripción: El usuario podrá registrarse desde la primera pantalla de la aplicación mediante una cuenta de *Google, Facebook* o usando email y contraseña.
Entrada: Pantalla de inicio de sesión. **Salida:** Pantalla de inicio.

Tabla 1. Registro

CU002 Inicio de sesión
Descripción: El usuario podrá iniciar sesión desde la primera pantalla de la aplicación mediante una cuenta de *Google, Facebook* o usando email y contraseña. Una vez iniciado, no necesitará volver a hacerlo hasta que cierre la sesión manualmente.
Entrada: Pantalla de inicio de sesión. **Salida:** Pantalla de inicio.

Tabla 2. Inicio de sesión

CU003 Escanear QR
Descripción: El usuario podrá, desde la pantalla de inicio, pulsar un botón en la parte superior para acceder a la pantalla de escaneo, dónde una vez escaneado un código QR válido, le llevará al sistema de pedidos de su mesa correspondiente.
Entrada: Pantalla de inicio. **Salida:** Sistema de pedidos.

Tabla 3. Escanear QR

CU004 Añadir plato al pedido
Descripción: El usuario podrá, desde un botón en la parte inferior del sistema de pedidos, acceder a la lista de platos, donde podrá añadir un plato al pedido eligiendo la cantidad y añadiendo, opcionalmente, una nota para el restaurante. Una vez añadido, podrá también editarse o eliminarse antes de enviar el pedido.
Entrada: Sistema de pedidos. **Salida:** Pantalla de selección de platos.

Tabla 4. Añadir plato al pedido

CU005 Enviar pedido
Descripción: El usuario podrá apretar un botón para mandar su pedido actual al restaurante, para que este, lo visualice y se lo entregue posteriormente. Los elementos enviados no podrán editarse o eliminarse pero pueden añadirse nuevos.
Entrada: Sistema de pedidos. **Salida:** Sistema de pedidos.

Tabla 5. Enviar pedido

CU006 Salir de pedido
Descripción: El usuario podrá salir del sistema de pedidos apretando el botón de la parte superior izquierda y no podrá volver a acceder a él, a menos que volviese a escanear el mismo código QR.
Entrada: Sistema de pedidos. **Salida:** Pantalla de inicio.

Tabla 6. Salir de pedido

CU007 Cambiar ubicación
Descripción: El usuario podrá desde la pantalla de inicio, en el botón a la derecha de la barra de búsqueda, cambiar su ubicación manualmente para poder ver los restaurantes sugeridos como si estuviera ubicado en ella.
Entrada: Pantalla de inicio. **Salida:** Pantalla de inicio.

Tabla 7. Cambiar ubicación

CU008 Hacer una búsqueda
Descripción: El usuario podrá desde la pantalla de inicio, apretando en la barra de búsqueda, acceder a la pantalla de búsqueda. Una vez aquí, podrá realizar una búsqueda de restaurante o de un conjunto de platos, filtrando mediante diferentes parámetros y eligiendo un criterio de ordenación.
Entrada: Pantalla de inicio. **Salida:** Pantalla de búsqueda.

Tabla 8. Hacer una búsqueda

CU009 Consultar mapa
Descripción: El usuario podrá consultar el mapa desde el botón de la barra inferior, en el cual, podrá ver los restaurantes dada una ubicación.
Entrada: Desde la barra inferior. **Salida:** Mapa.

Tabla 9. Consultar mapa

CU010 Consultar los Top
Descripción: El usuario podrá consultar los restaurantes y platos top de su zona desde el botón correspondiente en la barra inferior.
Entrada: Desde la barra inferior. **Salida:** Pantalla de top.

Tabla 10. Consultar los Top

CU011 Consultar las listas
Descripción: El usuario podrá consultar las listas personales de restaurantes, o de platos, desde el menú de perfil que se accede mediante el botón de la derecha de la barra inferior. Una vez entre el usuario, elegirá si ver las listas de platos o de restaurantes.
Entrada: Pantalla de perfil. **Salida:** Pantalla de listas de favoritos.

Tabla 11. Consultar las listas

CU012 Crear lista
Descripción: El usuario podrá crear una nueva lista mediante el botón inferior de añadir nueva lista. Se le dará un nombre y opcionalmente, una imagen.
Entrada: Pantalla de listas de favoritos. **Salida:** Pantalla de creación de lista.

Tabla 12. Crear lista

CU013 Consultar historial de valoraciones
Descripción: El usuario podrá consultar una lista de todas sus valoraciones a platos desde el menú de perfil, que se accede mediante el botón de la derecha de la barra inferior.

Entrada: Pantalla de perfil. **Salida:** Pantalla de historial de valoraciones.

Tabla 13. Consultar historial de valoraciones

CU014 Consultar notificaciones
Descripción: El usuario podrá consultar una lista de sus notificaciones sin leer desde el menú de perfil, que se accede mediante el botón de la derecha de la barra inferior.
Entrada: Pantalla de perfil. **Salida:** Pantalla de notificaciones.

Tabla 14. Consultar notificaciones

CU015 Enviar ticket
Descripción: El usuario podrá enviar un ticket de ayuda, seleccionando un asunto, desde el menú de perfil que se accede mediante el botón de la derecha de la barra inferior. Se le responderá a su correo electrónico.
Entrada: Pantalla de perfil. **Salida:** Pantalla de envío de ticket.

Tabla 15. Enviar ticket

CU016 Solicitar ser propietario
Descripción: El usuario podrá hacer una petición para que se le adjudique su restaurante como propietario si este ya existe en la base de datos. Se le pedirá que lo confirme mediante el teléfono o el email asociados al restaurante, o mediante el envío de su DNI para confirmar su identidad. Podrá acceder desde el menú de perfil al que se accede mediante el botón de la derecha de la barra inferior.
Entrada: Pantalla de perfil. **Salida:** Pantalla de solicitud de propietario.

Tabla 16. Solicitar ser propietario

CU017 Crear nuevo restaurante
Descripción: El usuario podrá crear un nuevo restaurante desde la pantalla de solicitud de propietario, si este no existe en la base de datos.
Entrada: Pantalla de solicitud de propietario. **Salida:** Pantalla de creación de nuevo restaurante.

Tabla 17. Crear nuevo restaurante

CU018 Consultar restaurante
Descripción: El usuario podrá consultar un restaurante, dónde se le mostrará una galería de imágenes, información sobre el restaurante y la carta y menú del día si tuviese.
Entrada: Desde diferentes ubicaciones de la aplicación. **Salida:** Pantalla de restaurante.

Tabla 18. Consultar restaurante

CU019 Reservar mesa
Descripción: El usuario podrá reservar una mesa en el restaurante, si este servicio está disponible desde el botón de reserva en la pantalla del restaurante. Elegirá un día, la cantidad de personas y una de las horas disponibles.
Entrada: Pantalla de restaurante. **Salida:** Pantalla de reserva de mesa.

Tabla 19. Reservar mesa

CU020 Consultar plato
Descripción: El usuario podrá consultar un plato, dónde verá su valoración si la ha realizado, la imagen de este en gran tamaño y una lista de alérgenos si los tuviese.
Entrada: Pantalla de restaurante. **Salida:** Pantalla de plato.

Tabla 20. Consultar plato

CU021 Valorar plato
Descripción: El usuario podrá valorar un plato con una nota de 0,5 a 5 estrellas en intervalos de 0,5 y pudiendo, escribir opcionalmente un comentario de texto.
Entrada: Pantalla de plato. **Salida:** Pantalla de plato.

Tabla 21. Valorar plato

CU022 Consultar valoraciones del plato
Descripción: El usuario podrá consultar las valoraciones que tengan texto de otros usuarios sobre un plato.
Entrada: Pantalla de plato. **Salida:** Pantalla de valoraciones del plato.

Tabla 22. Consultar valoraciones del plato

CU023 Consultar perfil de usuario
Descripción: El usuario podrá consultar el perfil de otro usuario desde la pantalla de las valoraciones de un plato, donde podrá ver su información y seguir o dejar de seguir a dicho usuario.
Entrada: Pantalla de valoraciones del plato. **Salida:** Pantalla de perfil de usuario.

Tabla 23. Consultar perfil de usuario

CU024 Consultar listas de otro usuario
Descripción: El usuario podrá consultar las listas de favoritos de otros usuarios a través de su perfil.
Entrada: Pantalla de perfil de usuario. **Salida:** Pantalla de listas de favoritos.

Tabla 24. Consultar listas de otro usuario

CU025 Consultar pantalla social
Descripción: El usuario podrá acceder a una pantalla, donde, podrá buscar nuevos usuarios o consultar las últimas valoraciones de los usuarios que sigue.
Entrada: Pantalla de inicio. **Salida:** Pantalla social.

Tabla 25. Consultar pantalla social

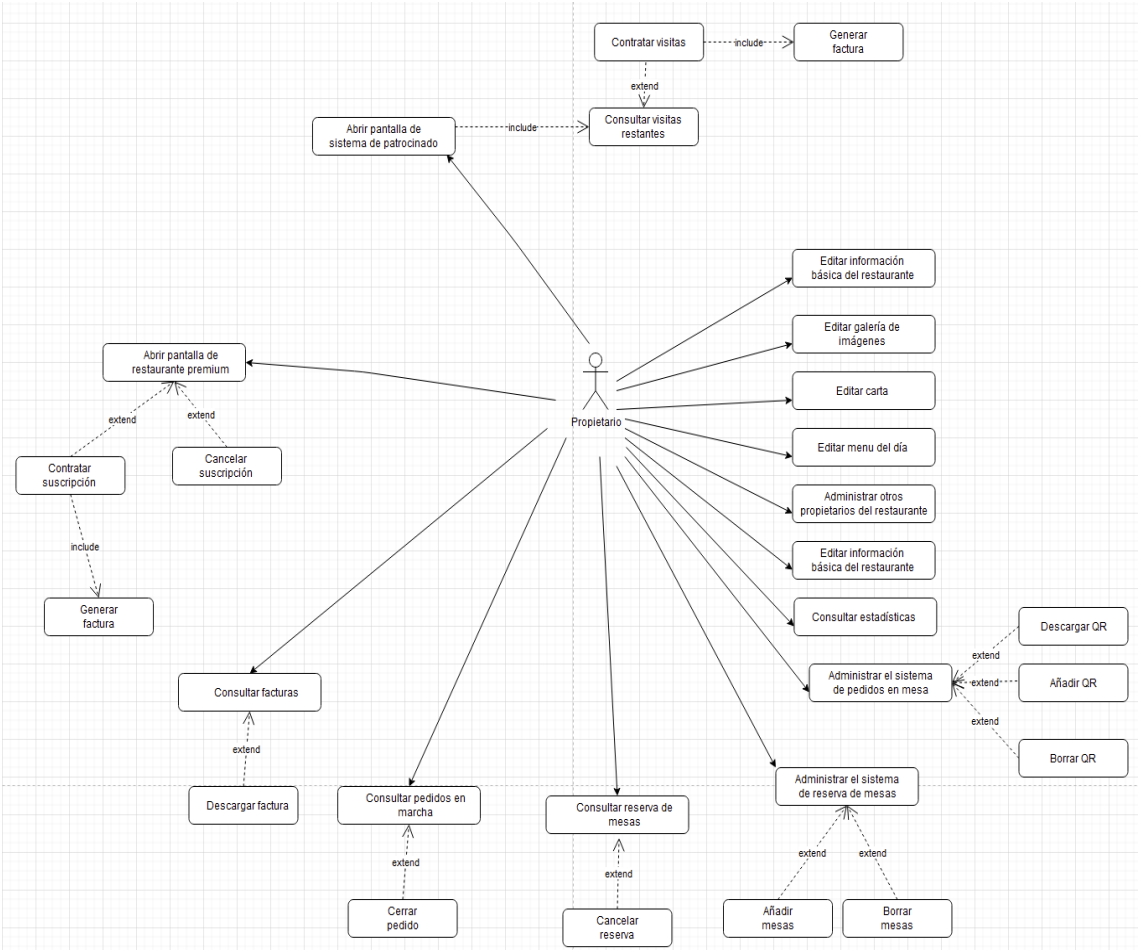

Ilustración 7. Casos de uso propietario

A continuación se observa una definición textual de los casos de uso del propietario:

CU026 Editar información básica del restaurante
Descripción: El propietario podrá acceder a una pantalla para la edición de la información del restaurante, como nombre, teléfono, email, tipos de cocina, etc.
Entrada: Pantalla de restaurante. **Salida:** Pantalla de edición de restaurante.

Tabla 26. Editar información básica de restaurante

CU027 Editar galería de imágenes
Descripción: El propietario podrá acceder a una pantalla para la edición de las imágenes del restaurante.
Entrada: Pantalla de restaurante. **Salida:** Pantalla de edición de galería de imágenes.

Tabla 27. Editar galería de imágenes

CU028 Editar carta
Descripción: El propietario podrá acceder a una pantalla para la edición de la carta del restaurante.
Entrada: Pantalla de restaurante. **Salida:** Pantalla de edición de carta.

Tabla 28. Editar carta

CU029 Editar menú del día
Descripción: El propietario podrá acceder a una pantalla para la edición del menú del día.
Entrada: Pantalla de restaurante. **Salida:** Pantalla de edición del menú del día.

Tabla 29. Editar menú del día

CU030 Administrar otros propietarios
Descripción: El propietario podrá acceder a una pantalla para añadir o quitar propietarios del restaurante.
Entrada: Pantalla de restaurante. **Salida:** Pantalla de administración de propietarios.

Tabla 30. Administrar otros propietarios

CU031 Consultar estadísticas
Descripción: El propietario podrá acceder a una pantalla para consultar estadísticas sobre su restaurante, como número de visitas o valoraciones en el tiempo.
Entrada: Pantalla de restaurante. **Salida:** Pantalla de estadísticas.

Tabla 31. Consultar estadísticas

CU032 Administrar el sistema de pedidos en mesa
Descripción: El propietario podrá acceder a una pantalla para la edición de los códigos QR para los pedidos en mesa.

Entrada: Pantalla de restaurante. **Salida:** Pantalla de administración de sistema de pedidos en mesa.

Tabla 32. Administrar el sistema de pedidos en mesa

CU033 Administrar el sistema de reserva de mesas
Descripción: El propietario podrá acceder a una pantalla para la edición de la cantidad de mesas y sus características para las reservas a través de la aplicación.
Entrada: Pantalla de restaurante. **Salida:** Pantalla de administración del sistema de reserva de mesas.

Tabla 33. Administrar el sistema de reserva de mesas

CU034 Consultar reserva de mesas
Descripción: El propietario podrá acceder a una pantalla para consultar las reservas de cada día desde un calendario, dónde podrán cancelarse si hiciese falta.
Entrada: Pantalla de restaurante. **Salida:** Pantalla de consulta de reserva de mesas.

Tabla 34. Consultar reserva de mesas

CU035 Consultar pedidos en marcha
Descripción: El propietario podrá acceder a una pantalla para consultar los pedidos en marcha para cada mesa.
Entrada: Pantalla de restaurante. **Salida:** Pantalla de consulta de pedidos.

Tabla 35. Consultar pedidos en marcha

CU036 Consultar facturas
Descripción: El propietario podrá acceder a una pantalla para consultar las facturas del restaurante.

Entrada: Pantalla de restaurante. **Salida:** Pantalla de consulta de facturas.

Tabla 36. Consultar facturas

CU037 Abrir pantalla de restaurante premium
Descripción: El propietario podrá acceder a una pantalla para contratar o cancelar el servicio premium, el cual incluye, el sistema de pedidos en mesa, el sistema de reserva de mesas y las estadísticas.
Entrada: Pantalla de restaurante. **Salida:** Pantalla de contratar premium.

Tabla 37. Abrir pantalla de restaurante premium

CU038 Abrir pantalla de sistema de patrocinado
Descripción: El propietario podrá acceder a una pantalla para comprar un número de visitas a su restaurante como patrocinado, el cual aparecerá en las búsquedas y en la pantalla inicial con prioridad sobre el resto.
Entrada: Pantalla de restaurante. **Salida:** Pantalla de sistema de patrocinados.

Tabla 38. Abrir pantalla de sistema de patrocinado

6. Análisis

Diagrama de clases

En base a los requisitos, aquí está el diagrama de clases correspondiente:

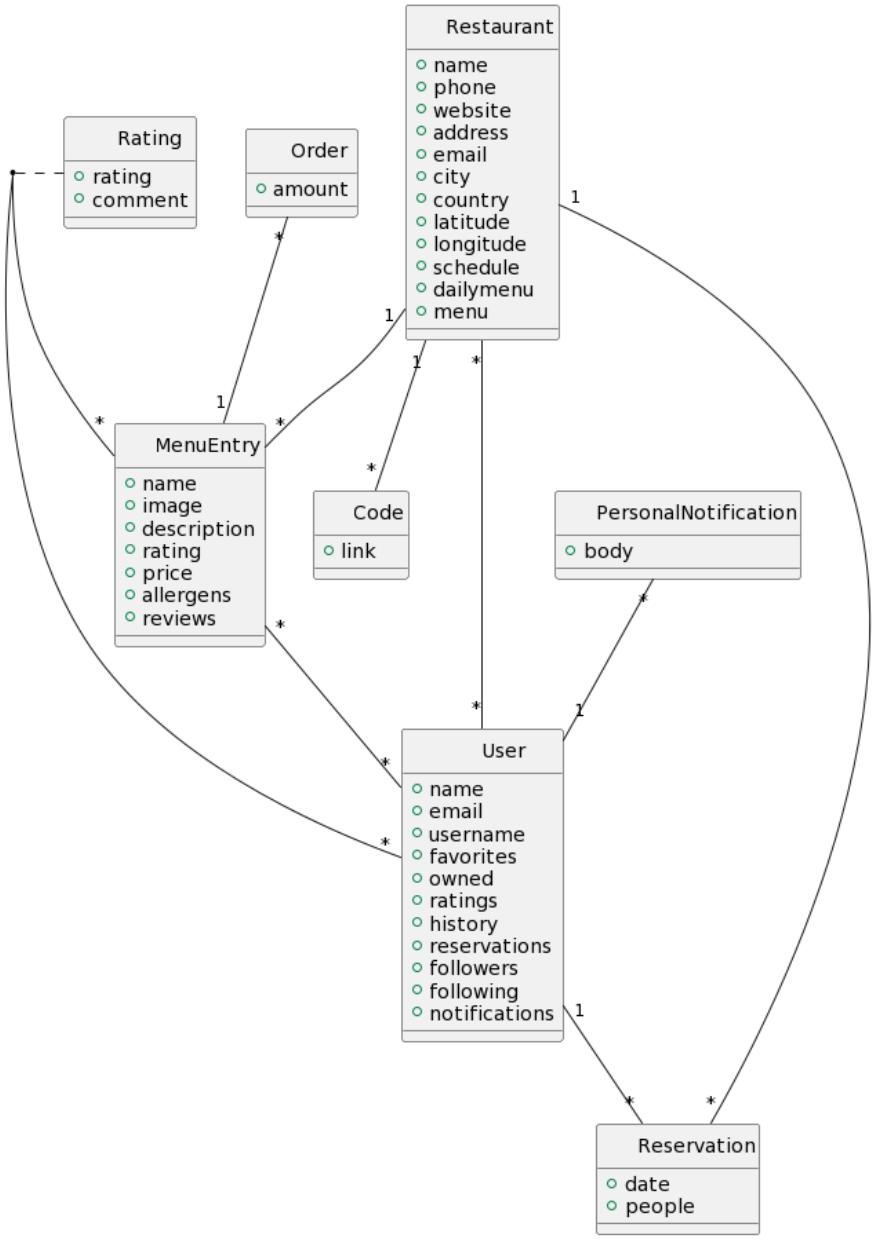

Ilustración 8. Diagrama de clases del análisis

Restaurant: Es la clase que modela la información del restaurante. Incluye atributos para la información básica de este: nombre, teléfono, web, ubicación, tipos de cocina, etc. Además de otros como las traducciones, las mesas o atributos de información de la carta.

Code: Es la clase para los códigos QR del restaurante para las mesas. Con un enlace codificado en el QR para acceder al sistema de pedidos o al perfil del restaurante.

Reservation: Es la clase correspondiente a la reserva de una mesa de un restaurante, para una fecha concreta, hecha por un usuario y para una cantidad de personas dada.

User: Es la clase que modela la información del usuario. Incluye atributos para la información básica como el nombre, email, país, nombre de usuario, etc. También existen otros atributos para la lista de restaurantes o platos favoritos, listas de restaurantes, historial de valoraciones, reservas actuales, restaurantes en propiedad, notificaciones y seguidores/seguidos.

PersonalNotification: Es la clase correspondiente a las notificaciones de un usuario, que siempre vienen por parte de un restaurante, ya sea para aviso de reserva de mesas o cancelación de reservas.

MenuEntry: Es la clase que modela un plato de la carta de un restaurante. Incluye la información básica de este: nombre, imagen, descripción, precio, alérgenos, valoración entre 0 y 5 y las *reviews* de los usuarios.

Rating: Es la clase para la valoración que tiene un usuario sobre un plato.

Order: Es la clase correspondiente a un pedido online de un restaurante. Representa el pedido de un *MenuEntry,* junto con una cantidad, una nota opcional, un atributo para confirmar si ha sido entregado y otro para confirmar que ha sido recibido y está en preparación.

7. Diseño

Arquitectura

La arquitectura general consiste en un sistema **cliente-servidor**. En la cual, los clientes hacen peticiones al servidor y este les contesta con la información deseada. La arquitectura se compone de la siguiente manera:

- Cliente: Los distintos clientes serían los usuarios con la app móvil.
- Servidor: Es un servidor implementado en *Node.js,* que recibe peticiones *HTTP* por parte de los clientes, y se comunica con la base de datos para las operaciones correspondientes a la petición, para posteriormente devolver una respuesta al cliente.
- Base de datos: Es un motor de base de datos *PostgreSQL*, el cual solo es accedido por el servidor.
- Servicios alternativos: También existen una serie de servicios ajenos al servidor y base de datos principal, que se utilizan a través de *Firebase,* como por ejemplo, el sistema de almacenamiento de imágenes y archivos, el sistema de autentificación, el sistema de base de datos en tiempo real (*Firestore*) para el sistema de pedidos en conjunto, o el sistema de notificaciones.

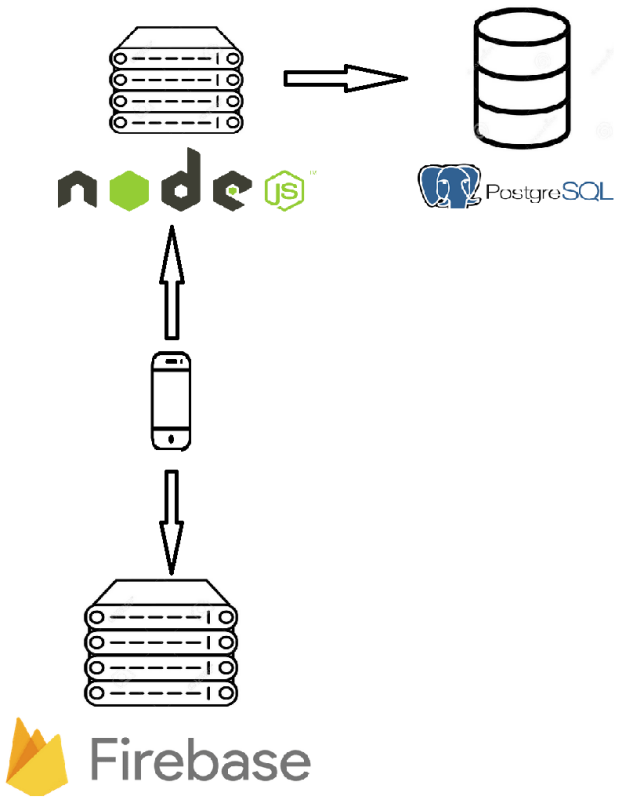

Ilustración 9. Arquitectura

La arquitectura de la aplicación consiste en un modelo de tres capas, donde cada una posee una utilidad propia y ayuda a la organización del código de la app, además de a su posible escalabilidad:

- Capa de presentación: Consiste en la capa externa, la interfaz, que es con la cual interactúa el usuario y donde se expone la información y las funcionalidades que se pueden emplear.
- Capa de lógica de negocio: Es la capa en la cual se ubican las funciones y servicios que se pueden utilizar.
- Capa de persistencia de datos: Es la capa encargada de interaccionar con el servidor, es decir, de recoger y adaptar los datos recibidos de este y pasarlos al resto de la app.

Diagrama de clases detallado

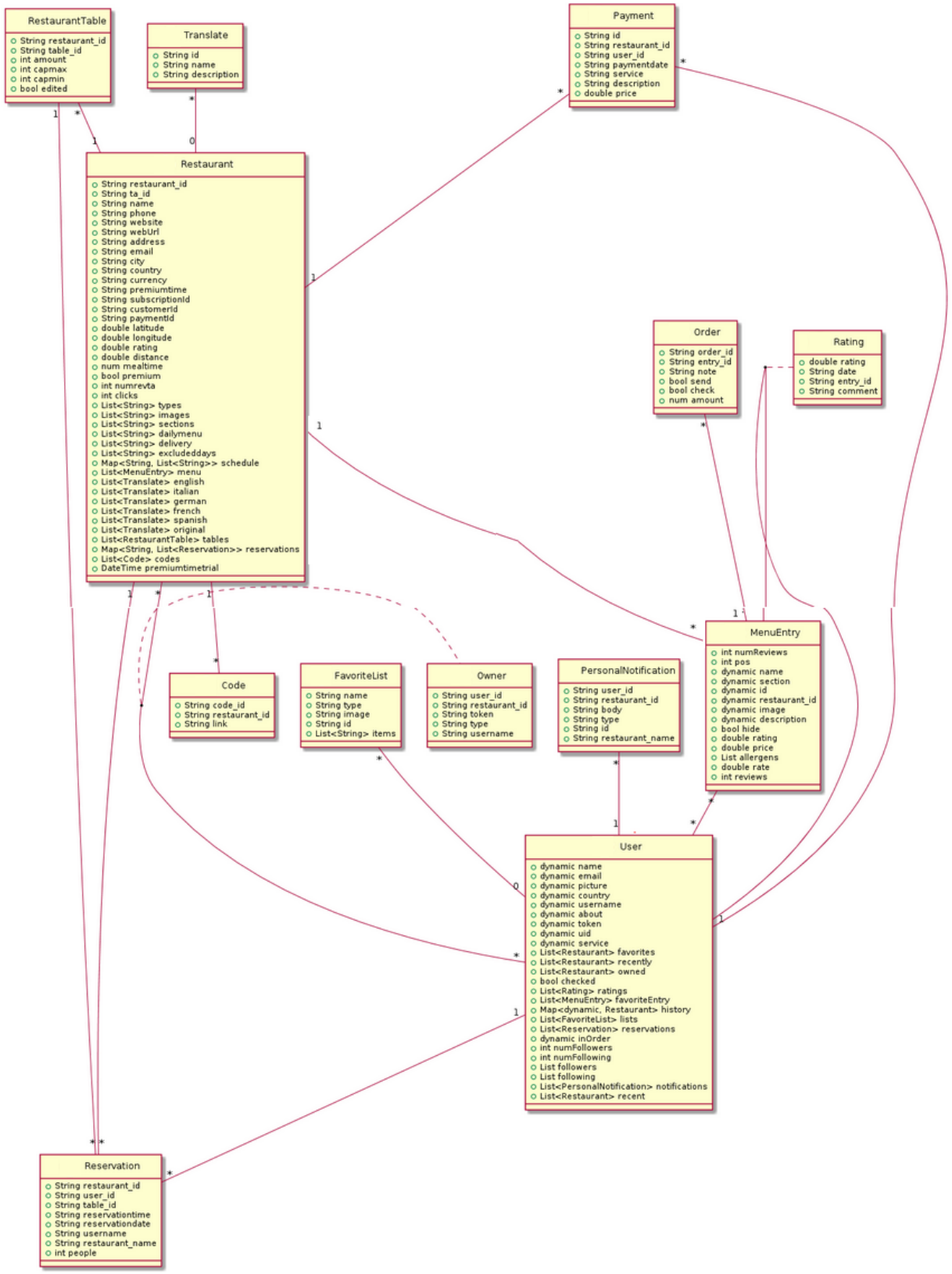

Ilustración 10. Diagrama de clases detallado

Capa de presentación (Interfaz gráfica)

El prototipado y diseño de la interfaz gráfica, se ha realizado a través de la herramienta *Figma*. Se han creado unos diseños aproximados a lo que se espera del producto final.

Pantallas de inicio de sesión y registro

Estas pantallas hacen referencia al requisito funcional REQ01. Si no se ha iniciado sesión aún o se ha cerrado la anterior, la *Ilustración 11* es la primera pantalla que se observa. Si se inicia sesión con *Google* o *Facebook* se saltará directamente a la pantalla de inicio. Si se quiere usar email, se verá la *Ilustración 12*.

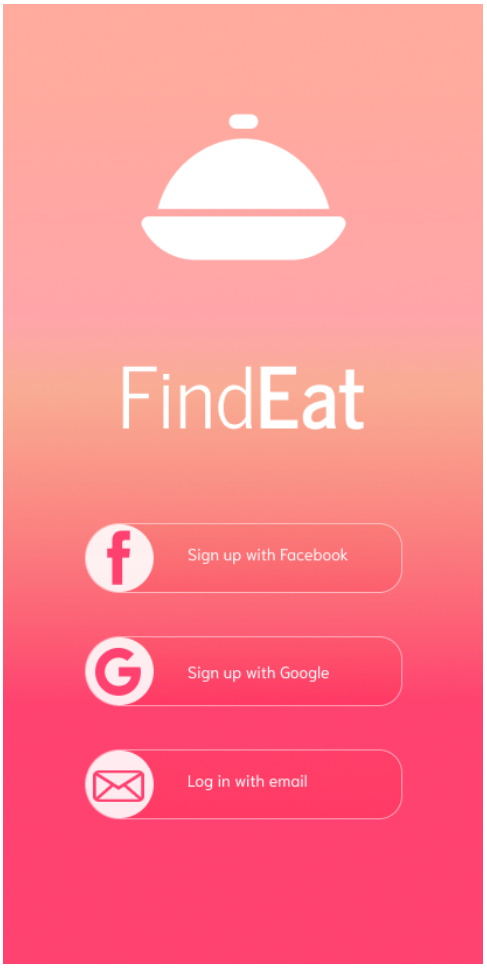

Ilustración 11. Primera pantalla

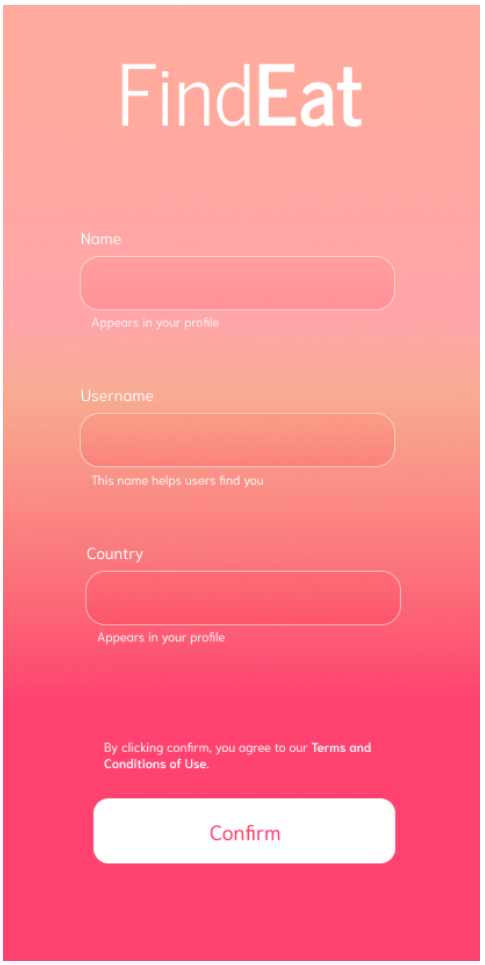

Ilustración 12. Registro email

Pantalla inicial y top

Estas pantallas hacen referencia al requisito funcional REQ05 y al REQ07. En la *Ilustración 13*, podrán observarse un conjunto de restaurantes y platos **recomendados**, además, de poder accederse a la pantalla de búsqueda, de cambio de ubicación, al escáner de códigos QR y a la pantalla de social. En la *Ilustración 14*, podemos observar un conjunto de los mejores platos y restaurantes de la zona.

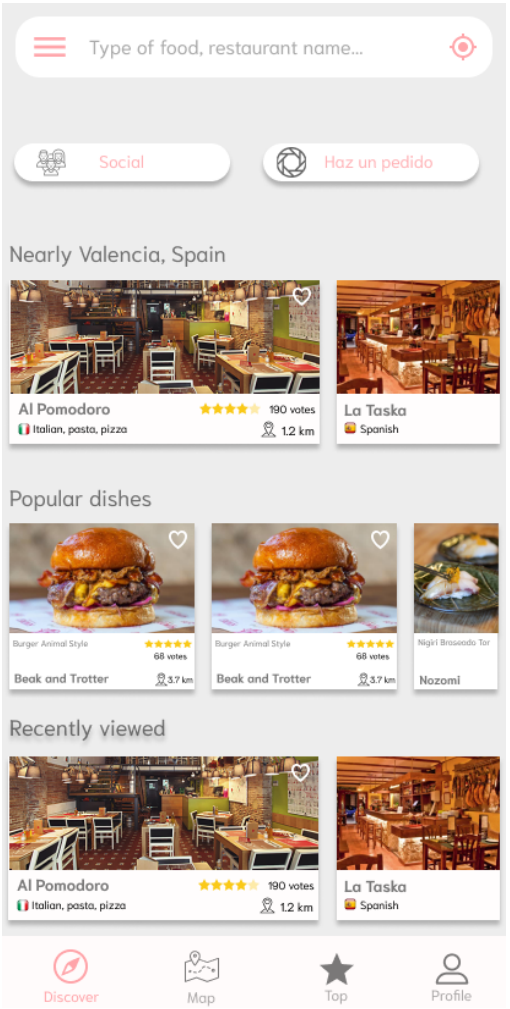

Ilustración 13. Pantalla principal

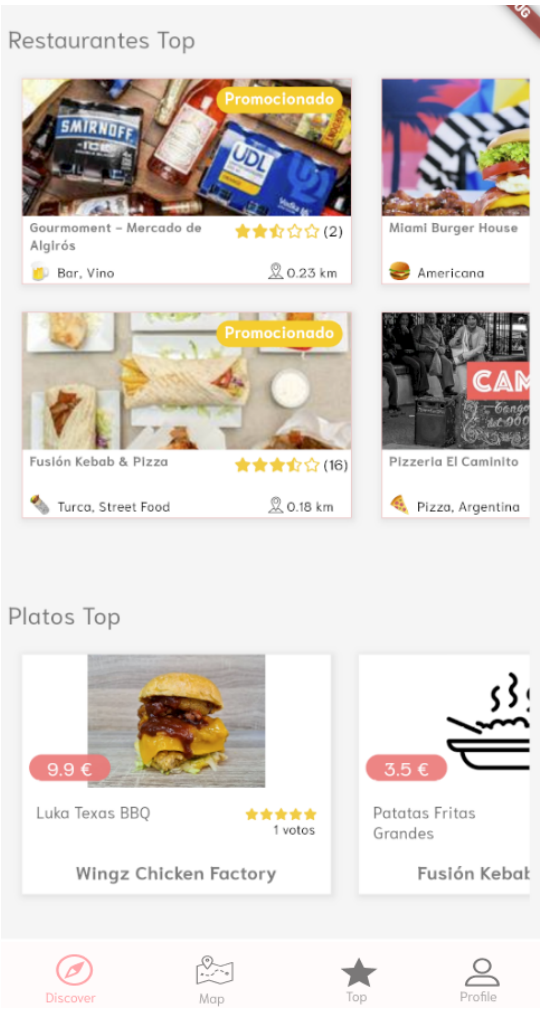

Ilustración 14. Pantalla de tops

Mapa y social

Estas pantallas hacen referencia al requisito funcional REQ04 y REQ21, se observará un **mapa** dónde se podrá buscar los restaurantes en la zona visible (*Ilustración 15*). El icono de cada restaurante, vendrá determinado por el tipo de cocina de este. Una vez se apriete en un icono, saldrá una pequeña ventana informativa del restaurante, la cuál, se podrá volver a apretar para ir a la pantalla del restaurante. En la *Ilustración 16*, tenemos la pantalla **social** dónde se podrá ver las valoraciones de los usuarios a los que se sigue, y otros sugeridos.

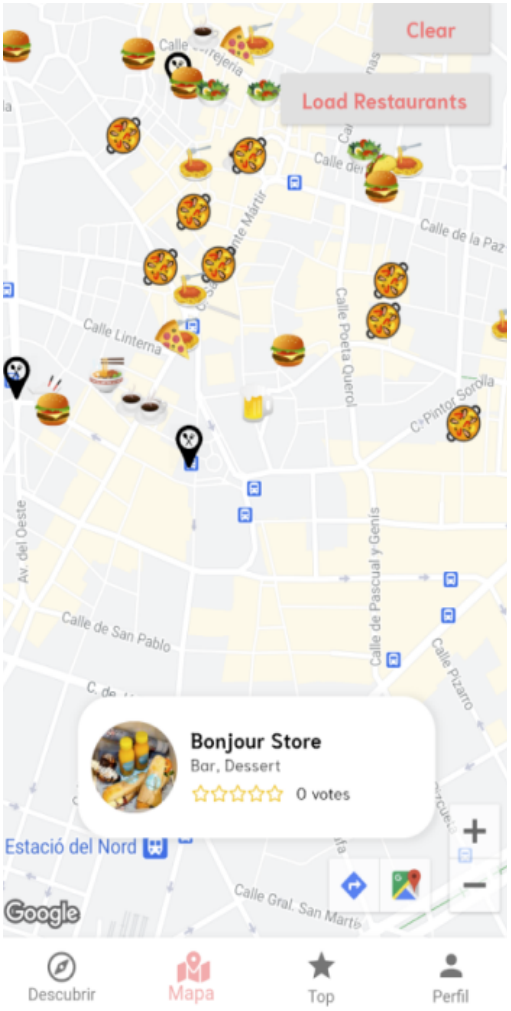

Ilustración 15. Mapa

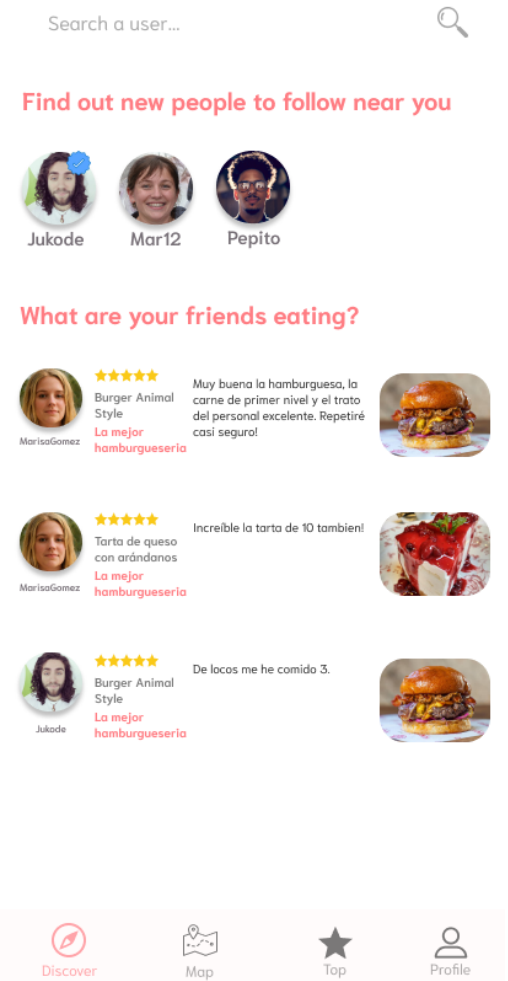

Ilustración 16. Social

Pantalla de restaurante, información y de plato

Estas pantallas hacen referencia a los requisitos funcionales REQ10, REQ11 y REQ13. En la *Ilustración 17*, se puede ver los botones para los usuarios **propietarios** (Reservas, Pedidos y Ajustes), un botón para acceder a la pantalla de información, el de reserva de mesas, un resumen de sus mejores platos y, a continuación, la **carta** y el menú del día si tuviera.

Ilustración 17. Pantalla de restaurante

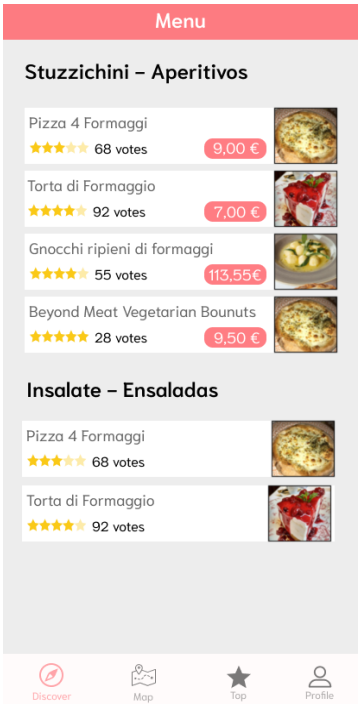

Ilustración 18. Carta

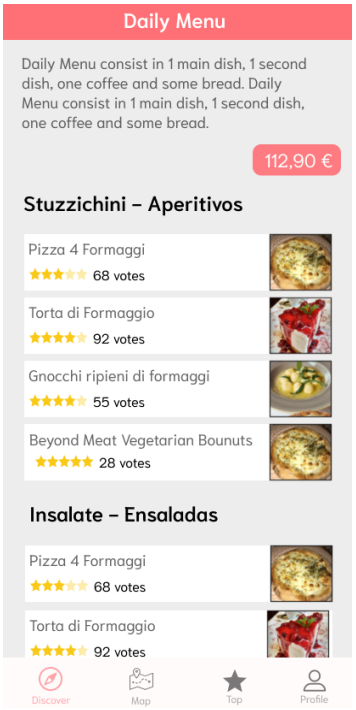

Ilustración 19. Menú diario

Ilustración 20. Pantalla de plato

Ilustración 21. Pantalla de información

Pantallas de usuario

Estas pantallas hacen referencia a los requisitos funcionales REQ08, REQ09, REQ12, REQ14, REQ18 y REQ19. En la *Ilustración 22*, se puede ver todos los menús disponibles para el **usuario**, de los cuales se ven a continuación la edición de perfil, el historial, una lista de favoritos, las notificaciones y la pantalla de selección de *tickets* de ayuda y soporte.

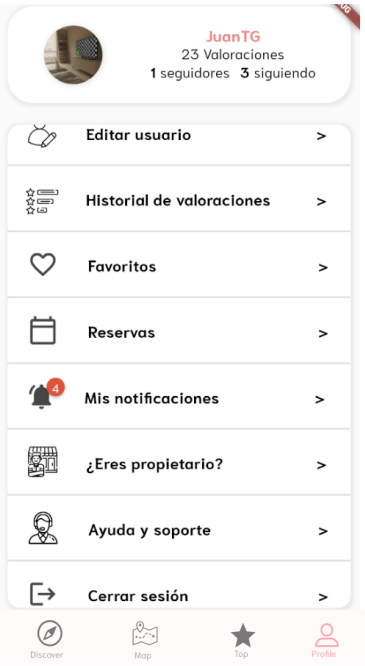

Ilustración 22. Perfil

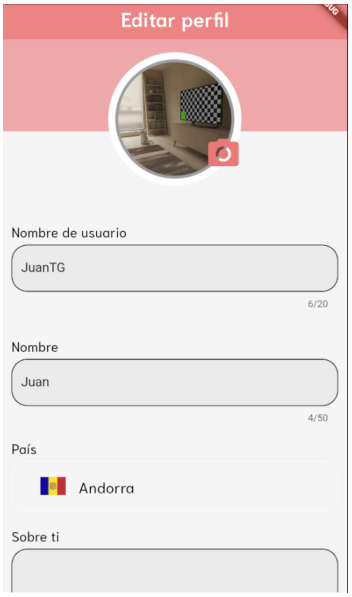

Ilustración 23. Edición

Ilustración 24. Historial

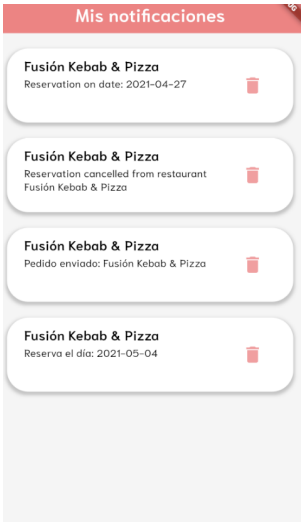

Ilustración 25. Notificaciones

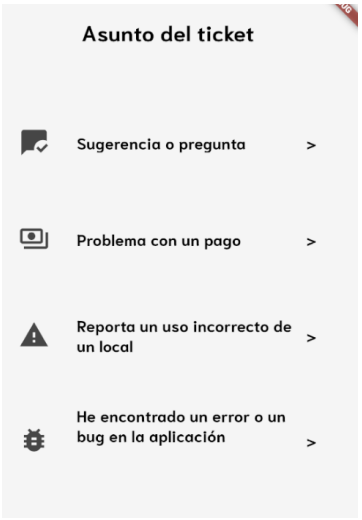

Ilustración 26. Ticket

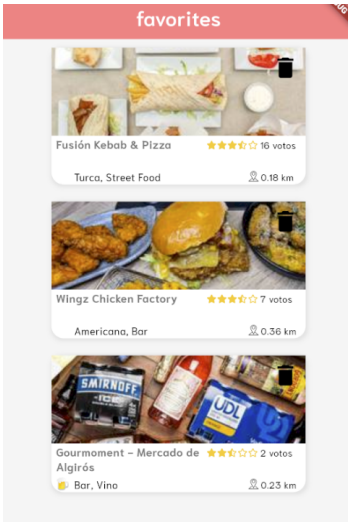

Ilustración 27. Favoritos

Pantallas de administración de restaurante

Estas pantallas hacen referencia a los requisitos funcionales REQ13, REQ14 y REQ16. En la *Ilustración 28*, se puede ver una lista con las opciones disponibles de **administración** del restaurante. En la edición de carta, se podrá marcar un plato como visible, añadir alérgenos, una descripción, una imagen o eliminar el plato además de cambiar su nombre y precio. Se podrá editar el orden de los platos y sus secciones. En la configuración de reservas, se pueden administrar las mesas añadiendo una capacidad mínima y máxima, y una duración estimada del tiempo de comida. Para el sistema de pedidos, se creará un código QR para cada mesa y se podrá descargar en PDF, una imagen con este para imprimir y colocar. Por último, se puede observar la pantalla con la lista de facturas para descargar, junto a un ejemplo de factura, y la pantalla de estadísticas.

Ilustración 28. Administración

Ilustración 29. Edición info.

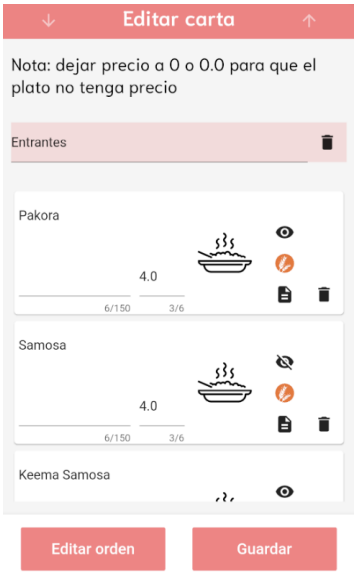

Ilustración 30. Edición carta

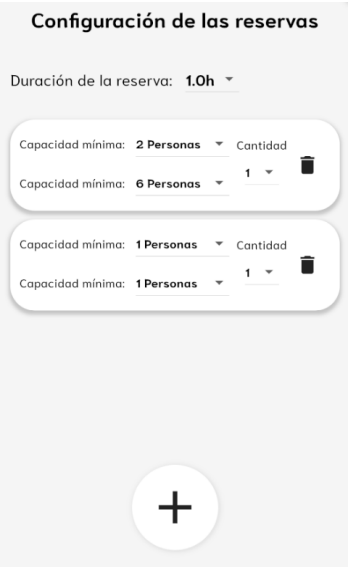

Ilustración 31. Reservas

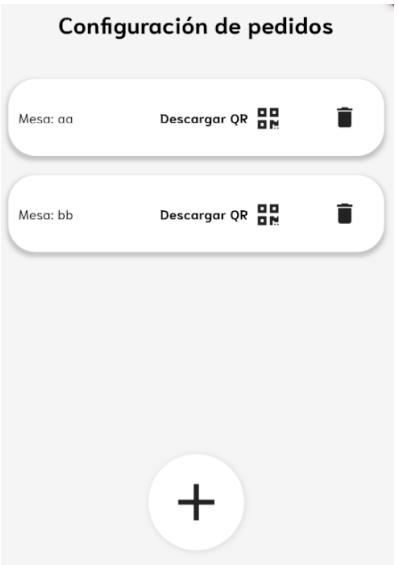

Ilustración 32. Pedidos

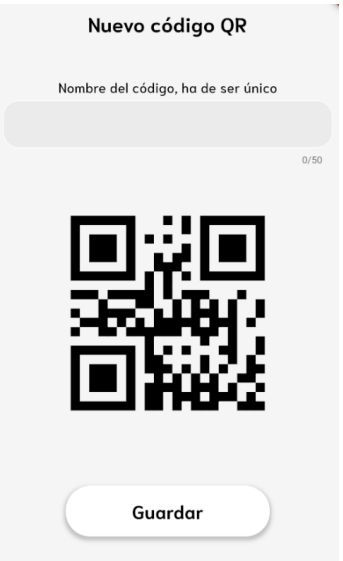

Ilustración 33. Nuevo QR

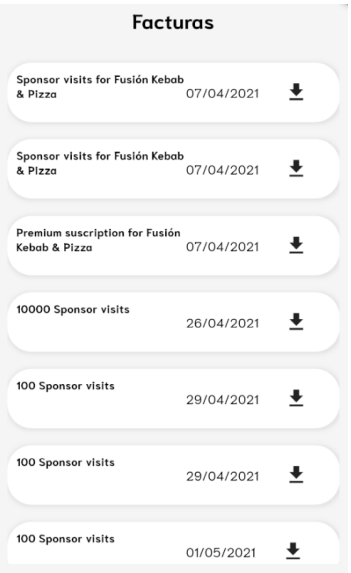

Ilustración 34. Facturas

Ilustración 35. Estadísticas

FindEat

Factura

Fecha: 07/04/2021
Fecha: 4
ID de usuario: l4z3wFH0SeTrVSqVtb8qAsXQijk1
ID del local: 833
Nombre del local: Fusión Kebab & Pizza

Servicio	Descripción	Precio
clicks	Sponsor visits for Fusión Kebab & Pizza	7.0 EUR

Incluye el 10% del IVA 0.7 EUR

Ilustración 36. Factura ejemplo

Pantallas de reserva de mesas

La *Ilustración 37*, hace referencia a los requisitos funcionales REQ13, dónde se observa el proceso de **reserva de mesas** que puede utilizar un usuario.

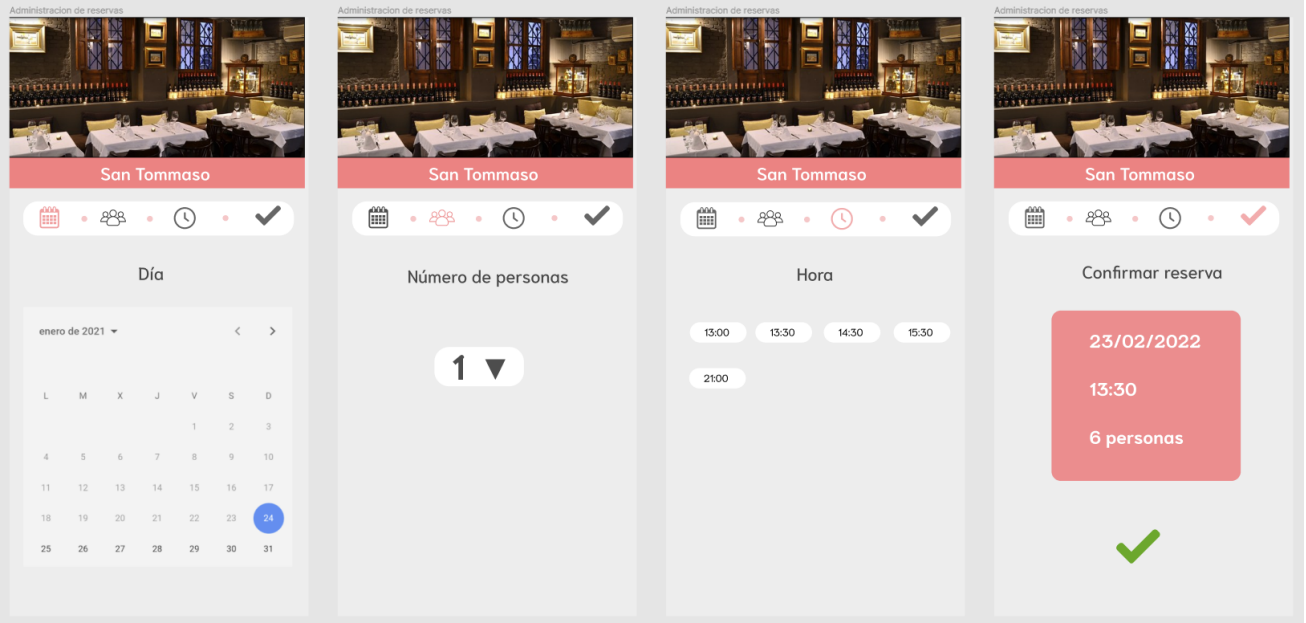

Ilustración 37. Proceso de reserva de mesa

La *Ilustración 38*, hace referencia a la pantalla de un administrador de restaurante para consultar las reservas de cada día.

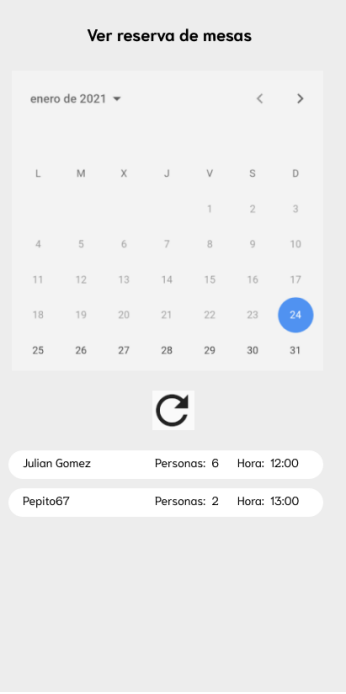

Ilustración 38. Consultar reservas administrador

Capa de persistencia (Base de datos)

En este apartado, se puede ver el modelo de datos que se utilizará para almacenar la información. De la *Ilustración 39*, podemos extraer las tablas: Usuario (*users*), con toda su información relacionada. Administrador (*owner*). Restaurante (*restaurant*). Notificaciones (*notifications*), las cuales siempre pertenecen a un único usuario y van siempre relacionadas con un único restaurante. Petición (*request*), que representaría una tabla para que un usuario pida ser administrador de un restaurante, validando ciertos datos, ya sea mediante un código al teléfono, al email o mediante un DNI. Seguidores (*followers*), que representaría que usuario sigue a otro. *Ticket* (*ticket*), para contactar con el soporte de la aplicación. ya sea por una petición o pregunta. Por último, listas de favoritos (*favoritelists*), dónde un usuario puede tener cero o varias listas de favoritos. ya sean para restaurantes o para platos.

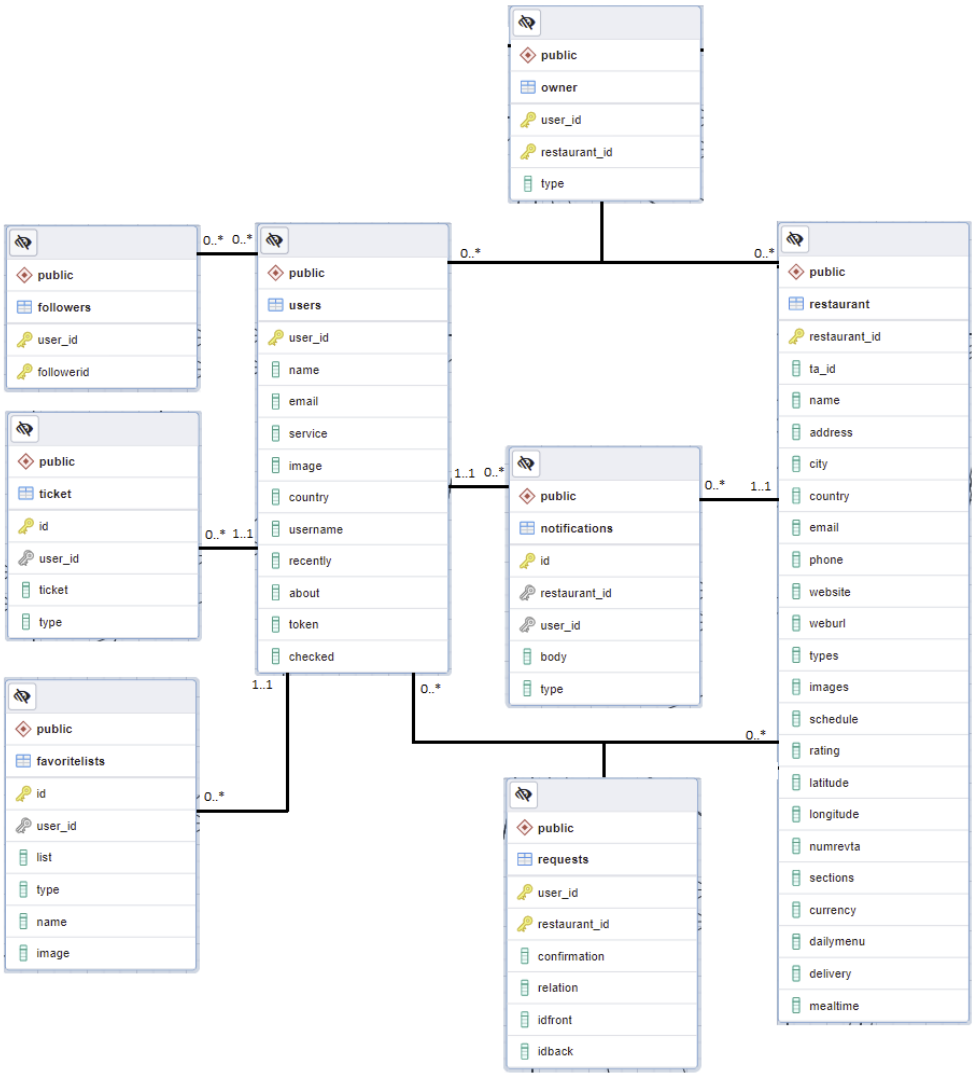

Ilustración 39. Diagrama base de datos 1

En la *Ilustración 40*, se pueden observar, además de restaurante y usuario ya mencionadas antes, las siguientes tablas y relaciones: Plato (*menuentry*), que representaría un plato asociado a un restaurante. Calificación (*rating*), que viene dada por la valoración de un plato por parte de un usuario, sólo se podrá valorar una única vez el plato, aunque se puede actualizar su puntuación o comentario. Código (*code*), que representa los códigos QR de cada mesa para los pedidos. Mesa (*tables*), dónde almacenamos la información de las mesas disponibles para el sistema de reserva. Por último, Reserva (*reservation*), para las reservas de dichas mesas.

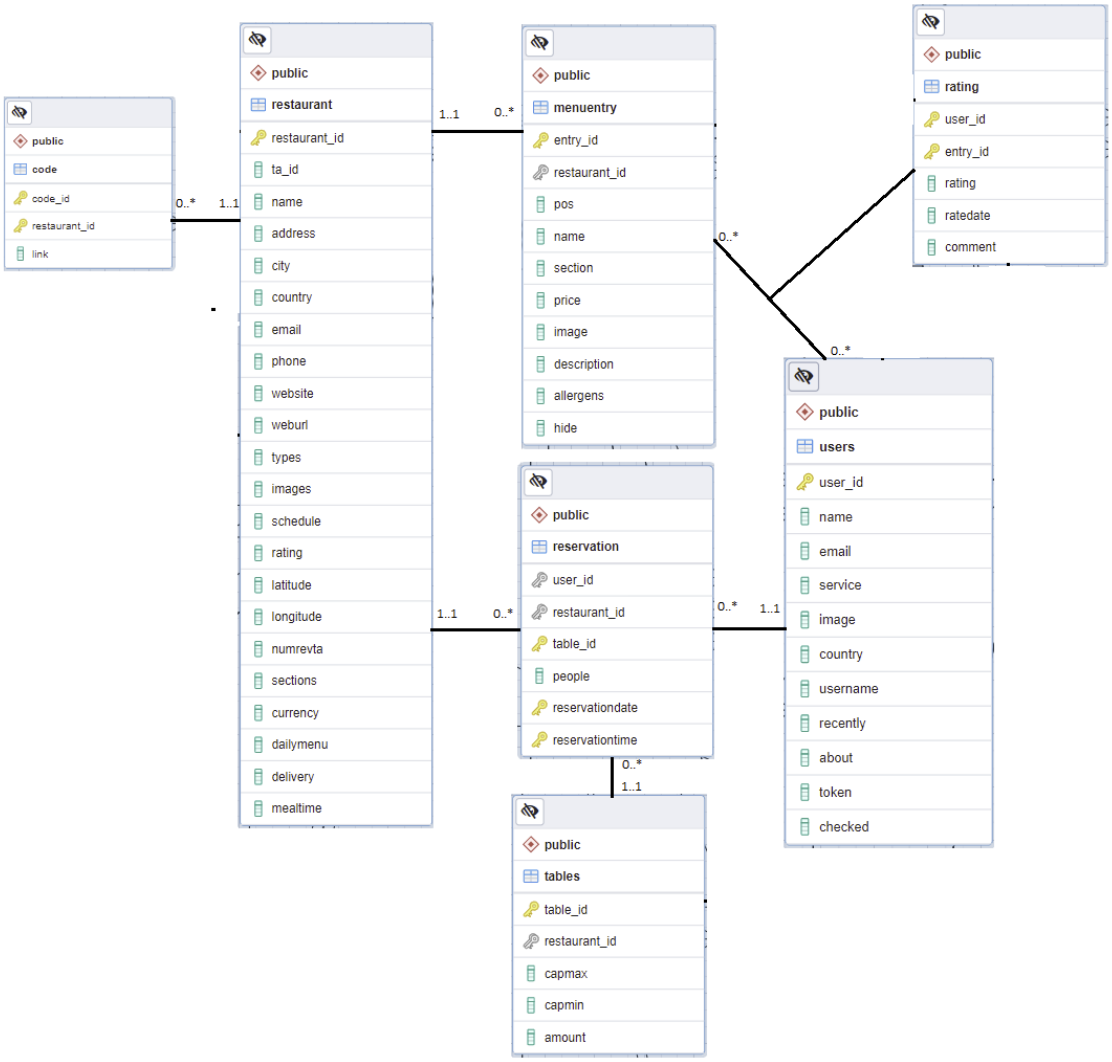

Ilustración 40. Diagrama base de datos 2

En la *Ilustración 41*, se pueden observar, además de restaurante y usuario ya mencionadas antes, las siguientes tablas y relaciones: Pago (*payment*), es una tabla dónde se almacena la información de algunos pagos, para generar una factura posteriormente. Premium (*premium*), para almacenar la información del servicio de suscripción premium del restaurante, que se realiza mediante Stripe. Patrocinado (*sponsor*), es la tabla para almacenar la información referente al sistema de patrocinados, y los clicks restantes que tiene un restaurante. Por último, Precios (*prices*), dónde se almacena la información de los precios del sistema premium y de patrocinados.

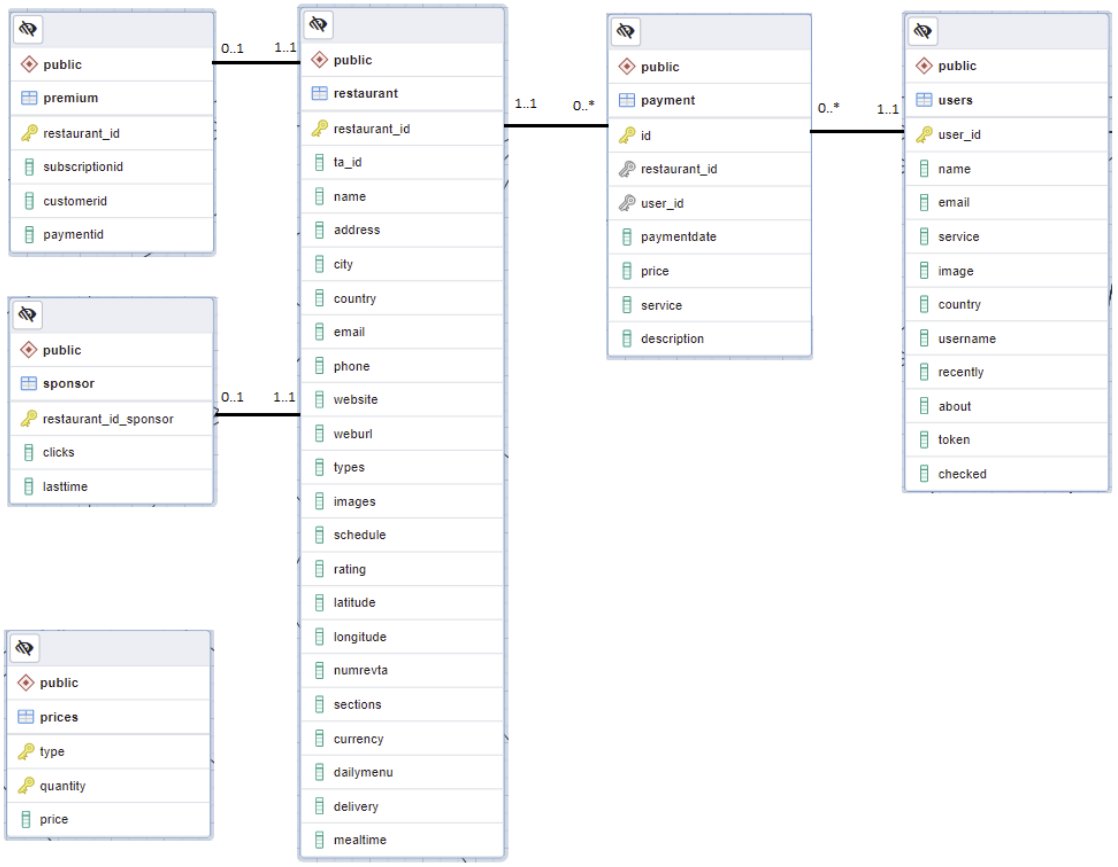

Ilustración 41. Diagrama base de datos 3

8. Implementación

Contexto tecnológico

A continuación se detallarán las distintas tecnologías, servicios, lenguajes de programación y entornos de desarrollo **requeridos** para este proyecto.

Entornos de desarrollo

Android Studio es el entorno de desarrollo o IDE (*Integrated Development Environment*) que se ha usado para el desarrollo de la aplicación en el lado del cliente con la versión 4.1.2.

Visual Studio Code es el editor utilizado para el desarrollo del lado del servidor con la versión 1.55.2.

Flutter

Flutter es un **framework** open-source, desarrollado por *Google,* basado en el lenguaje de programación *Dart,* para el desarrollo de interfaces de usuario para distintas plataformas como: *Android, iOS, Windows, Linux* y web. Es el framework utilizado para el desarrollo de la aplicación con versión 2.0.6 para el sistema operativo *Android*.

Node.js

Node.js es un framework open-source, basado en el lenguaje de programación *Javascript,* para el desarrollo del *back-end* de aplicaciones web. Es el entorno utilizado para desarrollar el **servidor** de la aplicación con versión 6.13.4.

Python

Python es un lenguaje de programación open-source, interpretado, multiparadigma y multiplataforma. Es el lenguaje utilizado, en su versión 3.8.10, en el proyecto para la generación de la base de datos mediante *scraping* (utilizando la librería *Selenium*) y la integración del servicio de OCR de *Nanonets*.

PostgreSQL

PostgreSQL o *Postgres,* es un sistema de gestión de bases de datos relacional orientado a objetos open-source. Es el sistema de gestión de bases de datos utilizado en el proyecto con versión 12.2.

Firebase

Firebase es una plataforma para el desarrollo de aplicaciones web y aplicaciones móviles, propiedad de *Google,* ubicada en la nube. Ofrece distintos servicios, de los cuáles, algunos han sido utilizados en el proyecto:

- *Firebase Auth*: Es un servicio que puede **autenticar** los usuarios utilizando únicamente código del lado del cliente. Además del clásico inicio de sesión con correo y contraseña, se han usado los proveedores de inicio de sesión *Facebook* y *Google*.
- *Firebase Storage*: Es un servicio de **almacenamiento** de archivos en la nube, que proporciona cargas y descargas seguras. Se ha utilizado en el proyecto principalmente para el almacenamiento de las imágenes.
- *Cloud Firestore*: Es un servicio de base de datos *NoSQL,* que funciona en la nube en **tiempo real**. Se organiza en forma de documentos agrupados en colecciones. Se ha utilizado en el proyecto para el sistema de pedidos en mesa.
- *Cloud Messaging*: Es una plataforma para mensajes y **notificaciones**, utilizada para recibir las notificaciones de la aplicación. [5]

Nanonets

Nanonets es un servicio web que ofrece un modelo de OCR (*Optical Character Recognition*) para distintos tipos de documentos, para **automatizar** la lectura de estos. Se ha utilizado este servicio para automáticamente **digitalizar** las cartas de los restaurantes, mediante una foto que pueda haber hecho algún usuario a una carta, con una posterior revisión humana. [6]

Stripe

Stripe es un servicio que proporciona una API (*Application Programming Interface*), que los desarrolladores pueden usar para integrar el procesamiento

de **pagos** en sus aplicaciones y webs. Ha sido utilizado en el proyecto para el sistema de suscripciones de los restaurantes al servicio premium. [7]

RevenueCat

RevenueCat proporciona un servicio para integrar el sistema de **compras** dentro de la aplicación mediante *Google Play* o *App Store,* sin necesidad de código en el lado del servidor. Ha sido utilizado, junto al sistema de compras de *Google Play,* para la compra de *packs* de visitas del sistema de patrocinados de los restaurantes. [8]

Apify

Apify es una plataforma que ofrece servicios de ***scraping*** para distintos portales. Se ha utilizado para hacer un *scraping* de *Tripadvisor,* para conseguir información sobre restaurantes y locales. [9]

Figma

Figma es una aplicación que se puede usar desde la web para el **diseño** y prototipado de interfaces de usuario. [10]

Desarrollo e implementación en Flutter

Introducción a Flutter y los Widgets

La estrategia de *Flutter*, todo es un ***widget***, sigue las bases de la programación orientada a objetos hasta la interfaz de usuario: la interfaz del programa consta de diferentes *widgets*, que pueden estar anidados entre ellos formando un **árbol**. Cada botón y texto mostrado es un *widget*. Estos cuentan con diferentes propiedades que se pueden modificar.

Pueden interactuar entre sí y reaccionar a cambios de estado externos mediante sus funciones integradas. Cuando el estado de un *widget* cambia, el *widget* reconstruye su descripción, el framework difiere de la descripción anterior, para determinar los cambios mínimos necesarios en el árbol de renderizado subyacente, para la transición de un estado al siguiente. Todos los elementos importantes de la interfaz de usuario, incluyen widgets que se corresponden con los diseños de *Android* y *iOS* o las aplicaciones web convencionales. Si se desea, estos widgets se pueden ampliar con funciones adicionales o se pueden crear *widgets* propios, que se pueden combinar fácilmente con los ya existentes [11]. Aquí tendríamos un ejemplo muy sencillo de código:

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(
    Center(
      child: Text(
        'Hello, world!',
        textDirection: TextDirection.ltr,
      ),
    ),
  );
}
```

Ilustración 42. Programa ejemplo

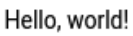

Ilustración 43. Hello world

Administración del estado: Provider y setState

Flutter es declarativo. Esto significa que *Flutter* crea la interfaz de usuario para reflejar el estado actual de la aplicación. [12]

Cuando el estado de la aplicación cambia (por ejemplo, si el usuario acciona un botón en la pantalla de configuración), se cambia el **estado** y se activa un rediseño de la interfaz de usuario. No existe una forma imperativa de cambiar la interfaz de usuario en sí (como por ejemplo widget.setText): cambia el estado y la interfaz de usuario se reconstruye desde cero. Para redibujar la interfaz de usuario cuando necesitamos hacer un cambio del estado, utilizamos la siguiente función:

```
setState(() { _myState = newValue; });
```

Puede parecer innecesario redibujar toda la interfaz para a veces solo cambiar un único valor en un texto por ejemplo, pero *Flutter* es lo suficientemente rápido para hacerlo.

El problema que se plantea ahora, es la forma de pasar la información del estado de un *widget* al resto de *widgets* que están por debajo de él en el árbol.

Si queremos pasar la información del estado de cada uno de los *widgets* de, por ejemplo, los que componen la lista de platos del menú:

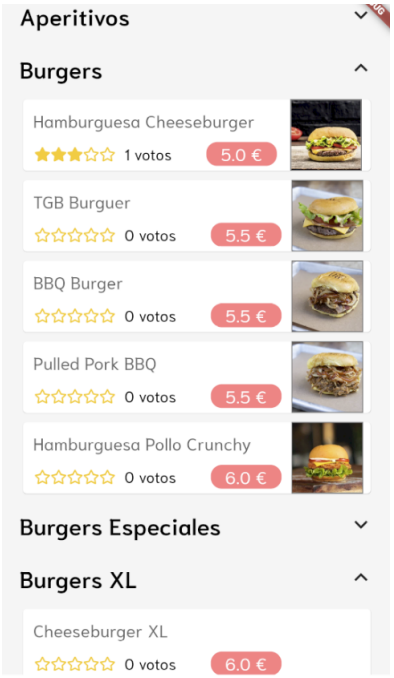

Ilustración 44. Lista widgets

Tendremos una lista compuesta de *MenuTile()*, las cuáles están envueltas por 3 *Provider*:

```
Provider.value(
  value: order,
  key: UniqueKey(),
  child: Provider<Restaurant>.value(
    key: UniqueKey(),
    value: restaurant,
    child: Provider<MenuEntry>.value(
        key: UniqueKey(),
        value: entry, child: MenuTile(daily: false,)), // Provider.value
  ), // Provider.value
); // Provider.value
```

Ilustración 45. Crear Provider

Provider es una clase para proveer de esta información a los *widgets* que estén por debajo en el árbol. Luego dentro de la función *build()* de *MenuTile* podemos recuperar esta información de forma sencilla:

```
@override
Widget build(BuildContext context) {
  entry = Provider.of<MenuEntry>(context);
  restaurant = Provider.of<Restaurant>(context);
  order = Provider.of<bool>(context);
```

Ilustración 46. Recuperar información del Provider

Cada *widget* tiene una función *build()* mediante la cual se construye el *widget*. *BuildContext* no es más que una referencia a la ubicación de un *widget,* dentro de la estructura de árbol de todos los *widgets* que se construyen.

Programación asíncrona: Futures, async y await

Las palabras clave *async* y *await,* proporcionan una forma declarativa de definir funciones **asincrónicas** y utilizar sus resultados. [13]

Siempre que se quiera definir una función asíncrona, se ha de añadir *async* antes del cuerpo de la función y esta siempre ha de devolver un objeto de tipo *Future*.

```
Future _loadNotifications() async{
  DBServiceUser.userF.notifications = await DBServiceUser.dbServiceUser.
  getNotifications(DBServiceUser.userF.uid);
  setState(() {
  });
}
```

Ilustración 47. Función asíncrona

```
Future<List<PersonalNotification>> getNotifications(String id) async
```

Ilustración 48. Función asíncrona 2

En este ejemplo, se utiliza la función *_loadNotificacions()* para recuperar las notificaciones del usuario cuando se abre la pantalla de notificaciones. La función *getNotifications(),* es una función asíncrona ya que está consultando la base de datos, por lo tanto, para poder utilizar su resultado se necesita añadir *await* antes de la llamada, esto convertirá el tipo *Future* al tipo que retorne la función, como podemos ver en la cabecera de la función. Como se puede observar, también se utiliza la función *setState()* para redibujar la interfaz una vez se ha recuperado la información que se deseaba.

Streams, Firestore y el sistema de pedidos conjunto en tiempo real

Un *Stream* es como una **tubería**, pones un valor en un extremo y si hay un oyente en el otro extremo, ese oyente recibirá ese valor. Un *Stream* puede tener varios oyentes, y todos esos oyentes, recibirán el mismo valor cuando se coloque en la tubería.

En la siguiente imagen, se observa el *Stream* que se comunica con *Firestore* y que es llamado desde la interfaz de usuario en la pantalla de pedidos. Actualiza la lista del pedido dados un ID de restaurante y un ID de mesa y devuelve una lista de elementos *Order,* que es la clase con la que se modela cada elemento del pedido.

```
Stream<List<Order>> getOrder(String restaurant_id, String table_id) {
  return orders.doc(restaurant_id).collection(table_id).snapshots().map(
      (event) =>
          event.docs.where((element) => element.id != "closed").map((s) {
            return Order(
                order_id: s.id,
                amount: s.data()['amount'],
                entry_id: s.data()['entry_id'],
                send: s.data()['send'],
                note: s.data()['note'],
                check: s.data()['check']);
          }).toList());
}
```

Ilustración 49. Función Stream

En la siguiente imagen, se puede observar la estructura de la base de datos para los pedidos desde *Firestore*.

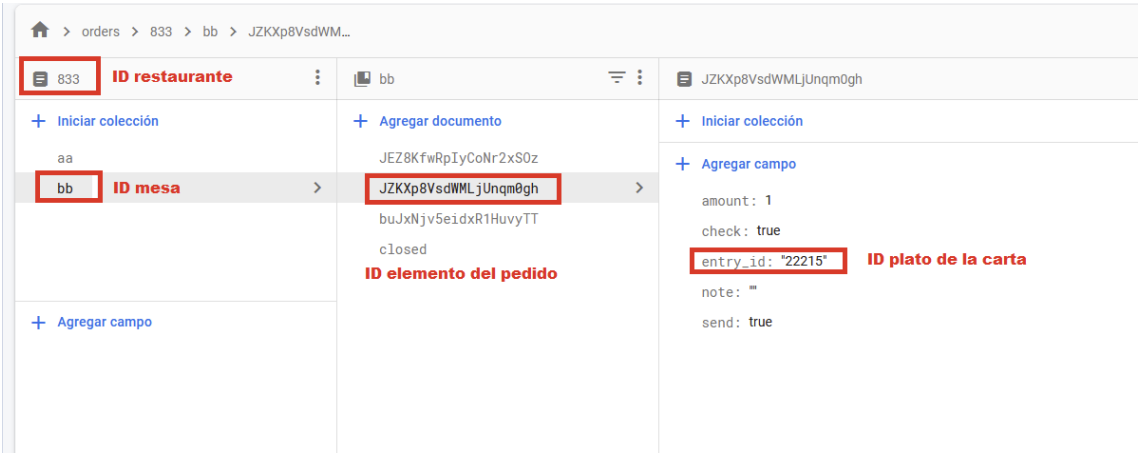

Ilustración 50. Interfaz de Firestore

Una vez un usuario añada un elemento, el cual está formado por el plato, la cantidad que se quiere pedir y una nota opcional, los cambios se reflejarán en el *Stream* de todos los usuarios que hayan entrado al pedido. Para entrar a este **pedido**, tendrán que escanear un código QR como el de la imagen, ya sea desde una opción disponible dentro de la app, o desde fuera de esta, ya que se podrá acceder gracias a los *Dynamic Links*. [14]

Ilustración 51. Imagen con código QR para la mesa

La forma del pedido consiste en dos listas: elementos enviados, los cuáles ya les aparecerán a los administradores como se muestra en la imagen , dónde pueden marcar cada elemento de forma que ya se esté preparando. Y los elementos pendientes, los cuales los administradores no ven hasta que se hayan enviado por algún usuario.

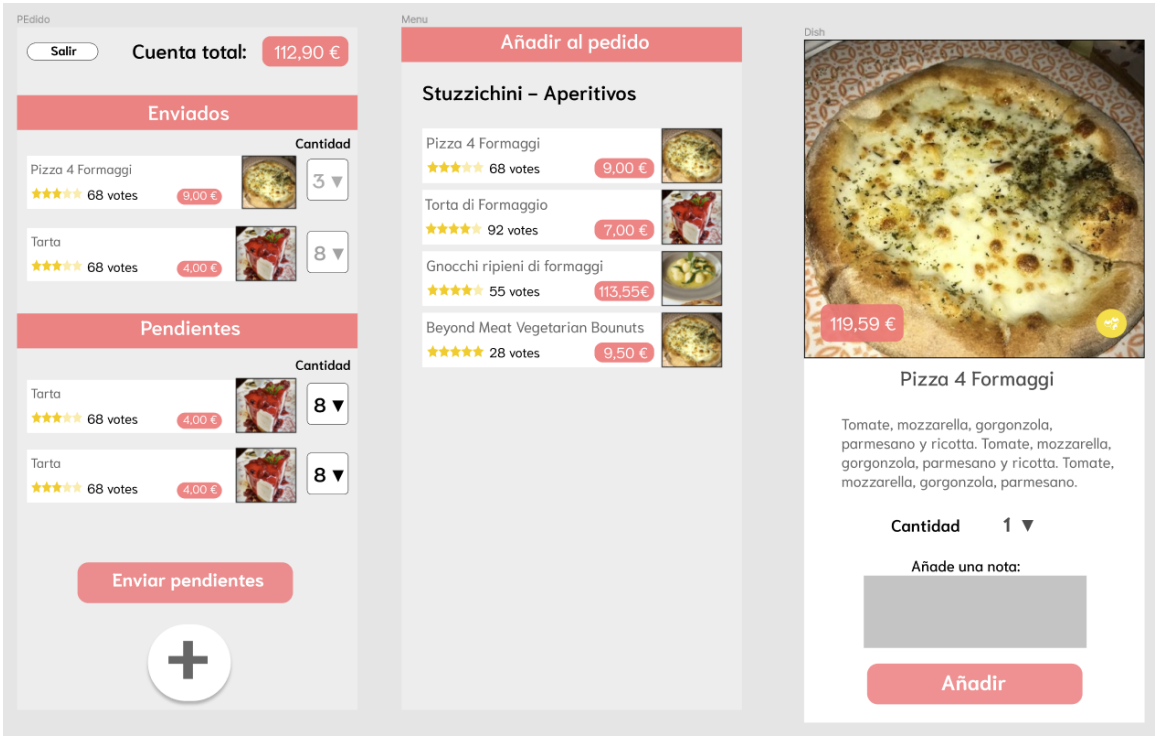

Ilustración 52. Interfaz del sistema de pedidos usuario

Ilustración 53. Interfaz del sistema de pedidos administrador

Compras dentro de la app mediante Google Play y RevenueCat

Las compras que se realizarán dentro de la app, serán únicamente los *packs* de visitas como patrocinado, un *pack* de 100, de 1000 y de 10000 visitas.

Para poder gestionar las compras de *packs* de visitas de *sponsor* dentro de la app mediante *Google Play*, hay que crear y pagar una cuenta de desarrollador que actualmente tiene un precio de 30€. Desde *Google Play Console* [15], se crea un proyecto nuevo y se configura la app mediante una serie de pasos, para poder generar un archivo APK para subir a la plataforma, y así tener una primera versión.

Una vez hecho esto, desde *Google Play Console* podemos crear productos que luego podrán ser comprados desde la app:

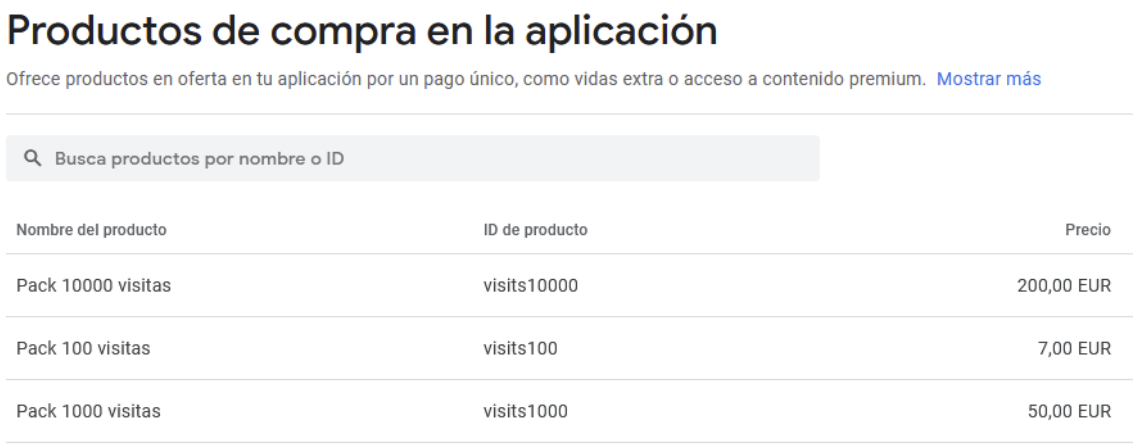

Ilustración 54. Productos en Google Play

Para simplificar la programación a la hora de gestionar las compras, tanto en el cliente como en el servidor, se puede utilizar *RevenueCat,* que es un servicio dedicado a mantener y generar toda la **infraestructura** de los pagos para el desarrollador, y que es gratuito mientras el beneficio de la app no supere los 10.000€ mensuales.

Una vez creada una cuenta y un proyecto, se pueden crear productos para comprar de forma similar a como se hacía en *Google Play Console*, siendo necesario respetar el identificador dado a los productos en ambos lugares.

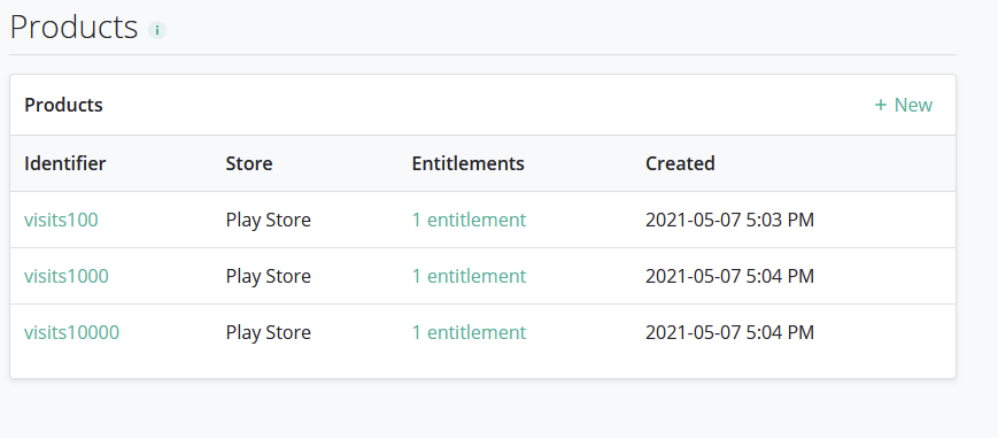

Ilustración 55. Productos en RevenueCat

La programación solo es necesaria en el lado del cliente y no en el servidor, pudiendo hacerlo todo con unas pocas líneas de código.

```
static Future initPlatformState() async {
  await Purchases.setDebugLogsEnabled(true);
  await Purchases.setup("API KEY");
}

Future<bool> buyProduct(String id) async{
  try {
    Offerings offerings = await Purchases.getOfferings();
    Offering offering = offerings.getOffering(id);
    if (offering != null && offering.lifetime != null) {
      Product product = offering.lifetime.product;
      PurchaserInfo info = await Purchases.purchaseProduct(product.identifier, type: PurchaseType.inapp);
      print(product.title);
      print(info.entitlements.all[id].identifier);
      return true;
    }
  } on PlatformException catch (e) {
    print(e);
    return false;
  }
}
```

Ilustración 56. Código para comprar Productos

Así quedaría la pantalla de compras de *packs* de visitas dentro de la app, dónde una vez se elija una opción, se abrirá el menú de *Google Play* dónde se gestionará la compra.

Ilustración 57. Pantalla de compra de visitas patrocinadas

Suscripciones premium para restaurantes mediante Stripe

El sistema de suscripciones se gestionará a través de *Stripe*. Es necesario crear una cuenta y dar de alta una suscripción, en este caso costará 15 euros y se renovará de forma automática cada mes si no es cancelada.

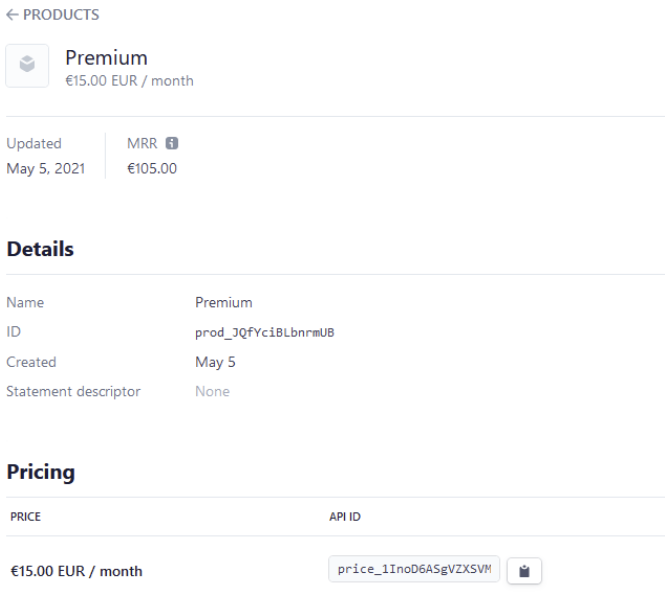

Ilustración 58. Interfaz Stripe

Para la implementación del sistema de suscripciones, es necesario interactuar con el servidor *Node.js* de la app, además de con el propio servicio de *Stripe*, por seguridad, y para para poder almacenar en la base de datos *PostgreSQL* los **identificadores** de cliente y suscripción que genera *Stripe,* para poder cancelar o renovar una suscripción en un futuro.

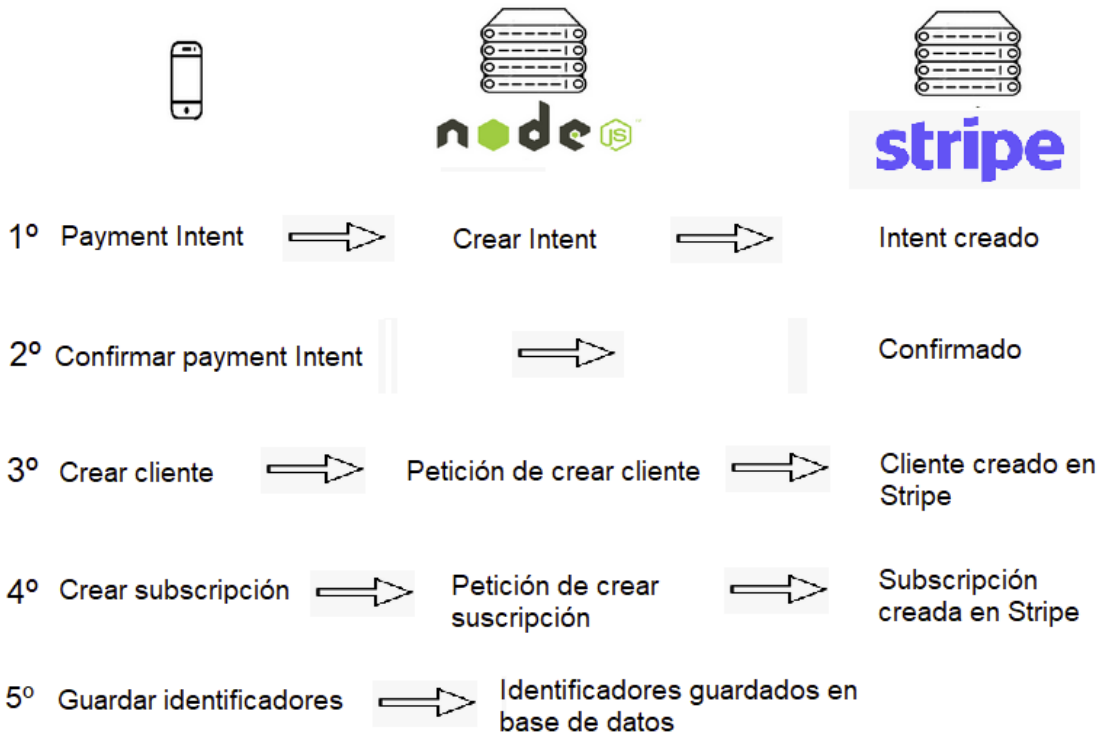

Ilustración 59. Esquema de pago de suscripciones

Siguiendo el esquema del proceso de pago y creación de la suscripción, una vez se ha recogido la información para el pago, que en este caso serán los datos de una tarjeta de crédito o de débito, la app se comunicará con el servidor para crear un *payment intent* (intento de pago) y este, posteriormente, se comunicará con el servicio de *Stripe* para crearlo si los datos de pago son correctos. Una vez hecho esto, la app se comunicará de forma directa con *Stripe* para confirmar que el pago ha sido correcto, en caso contrario, el proceso no continuará y aparecerá un aviso de error en la app. Si el proceso continúa, la app se comunicará con el servidor de nuevo para crear un elemento cliente y suscripción cuando este se comunique con el servicio de *Stripe*, y por último, guardará en base de datos los identificadores generados por *Stripe*, que será uno para el pago, uno para el cliente y otro para la suscripción.

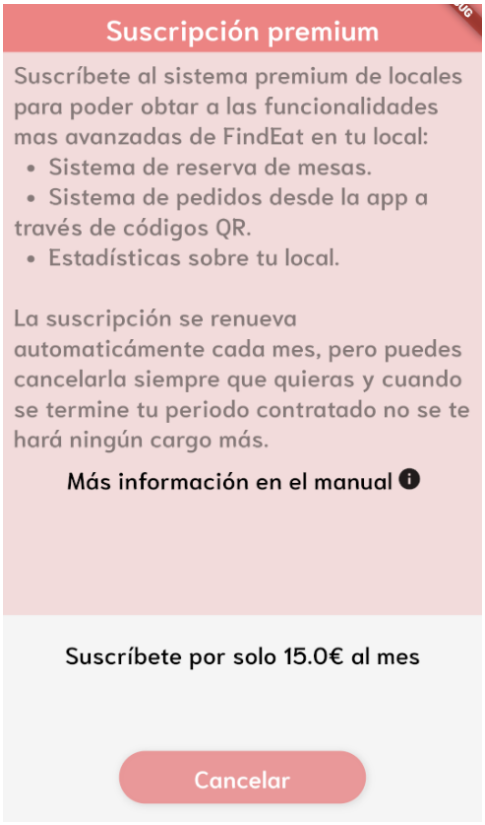

Ilustración 60. Pantalla de suscripción

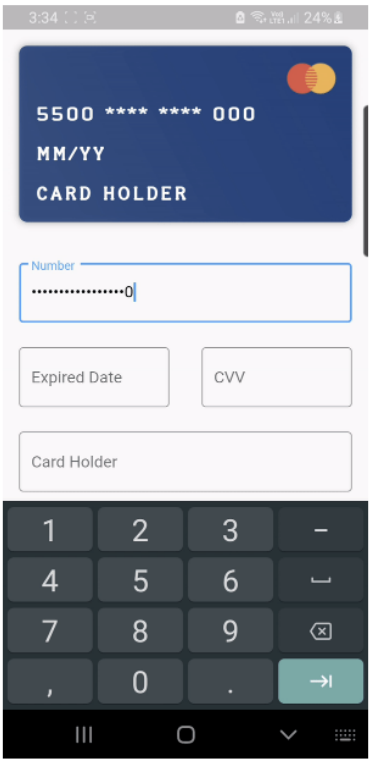

Ilustración 61. Datos de tarjeta

Por último, se puede observar la pantalla dónde se podrá activar la suscripción o cancelarla mediante un único botón, en el primer caso, aparecerá un formulario donde rellenar la información de la tarjeta de crédito o débito.

Desarrollo e implementación del servidor Node.js

Servidor

El servidor, funciona principalmente para interaccionar con la base de datos mediante peticiones del cliente. Está implementado en *Node.js* mediante la librería *Express*.

Cuando se inicializa el servidor, escuchamos en un puerto a elección.

```
const app = express()
app.listen(port, () => {
  console.log(`App running on port ${port}.`)
})
```

Ilustración 62. Inicialización servidor

Inicializamos las funciones GET, POST, PUT y DELETE, como por ejemplo las siguientes:

```
app.get('/codes', dbc.getCodes)
app.post('/codes', dbc.createCode)
app.delete('/codes', dbc.deleteCode)

app.get('/ownerres', dbu.getRestaurantOwners)
app.post('/owner', dbu.createOwner)
app.delete('/owner', dbu.deleteOwner)
app.put('/owner', dbu.updateOwner)
```

Ilustración 63. Inicialización funciones HTTP

Las cuáles, tienen una forma siempre similar a la siguiente, dónde tenemos escrito el código SQL necesario para interaccionar con la base de datos:

```
const getDailyMenu = (request, response) => {
  const {restaurant_id} = request.headers;
  pool.query(
    `SELECT dailymenu from restaurant where restaurant_id = $1`,[restaurant_id],
    (error, results) => {
      if (error) {
        throw error
      }
      response.status(200).json(results.rows)
    }
  )
}
```

Ilustración 64. Ejemplo de función servidor

Dónde *pool,* es una librería que utilizamos para conectar con la base de datos PostgreSQL.

```
require('dotenv').config()
const Pool = require('pg').Pool
const pool = new Pool({
  user: process.env.DB_USER,
  host: process.env.DB_HOST,
  database: process.env.DB_DATABASE,
  password: process.env.DB_PASSWORD,
  port: process.env.DB_PORT,

})
module.exports ={pool}
```

Ilustración 65. Conexión con base de datos

Sistema de búsqueda de restaurantes

Para la búsqueda de restaurantes, se podrá seleccionar un filtro de distancia máxima y de tipo de cocina, además de un criterio de ordenación por cercanía o relevancia.

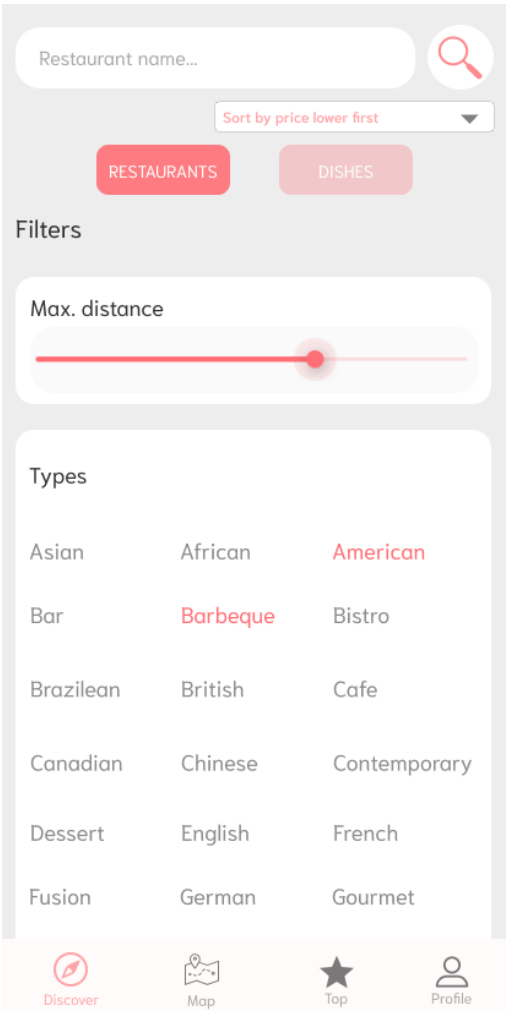

Ilustración 66. Filtros restaurantes

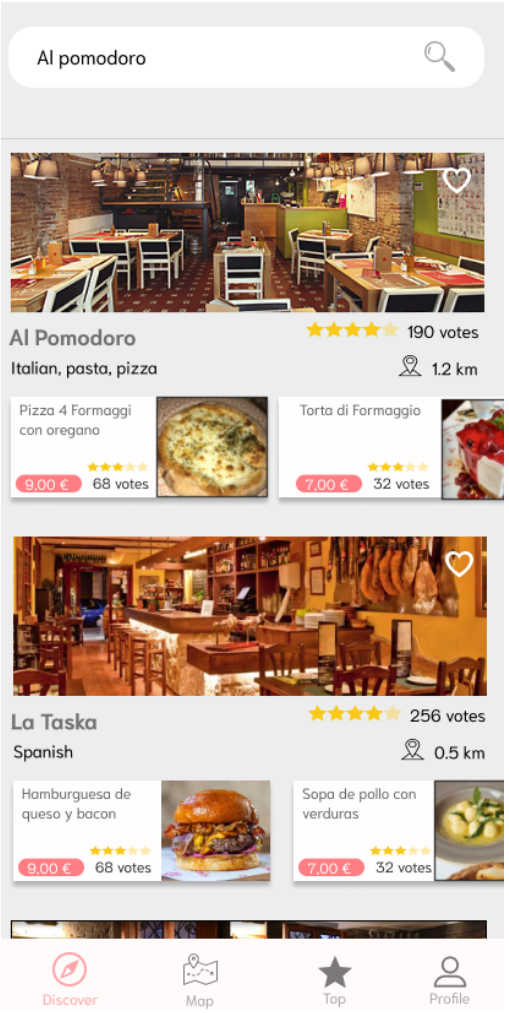

Ilustración 67. Búsqueda restaurantes

A la hora de buscar por el nombre introducido en la barra de búsqueda, se sigue el siguiente criterio de búsqueda, tomando de ejemplo la búsqueda *Al pomodoro,* que es el nombre de un restaurante:

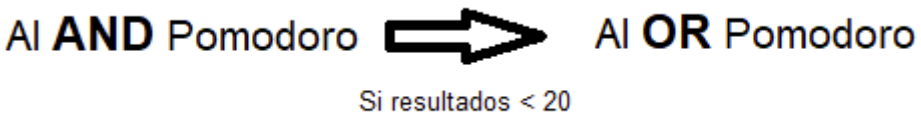

Ilustración 68. Criterio de búsqueda

Primero, se hace una búsqueda donde todas las palabras introducidas sean **prefijo**, de al menos, una de las del nombre del restaurante. Si el resultado de esta búsqueda es menor a un número de restaurantes deseado, como por ejemplo 20, se realizará una nueva búsqueda, en este caso, donde al menos una de las palabras de la búsqueda sea prefijo de alguna del nombre del restaurante.

Destacar que a la hora de comparar palabras se busca por prefijo, no la palabra completa, si por ejemplo se busca *Task,* aparecerá probablemente *La Taska* como primer restaurante.

El criterio de ordenación por relevancia viene dado por un cálculo de la siguiente forma:

```
sum(rating.rating)*X/count(rating) + count(rating)*Y
```

Dónde se tiene en cuenta la media de las valoraciones dadas a todos los platos, multiplicado por un factor de ajuste X, más el número de valoraciones totales de todos los platos multiplicado por un factor de ajuste Y. A la hora de implementarlo, los valores dados han sido X=0,5 e Y=0,0025.

Sistema de búsqueda de platos

Para la búsqueda de platos, se podrá seleccionar un filtro de precio máximo, nota mínima y exclusión de alérgenos (es decir, los platos que contengan ese alérgeno marcado no saldrán en la búsqueda), además, de un criterio de ordenación de: precio más bajo, cercanía o relevancia. También se podrá hacer una búsqueda avanzada de hasta 3 platos distintos, sacando como resultado los restaurantes que contengan al menos un plato que coincida con cada una de las búsquedas.

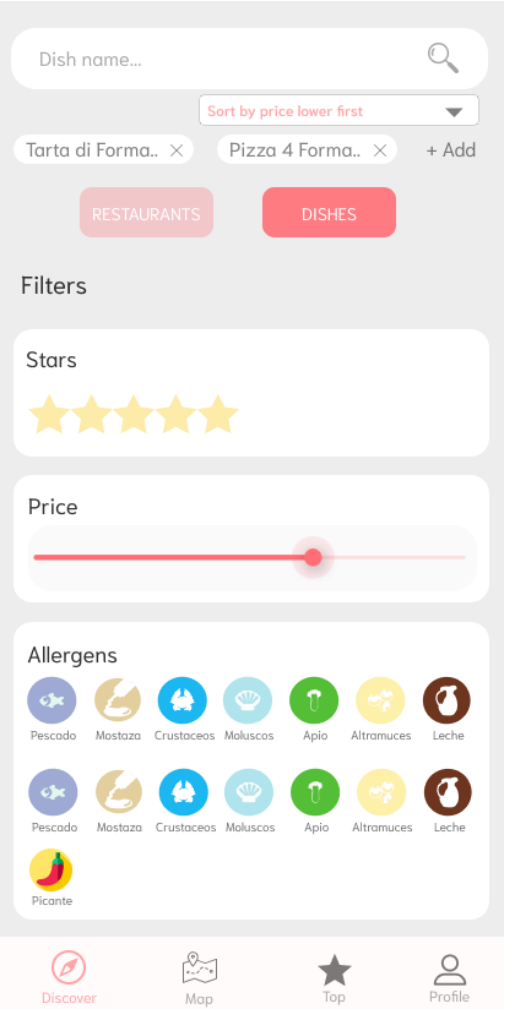

Ilustración 69. Filtros platos

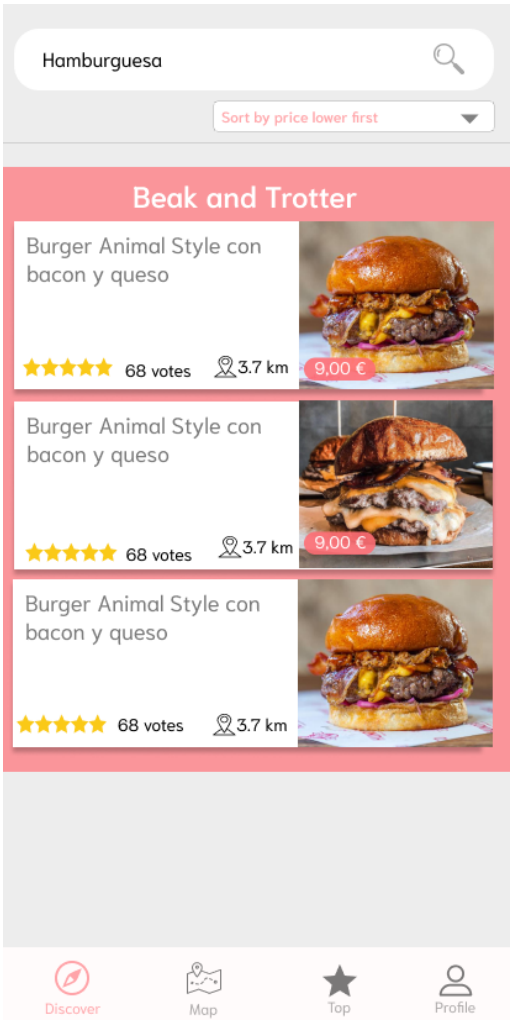

Ilustración 70. Búsqueda platos

Al hacer una búsqueda conjunta, se separa cada una de ellas en una búsqueda independiente dónde se le pone los filtros deseados de precio, alérgenos y valoración, además del nombre del plato.

Si en el restaurante existe al menos un plato satisfaga cada una de las búsquedas, se obtendrá como resultado.

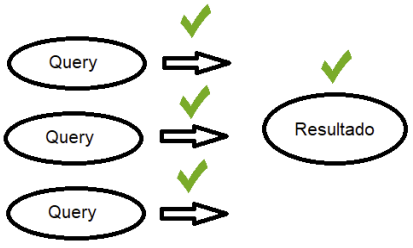

Ilustración 71. Resultado positivo

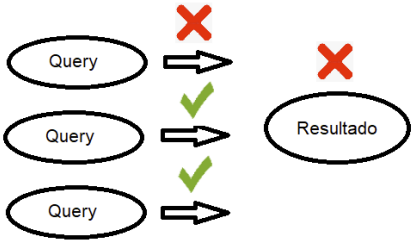

Ilustración 72. Resultado negativo

Si alguna de ellas no coincide, no se dará el restaurante como resultado.

Los criterios de búsqueda de los nombres y de ordenación por valoración, siguen la misma estructura que en la búsqueda de restaurantes:

```
sum(rating1.rating)*X/count(rating1)      +      count(rating1)*Y      +
sum(rating2.rating)*X/count(rating2)      +      count(rating2)*Y      +
sum(rating3.rating)*X/count(rating3) + count(rating3)*Y
```

En el caso del criterio por relevancia, se sumaría el de todos los platos encontrados. Si la ordenación es por precio, se sumará el precio de los platos encontrados.

Scraping para generar una base de datos

Scraping de Tripadvisor mediante Apify

Apify es una plataforma que ofrece servicios gratuitos de *scraping* para distintos portales, existe uno para *Tripadvisor*, el cual se configura eligiendo restaurantes para una ciudad en concreto:

Ilustración 73. Opciones de Apify

La salida del servicio una vez se ha ejecutado, es un archivo JSON que contiene una lista de restaurantes con la siguiente estructura:

```
"id": "4464249",
"type": "RESTAURANT",
"name": "Minamo",
"rankingPosition": "159",
"priceLevel": "$$$$",
"category": "restaurant",
"rating": "4.5",
"isClosed": false,
"isLongClosed": false,
"phone": "+34 934 67 38 00",
"address": "Bruc 65 Next to Consell de Cent, 08009 Barcelona Spain",
"email": "mail@minamobarcelona.com",
"cuisine": [
  "Japanese",
  "Sushi",
  "Asian",
  "Gluten Free Options"
],
"hours": [],
"latitude": "41.39391",
"longitude": "2.170124",
"webUrl": "https://www.tripadvisor.com/Restaurant_Review-g187497-d446424
"website": "http://www.minamo.es",
"numberOfReviews": "386",
"rankingDenominator": "9494",
"rankingString": "#142 of 9,547 Restaurants in Barcelona",
"reviews": []
```


Ilustración 74. JSON salida de Apify

Este archivo JSON, será tratado luego junto a otros que se explicarán a continuación para generar la base de datos.

Scraping de UberEats y JustEat

Una de las formas para generar una base de datos de platos para los restaurantes, es hacer un *scraping* de algún servicio que ya las posea, en este caso, de los portales *UberEats* y *JustEat*. Para conseguir esto, se ha realizado un *script* en *Python* utilizando *Selenium,* que es una herramienta para controlar navegadores web y realizar tareas de **automatización**. Como navegador se ha utilizado *Mozilla Firefox*.

Ilustración 75. Búsqueda del input UberEats

La forma de trabajar, consiste en **inspeccionar** el código HTML de la página web, e ir buscando los elementos que necesitamos para interactuar con ellos. Por ejemplo, para comenzar con *UberEats*, una vez hemos cargado la página principal, hemos de buscar el elemento para introducir una dirección dónde buscar restaurantes. El elemento es un *input,* con id="location-typeahead-home-input". A la hora de plasmar esto en el código, tenemos distintas funciones que nos proporciona *Selenium* para buscar dentro del código HTML de la web.

```
driver = webdriver.Firefox(executable_path=r'C:\D\lookin_mealDB\geckodriver.exe')
driver.get("https://www.ubereats.com/")
input_name = driver.find_element_by_css_selector("input[id='location-typeahead-home-input']")
input_name.send_keys(address)
firstLoc = driver.find_element_by_css_selector("li[id='location-typeahead-home-item-0']")
firstLoc.click()
```

Ilustración 76. Código inicio del scraping

La función *find_element_by_css_selector,* recibe como parámetro una cadena de texto, en la cuál hay que especificar la etiqueta, en este caso *input*, y el atributo en cuestión mediante el cuál queremos hacer la búsqueda, en este caso

el id="location-typeahead-home-input". Por último, se introduce la dirección deseada y se localiza el primer resultado de la búsqueda para hacer click en él.

Ilustración 77. Selección de la dirección

El siguiente paso, consiste en localizar el botón que se sitúa al final de la siguiente página, que sirve para cargar más resultados de la búsqueda y mediante un bucle, ir haciendo click en él para cargar resultados hasta que el botón desaparezca de la página por completo.

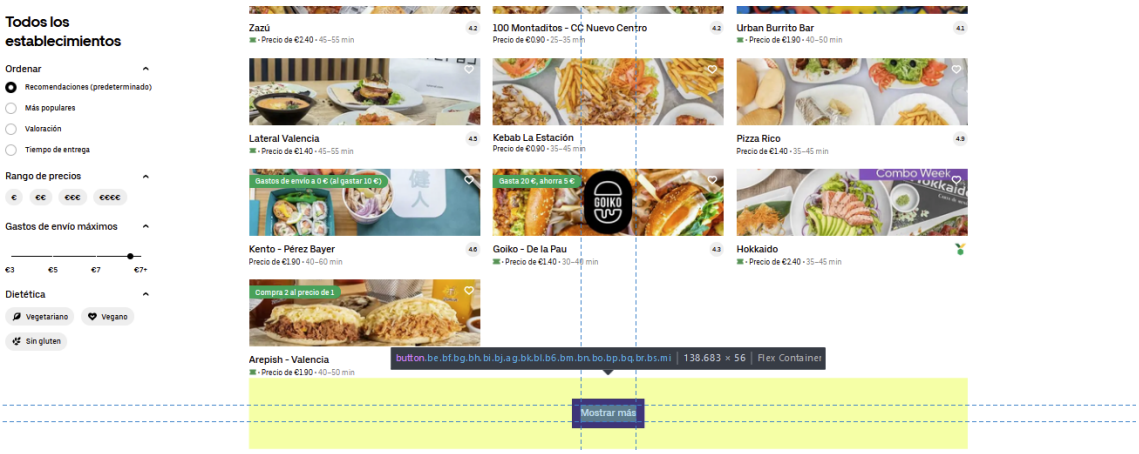

Ilustración 78. Localización del botón de Mostrar más

Una vez hecho esto, mediante otro bucle vamos consultando cada uno de los restaurantes que se han cargado en la página.

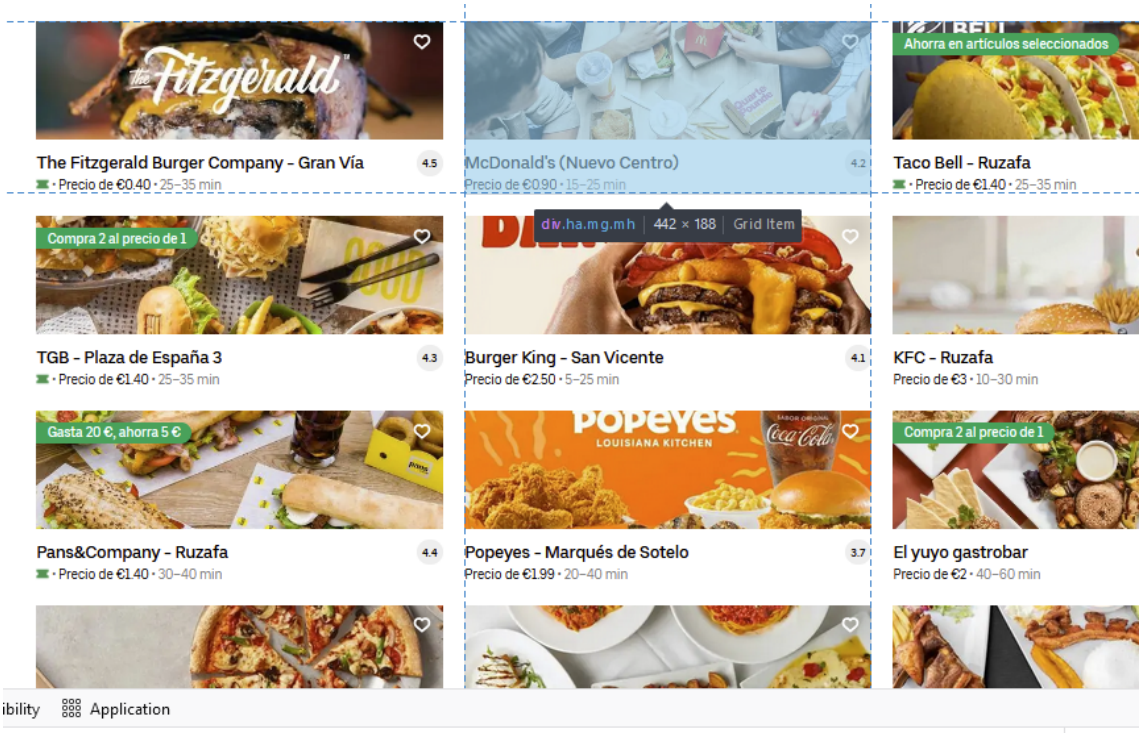

Ilustración 79. Selección de un restaurante de la lista

Otras funciones que se utilizan para buscar los elementos son *find_element_by_xpath* y *find_elements_by_xpath*, en estos casos, las funciones reciben como parámetro de entrada una cadena de texto dónde se especifica una ruta dentro del código HTML, para buscar uno o varios elementos. En la siguiente figura, se observa el código de todo el proceso de cargar los restaurantes, dónde por último, para obtener una lista de ellos se utilizan las dos funciones mencionadas anteriormente.

```
try:
  nextButton = driver.find_element_by_css_selector("button[class*='ce bh ca c3 cf er aq']")
except:
  nextButton = None
if nextButton != None:

  while nextButton != None :
    nextButton.click()
    sleep(9)
    try:
      nextButton = driver.find_element_by_css_selector("button[class*='ce bh ca c3 cf er aq']")
    except:
      nextButton = None

  results = driver.find_element_by_xpath("/html/body/div/div/main/div/div[3]/div[2]/div/div[2]").find_elements_by_xpath("./div")
```

Ilustración 80. Código para cargar los restaurantes

Es necesario mencionar que para que el programa funcione, la página web tiene que mantener la misma estructura, si en algún momento cambiase algunos de los elementos con los que se interactúa, el programa ya no funcionaria correctamente.

Por último, los elementos guardados de cada restaurante son el nombre de este, sus tipos de cocina, su dirección, su imagen principal y el enlace para acceder al restaurante. Posteriormente, se almacena cada una de las secciones de la carta con sus platos. De cada plato, se coge toda la información disponible, su nombre, su precio, su descripción y su imagen.

Las imágenes se almacenan como una cadena de texto con la URL necesaria para acceder directamente a ellas.

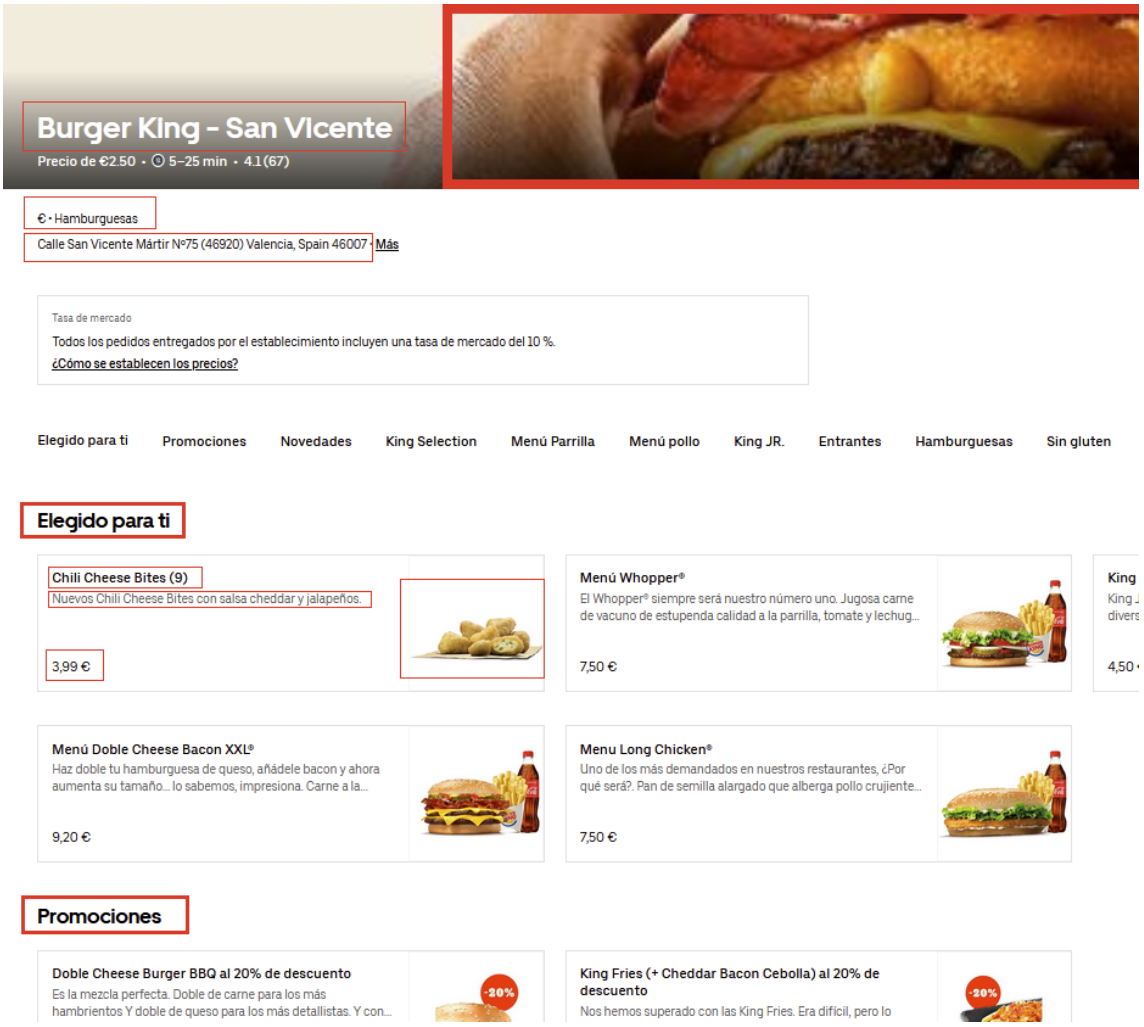

Ilustración 81. Selección de datos del restaurante

Una vez terminado el scraping, toda la información se vuelca en un archivo JSON con la siguiente estructura.

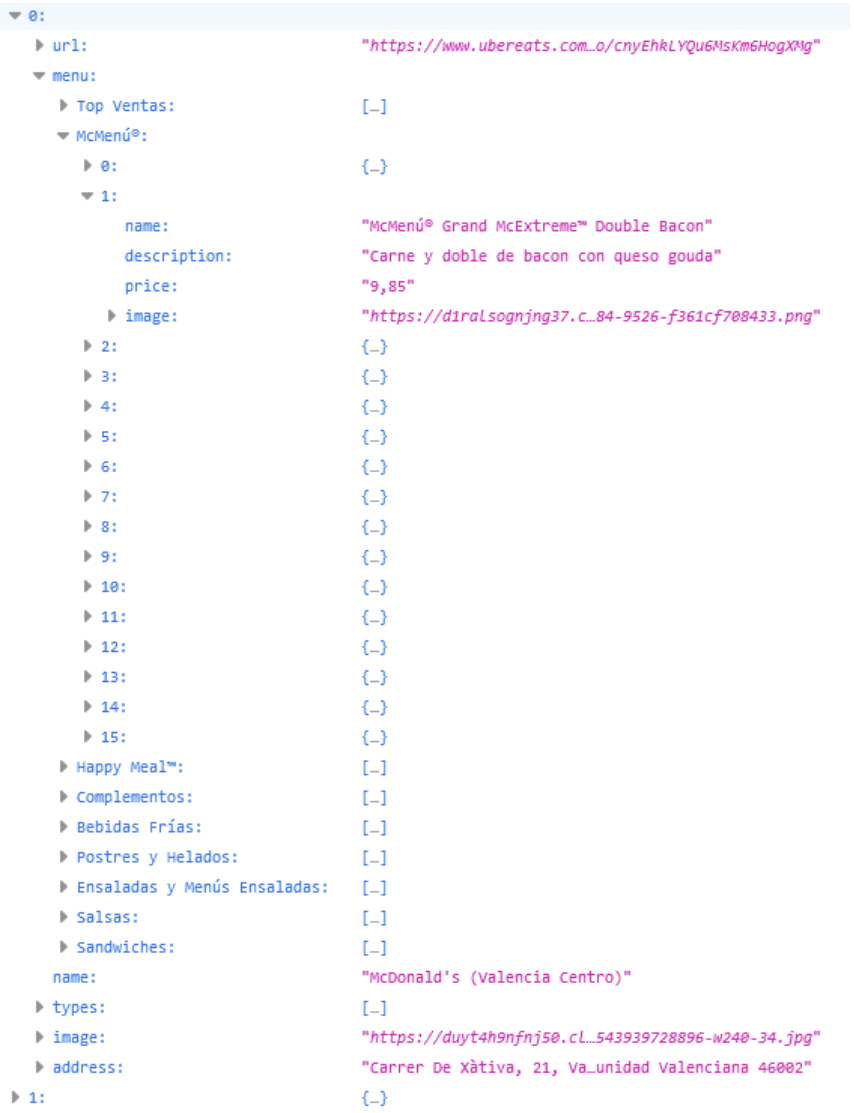

Ilustración 82. JSON de salida del scraping

Para el *scraping* de *JustEat,* se sigue una forma de trabajo idéntica al *scraping* de *UberEats*. Primero, se busca una dirección desde su página principal.

Ilustración 83. Búsqueda del input JustEat

Luego, se carga toda la lista de restaurantes posibles para esa dirección y se va consultando uno a uno cada uno de ellos.

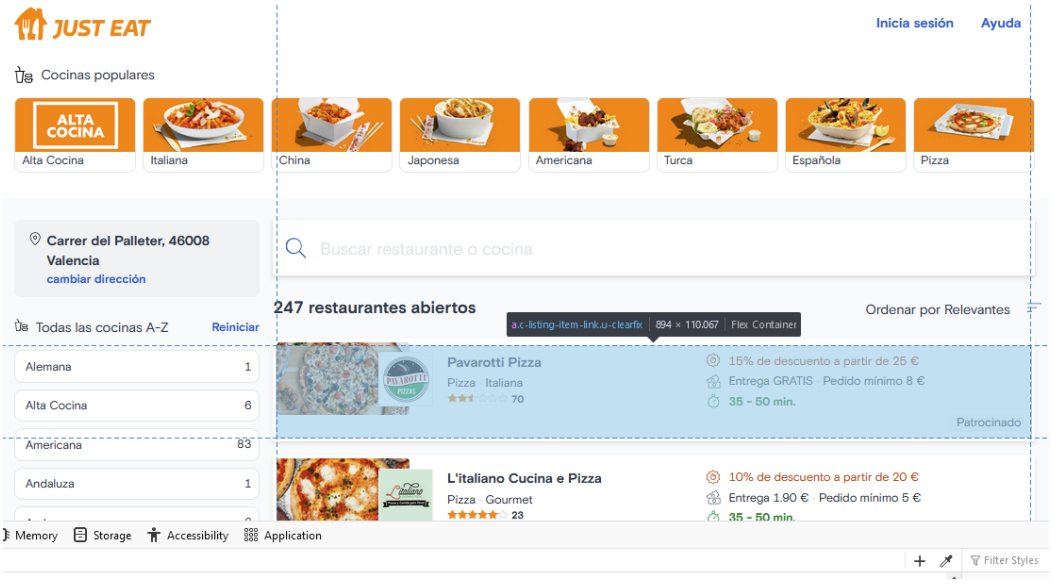

Ilustración 84. Selección de un restaurante de la lista

Para cada restaurante se almacena la misma información que en el caso de *UberEats,* ya que así generamos luego un archivo JSON con una estructura idéntica. Los elementos guardados son el nombre, sus tipos de cocina, su dirección, su imagen principal y el enlace para acceder al restaurante. Luego se almacena cada una de las secciones de la carta con sus platos. De cada plato, se coge toda la información disponible, su nombre, su precio, su descripción y su imagen.

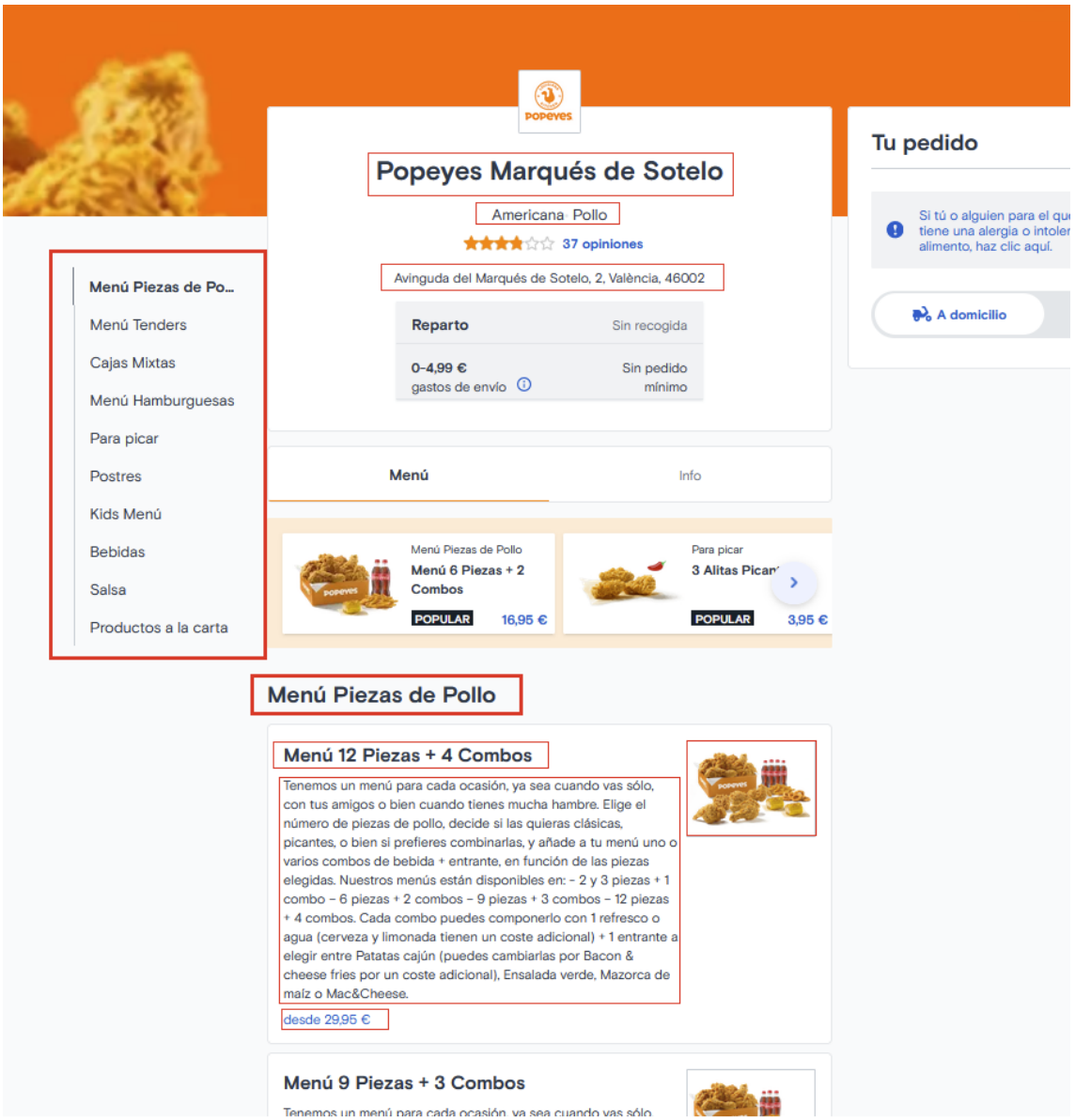

Ilustración 85. Selección de datos del restaurante

Fusionar datos de Tripadvisor con los menús de UberEats y JustEat y generar la base de datos

El primer paso es comparar los archivos JSON obtenidos de *UberEats* y *JustEat,* para saber si existe algún restaurante en ambos archivos y no salga repetido cuando se concatena uno con otro. Para ello, mediante la dirección obtenida en el scraping obtenemos unas **coordenadas**, latitud y longitud, utilizando la API de *Google Maps*. Estas coordenadas, se dan en un número con siete decimales.

Se compara uno a uno cada restaurante del archivo JSON de *UberEats* con el archivo JSON de *JustEat*. La forma de decidir si un restaurante es el mismo que otro viene dada de la siguiente manera:

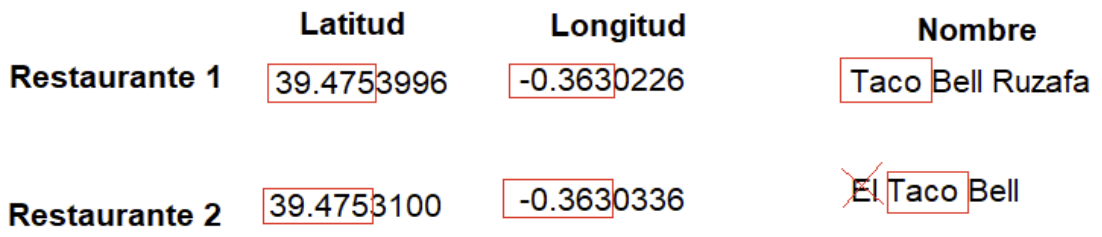

Ilustración 86. Criterio de emparejamiento de dos restaurantes

Se comparan las coordenadas, de forma que, cogiendo los tres primeros decimales de cada número, coincidan con los del otro restaurante. Después se comparan los nombres de los restaurantes, de forma que primero se eliminan de estos una lista de stop words.

```
stop_words_es = ['a', 'al', 'con','de', 'del', 'e', 'el', 'en', 'la', 'las', 'lo', 'los', 'y']
stop_words_en = ['a', 'an', 'and', 'the', 'of']
```

Ilustración 87. Stop words

Y por último se compara que, al menos, exista en común una de las palabras en cada nombre. Si todas estas condiciones se dan, se elegirá únicamente uno de los restaurantes almacenados en alguno de los archivos JSON, y se eliminará el otro. También se almacenan las coordenadas obtenidas como nuevo elemento.

El resultado de este proceso, se almacena en un nuevo archivo JSON, el cuál siguiendo el mismo proceso otra vez, se compara con el archivo JSON obtenido anteriormente gracias a *Apify,* con la información más detallada de los restaurantes de *Tripadvisor*. Se utiliza el mismo criterio de comparación y si dos restaurantes coinciden, se fusiona la información de ambos.

Se darán bastantes casos donde no existirá una coincidencia, por lo que en la base de datos habrá restaurantes sin carta, o restaurantes con carta pero con una información del propio restaurante más limitada.

Una vez terminada la comparación, se obtendrá la lista definitiva de restaurantes ya lista para la base de datos. Una cosa importante a tener en cuenta antes son los **tipos de cocina**, cada una de las tres plataformas tiene unos tipos distintos que pueden hacer referencia a lo mismo y además, habría una variedad demasiado grande. Para ello,primero hay que hacer una reducción de los posibles tipos de comida para la base de datos.

```
acceptedTypes = ["African", "American", "Argentinean", "Asian", "Bar", "Barbecue",
"Bistro", "Brazilian", "Cafe", "Canadian", "Chinese", "Dessert", "English", "French",
 "Fusion", "German", "Greek", "Grill", "Gourmet", "Hamburgers", "Hawaiian", "Healthy",
  "Indian", "Indonesian", "Italian", "Japanese", "Korean", "Lebanese", "Mexican", "Morocco",
   "Organic", "Pizza", "Pub", "Seafood", "South American", "Spanish", "Street Food", "Sushi",
    "Tapas", "Thai", "Turkish", "Vegan Options", "Vegetarian Friendly", "Wine Bar"]
```

Ilustración 88. Lista de tipos de cocina aceptados

Y se hará una conversión de los tipos de comida que sean similares a estos con el correspondiente. Como por ejemplo:

```
'Estadounidense':"Hamburgers", 'Tapas y raciones':'Spanish',
```

Ilustración 89. Conversión tipos de cocina

Además, se aprovechará para almacenar todos los tipos de comida en inglés, para luego poder hacer una traducción consistente si fuera necesario.

Por último, solo falta hacer unas pequeñas adaptaciones a los datos para que sean compatibles con la base de datos y subirlos a esta, dónde primero se subirá la información del restaurante y la base de datos generará un id único para este, que posteriormente, se utilizará para subir la información de la carta.

```
response = requests.request("POST",ip + "/restaurants", data = body)
restaurant_id = json.loads(response.text)[0]["restaurant_id"]
```

Ilustración 90. Subir restaurante

```
response = requests.request("PUT", ip + "/sections", data =  {
    "restaurant_id": restaurant_id,
    "sections":str(sections).replace("[", "").replace("]", "")
})
for entry in finalMenu:
    response = requests.request("POST", ip + "/menus", data = entry)
```

Ilustración 91. Subir carta

Sistema de ORC de Nanonets para digitalizar las cartas

Otra forma de conseguir ampliar la base de datos de las cartas, consistiría en **digitalizar** el contenido de estas si se presentan en forma de imágen. Para ello utilizamos OCR (*Optical character recognition*).

Envío de las imágenes de la carta

Si un restaurante no tuviera una carta, cualquier usuario tiene la opción de poder subirla a través de la app, mediante una o varias imágenes, ya sea abriendo directamente la cámara del móvil o la galería de fotos. También existe la opción de poder subir un archivo PDF de la carta desde el almacenamiento del teléfono.

Ilustración 92. Pantalla de foto para la carta

Si se han seleccionado imágenes, se generará un archivo PDF concatenando cada una en una página distinta. Este archivo, contendrá el identificador del restaurante y será primero enviado al servidor *Node.js,* el cuál posteriormente lo enviará al servicio de *Nanonets*. También se almacenará en una tabla de la base de datos, el identificador de usuario y restaurante para anotar que el usuario ya ha mandado una carta, ya que solo se podrá enviar una carta de cada restaurante por usuario.

```
const sendFile = (request, response) => {
    const {restaurant_id, user_id, file} = request.body

    const form_data = {
        urls: file,
    }

    const options = {
        url : "https://app.nanonets.com/api/v2/OCR/Model/92368006-2d25-4b9e-a188-17c5b837b0a2/LabelUrls/",
        body: querystring.stringify(form_data),
        headers: {
            'Authorization' : 'Basic ' + Buffer.from('API' + ':').toString('base64'),
            'Content-Type': 'application/x-www-form-urlencoded'
        }
    }


    req.post(options, function(err, httpResponse, body) {
        //console.log(err)
        if(err){
            response.status(201).send("BAD")
        }
        else{
            pool.query('INSERT INTO pdfrequest (restaurant_id, user_id) VALUES ($1, $2)',
            [restaurant_id, user_id], (error, results) => {
             if (error) {
               throw error
             }
             response.status(201).send("OK")
            })
        }

    });
}
```

Ilustración 93. Código para subir las imágenes a Nanonets

Análisis y corrección del resultado en Nanonets

Nanonets es una herramienta, que se puede utilizar desde su página web, que nos permite mediante una red neuronal ya entrenada, pasar la imágen de una carta a texto, además de identificar distintos tipos de campos dentro de esta, como puede ser el nombre de un plato, el nombre de una categoría, el precio de un plato o la descripción. Cuenta con un servicio gratuito para analizar hasta 100 documentos al mes, para más cantidad es necesario contratar el servicio de pago.

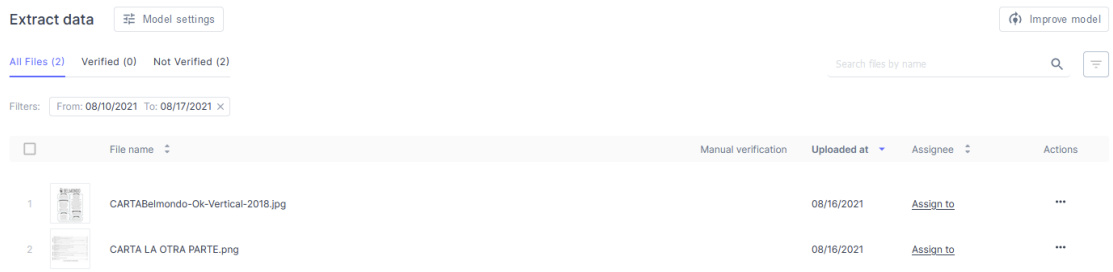

Ilustración 94. Lista de documentos subidos a Nanonets

Una vez subido un documento, este aparecerá ya **analizado** y con los campos **identificados**. La ventaja que ofrece, es que se puede supervisar el resultado de manera sencilla y cómoda para arreglar todos los errores que el modelo ha cometido.

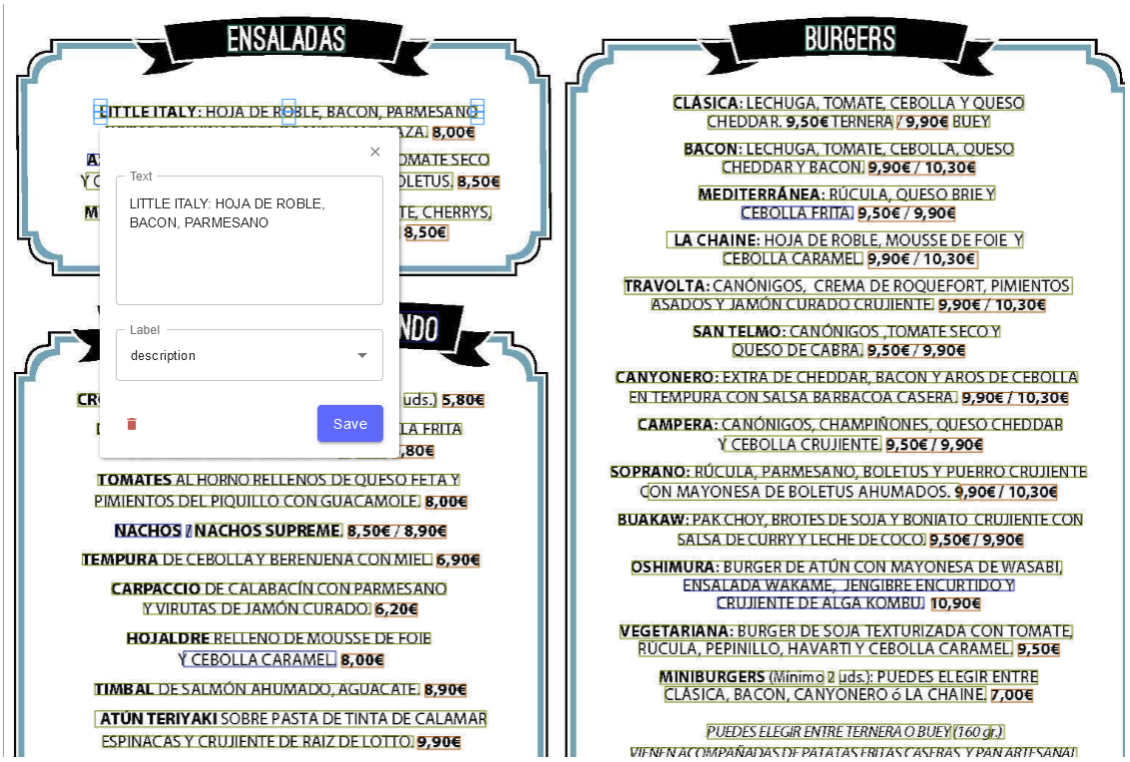

Ilustración 95. Edición del resultado dado por Nanonets

El modelo captura, en distintas celdas rectangulares, cada trozo de texto y le asigna un tipo de campo. Se puede editar manualmente el texto por si fuera erróneo o cambiar el tipo de campo.

También es posible crear un rectángulo nuevo para poder capturar algún elemento que al modelo se le haya escapado.

Cabe destacar que una cosa que no hace el modelo, es enlazar elementos que deban estar relacionados, como por ejemplo, enlazar un precio con su plato correspondiente o un plato con la categoría a la que pertenece. Cada elemento detectado es en principio independiente del resto.

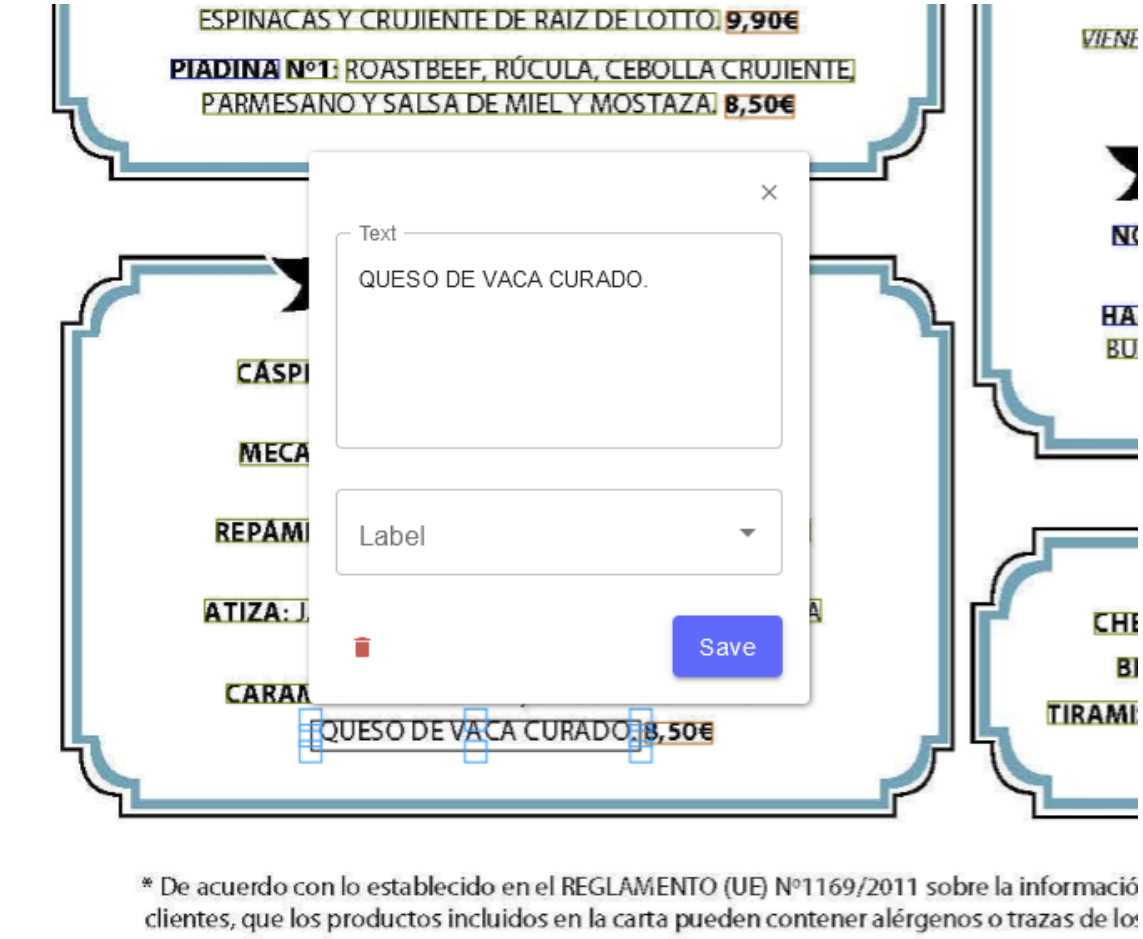

Ilustración 96. Creación de nuevo rectángulo

Una vez se ha arreglado el resultado, este se puede descargar en forma de archivo XML con la siguiente estructura.

```
{
   "label": "category",
   "ocr_text": "BELWONDO",
   "score": 0.91005206,
   "xmin": 192,
   "xmax": 1431,
   "ymin": 256,
   "ymax": 544
},
{
   "label": "category",
   "ocr_text": "ENSALADAS",
   "score": 0.9350949,
   "xmin": 358,
   "xmax": 516,
   "ymin": 592,
   "ymax": 628
},
{
   "label": "category",
   "ocr_text": "BURGERS",
   "score": 0.9901108,
   "xmin": 1140,
   "xmax": 1261,
   "ymin": 593,
   "ymax": 626
},
```


Ilustración 97. Salida de Nanonets

Dónde además del texto y su tipo de campo, se especifican dos cosas más, la primera es el *score,* que es un valor que representa la fiabilidad del texto, es decir, cuanto más cercano a 1 más probable es que el texto analizado sea correcto. Por último, tenemos cuatro números que representan los valores **máximos y mínimos** en el eje de las **X y las Y** que toma el rectángulo, dentro de la imágen y con los cuáles se pueden ubicar los vértices de este en forma de coordenadas. Gracias a esto, se puede intentar relacionar algunos elementos con otros siguiendo un algoritmo.

Ilustración 98. Ubicación de las variables

Cuánto más hacia abajo las Y tendrán un valor más alto, y cuánto más hacia la derecha las X tendrán un valor más alto también.

Construcción de la carta

Por complejidad, y ya que es un elemento más secundario, las descripciones no se intentarán relacionar con su plato, solo el precio y la categoría.

Para cada plato, se intentará asignar la **categoría** a la que pertenece y su **precio**.

El programa para el análisis del XML y la construcción de la carta se ha hecho también mediante *Python*.

Elección de la categoría

Por normal general, en una carta la categoría suele ser un título que se sitúa por **encima** en el eje de las Y de los platos. Por lo tanto, de primeras se pueden descartar todas las que se sitúen por debajo.

En el siguiente ejemplo se explicará el proceso de asignar una categoría al plato con nombre *Cáspita*.

Ilustración 99. Categorías válidas

Dentro del código, *sections* hace referencia a categorías, *entry* al plato y *closest* a la categoría más cercana en el punto actual.

`["coordinates"][0]` hace referencia al X min.

`["coordinates"][1]` hace referencia al Y min.

`["coordinates"][2]` hace referencia al X max.

`["coordinates"][3]` hace referencia al Y max.

El siguiente paso, consiste en calcular la media de las distancias en el eje de las X del X min. del plato y los X min. de las categorías restantes a comprobar.

```
for section in validSections:
  total += abs(int(entry["coordinates"][0]) - int(section["coordinates"][0]))
avg = total/len(sections)
```

Ilustración 100. Código para la media de distancias en el eje de las X

También es necesario decidir un margen de error para hacer las comparaciones, ya que dentro de la imágen las medidas no pueden ser exactas. Este número puede ser un valor fijo, o calcularlo acorde a ciertas medidas como puede ser el alto de un rectángulo de alguna categoría, por ejemplo.

```
threshold = 20
```

Se inicializa una categoría cualquiera como la categoría correspondiente al plato en cuestión, para poder ir haciendo las comparaciones con el resto. A continuación, se va comparando en un bucle cada una de las categorías.

Se calcula primero la distancia en el eje de las X del plato con la categoría en la que estamos iterando.

```
distance=abs(int(section["coordinates"][0])-int(entry["coordinates"][0]))
```

Ahora, se compara que la Y min. de la categoría a comprobar sea mayor que la Y min. de la categoría anotada cómo más cercana, restándole el margen de error, es decir, que sea más cercana al plato en el eje de las Y.

```
if int(section["coordinates"][1]) >= int(closest["coordinates"][1]) - threshold and distance <= avg + threshold:
    closest = section
```

Ilustración 101. Código para comparar las Y min.

Ilustración 102. Siguientes categorías válidas

Se irán descartando con solo esta primera comprobación, todas las categorías que estén claramente demasiado arriba.

Por último, se compara que la distancia en el eje de las X del plato con la categoría que estamos iterando, sea menor o igual que a la distancia media calculada antes más el margen de error.

```
if int(section["coordinates"][1]) >= int(closest["coordinates"][1]) + threshold and distance <= avg + threshold:
    closest = section
```

Ilustración 103. Código para comparar la distancia con la media

Ilustración 104. Categoría elegida

Con esta última comprobación, se descarta el resto de categorías que estaban cercanas en el eje de las Y, pero que están alejadas respecto al eje de las X.

```
for section in validSections:
    if closest == None:
        closest = section
    distance = abs(int(section["coordinates"][0]) - int(entry["coordinates"][0]))
    if int(section["coordinates"][1]) >= int(closest["coordinates"][1]) + threshold and distance <= avg + threshold:
        closest = section
```

Ilustración 105. Código completo

Elección del precio

Por norma general, el precio es un elemento que se sitúa a la **derecha** del nombre del plato y a una distancia en el eje de las Y, relativamente próxima. Por lo tanto, todos los precios que se encuentren a la izquierda del plato en el eje de las X pueden ser inicialmente descartados, igual que ocurría con el caso de las categorías.

En el código, *price* hace referencia al precio actual con el que se está iterando y *closestPrice,* hace referencia al precio más cercano que hay elegido actualmente

```
int(price["coordinates"][0]) > int(entry["coordinates"][2])
```

A continuación, se elegirá el ejemplo del plato 'Tomates' para la elección del precio.

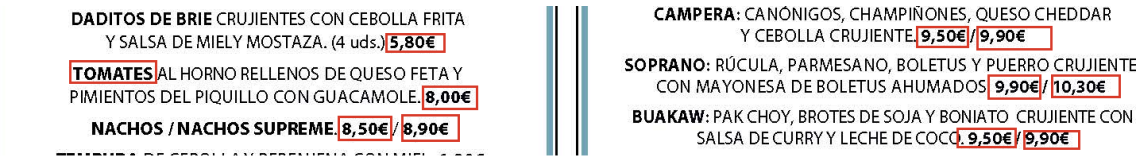

Ilustración 106. Precios iniciales

Para evitar errores, si algún plato no tuviera precio o la estructura de la carta está muy alejada de lo común, se descartan directamente los precios que en cuanto al eje Y, comparados con el plato, estén demasiado alejados dado un umbral. El valor de este umbral puede, por ejemplo, decidirse en base a la distancia en el eje Y del plato o de un número fijo. Si después de este paso no queda ningún precio válido, no se asignará ninguno.

```
threshold = 2*int(entry["coordinates"][3]) - int(entry["coordinates"][1])
```

Entonces, tomando la distancia entre los puntos medios en el eje de las Y de los rectángulos del plato y de los precios restantes, si esta es menor o igual que el valor del umbral, no serán descartados.

Ilustración 107. Distancias para la selección de los precios válidos

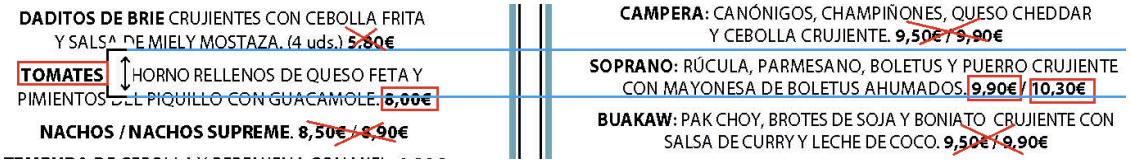

Ilustración 108. Precios válidos

```
abs((int(price["coordinates"][3]) + int(price["coordinates"][1]))/2
- (int(entry["coordinates"][3]) + int(entry["coordinates"][1]))/2) <= threshold
```

Ilustración 109. Código para decidir si un precio es válido

Por último, solo quedaría elegir respecto al eje de las X cuál es el precio más próximo, por lo que primero se asigna directamente como precio más próximo uno cualquiera de los disponibles y después, se iría iterando en un bucle con todos los precios restantes para ir comparándolos.

```
if int(price["coordinates"][0]) < int(closestPrice["coordinates"][0]):
    closestPrice = price
```

Ilustración 110. Código para elegir el precio más cercano

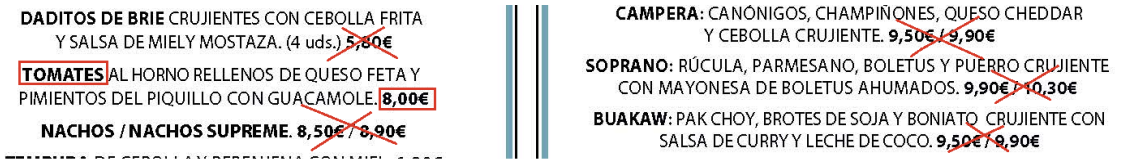

Ilustración 111. Precio elegido

Una vez asignados a cada plato su precio y la categoría a la que pertenece, solo faltaría subir a la base de datos toda esta información, y ya existiría una carta para el restaurante.

9. Conclusiones y líneas futuras

Vista la necesidad de tener una aplicación dónde poder **consultar** los platos de la carta de cualquier restaurante, u otra centrada en la **valoración** de éstos, la aplicación tiene el potencial de poder resolver estos problemas, además, de otorgar funcionalidades extra que la convertirían en una aplicación más completa y con un uso más amplio.

La aplicación posee diferentes formas de monetización, lo cuál probablemente la convertiría en un proyecto rentable si se lanzara al mercado de forma total.

Los principales problemas que aparecen para que la aplicación funcione comercialmente son:

- Es necesaria una cantidad elevada de usuarios activos, para que proporcionalmente, exista una cantidad elevada de valoraciones para poder discernir qué platos son más gustados o cuáles no.
- Es necesaria una base de datos de restaurantes y cartas amplia, para ello, se podría utilizar los sistemas de *scraping* planteados, pero solo a nivel nacional ya sería necesario utilizarlos en un número de ubicaciones muy alto.
- Si la cantidad de imágenes de cartas enviadas por los usuarios es muy elevada, se necesitaría mucho tiempo para la supervisión del resultado del sistema OCR de *Nanonets*.

Se han podido completar todos los objetivos planteados en la planificación del proyecto.

Como posible futuro trabajo, lo ideal sería poder adaptar la aplicación para el sistema operativo *iOS* y para web. *Flutter* ofrece la posibilidad de desarrollar para las tres plataformas, para *iOS* probablemente podría utilizarse el mismo código que para *Android,* pero haciendo todas las adaptaciones necesarias. Para la web, sería necesario crear un proyecto nuevo, pero se podrían reutilizar muchas partes del código de este proyecto.

Personalmente, los conocimientos obtenidos en diversas asignaturas cursadas a lo largo de la carrera han sido claves para el desarrollo del proyecto. Los contenidos de la asignatura Bases de datos y sistemas de información, han sido totalmente necesarios para la gestión de la base de datos. Conocimientos aprendidos sobre *Javascript* y *Node.js* en Tecnología de sistemas de información en la red, han sido muy útiles para la parte del servidor. También ha sido de gran utilidad la asignatura Sistemas de almacenamiento y recuperación de información, debido al aprendizaje en detalle del lenguaje *Python* y el tratamiento de datos. Por último *Flutter*, que es el desarrollo que más ha abarcado del proyecto, lo he aprendido por cuenta propia. La elección de esta tecnología sobre el desarrollo nativo en *Android* con *Java* o *Kotlin,* se debe a que existe la opción desarrollar para varias plataformas si se desea, sin sacrificar mucho rendimiento de la aplicación.

10. Bibliografía

[1] El Tenedor. (2021, 31 de Agosto). *Discover and book the best restaurant.*

https://www.thefork.com/

[2] Foodyt. (2021, 31 de Agosto). *Find where to eat your favourite dish.*

https://www.foodyt.com/en/esp/

[3] QRCarta. (2021, 31 de Agosto). *Carta digital QR para su establecimiento.*

https://www.qrcarta.com/

[4] Maida, Esteban Gabriel & Pacienzia, Julián. (2015). Metodologías de desarrollo de software. Tesis de Licenciatura en Sistemas y Computación. *Metodologías clásicas. Incremental* (pp 44-45). Facultad de Química e Ingeniería "Fray Rogelio Bacon".

[5] Firebase. (2021, 31 de Agosto). *Firebase helps you build and run successful apps.*

https://firebase.google.com/

[6] Nanonets. (2021, 31 de Agosto). *Automate manual data entry using AI.*

https://nanonets.com/

[7] Stripe. (2021, 31 de Agosto). *Infraestructura de pagos para Internet.*

https://stripe.com/es

[8] RevenueCat. (2021, 31 de Agosto). *Connect in-app purchase data everywhere you need it.*

https://www.revenuecat.com/integrations

[9] Apify. (2021, 31 de Agosto). *Make the web work for you.*

https://apify.com/

[10] Figma. (2021, 31 de Agosto). *Figma connects everyone in the design process so teams can deliver better products, faster.*

https://www.figma.com

[11] IONOS. (2020, 09 de Octubre). *Flutter: introducción al framework multiplataforma.*

https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-flutter/

[12] Documentación Flutter. (2021, 31 de Agosto). *Start thinking declaratively.*

https://flutter.dev/docs/development/data-and-backend/state-mgmt/declarative

[13] Documentación Dart. (2021, 31 de Agosto). *Asynchronous programming: futures, async, await.*

https://dart.dev/codelabs/async-await

[14] Documentación Firebase. (2021, 31 de Agosto). *Firebase Dynamic Links.*

https://firebase.google.com/docs/dynamic-links

[15] Google Play Console. (2021, 31 de Agosto). *Built by you. Powered by us. Experienced by billions.*

https://play.google.com/console/developers

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No procede
Fin de la pobreza.				X
Hambre cero.				X
Salud y bienestar.		X		
Educación de calidad.				X
Igualdad de género.				X
Agua limpia y saneamiento.				X
Energía asequible y no contaminante.				X
Trabajo decente y crecimiento económico.			X	
Industria, innovación e infraestructuras.			X	
Reducción de las desigualdades.				X
Ciudades y comunidades sostenibles.				X
Producción y consumo responsables.				X
Acción por el clima.			X	
Vida submarina.				X
Vida de ecosistemas terrestres.				X
Paz, justicia e instituciones sólidas.				X
Alianzas para lograr objetivos.				X

De los anteriores objetivos de desarrollo sostenible, el proyecto está relacionado con:

- **Salud y bienestar:** Dentro de la aplicación, el hecho de poder consultar en los platos de los restaurantes los alérgenos, facilita y promueve el consumo de forma segura de personas con alergias alimenticias.

- **Trabajo decente y crecimiento económico:** El proyecto en general es una forma clara de apoyo a los restaurantes para su crecimiento y desarrollo, debido a los servicios digitales que aporta y la publicidad que le puede otorgar para que nuevos usuarios lo descubran.

- **Industria, innovación e infraestructuras:** Gracias al importante *feedback* de las valoraciones de los platos, es más fácil poder llegar a innovar en otros nuevos, al rápidamente conocer las opiniones de los comensales sobre estos.

- **Acción por el clima:** En cierto modo, debido otra vez al conocimiento de las opiniones de los platos, es más sencillo intuir que comida priorizar a la hora de comprar y elaborar en el día a día, optimizando la producción y evitando parcialmente el derroche de comida que no se ha consumido.