



iesperemariaorts

**IES PERE MARIA ORTS I BOSCH
CICLOS FORMATIVOS DE GRADO
SUPERIOR**



**Familia Profesional: Informática y
Comunicaciones**

Sei Vegan, Sei Grün

Alumno: Alejandro Martínez Morillo

IES PERE MARIA ORTS I BOSCH

CICLOS FORMATIVOS DE GRADO SUPERIOR



**Familia Profesional: Informática y
Comunicaciones**



iesperemariaorts

PROYECTO DE FIN DE CICLO
Desarrollo de Aplicaciones Multiplataforma

Sei Vegan, Sei Grün

AUTOR: Alejandro Martínez Morillo
TUTOR: Jose Rafael Caturla Palao

Benidorm, 9 de junio de 2023

Agradecimientos

Este proyecto ha resultado ser un reto, tanto por lo que ha sido el proyecto como por causas externas, por lo que me gustaría agradecerse a muchas personas.

A las primeras personas que me gustaría agradecer son a mis profesores y quiero nombrar a todos, desde Antonio y Claudia, que son asignaturas algo más alejadas del ciclo, hasta a Marina, Jose, Jaume, Fran y Noe. Gracias a todos por ayudarme a crecer académicamente hablando.

También quería agradecer a esas personas que han estado conmigo y han aguantado infinidad de tonterías, buenos y malos ratos y, sobretodo, risas. Gracias a Guillermo, Salva y David por hacer que este año haya sido increíble y haber hecho que ir a clase sea especial, porque aunque me pasase algo malo o no, estabais ahí para hacérmelo olvidar. Y, ya que hablamos de amistades, también me gustaría nombrar a mi amiga Toñi, que me ha apoyado y aguantado como una campeona. Gracias por todo chicos.

A las siguientes personas que quiero agradecer, y también dedicar, este proyecto es a mis padres, sobre todo a mi madre, que desde que empezó el año no ha tenido, ni de lejos, una buena temporada. Me han ayudado y apoyado en muchas situaciones y han soportado alguna que otra mala contestación cuando estaba estresado o ansioso y, claramente, no se la merecían. Gracias por todo, os quiero mucho y os merecéis un mundo entero.

Por último, pero no menos importante, quería agradecer, y también dedicar, este proyecto a mi pareja Angie. Ella es la única realmente que sabe lo mucho que eh trabajado durante este curso, lo mucho que he sufrido haciendo este proyecto, y más aún en el intermodular. Ella es la que siempre me ha animado y apoyado en todas y cada una de las decisiones que he tomado, ayudándome siempre a elegir la mejor. Gracias a ella he crecido mucho como persona y me ha hecho ser quien soy ahora mismo. Eres única Angie, te quiero mucho, gracias por todo lo que has hecho por mí.

Contenido

Agradecimientos	i
Resumen.....	vi
Abstract.....	vii
1. Introducción	2
1.1. Objetivos iniciales.....	3
1.2. Objetivos cumplidos	4
1.3. Justificación del proyecto	4
2. Estudio económico.....	6
2.1. Idea de negocio	6
2.2. Elección de forma jurídica	7
2.3. Estudio de mercado.....	8
2.3.1. Macroentorno.....	8
2.3.1.1. Económico	8
2.3.1.2. Sociocultural	9
2.3.1.3. Político	10
2.3.1.4. Legal	10
2.3.1.5. Tecnológico	11
2.3.1.6. Medioambientales	11
2.3.1.7. Internacionales	11
2.3.2. Microentorno	12
2.3.2.1. Competidores	12
2.3.2.2. Clientes	12
2.3.2.3. Proveedores.....	12
2.3.2.4. Productos sustitutivos	13
2.3.3. Ubicación del negocio y plano del local.....	13
2.3.4. Análisis DAFO	14
2.4. Plan de marketing	14

2.4.1. Producto	14
2.4.2. Precio	14
2.4.3. Promoción	15
2.4.4. Place/distribución.....	15
2.5. Plan de producción y recursos humanos	15
2.5.1. Inversiones y gastos	15
2.5.2. Origen del financiamiento	16
2.5.3. Ayudas y subvenciones.....	17
2.5.4. Distribución de costos.....	17
2.5.5. Umbral de rentabilidad o punto muerto	17
2.6. Trámites de constitución, puesta en marcha e impuestos	18
3. Análisis	20
3.1. Planificación temporal	20
3.2. Casos de uso.....	21
3.2.1. Aplicación administrativa	22
3.2.1.1. Caso de uso – Gestionar recursos del restaurante.....	22
3.2.2. Aplicación trabajadores	24
3.2.2.1. Caso de uso – Camarero	24
3.2.2.2. Caso de uso – Cocinero	24
3.2.3. Aplicación clientes.....	25
3.2.3.1. Caso de uso – Usuario- Crear pedido online	25
3.2.3.2. Caso de uso – Usuario- Ver menú	25
3.2.3.3. Caso de uso – Usuario- Editar perfil.....	26
3.3. Diagrama de clases	27
3.4. Modelo relacional	29
4. Entorno de programación.....	30
4.1. Lenguajes y tecnologías que vamos a utilizar	30
4.1.1. Base de datos	30

4.1.2. Backend	31
4.1.2.1. Node.js y Express	31
4.1.2.2. Librerías utilizadas.....	31
4.1.3. Frontend	33
4.1.3.1. Dart	33
4.1.3.2. Flutter.....	33
4.1.3.3. Provider.....	34
4.1.3.4. Bloc	34
4.1.3.5. Librerías utilizadas.....	35
4.2. Preparación y configuración del entorno de programación	38
4.2.1. Flutter.....	38
4.2.2. Visual Studio Code	39
4.2.3. Máquinas virtuales y dispositivo físico	43
4.2.4. Otras aplicaciones	45
4.3. Sistema de control de versiones	45
5. Implementación: Prototipado, codificación y validación	47
5.1.1. Prototipado inicial.....	47
5.1.2. Primeras aproximaciones	48
5.2. Aspectos relevantes de la codificación y validación.	53
5.2.1. API REST	53
5.2.1.1. Autorización.....	55
5.2.1.2. Validación	55
5.2.1.3. Comprobación de errores	56
5.2.1.4. Función del <i>endpoint</i>	57
5.2.2. Aplicación cliente	58
5.2.2.1. Splash Screen.....	58
5.2.2.2. Login Screen	60
5.2.2.3. Register Screen	61

5.2.2.4. Main Screen – Home	62
5.2.2.5. Main Screen – Menú	65
5.2.2.6. Main Screen – Profile	71
5.2.3. Aplicación Trabajador – Camarero.....	73
5.2.4. Aplicación Trabajador – Cocinero.....	75
5.2.5. Aplicación administradora.....	76
5.3. Pruebas de funcionamiento y accesibilidad	80
5.3.1. Aplicación administrativa	80
5.3.2. Aplicación camarero	85
5.3.3. Aplicación cliente	89
5.3.3.1. Registro y edición de datos	89
5.3.3.2. Inicio de sesión y creación de pedido	91
5.3.4. Aplicación cocinero.....	93
6. Despliegue del proyecto.....	95
6.1. Ejecución de los proyectos	95
7. Conclusión del proyecto	98
8. Bibliografía	100
8.1. Apartado económico	100
8.2. Extensiones VSCode.....	101
8.3. Paquetes Flutter.....	103
8.4. Otras referencias	105

Resumen

Este documento representa la creación y el desarrollo del proyecto final de curso del Ciclo Superior de Desarrollo de Aplicaciones Multiplataforma.

Para la creación del mismo, se han creado distintos elementos ficticios para que el proyecto sea lo más cercano al mundo laboral. Como primer elemento, se ha creado una sociedad limitada de un único socio llamada *SoftDev*. En el apartado 2 de este mismo proyecto, observaremos el proceso completo de lo que es crear una sociedad limitada con unas características concretas. Además, el segundo ficticio al cual está ligado es el restaurante el cuál necesita nuestros servicios, *Grün*. Este restaurante aporta una oferta vegana al mercado, la cual es escasa actualmente y refleja unos valores que el creador de dicho proyecto defiende.

Apartando el tema empresarial y económico, en este documento veremos reflejado el desarrollo de 4 aplicaciones, centradas en distintos usuarios finales.

El primer paso de todos fue analizar el proyecto a grandes rasgos y ver cómo enfocar el desarrollo del mismo. Un diagrama de *Gantt* inicial o analizar y crear los posibles casos de usos de las aplicaciones son algunos de los primeros pasos que se deben hacer.

Ya analizada la aplicación y observar las consideraciones a tener en cuenta, llega el momento de preparar el entorno de programación y sus librerías, el control de versiones de los proyectos y el estudio de nuevas tecnologías que se pondrán en uso.

Por último, podremos observar distintos apartados destacables de la aplicación, ya sean por su complejidad o por lo interesante y útil que ha resultado implementar dicho código. Además, pondremos a prueba la aplicación utilizando los casos de uso creados con anterioridad.

Abstract

This document represents the creation and development of the final course project of the course of Multiplatform Applications Development.

For the creation of the project, different fictitious elements have been created to make the project as close as possible to the working world. As a first element, a single-partner limited company called SoftDev has been created. In section 2 of this project, we will observe the complete process of creating a limited company with specific characteristics. In addition, the second fictitious to which it is linked is the restaurant which needs our services, Grün. This restaurant brings a vegan offer to the market, which is currently scarce and reflects some values that the creator of this project defends.

Apart from the business and economic issue, in this document we will see reflected the development of 4 applications, focused on different end users.

The first step of all was to analyze the project in broad strokes and see how to approach its development. An initial Gantt chart or analyze and create the possible use cases of the applications are some of the first steps to be done.

Once the application has been analyzed and the considerations to be taken into account have been observed, it is time to prepare the programming environment and its libraries, the project version control and the study of new technologies to be used.

Finally, we will be able to observe different remarkable sections of the application, either because of their complexity or because of the interesting and useful result of implementing the code. In addition, we will test the application using the use cases created previously.

1. Introducción

Sei Vegan, Sei Grün es un proyecto realizado por la empresa *SoftDev*, una empresa de desarrollo software que creará y desplegará las aplicaciones que necesita el restaurante Grün, centrado en la comida vegana.

La representación de este proyecto se divide en tres partes, una centrada en el uso del administrador, una segunda centrada en el uso de los trabajadores, tanto los cocineros como los camareros, y una tercera y última parte centrada en los clientes.

En la primera aplicación, centrada en la administración, el dueño o la persona designada en la gestión de recursos podrá encargarse de añadir, borrar y actualizar los datos del restaurante, ya sean los datos relacionados con los trabajadores como los relacionados con los productos del negocio.

Por otro lado, la segunda aplicación que se desarrollará para este negocio estará centrada en el uso de los trabajadores a la hora de atender a los clientes en el local, pensada para ser utilizada en un teléfono móvil o tableta. Por un lado, los camareros tendrán acceso a un TPV para crear y gestionar los pedidos realizados en el local, ya sean para degustarlos en el lugar o para llevar. Por otro lado, los cocineros tendrán una vista, pensada para que se utilice en una tableta o AIO (*All in one*). En este apartado, los cocineros podrán ver todos los pedidos, junto a sus detalles, y marcarlos como “Finalizados”.

Por último, se desarrollará la aplicación centrada en el uso del cliente. En dicho programa, los usuarios tendrán dos acciones principales: visualizar el menú y crear un pedido. La primera opción está diseñada y orientada para el uso en el local, de esta manera los clientes pueden visualizar el menú en cualquier momento y no tendrán por qué esperar a que un camarero les entregue la carta físicamente. La segunda función está centrada en el uso desde el domicilio. Desde la comodidad de su casa, podrán pedir cualquier producto de la carta para que lo entreguen en su domicilio.

1.1. Objetivos iniciales

A la hora de presentar la propuesta de proyecto, se estipularon una serie de objetivos que, como veremos más adelante, se cumplen. A grandes rasgos, el objetivo principal del proyecto es desarrollar tres tipos de aplicaciones, las cuáles han sido desarrolladas *grosso modo*. Para la parte administrativa, tenemos como objetivos desarrollar un CRUD (*Create, Read, Update Delete*) completamente funcional.

Por otro lado, a la hora de desarrollar la aplicación para los trabajadores, concretamente para los camareros, el objetivo es crear distintas funciones, las cuales son:

- ⇒ Creación de pedidos.
- ⇒ Creación de facturas.
- ⇒ Diseño de un menú e implementarlo en la aplicación actual.

En cambio, en la vista de los cocineros tendremos una única función; visualizar los pedidos y todos sus detalles.

Como objetivo adicional, se desea implementar un *socket* para que dichas aplicaciones sean en tiempo real y que los cocineros no tengan la necesidad de darle un botón para recibir los nuevos pedidos

La tercera aplicación que se desea desarrollar tiene como función principal la de crear pedidos a domicilio. También se quiere desarrollar una vista con el menú completo con el objetivo de ayudar al usuario final en la elección de sus productos. Asimismo, como en la anterior aplicación, se intentará implementar un *socket* para que sea una aplicación en tiempo real, facilitando tanto la llegada de los pedidos como las actualizaciones del mismo.

Como último objetivo prioritario de este proyecto, se desarrollará un software intermediario que tiene como finalidad la comunicación entre aplicaciones y la base de datos. En ella, se desarrollarán distintas funciones que ayudarán en la comunicación del software anteriormente nombrado.

Por último, como objetivo ajeno a la programación en sí, se desea diseñar y recrear el menú del restaurante y realizar las fotografías correspondientes para utilizarlas en las distintas aplicaciones.

1.2. Objetivos cumplidos

Tras realizar el proyecto, podemos analizar todos los puntos obligatorios y observar que todos los puntos han sido cumplidos correctamente. Además de todo esto, algunos puntos opcionales también han sido añadidos, como que algunas aplicaciones sean a tiempo real, y también han sido añadidas algunas características que he creído que serían buenas para el proyecto, como la impresión de los pedidos en forma de ticket o la implementación de gráficos para que el dueño pueda observar qué productos son los más/menos vendidos.

1.3. Justificación del proyecto

La decisión de desarrollar este proyecto tiene un trasfondo totalmente personal. Mis padres, desde pequeño, me han transmitido un amor y una pasión por la comida que, a día de hoy, sigo manteniendo. Además, siempre me han ido enseñando a cocinar, desde una simple merienda hasta algún plato que pueda alimentar a una familia entera.

Por otro lado, otra parte característica de este proyecto es que mantenga una dieta vegana. Desde hace unos años, mi curiosidad sobre este tipo de dietas ha ido en auge, siendo el confinamiento uno de los momentos en los que más me centré en ello. Desde entonces, he descubierto un mundo culinario muy interesante, tanto como plantear llevar este tipo de dieta, aunque no es correcto llamarlo de tal manera.

En conclusión, he juntado dos facetas que me hacen feliz, la programación y la cocina, junto a un mundo culinario que no está para nada explotado. Además de crearme un reto en cuánto a la programación, decidí conjuntarlo con la creación de un menú propio para fusionar al máximo posible estas dos facetas propias.

2. Estudio económico

2.1. Idea de negocio

Mi idea de negocio es crear una empresa en la que, como anteriormente he dicho, pueda desarrollar *software* que sea moderno. El término “moderno” se puede interpretar de distintas maneras. Quiero enfocar este término tanto a la estética como a la programación, para que las empresas tengan siempre una opción óptima. Como idea principal, quiero mantener siempre con una continua formación de mí mismo, o mis futuros empleados, para crear aplicaciones con los lenguajes de programación más actuales.

Actualmente, estoy centrado en el estudio y la implementación de lenguajes de programación como *Dart* y *Flutter*, lenguaje multiplataforma que nos permite, con el mismo código, exportar a otros sistemas operativos sin problema, y Kotlin con *Jetpack Compose*, el cual nos permite desarrollar aplicaciones en móvil y, con *Desktop Compose*, en escritorio. Añadir que ambos lenguajes son desarrollados por *Google*.

Por otro lado, en cuanto a estética, actualmente la más común en la mayoría de lenguajes es *Material Design*, desarrollado por *Google* también. A mi empresa le viene como anillo al dedo que esta estética sea muy reclamada y utilizada porque, tanto *Flutter* como *Jetpack Compose*, la utilizan dicha en todos sus componentes.

Por último, en lo que se centra justamente este proyecto dentro de la empresa es sobre una aplicación móvil para pedir comida a domicilio de un restaurante vegano. Además, centrado en el ámbito empresarial, desarrollaré una aplicación TPV (Terminal de Punto de Venta) centrada para el uso de los camareros y una aplicación centrada en el administrador del restaurante para gestionar todos los datos del mismo.

2.2. Elección de forma jurídica

Tras analizar nuestras expectativas, nuestro capital inicial y otros factores, se decide que la forma jurídica para este proyecto es una sociedad de responsabilidad limitada. Además de ser la forma social muy común a la hora de realizar proyectos, otra razón tiene que ver con lo dicho inicialmente, las expectativas. Si tuviésemos un capital mucho más alto y tuviéramos en mente salir a bolsa, podríamos optar para elegir una sociedad anónima como forma social, por ejemplo. Por otro lado, esta forma social es apropiada para las PYME, las pequeñas y medianas empresas, por lo que es la opción perfecta para nuestra empresa.

Como se comentó en el apartado Idea de negocio, este proyecto lo realizaré por mi cuenta, por lo que el número de socios de esta SL es de uno. De igual manera, la responsabilidad frente a terceros de los socios es limitada. Esto significa que, si en el peor de los casos la sociedad tiene pérdidas, éstas quedan limitadas a los bienes, los derechos y el capital al nombre de la empresa, al contrario que los autónomos o personas físicas que tienen una responsabilidad ilimitada.

Enlazando con el anterior apartado, el capital inicial de la sociedad será de 5000€. A la hora de crear una S.L., no es necesario tener un capital social elevado, al menos 1€, pero al momento de hacer el plan de tesorería es mucho más sencillo tener, al menos, 3000€. Esto es debido a que, si no superas los 3000€, deberás apartar, al menos, un 20% de las ganancias hasta que el capital social alcance los 3000€ estipulados.

Antes de comentar que debemos presentar para crear nuestra S.L., debemos comentar el aspecto fiscal de la misma. Si fuésemos personas físicas o autónomos, nuestra forma jurídica se encontraría sometida al impuesto sobre la renta de las personas físicas, es decir, el IRPF, el cual es variable según la renta que tengamos, por lo que este impuesto aumentaría si el rendimiento de nuestra empresa aumenta. En cambio, al ser una sociedad limitada, nuestra forma judicial es sometida al impuesto de sociedades. En nuestro caso, tendremos que pagar un 15% durante el primer año en el que tengamos beneficios y, a partir de este, pagaremos un 25%

Por último, para crear la S.L. debemos aportar distintos datos, los cuales son:

- Una escritura pública que se inscribe en el Registro Mercantil, el cual nos otorga una personalidad jurídica.
- La identidad de los socios, en este caso la identidad de la madre e hijo.
- La voluntad de constituir una S.L.
- Las aportaciones que otorgue cada socio y la numeración de las participaciones.

- Los estatutos de la sociedad.
- La manera en la que se va a administrar la sociedad.
- La identidad de las personas, ya sea única o varias, que se encargará de la administración y la representación social.

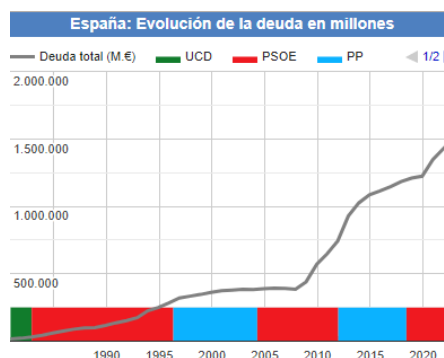
2.3. Estudio de mercado

En este apartado vamos a analizar a nuestra futura empresa, tanto el entorno que la rodea como los puntos fuertes y debilidades que tiene.

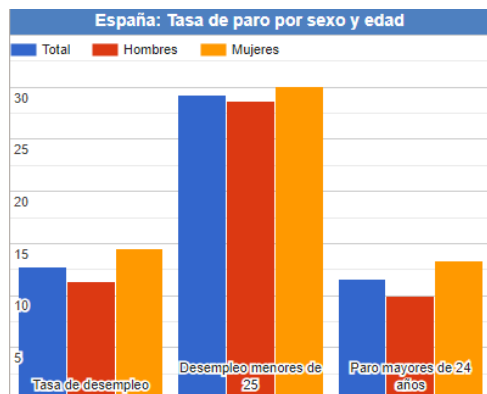
2.3.1. Macroentorno

2.3.1.1. Económico

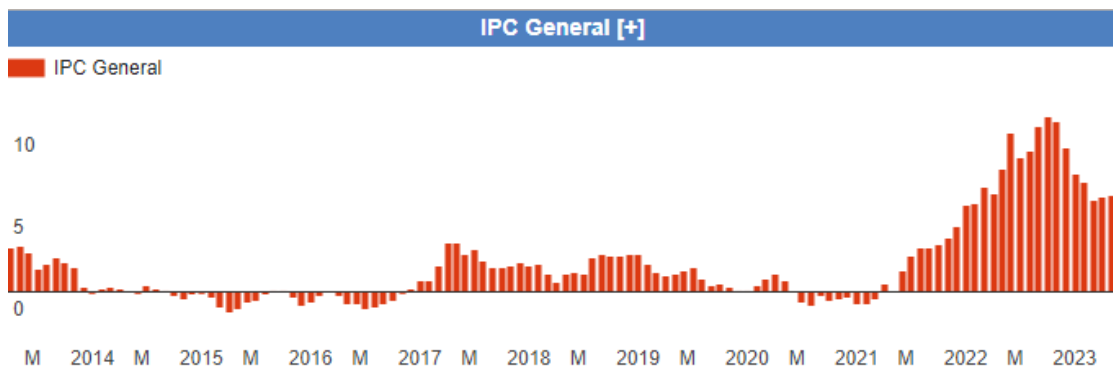
España está viviendo una situación económica complicada, la cual aumentó hace unos años con la pandemia del Covid-19. En noviembre del anterior año, la deuda pública aumento en 8.384 millones de euros respecto al mes anterior, pasando de 1.497.154 millones a 1.505.538 millones. Si recapitulamos a noviembre del 2021, observamos que la deuda ha crecido 1.484 € por habitante. Si esto se expresa mediante el PIB, observamos que, en este último trimestre, la deuda alcanzó el 115.6% del PIB en España, un 0.5% menos que el trimestre anterior. En cambio, si comparamos este último trimestre con el último trimestre de 2021, veremos que la deuda ha incrementado en 71.458 millones de euros.



Por otro lado, si hablamos del desempleo en España, observamos que, durante el último año, ha disminuido un 0.9%, posicionando el paro en un 12.4%. Actualmente, el paro está en un 12.67%, el cual ha aumentado desde un 0.27% desde noviembre del 2021. En cuánta de persona, el número de parados actualmente es de 2.980. 200 personas.



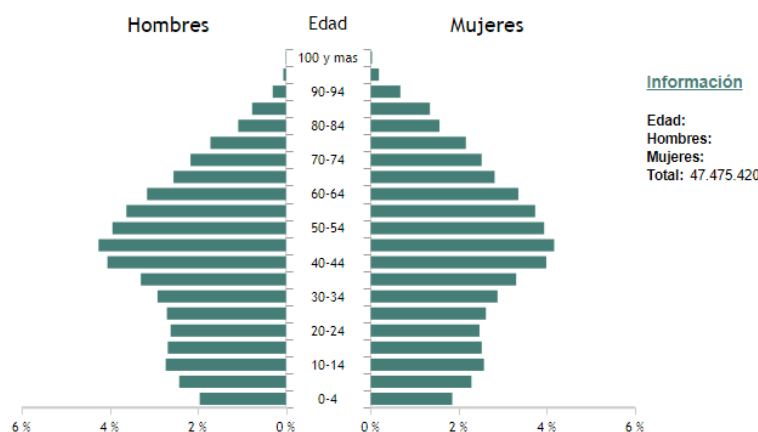
Por último, para hablar sobre el IPC, un tópico bastante recurrido en los debates y hablado por la población. La inflación en España, debido al conflicto bélico entre Ucrania y Rusia, ha aumentado un 6.4% desde diciembre del 2021. Aunque pensemos o, desde los medios de comunicación, parezca que ha incrementado mucho la inflación, si comparamos nuestro IPC interanual con los diferentes miembros de la Unión Europea, observamos que España es uno de los países con menor IPC. Por ejemplo, nuestro país vecino, Portugal, ha acumulado un 9.6% y Alemania, un país referente, ha aumentado un 8.6%.



Todos estos datos están obtenidos de Datos Macro y sus distintas secciones.

2.3.1.2. Sociocultural

Actualmente en España, con una población de 47.615.034 personas, se encuentra en la posición 30 de la tabla de poblaciones del mundo, con una densidad de 94 habitantes/Km².



Si observamos el gráfico anterior, observamos que los picos más notables son de las personas entre los 40 y los 54. La conclusión clara que obtenemos de este gráfico es que tenemos más adultos, entre 40 y 54 años, que jóvenes, algo que comentaremos más adelante.

Si hablamos de nuestra sociedad y sus valores, ahora mismo estamos viviendo una situación en la que la tecnología está en auge. Tanto gente mayor como más joven tiene a su disposición un teléfono móvil, tableta u ordenador. Actualmente, pensar en un día a día sin un teléfono móvil es algo difícil de pensar, sobre todo para el público joven.

Si enlazamos los puntos anteriores con nuestra empresa, podemos ver una ventana de negocio muy rentable. La mayor cantidad de personas ronda entre los 40 y 55 años, además de haber una cantidad considerablemente de jóvenes. Esto nos va a beneficiar sin ninguna duda, ya que nuestro futuro producto, que será una aplicación de un restaurante vegano, estará orientado a gente joven-adulta.

2.3.1.3. Político

Nuestro país, España, es un estado democrático cuya forma política es una monarquía parlamentaria. Los ciudadanos participan en asuntos públicos a través de los representantes que se eligen en elecciones plurales, universales y plurales.

La situación actual en nuestro país es bastante tranquila y constante, con algunos picos de enfrentamientos políticos pacíficos entre el actual gobierno y los demás partidos. Actualmente, en España gobierna el Partido Socialista Obrero Español, con el apoyo de Unidas Podemos, el Partido de los Socialistas de Cataluña e Izquierda unida/Partido Comunista de España. Este conjunto de partidos de “izquierdas” se enfrenta, en unos meses, a unas elecciones generales en la que lucharán para mantener dicho gobierno de coalición y parar el auge de la extrema derecha.

2.3.1.4. Legal

A nivel estatal, existe un Convenio Colectivo que regula el ámbito de servicios informáticos, además de otros servicios como gestoría. Este CC fue publicado en el BOE el 6 de marzo de 2018, con resolución el 22 de febrero del mismo año

Por otro lado, una ley que regula una parte de todos los servicios informáticos es la LOPD, es decir, la Ley Orgánica de Protección de Datos.

Por último, existen distintas leyes que regulan las sociedades limitadas, como en cualquier otra forma jurídica. Por ejemplo, nuestra S.L está regulada por la ley del Impuesto sobre el Valor Añadido (IVA), la ley que regula el Impuesto sobre sociedad o el RD refundido de la Ley de Sociedades de Capital.

Todas las referencias a dichos documentos están en la bibliografía, al final de este documento.

2.3.1.5. Tecnológico

En los últimos años, la tecnología está dando pasos agigantados. Hace una década, nadie se imaginaría que, al visitar un restaurante, el camarero que te trae los platos es un robot. Este salto no solo ha sido en la informática y en inteligencia artificial, muchos ámbitos han sido beneficiados de dicho salto, como puede ser el alimenticio. Desde nuevos robots de cocina hasta, lo nombrado anteriormente, un camarero robot son unos de los muchos avances que ha traído la tecnología a nuestras cocinas.

Para mi empresa, este avance de la tecnología es algo complejo, porque a la par que comenzamos a adquirir conocimientos de nuevos lenguajes, puede que aparezcan nuevos lenguajes de los cuales aprender. La parte buena de todo esto es que, conforme pasen el tiempo, seguiremos teniendo trabajo constante y empresas a las cuales ofrecer nuestro servicio.

2.3.1.6. Medioambientales

El cambio climático, la contaminación y las distintas causas medioambientales han supuesto un cambio de mentalidad mundialmente. Aunque muchos gobiernos no están tomando medidas, otros tantos las están tomando para frenar el calentamiento global. Centrado en la informática, esta no produce una cantidad notable de residuos si lo miramos de una manera general, pero existe alguna rama de la informática que sí que es contaminante. La rama de la cual hablo es sobre el tema de criptomonedas. Estas se consiguen, o, mejor dicho, se minan mediante el uso de tarjetas gráficas. Mucha gente mantiene habitaciones repletas de tarjetas gráficas, las cuáles necesitan una cantidad considerable de energía y, debido a su uso tan continuado y de una manera tan fuerte, estas no duran tanto como en una situación normal, por lo que deben cambiarse frecuentemente. A consecuencia de esto, se genera mucha basura electrónica, la cual contienen productos químicos que se pueden filtrar al suelo provocando una contaminación en las zonas donde penetran. Se calcula, aproximadamente, que Bitcoin, la criptomoneda más importante del momento, ha aumentado severamente las emisiones contaminantes.

2.3.1.7. Internacionales

En los últimos meses, la situación internacional es bastante cruda. Ucrania lleva algo más de un año en conflicto con Rusia. A consecuencia de esto, la UE está viviendo una crisis energética que afecta a todos los países centrales y del norte, dado que Rusia era su primera fuente de gas. A causa de esto, países del sur, como España o Italia, han tenido que reducir su consumo energético para que los países del norte, como Alemania, puedan disfrutar de un

invierno sin ninguna dificultad. Además de la energía, este conflicto bélico ha hecho que las materias primas, como la comida, se hayan encarecido notablemente.

2.3.2. Microentorno

2.3.2.1. Competidores

Tanto en España como en el mundo, el desarrollo software es un campo solicitado y, a su vez, tiene una oferta escasa. Aun así, existen distintas empresas grandes que se encargan de dar este servicio a otras grandes empresas, siendo el presupuesto de estas mucho mayor que las PYME. Aunque existen empresas que sí proporcionan estos servicios a un presupuesto más acertado para las PYME, creo que existe una ventana muy interesante y amplia para explotar. Además, esta situación creo que es perfecta dado la situación de mi empresa, una primeriza y con poca experiencia, por lo que mis precios se ajustan mucho mejor a los pedidos de las PYME.

2.3.2.2. Clientes

Como he comentado ligeramente en el apartado anterior, mi idea es centrar mi servicio en las pequeñas y medianas empresas. Dado que soy un empresario primerizo y con nula experiencia en el ámbito laboral, concretamente en el desarrollo software, mis precios y mis servicios no serán ni tan solicitados ni tendrán una calidad tan extraordinaria como la que puede otorgar una empresa grande.

Conforme pase el tiempo y gane experiencia, tanto la calidad como el precio subirán poco a poco y, además de ofrecer mis servicios a PYME, podré contratar un equipo mejor con el que podré dar servicios a grandes empresas.

2.3.2.3. Proveedores

Antes de hablar sobre los proveedores, para dar contexto, el local será mi propia vivienda, por lo que los proveedores serán los necesarios para vivir en ella.

A continuación, voy a mostrar una tabla con las distintas necesidades que posee mi negocio.

Necesidades	Proveedores
Luz	Iberdrola
Gas	Endesa
Agua	Aguas de Alicante
Comida	Distintos supermercados (Mercadona, Carrefour, Dia...)
Telefonía e Internet	Jazztel
Servidores	Amazon Web Services
Software	Microsoft, Google, etc.*

*Este apartado puede cambiar debido a que mi empresa estará en constante aprendizaje, por lo que los distribuidores de los nuevos lenguajes de programación cambiarán

2.3.2.4. Productos sustitutivos

En mi empresa, realmente no existen productos sustitutivos como tal, dado que creamos un producto según el cliente quiere. La única parte que podría nombrarse como producto sustitutivo es a la accesibilidad de la aplicación, la cual depende de cada cliente. Con la accesibilidad, podemos ayudar a que gente que tiene ciertos problemas, como puede ser daltonismo o deficiencia visual, para que utilicen la aplicación sin complicaciones.

2.3.3. Ubicación del negocio y plano del local

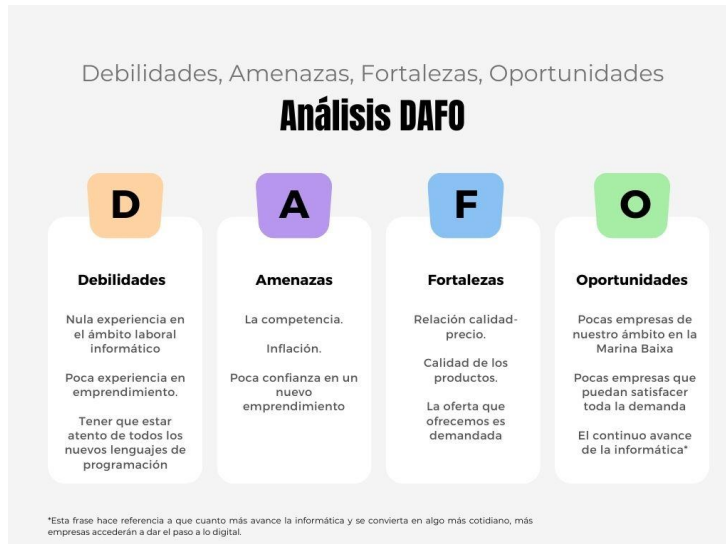
Mi negocio está en el centro de Benidorm, concretamente en la Avinguda Rei Jaume I. Es una zona muy buena dado que, al ser en el centro, puedo visitar a cualquier cliente sin problema. Además, aunque viviese a las afueras, es un trabajo que puede realizarse telemáticamente sin problema.

El plano de mi oficina será el siguiente.



2.3.4. Análisis DAFO

El análisis DAFO es imprescindible a la hora de crear una empresa para observar, detenidamente, sus inconvenientes (debilidades y amenazas) y sus ventajas (amenazas y fortalezas).



2.4. Plan de marketing

2.4.1. Producto

El producto, o más bien servicio, es la creación de aplicaciones para distintos dispositivos con los lenguajes de programación más actuales. Este servicio se ajusta a los requerimientos que el cliente quiera, por lo que el diseño y la funcionalidad de la aplicación depende del cliente. A su vez, también dependerá del cliente y de la situación actual el lenguaje de programación en el cual se desarrollará la aplicación.

2.4.2. Precio

El precio también será algo variable según el tamaño del proyecto y su complicidad. Actualmente, un proyecto de 6 meses en una situación normal tendría un coste de 12000€ aproximadamente. Este cálculo está realizado mediante un cálculo aproximado de lo que podría costar un sueldo mensual de un desarrollador software con un añadido en el caso de que el proyecto acabe antes de tiempo o se demore un poco. Además, se realiza un primer abono del coste total, entre un 25% y un 50%. Esta medida es así para asegurar que el cliente no cancele dicho proyecto durante el proceso del mismo y dejar al empresario, en este caso mi persona, tirado y un ingreso por ese esfuerzo.

2.4.3. Promoción

Muchas empresas están buscando desarrolladores para que creen una aplicación para el uso de sus clientes o de la propia empresa. Mi manera de llegar a estas empresas va a ser de distintas formas. La primera de ellas será mediante las redes sociales, como Instagram o Facebook. En Instagram, publicaré proyectos realizados en la empresa para que futuras empresas observen el trabajo que se ha realizado y, si les gusta, se lancen a pedirnos un producto. Por otro lado, en Facebook haremos algo semejante, pero añadiendo que contactaremos con las empresas. Muchas empresas tienen Facebook en la que publican distinta información del negocio. A partir de esta información, la analizaré y contactaré con ellos para sugerir un tipo de aplicación.

Por otro lado, añadiremos nuestra empresa a las páginas amarillas, de esta manera una multitud de empresas podrán observar nuestra información y podrán contactar con nosotros.

Estas medidas, al ser sencillas y qué, según mi parecer, puede hacerlo una única persona, yo seré el encargado de todas las redes sociales y el mantenimiento de las mismas, además de añadir a la empresa en las páginas amarillas, por lo que el costo de todo esto es nulo.

2.4.4. Place/distribución

Nuestra vía para poder administrar nuestros servicios a nuestros clientes es íntegramente digital. Tanto en las páginas amarillas, como en redes sociales, tendremos toda nuestra información y demostraciones de nuestros servicios para que las empresas que vean nuestros perfiles puedan contactar con nosotros para un servicio personalizado.

2.5. Plan de producción y recursos humanos

2.5.1. Inversiones y gastos

A la hora de iniciar una actividad emprendedora siempre hace falta un capital inicial. En este proyecto, tenemos un total de 5000€ ahorrados que nos ayudarán en la creación de la S.L. y con los gastos de los mismos. A continuación, analizamos los gastos y las inversiones de nuestro proyecto para saber diferenciarlos y tener un recuento del capital que gastamos. En la siguiente tabla diferenciamos entre inversiones y gastos.

Inversiones	Gastos
Hardware (2.400€)	Alquiler (6.000€/a)
	Luz (960 €/a)
	Comida (1440€/a)
	Agua (240 €/a)
	Gas(600€/a)
	Telefonía e Internet (480 €/a)
	Seguro (1.000 €/a)
	Comunidad (Se encarga el casero)
	Gastos de constitución (500 €)
	Sueldo propio (18.000€/a)
	SS administrador (960 €/a)
	Servidores (100€/a)
	Licencias software(100€/a)
TOTAL: 2400 €	TOTAL: 30280€
TOTAL: 32680€	

2.5.2. Origen del financiamiento

Como hemos comentado en apartados previos, el origen de nuestro financiamiento inicial es, en su totalidad, propio. Gracias a mis propios ahorros que he ido consiguiendo con trabajos temporales, tenemos la libertad de no pedir un préstamo bancario para comenzar nuestra actividad.

2.5.3. Ayudas y subvenciones

Para el comienzo de todas las empresas, las ayudas son más que importantes. Hay de todo tipo de ayudas, como la ayuda para jóvenes emprendedores o para mujeres emprendedoras. En el caso de nuestra empresa, hemos decidido obtener la bonificación en la SS para las altas iniciales. Esta ayuda lo que nos otorga es una cuota mensual de 80€ en el apartado de SS, lo que supone una reducción más que notable del total que deberíamos pagar.

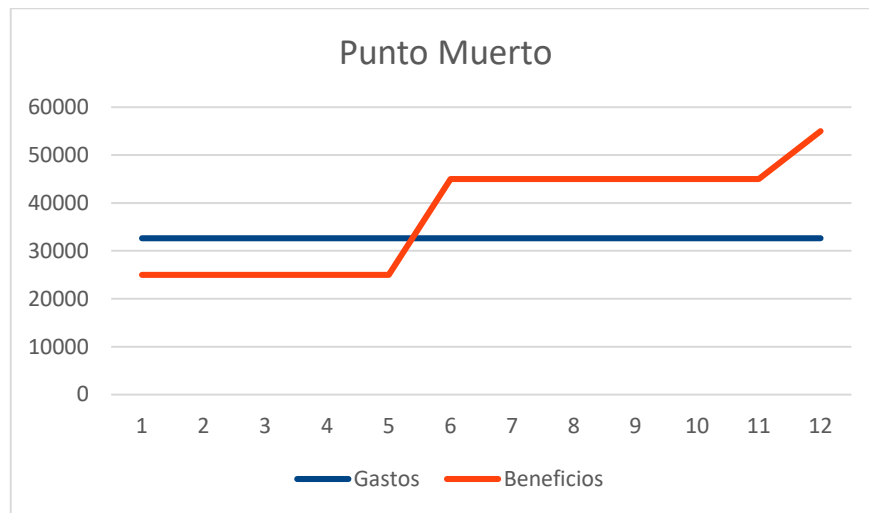
2.5.4. Distribución de costos

A continuación, debemos diferenciar entre gastos fijos y variables.

FIJOS	VARIABLES
Alquiler	Luz
Telefonía e Internet	Agua
Seguro	Comida
Gasto de constitución	Gas
Sueldo propio	
SS propio	
Servidores	
Licencias	

2.5.5. Umbral de rentabilidad o punto muerto

Antes de ver el umbral de rentabilidad, debemos saber qué es exactamente. El umbral de rentabilidad o punto muerto representa el momento en el que existe el equilibrio entre los ingresos y gastos totales, es decir, el punto en el que la empresa no pierde dinero, pero tampoco gana. El umbral de rentabilidad de nuestra empresa es el siguiente.



Como se puede observar en el anterior gráfico, nuestro punto muerto se encuentra entre el quinto y sexto mes, debido a que en esa fecha se realiza el segundo pago del proyecto. A partir del sexto mes, todos los ingresos pasan a ser ingresos netos.

2.6. Trámites de constitución, puesta en marcha e impuestos

INSTITUCIÓN	TRÁMITES	IMPUESTOS/TASAS
Ayuntamiento	Licencia de actividades e instalaciones	Gratuito
	Licencia de apertura	Depende del ayuntamiento
Hacienda	Solicitud NIF temporal	Gratuito
	ITPAJD → Hacienda autonómica	Gratuito
	NIF definitivo/Inicio habitual de actividades	Gratuito
	Declaración censal	Gratuito
	Alta en el IAE	Gratuito solamente el primer año
TGSS	Alta al representate/s en el régimen correspondiente de la SS	Gratuito

	Código de Cuenta de Cotización de la empresa	Gratuito
Registros oficiales	Certificación negativa del nombre en el Registro Mercantil	14 €
	Legalización de los libros de cuentas anuales	Gratuito
	Inscripción en el RGPD	Gratuito
Notaría	Estatutos + Escritura constitución	462 €
Otros	Certificación del cargo de administrador o certificado digital => FNMT	24 €

3. Análisis

En el siguiente apartado nos centraremos en el análisis previo que ha de hacerse siempre a la hora de desarrollar un proyecto.

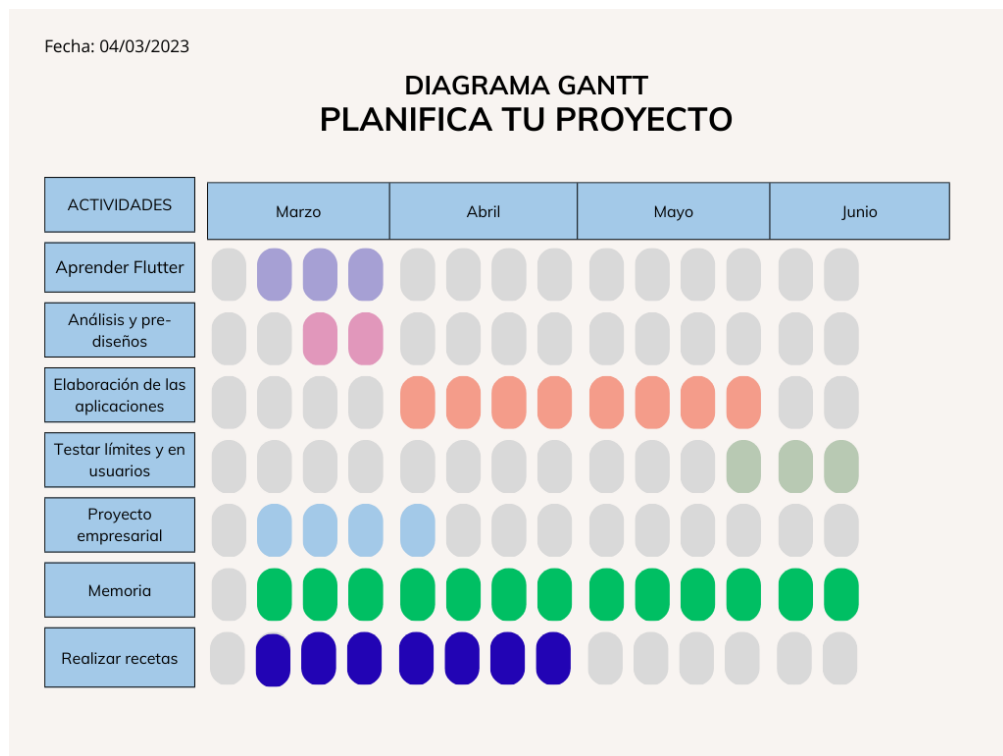
A continuación, desarrollaremos distintos puntos, desde una planificación inicial, con el Diagrama de Gantt, hasta casos de usos y diagrama de clases.

3.1. Planificación temporal

La planificación temporal es un apartado más que importante a la hora de desarrollar un proyecto de cualquier tipo. De esta manera, podemos tener un control sobre las acciones que realizamos durante el mismo. Además, realizar esta planificación nos puede ayudar en futuros proyecto a la hora de organizarlo mejor en el tiempo.

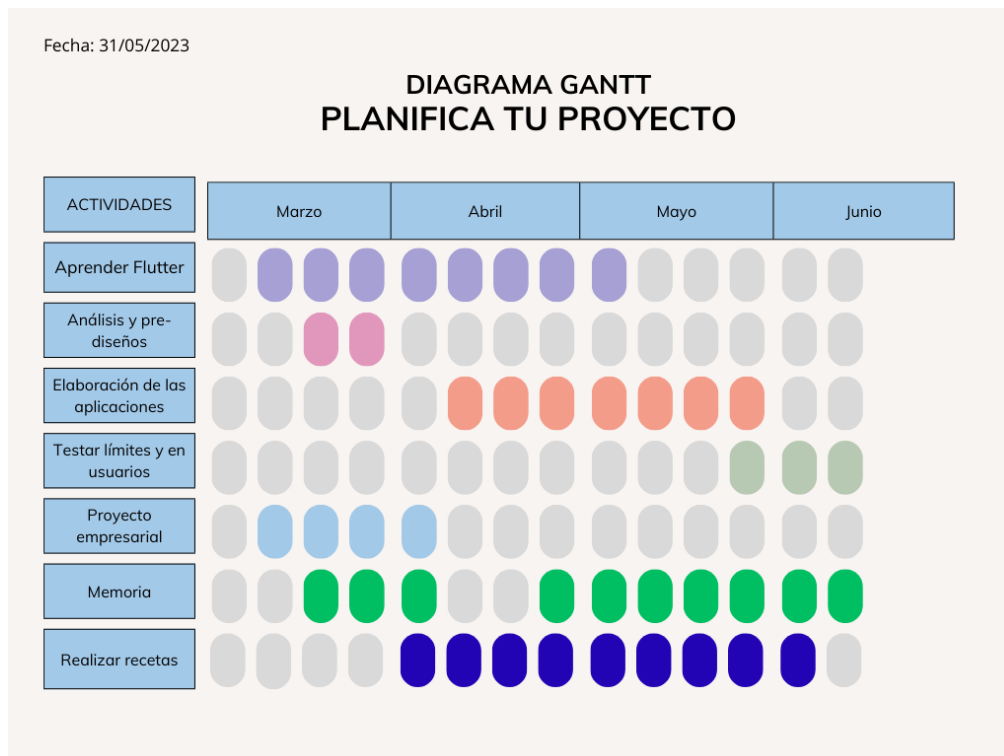
Uno de los procedimientos más conocidos y utilizados en este ámbito es el “Diagrama de Gantt”. Este diagrama te proporciona una vista general de las tareas programadas. Además, muestra datos de interés como la fecha de inicio y final del proyecto, en cuántas tareas está dividido el proyecto, una estimación aproximada de cuánto tiempo se llevará a cabo en cada tarea y la manera en la que las tareas están relacionadas entre sí.

A continuación, mostraré el diagrama de Gantt realizado al comienzo de este



proyecto.

Tras finalizar el proyecto, he realizado otro diagrama de *Gantt* para poder comparar las expectativas, mostradas en el diagrama anterior, con el resultado real y final, que es el siguiente diagrama.



Se pueden observar ciertas diferencias entre diagramas. La primera diferencia, y la más grande, es sobre el aprendizaje de *Flutter*. En un principio, el aprendizaje de *Flutter* fue rodado, pero conforme pasó el tiempo fue más tranquilo.

La segunda diferencia, prácticamente igual de notable que la anterior, es la realización de las recetas del menú, las cuáles han sido movidos en el tiempo, dado que he tardado algo más de lo esperado tanto realizando las recetas como pensando el menú.

Por último, pero no más importante, ha habido ligeros cambios tanto en la memoria, con la que no he sido del todo constante, y la elaboración de las aplicaciones, que he tardado algo menos de lo esperado.

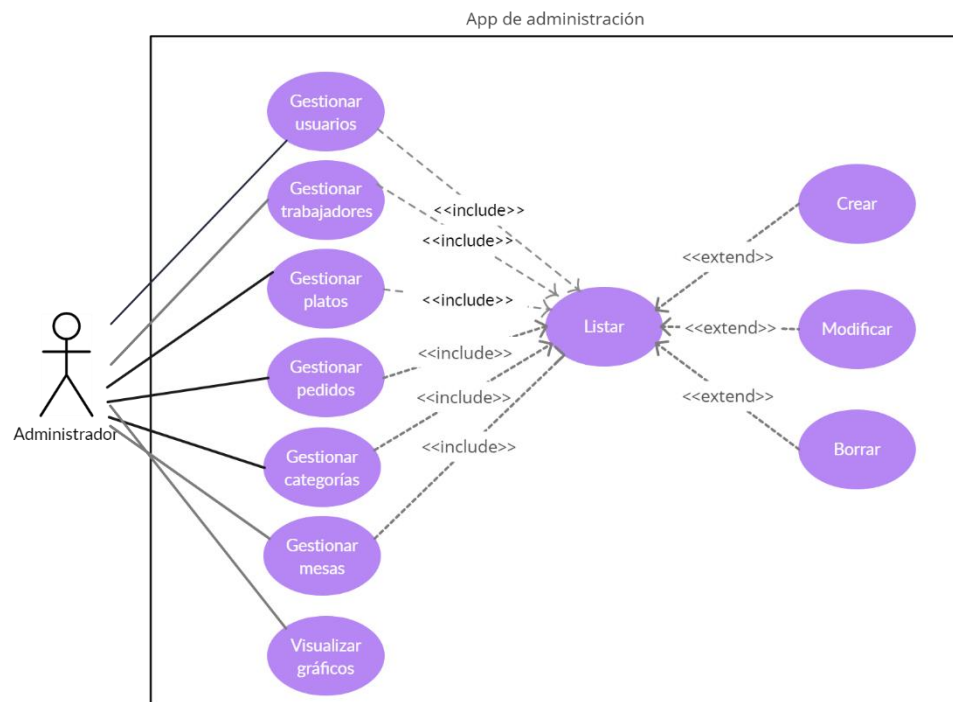
3.2. Casos de uso

Antes que nada, se debe definir correctamente que es un caso de uso. Como su nombre indica, un caso de uso es una secuencia de acciones que debe realizar alguien o algo, llamado actor, para llevar a cabo un proceso.

Como en cualquier aplicación, antes de su desarrollo se deben crear distintos casos de usos ajustado al, valga la redundancia, su uso. Por ello, he realizado los distintos casos de uso para las aplicaciones que desarrollaré.

Para comenzar, desarrollaremos los casos de uso de la aplicación orientada en el uso administrativo.

3.2.1. Aplicación administrativa



3.2.1.1. Caso de uso – Gestionar recursos del restaurante

Antes de empezar en sí con este caso de uso, agruparemos las tres acciones que puede realizar el usuario administrador en nuestra aplicación porque realmente es la misma acción, únicamente cambia la entidad a la que hace efecto.

Descripción: El administrador desea gestionar una entidad, ya sea para crear, borrar o modificar.

Actor: Administrador.

Precondiciones: El usuario sea ha registrado y tiene el rol “Administrador”.

Curso normal de la acción:

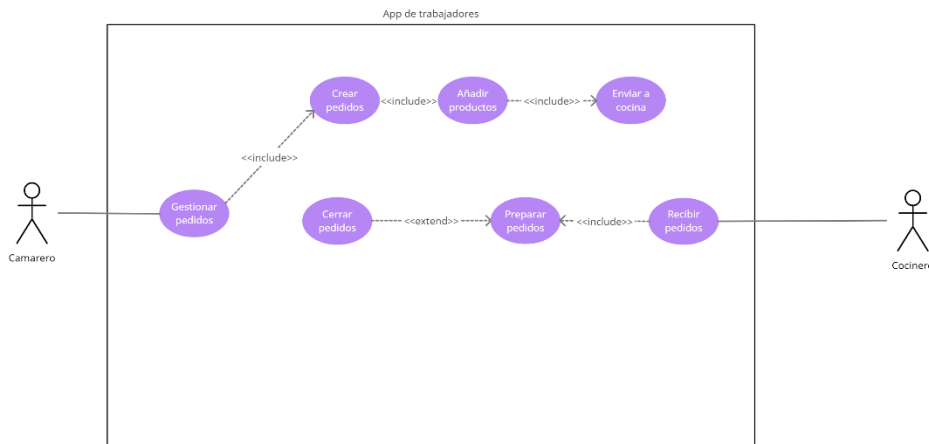
1. El administrador accede a la pestaña correspondiente a la entidad seleccionada.

2. La aplicación lista todas las entidades de la base de datos y el administrador elige una opción (Crear, Borrar o Modificar).
3. Si elige *Crear*, existirán los siguientes sucesos:
 - 3.1. El administrador introduce los datos de la nueva entidad.
 - 3.2. El sistema comprueba si los datos son correctos.
 - 3.3. Ingresa la nueva entidad en la base de datos.
4. Si elige *Modificar*, ocurrirá lo siguiente:
 - 4.1. El administrador introduce los nuevos datos.
 - 4.2. El sistema comprueba si los datos son correctos.
 - 4.3. Actualiza la entidad en la base de datos.
5. Si elige *Borrar*, los sucesos serán los siguientes:
 - 5.1. El sistema comprueba que esa entidad exista actualmente
 - 5.2. El sistema elimina toda la información relacionada con esa entidad en concreto.

Errores/Alternativas:

1. El usuario elige la opción “Visualizar gráficos”, la cuál muestra unos gráficos sobre los platos, sin opción a poder realizar alguna acción *CRUD*.
3. El sistema muestra, con una ventana emergente, que los datos son incorrectos, ya sea porque no cumplen un formato o porque ya están en uso.
4. El sistema muestra, con una ventana emergente, que los datos son incorrectos, ya sea porque no cumplen un formato o porque ya están en uso.
5. Algún dato no se borra correctamente.

3.2.2. Aplicación trabajadores



3.2.2.1. Caso de uso – Camarero

Descripción: El camarero gestiona los pedidos.

Actor: Camarero

Precondiciones: El usuario tiene que haberse registrado y tener el rol de “Camarero”

Curso normal de la acción:

1. El camarero gestiona los pedidos
2. El camarero crea un pedido
3. El camarero añade productos según quiera el cliente
4. Por último, el camarero envía el pedido a cocina.

Errores/Alternativas: Ninguna

3.2.2.2. Caso de uso – Cocinero

Descripción: El cocinero recibe y prepara pedidos.

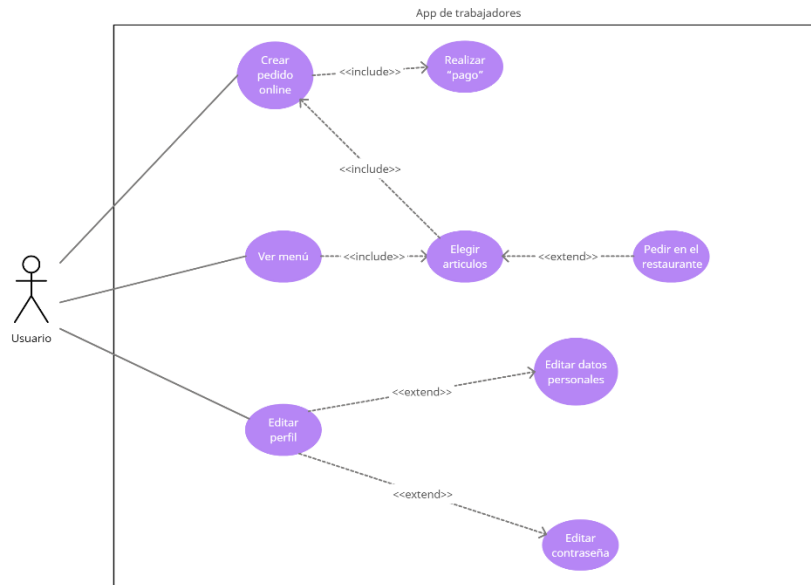
Actor: Cocinero.

Curso normal de la acción:

1. El cocinero recibe pedidos.
2. Tras esto, los prepara y tacha los pedidos según los va realizando.
3. Cuando el pedido está acabado, lo finaliza.

Alternativas/Errores: Ninguna.

3.2.3. Aplicación clientes



3.2.3.1. Caso de uso – Usuario- Crear pedido online

Descripción: El usuario crea un pedido online

Actor: Usuario

Precondiciones: El usuario debe de estar registrado y con una sesión iniciada.

Curso normal de la acción:

1. El usuario empieza a crear el pedido
2. Elige artículos para su pedido
3. Paga dicho pedido y se envía a su domicilio.

Alternativas/Errores:

3.2.3.2. Caso de uso – Usuario- Ver menú

Descripción: El usuario observa el menú.

Actor: Usuario

Precondiciones: El usuario debe de estar registrado con el rol de “Usuario”

Curso normal de la acción:

1. El usuario observa el menú
2. A continuación, elige los productos que desea
3. Le comunica los productos que desea al camarero

Errores/Alternativas:

3. En el caso de comunicarle los productos al camarero, ese el camarero el que crea el pedido y añade los productos

3.2.3.3. Caso de uso – Usuario- Editar perfil

Descripción: El usuario desea cambiar sus datos personales.

Actor: Usuario.

Precondiciones: El usuario debe de estar registrado con el rol de “Usuario”.

Curso normal de la acción:

1. El usuario selecciona su perfil.
2. Elige entre cambiar sus datos personales o la contraseña.
3. Depende de lo que seleccione, podrá realizar los siguiente:
 - 3.1. Si elige datos personales, podrá cambiar datos como el correo, el teléfono móvil o su nombre y apellidos.
 - 3.2. Si elige la contraseña, podrá cambiar la contraseña.

Alternativas/Errores:

Los datos ya están en uso o son incorrectos.

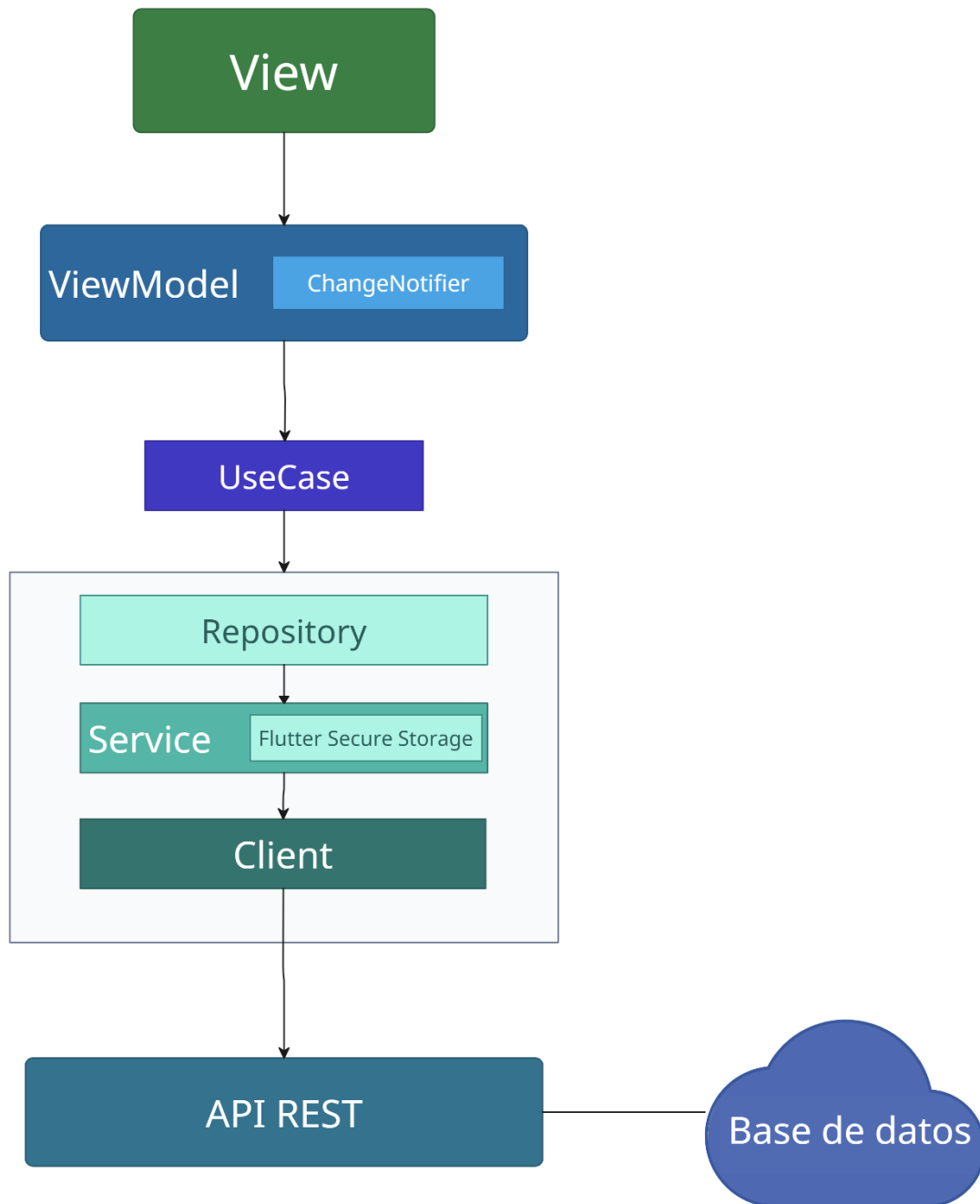
3.3. Diagrama de clases

En este apartado se debería visualizar un diagrama de clases común y corriente, viendo las relaciones entre clases y sus propiedades. En cambio, Flutter es especial en este sentido, dado que, si queremos tener un código legible y algo estructurado, debemos separar los componentes y, al separarlos, debemos crear una nueva clase. En una aplicación pequeña, puede llegar a ser algo viable, pero para un proyecto mediano como este, se crea un modelo de clases absurdamente grande.

Además, la estructura que hemos utilizado en este proyecto se podría decir que es por módulos, imitando la estructura MVVM y, que, en un futuro cercano, será cambiada para que sea lo más óptima posible. Al trabajar con una estructura dividida en módulos, la estructura que tiene todos los módulos es igual, lo cuál puede llegar a ser un contratiempo, pero está hecho de esta manera para qué, a la hora de exportarlos y querer utilizarlos en otro proyecto, sea lo más sencillo y rápido posible.

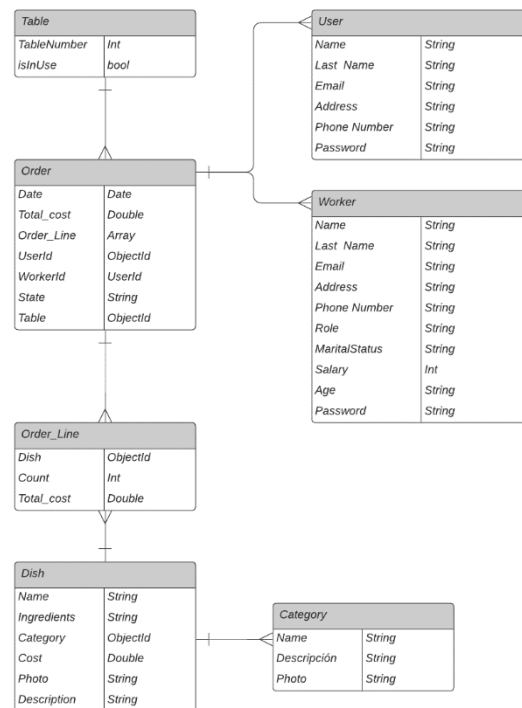
Se podría pensar que se debería crear un diagrama de clases de todas formas, con la estructura básica de todos los módulos para que, de alguna manera, se pueda ver un diagrama claro. Este punto también ha sido pensado, pero según mi criterio no es una opción viable. La razón más clara de rechazar esta idea es que, aunque todos tenga una estructura similar, a su vez todos son distintos. No se puede simular una estructura básica porque cada módulo tiene sus funciones, sus casos de uso, etc., faltando así muchos datos y archivos (o, mejor dicho, clases), por lo tanto, faltaría información para entender correctamente el diagrama de clases.

Para concluir este apartado, me gustaría añadir, al menos, un diagrama de la estructura que he utilizado en el proyecto, basada en una ya aprendida y aplicada en Android.



3.4. Modelo relacional

Para finalizar el apartado del análisis del proyecto, tenemos el modelo relacional de la base de datos. Como podemos observar a la derecha del documento, vemos las colecciones de la base de datos junto a sus campos y los tipos. En cada documento, además, hay un campo obligatorio en todas las colecciones, que es el id del documento, generado de manera automática por MongoDB. Sabiendo esto, las referencias que existen de otras colecciones, como puede ser en la colección *order*, son del tipo *ObjectId*.



4. Entorno de programación

En el punto 4 de esta memoria se va a exponer y explicar los distintos lenguajes y librerías que se van a emplear en el desarrollo de todo el software. Además, hablaremos ligeramente del control de versiones y cómo lo hemos empleado.

4.1. Lenguajes y tecnologías que vamos a utilizar

4.1.1. Base de datos

La base de datos elegida para este proyecto ha sido *MongoDB*, una base de datos no relacional. En un principio, existían dudas sobre qué tipo de base de datos se adaptaría mejor a este tipo de proyecto.

Por un lado, una base de datos relacional nos proporcionaba una estructura rígida que nos permite ser menos vulnerable a fallos. La parte negativa de utilizar este tipo de base de datos es que no tenía, a comienzo de dicho proyecto, conocimientos suficientes para enlazar el proyecto de backend, del cual se hablará más adelante, con la base de datos.



Por otro lado, las bases de datos no relacionales nos permiten tener más flexibilidad a la hora de albergar datos y que, al contrario que con las bases de datos relacionales, sí tengo conocimientos para enlazar el backend, desarrollado en *Node.js*, y una base de datos como *MongoDB*.

MongoDB tiene distintas diferencias frente a las bases de datos relacionales como *MySQL*. Esta base de datos, y todas las bases de datos no relacionales, se divide en “colecciones”, las cuales son equivalentes a las tablas en base de datos relaciones. Cada una de estas tablas, tiene “documentos” los cuales albergan atributos con sus respectivos datos. Al contrario que en *SQL*, en *MongoDB* generalmente no hay referencias a otras tablas, el propio documento suele albergar estos datos, por lo que para aplicaciones donde sea necesario guardar datos de manera “masiva” y sin restricciones es perfecta.



Al analizar todos estos pros y contras de ambos tipos de bases de datos, mi proyecto puede utilizar cualquiera de las dos bases de datos, incluso ambas. La razón que inclinó la balanza a favor de *MongoDB* es la experiencia que tenía desarrollando APIs en *Node.js* y *MongoDB* y la facilidad que te otorga desarrollar APIs en *Node.js* junto a *Express*, un entorno que se explicará en la próxima sección.

4.1.2. Backend

Para comenzar esta sección, debemos saber diferenciar entre *frontend*, que definiremos más adelante, y el *backend*. El *backend* de una aplicación es la parte no visible, la cual se encarga de gestionar los pedidos, confirmaciones de pago, etc. Esta parte, además, se encarga de la conexión del *frontend* y sus acciones con la base de datos para qué, de manera unitaria, todas las aplicaciones desplegadas tengan los mismos datos.

Este *backend* está desarrollado en *Node.js*, un entorno basado en *JavaScript*, y *Express*, un entorno enfocado en el desarrollo de aplicaciones web o APIs, el cuál es nuestro caso.

A continuación, se desarrollará de manera más concreta tanto los entornos anteriormente nombrados, *Node.js* y *Express*, como las librerías empleadas en el desarrollo de este *backend*.

4.1.2.1. Node.js y Express

Node.js es un entorno de ejecución orientado a eventos asíncronos, el cual está enfocado para crear aplicaciones web escalables. Una de las características más destacables de este entorno es su enfoque en el modelo E/S sin bloqueo. Además, *Node.js* utiliza un único hilo de ejecución que maneja solicitudes de manera paralela. Por último, también cuenta con un sistema de gestión de paquetes, llamada *npm*, que facilita la instalación e incorporación de nuevos módulos y librerías a nuestro proyecto. De hecho, debemos utilizar este sistema de gestión para instalar *Express* y las demás librerías que explicaremos más adelante.

Express, abreviación de *Express.js*, es un *framework* el cuál se utiliza para el desarrollo de APIs de manera eficiente, flexible y modular. Permite a los desarrolladores crear APIs de manera rápida y sin una estructura rígida. Además, simplifica algunas tareas como el enrutamiento, gestión de sesiones, etc. y permite una alta compatibilidad con otros módulos y librerías de *Node.js*.

Tras explicar los dos pilares de nuestro backend, entraremos más a fondo sobre las librerías utilizadas y qué funcionalidades tiene.

4.1.2.2. Librerías utilizadas

El siguiente apartado trata de analizar las librerías que he utilizado en este proyecto. Las librerías más destacables serían *bcrypt*, *mongoose*, *jwt-simple* y *socket.io*, dado la importancia que tienen en el proyecto, pero se analizarán todas las librerías.

La primera de ellas, y la más importante de nuestro backend, es *mongoose*. Esta librería, como parte de su nombre indica, permite la conexión entre el backend y nuestra base de datos. *Mongoose* es una librería esencial en este proyecto dado que, sin ella, no existiría una

uniformidad en todas nuestras aplicaciones de manera dinámica. Además de permitirnos extraer e introducir documentos en nuestras distintas colecciones, nos facilita mucho el trabajo a la hora de hacer algunas peticiones más complejas.

Otra librería muy importante para el desarrollo del proyecto es *jwt-simple*, una librería que nos añade una capa de seguridad más que necesaria, dado que nos permite codificar y decodificar los *json web token*, los cuáles nos permite autorizar y denegar el acceso a nuestros datos. Esta librería, junto a la siguiente, son muy necesarias si queremos un *backend* seguro.



La siguiente librería, junto con la anterior, son las dos más importantes si hablamos de seguridad. En este caso, hablamos de *bcrypt*, una librería que nos permite encriptar datos sensibles para albergarlos en la base de datos, como puede ser la contraseña del usuario. De esta manera, si atacan nuestra base de datos, algo tan importante como es la contraseña no puede ser robada.

La última librería a destacar de nuestro backend es *socket.io*. Esta librería, en un principio, no iba a incluirse en el proyecto, pero al final es necesaria. *Socket.io* nos permite crear una conexión constante con un cliente mediante eventos, por lo que se puede mantener una conexión a tiempo real con los datos. Este socket lo utilizamos a la hora de la creación de pedidos para que, sin que el usuario tenga que realizar ninguna acción, se actualice el estado del pedido y para que los cocineros reciban los pedidos al momento.



Tras comentar las cuatro librerías más importantes de nuestra API, toca explicar las demás, que no son tan importantes, pero son necesarias. Son las siguientes:

- *DotEnv*. Esta primera librería nos permite tener variables de entorno, las cuales nos permiten guardar datos sensibles, como puede ser la dirección IP de la base de datos o el token secreto.
- *Nodemon*. Esta librería se podría catalogar como la prescindible, dado que únicamente nos ayuda a la hora de desarrollar a la API, permitiéndonos usar el *hot reload* para aplicar los cambios sin tener que reiniciar el programa.
- *Moment*. *Moment* es una librería que nos ayuda a trabajar con fechas, como puede ser validándolas o transformándolas.
- *Express-validator*. Esta es una de las más importantes de estas cuatro librerías. Nos permite añadir un middleware entre la autenticación, gestionada con *jwt-simple*, y la función principal de la petición con la cual podemos crear validaciones para rechazar o aceptar dichas peticiones.

Ya explicado toda la parte del *backend*, pasamos a explicar la parte más grande del proyecto, el *frontend*.

4.1.3. Frontend

El *frontend* es la capa con la que el usuario interactúa y todo lo que ello conlleva, como son los diseños, funcionalidad que se encarga de la interactividad con los usuarios, etc.

En el caso del proyecto *Sei Vegan, Sei Grün*, el *frontend* está desarrollado únicamente en el *framework Flutter*, con base del lenguaje de programación *Dart*, los cuales van a ser explicados en el siguiente punto.

4.1.3.1. Dart

Dart es un lenguaje de programación de código abierto desarrollado por *Google*, publicado por primera vez en 2011. Este lenguaje, en sus inicios, era una alternativa para JavaScript, siendo Dart un lenguaje orientado a objetos (POO) y de tipado estático, un estilo parecido a Java. Actualmente, Dart puede ser utilizado en distintos ámbitos, como aplicaciones web, aplicaciones multiplataforma o servidores.



Dart, como hemos comentado de manera leve anteriormente, es un lenguaje de programación con similitudes con *JavaScript*, *Java* o *C++*. Por lo tanto, al tener esta base tan similar con lenguajes tan conocidos, a los nuevos estudiantes, en los que me incluyo, se les hace muy sencillo aprender de él.

4.1.3.2. Flutter

Como hemos nombrado anteriormente, vamos a utilizar el *framework Flutter*, basado en un único lenguaje, *Dart*. A diferencia de otros *frameworks* de otros lenguajes o del propio Dart, Flutter compila a código nativo, consiguiendo así un rendimiento mayor que en otras aplicaciones basadas en *web-views*.



El mayor potencial, y unas de las razones por las que se ha elegido este SDK, es la facilidad en la que se pueden crear aplicaciones en distintas plataformas con un único código. Flutter permite crear aplicaciones nativas para *Android*, *iOS*, aplicaciones web y aplicaciones en *Windows*, *Linux* y *MacOS*. Además, también tiene integrado el *Hot Reload*, una característica muy útil a la hora de desarrollar. A continuación, indagaremos en características de *Flutter*, y a su vez de *Dart*, y veremos qué ventajas tiene.

La primera ventaja de la cual vamos a hablar es sobre su rápido desarrollo. Como hemos avanzado anteriormente, *Dart* compila en lenguaje 100% nativo, por lo que su velocidad de compilado es una de las más óptimas actualmente. Además de todo esto, tiene implementado

la función *Stateful Hot Reload*, que te permite ver, de forma instantánea, los cambios realizados con la aplicación corriendo.

Otra de las ventajas que tiene *Flutter* es tener una amplia librería de widgets. Los widgets son los componentes que conforman nuestra aplicación. Estos componentes, mediante clases y *builders*, pueden hacerse reutilizables y personalizables, algo muy interesante a la hora de optimizar y ahorrar código. Algunos de los *widgets* más básicos pueden ser:

- *Text* ➡ Nos permite ver texto
- *AppBar/TabBar* ➡ Son la parte superior e inferior de la aplicación.
- *Button* ➡ Componente que, al dar click, realiza una función.

Los *widgets*, además, tienen dos tipos: Con estado y sin estado. El estado es la información que se puede utilizar una vez se crea el *widget* y que cambie durante su vida útil. Dicho estado puede ser administrado/controlado de distintas maneras, como puede ser con un *Stateful Widget* o con gestores de estado, como son *Provider* o *Bloc*. A continuación, vamos a hablar de los más conocidos.

4.1.3.3. Provider

Provider es, posiblemente, el gestor de estado más conocido de *Flutter* junto a *Bloc*. Este gestor es muy fácil de implementar, lo cual no significa que sea incompleto o perfecto para aplicaciones pequeñas. *Provider* no tiene nada que envidiarles a gestores como *Bloc*, *GetX* o *Cubic*. Este gestor sirve tanto para aplicaciones pequeñas como medianas/grandes, aunque para aplicaciones grandes es más recomendado *Bloc*.

Como gestor de estado, nos permite administrar el estado de las pantallas y widgets de nuestra pantalla. Además, podemos crearlo nada más se ejecuta la aplicación, por lo que tendremos acceso a la instancia de nuestro provider en cualquier lugar de nuestro árbol de widgets. Esto, además de proporcionarnos mucha facilidad a la hora de centralizar nuestros datos, nos permite acceder a datos entre páginas sin perder el estado de las mismas. En mi opinión, y viendo la magnitud del proyecto, *Provider* es el gestor de estados perfecto para ello, aunque nunca se debe descartar el cambio hacia *Bloc*, otro gestor de estado la mar de interesante. A continuación, hablaremos de él.

4.1.3.4. Bloc

Anteriormente, hemos nombrado *Bloc* como un gestor de estado, pero se puede nombrar a *Bloc* como una parte del patrón de *Clean Architecture*. Si se tiene conocimientos de algún tipo de *Clean Architecture*, *Bloc* es algo parecido a *MVVM*. El concepto principal es que

exista una capa entre el modelo y la vista, justamente una de los conceptos que sustenta *MVVM*. Además, en esta capa se gestionarán, además de los datos, los estados de los widgets.

El patrón *Bloc* se puede dividir en cuatro importantes partes; *Events*, *States*, *Bloc* y *UI*. Del primero que vamos a hablar es de los estados, dado que es similar al estado de los widgets. Un estado en el patrón *Bloc* almacena las condiciones en las que se encuentra nuestra pantalla o aplicación, al igual que en los widgets guarda las condiciones de los mismos. Esto nos puede ayudar a la hora de mostrar un contenido u otro dependiendo del estado de la aplicación, como puede ser el mostrar un texto si ha saltado un error o una barra de carga mientras se obtienen los datos desde la API.

Otro de las partes de *Bloc* que va de la mano con los estados son los eventos. Estos eventos suelen ser lanzados cuando la pantalla o aplicación tienen un estado en concreto. Dicho de otra manera, y para enlazar con el apartado anterior, cuando un estado se establece en la pantalla, este puede lanzar un evento que ejecutará una acción. Poniendo por caso el ejemplo anteriormente nombrado sobre los errores, la pantalla debería cambiar de estado y, nada más lanzar dicho estado, se ejecutará un evento que está ligado a este estado.

Como tercer componente del patrón *Bloc*, tenemos el propio *Bloc*. Este componente es el encargado de manejar los estados de la aplicación y sus eventos, o, dicho de otra manera, es el gestor de estados de la aplicación. Dicho componente es la unión, por decirlo de alguna manera, entre los estados y los eventos y el usuario.

Para este proyecto, concretamente para la aplicación de los clientes, *Bloc* como gestor de estados hubiese sido perfecto, dado que se una aplicación mediana en la que la implementación de dicho gestor hace que el desarrollo de la misma sea más sencillo. Además, si a este patrón le añadimos la librería *injector*, librería que nos permite inyectar dependencias, obtenemos un gestor maravilloso. Aun teniendo estas ventajas, descubrí demasiado tarde este gestor de estados y esta librería, las cuales, y como he dicho antes, ahorran mucho trabajo. De todas formas, desarrollar un proyecto como este con un gestor de estados como *Provider* no ha resultado una aventura extremadamente difícil.

4.1.3.5. Librerías utilizadas

Tras comentar y analizar los gestores de estado que se podrían haber utilizado en este proyecto, toca dar paso a analizar y explicar qué librerías hemos utilizado en el proyecto y el porqué. Antes de comenzar a nombrarlas, me gustaría aclarar que no todas las librerías están en todas las aplicaciones, la siguiente lista es un conjunto de todas las librerías usadas.

La primera librería de la que vamos a hablar es una alternativa a las *snackbar* que vienen por defecto en *Flutter*. Dicha librería se llama *another_flushbar* y nos permite notificar al usuario, de una manera más fluida y personalizable que si utilizamos las *snackbar* o *toast*. De todas maneras, he utilizado tanto las que vienen por defecto como las de este paquete para no tener únicamente de un tipo y poder cambiar dependiendo del usuario final.

La siguiente librería, llamada *auto_size_text*, nos ayuda a la hora de crear una aplicación *responsive*. Este paquete nos proporciona unas características que con un *widget* de texto nativo no podríamos. Por ejemplo, nos permite ajustar un texto a un número de líneas y se irá adaptando de manera dinámica a esta restricción. Además, si el texto sobrepasa estos límites, podemos añadir un *widget* para sustituir a dicho texto. En este proyecto, esta librería se ha utilizado junto a *marquee*, una librería que permite a texto desplazarse de manera infinita, obteniendo un efecto muy bueno.

Este tercer paquete es exclusivo de las plataformas de escritorio, como son *Linux*, *MacOS* o *Windows*. Dicho paquete es *bitsdojo_window* y nos permite añadir y personalizar una barra superior y añadirle los botones deseados, como el de minimizar, maximizar y cerrar. Esto nos ayuda en el diseño de la aplicación, dado que la barra superior que viene por defecto con *Flutter* no es muy agradable a la vista.

Los dos siguientes paquetes están únicamente añadidos en la aplicación de administrador, los cuales son *cloudinary_sdk* y *file_selector*. Por un lado, tenemos el primer paquete que está enfocado en subir archivos a la plataforma *Cloudinary*. Esta plataforma es la que he seleccionado para la administración de imágenes. Por otro lado, *file_selector* nos permite abrir un selector de archivos, en nuestro caso fotos, y almacenarlo en una variable.

Siguiendo el tema de las imágenes, tenemos el paquete *cached_network_image*, el cual nos ayuda a ahorrar recursos a la hora de obtener las fotos desde *cloudinary*. El *widget* que nos proporciona este paquete guarda en la memoria caché las fotos que sean cargadas desde la red, ahorrando así ancho de banda cuando se vuelva a cargar el *widget*.

Este ahorro de ancho de banda que acabamos de comentar, se nota sobre todo en *widgets* como el que nos proporciona *carousel_slider*, un *widget* que genera una lista deslizable automática, la cual podremos observar en la aplicación de los clientes. Además de facilitarnos la creación de un deslizable automático, también nos permite personalizar en gran medida dichas funciones, como la duración de la animación, el estilo de animación o el tiempo de espera entre animaciones.

En lo referente a las animaciones, tenemos que hablar de las dos librerías utilizadas para esto, que son *simple_animations* y *animate_do*. Estas dos librerías han sido utilizadas para la mayoría de animaciones de este proyecto. Para concretar, *simple_animations* es utilizado en la animación del inicio de sesión y del registro y *animate_do* es utilizado en las demás animaciones.

En relación con los gestores de estado y todo lo utilizado en ellos, tenemos los paquetes *provider* y *http*. El primero de ellos nos provee, nunca mejor dicho, del gestor de estados *Provider*. El segundo de ellos, *http*, nos ayuda con cualquier conexión http, en nuestro caso nos ayuda con la conexión hacia la API. Además, si hablamos de peticiones, debemos mencionar como almacenamos el JWT, es decir, el token. Aquí es donde entra en acción *flutter_secure_storage*, una librería que nos permite guardar datos sensibles de manera segura.

Los próximos paquetes son del mismo creador, pero están separados para que la importación del paquete no sea muy pesada. Estamos hablando de los *widgets* de *Syncfusion*, una empresa especializada en crear componentes en distintos lenguajes de programación, como *WPF*, *Flutter* o *MAUI*. Para este proyecto, han sido necesarios tres de tantos que nos proporcionan, los cuáles son:

- *syncfusion_flutter_charts*, especializado en los gráficos.
- *syncfusion_flutter_datagrid*, especializado en tablas de datos.
- *syncfusion_flutter_datepicker*, especializado en los calendarios.

Todos ellos han sido utilizados en la aplicación del administrador, para proporcionar al dueño del restaurante una experiencia satisfactoria. Además, junto al último paquete de estos tres, ha sido utilizada la librería *flutter_localizations*, el cual nos permite traducir, de manera automática, el calendario.

Las siguientes librerías no son tan importantes como las últimas mencionadas, pero la funcionalidad que nos proporciona es más que correcta. Las librerías nombradas son *font_awesome_flutter*, *scrollable_positioned_list* y *flutter_credit_card*. La primera de ellas, *font_awesome_flutter* nos ofrece una grandiosa cantidad de nuevos iconos que no están de manera predeterminada en *Flutter*, por ejemplo, iconos de empresas muy conocidas, como la G de *Google* o la F de *Facebook*. La siguiente librería es *scrollable_positioned_list*, la cual nos facilita un widget muy parecido al *ListView* nativo de *Flutter*, pero qué añadí a este proyecto para facilitar la creación de ciertas partes del proyecto. Por último, tenemos *flutter_credit_card*, librería utilizada para mostrar las tarjetas de créditos guardadas a la hora de pagar el pedido en la aplicación del cliente.

Los últimos paquetes, pero no por ello menos importante, son los utilizados en la aplicación de los camareros. El primero de ellos es *path_provider*, paquete que nos provee la herramienta necesaria para obtener rutas absolutas y/o relativas de archivos. Si combinamos esta propiedad con el siguiente paquete, *pdf*, podemos crear PDFs de manera sencilla y guardarlo en nuestro sistema. Por último, y para completar los paquetes de *Flutter*, utilizaremos el paquete *printing* que nos permitirá imprimir estos documentos, o cualquier otro. En nuestro caso, utilizaremos estos tres paquetes para la creación e impresión de los tickets de los pedidos creados en el restaurante. Cabe decir que, a falta de tener una impresora térmica como la que utilizan normalmente los restaurantes, he decidido implementar el paquete de *printing* que, al menos, nos permitirá imprimir en una impresora tradicional.

4.2. Preparación y configuración del entorno de programación

El siguiente punto trata sobre cómo debemos preparar nuestro entorno para programar este proyecto. Lo primero que tenemos que tener instalado es Flutter, sin esto no será imposible ejecutar aplicaciones de este lenguaje de programación. Lo segundo, debemos tener instalado *Visual Studio Code*, un entorno de programación muy conocido y con mucho potencial, el cual nos ayudará con todas las aplicaciones. Además, contamos con otro tipo de herramientas de las cuáles también hablaremos, como pueden ser *Postman*, *MongoDB Compass* o alguna máquina virtual, las cuáles crearemos y configuraremos en este apartado.

4.2.1. Flutter

La instalación de *Flutter* para dispositivos *Windows* es algo extraña. Para comenzar, debemos dirigirnos a la página oficial de *Flutter*, <https://flutter.dev>, y hacer clic en *Get Started*. Seleccionamos nuestro sistema operativo, en mi caso es *Windows*, y le daremos clic al botón de descargar.

A blue rectangular button with white text that reads "flutter_windows_3.10.3-stable.zip".

El siguiente paso que debemos hacer es extraer el contenido de dicho comprimido en el lugar donde queramos instalar *Flutter*, en mi caso fue en la raíz del disco duro primario.

De esta manera, ya tendríamos instalado *Flutter*, pero si queremos realizar algún comando de *Flutter*, no vamos a poder dado que no está registrado como un comando. Por ello, lo que debemos de hacer es agregar una nueva variable de entorno. Para realizar este paso, en el navegador de *Windows* buscaremos *Editar las variables de entorno del sistema de esta*

cuenta. En la nueva pestaña, debemos editar la variable *Path*. En esta variable, debemos añadir la ruta hasta la carpeta *bin* de *Flutter*. En mi caso sería algo así.



A continuación, si ejecutamos en una línea de comando *flutter doctor*, podremos observar si está configurado todo. A estas alturas, seguramente haya algunos fallos, pero con las instalaciones que haremos *a posteriori* todo esto quedará resuelto.

4.2.2. Visual Studio Code

El entorno de programación elegido ha sido Visual Studio Code, un entorno muy versátil desarrollado por *Microsoft*. Este editor de código es multiplataforma, por lo que podríamos utilizarlo en *Windows*, *MacOs* o *Linux*, en nuestro caso utilizaremos *Windows*. Además de todo esto, *Visual Studio Code* tiene un sinnúmero de extensiones que nos ayudarán a programar en, prácticamente, cualquier lenguaje de programación. También, tiene una buena sinergia con *Git*, por lo que nos facilita mucho las cosas a la hora de realizar acciones como *pull*, *push*, o *commit*, aunque de estas acciones hablaremos posteriormente.

Antes de configurar este editor de código, debemos comprobar si tenemos instalado *VSCode*. En el caso de tenerlo ya instalado, esta parte puede ser ignorada.

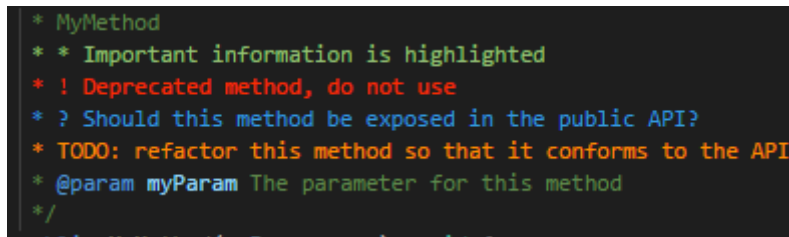
Para instalar *VSCode*, debemos dirigirnos a <https://code.visualstudio.com> y pulsar en el botón de *Descargar*. Cuando el ejecutable esté totalmente descargado, debemos abrirlo dándole doble clic en él y empezaremos a instalarlo. Al ser una instalación de *Microsoft*, es muy guiada y solo debemos darle a *Siguiente*. Cabe recalcar que hay algunas opciones personalizables, como el lugar de instalación, si queremos un icono en el escritorio o en el menú de Inicio.

Tras instalar *VSCode*, damos paso a instalar las extensiones que he utilizado durante el desarrollo de este proyecto, con un total de 18 extensiones. Me gustaría dejar claro que NO todas estas extensiones son necesarias para desarrollar este proyecto, algunas, por ejemplo, solamente ayudan a leer mejor líneas de código o remarcar de distinta manera cierto tipo de comentarios. Además, también me gustaría aclarar que todas estas extensiones están instaladas desde el *Marketplace* que está incorporado en *VSCode*.

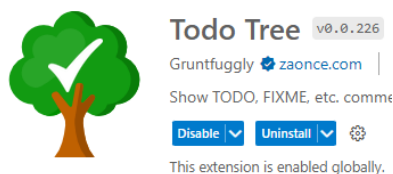
La primera extensión de la cual vamos a hablar es *Awesome Flutter Snippets*. Esta extensión nos proporciona una colección de accesos rápidos que son frecuentemente usados en el desarrollo de aplicaciones con *Flutter*. Por ejemplo, en vez de tener que escribir *SingleChildScrollView* entero, podemos escribir *singleChildSV* y creará el widget sin problema.



La segunda extensión que debemos instalar es una comentada anteriormente, *Better Comments*. *Better Commands* mejora los comentarios que podemos hacer remarcándolos con colores. En la siguiente foto podemos observar algunos de ellos.



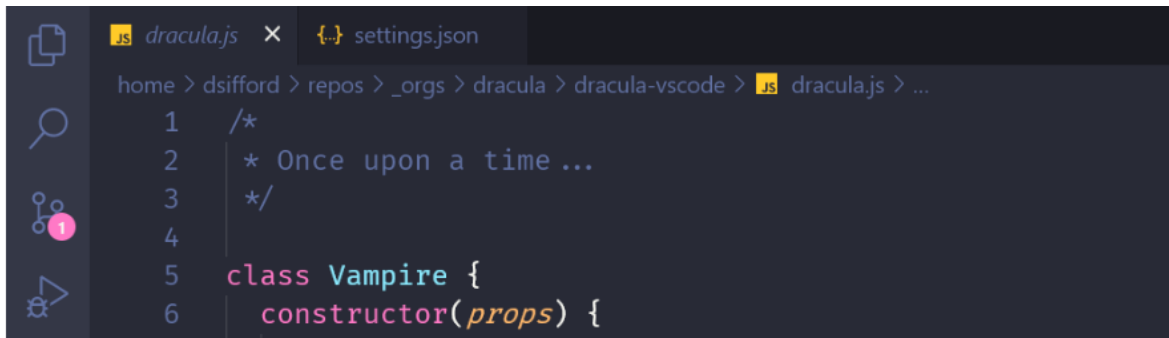
Hay algo más que mencionar sobre los comentarios. Con la extensión *TODO Tree*, podemos ver rápidamente los cometarios *TODO* (y otro tipo de comentarios, pero en mi caso solo lo utilicé para este tipo de comentarios) que tengamos en el proyecto. De esta manera, podemos ir a esa parte del código rápido y ver qué funcionalidades quedan por hacer.



En *VSCode*, a la hora de ponerte a programar en cualquier tipo de lenguaje de programación, debemos instalar la extensión de ese mismo lenguaje. Por ello, la tercera y cuarta opción son *Dart*, *Flutter* y *JavaScript*.



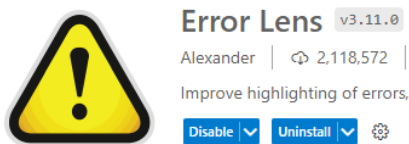
Las próximas dos extensiones son exclusivamente para el diseño de *VSCode*. Hablo de *Dracula Official*, extensión que cambia el tema de *VSCode* a un tema oscuro, pero algo más claro que el tema oscuro predeterminado. Podemos ver el tema en la siguiente foto.



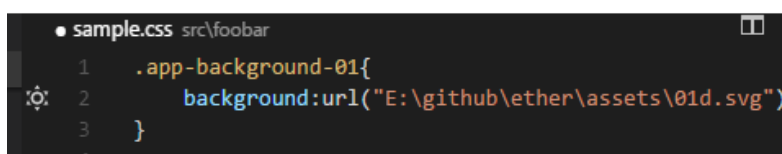
La segunda extensión sobre el diseño de *VSCode* es *Material Icon Theme*. Básicamente, esta extensión cambia el icono de todos los archivos y carpetas. Por ejemplo, los archivos de Dart y JavaScript son tal que así.



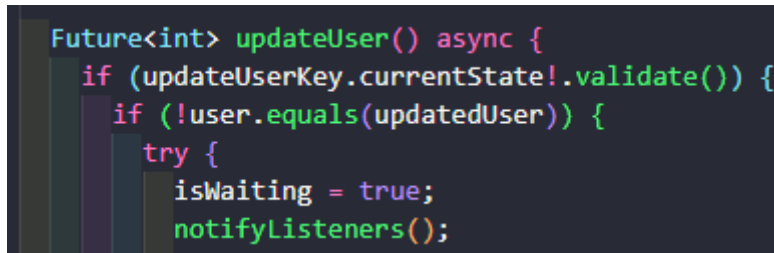
De la misma manera que algunas de las anteriores extensiones nos ayudan a programar en ciertos lenguajes, la extensión *Error Lens* nos ayuda remarcando errores, partes de código a mejorar, etc. *Error Lens* es una extensión fundamental, a mi parecer, a la hora de programar para detectar rápidamente errores que, a simple vista, no vemos.



Enlazando con el párrafo anterior, si hablamos de ver mejor ciertas cosas de nuestro código, las extensiones *Image Preview* y *Indent-Rainbow* son muy buenas. La primera de estas dos extensiones, *Image Preview*, nos permite ver en el propio código una pequeña previsualización de la imagen que hemos referenciado, como por ejemplo en la siguiente línea de código.



Si hablamos de la segunda extensión, *Indent-Rainbow*, nos obsequia, como en su nombre indica, con un arcoíris en cada “sangrado” del código, permitiéndonos así visualizar de una mejor manera si tenemos el código con una estructura correcta. Esta extensión, en lenguajes de programación como puede ser *Flutter*, va de maravilla porque son lenguajes de programación con muchos componentes dentro de otros componentes. He aquí un ejemplo de esta extensión.

A screenshot of a code editor showing a Dart function `updateUser()` with rainbow-colored indentation. The code is as follows:

```
Future<int> updateUser() async {  
  if (updateUserKey.currentState!.validate()) {  
    if (!user.equals(updatedUser)) {  
      try {  
        isWaiting = true;  
        notifyListeners();  
      }  
    }  
  }  
}
```

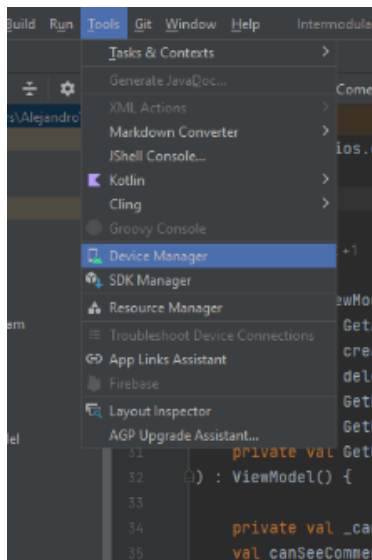
Por último, las dos extensiones restantes van enfocadas en el ahorro de tiempo. Las dos extensiones son *Pubspec Assist* y *Path Intellisense*. Por un lado, *Pubspec Assist* nos permite añadir desde cualquier lugar de nuestro proyecto una dependencia en su última versión a nuestro archivo `pubspec`. Esto nos facilita algo el trabajo porque nos ahorramos de buscar en <https://pub.dev> esta dependencia y copiar y pegar la dependencia al archivo y sin preocuparnos de la indentación. Por otro lado, *Path Intellisense* nos muestra, a la hora de escribir una importación en el código, la ruta en la que estamos y los archivos/carpetas que tenemos disponibles en ese momento. Esto previene fallos en rutas relativas o archivos mal escritos.

4.2.3. Máquinas virtuales y dispositivo físico

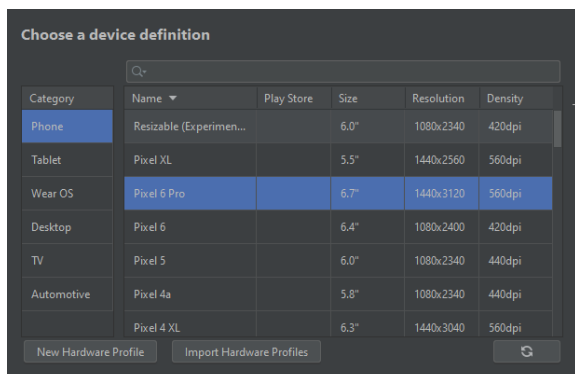
Para ejecutar nuestras aplicaciones, tenemos dos opciones: ejecutarlas en una máquina virtual o en nuestro dispositivo físico. En este apartado se comentarán estas dos opciones, pudiendo así ejecutar el código en distintos dispositivos y formatos.

Primero, comenzaremos a configurar la máquina virtual. Antes que nada, debemos tener instalado Android Studio, el cual nos ayudará a crear dicha máquina virtual. Nos dirigiremos a <https://developer.android.com/studio> y haremos clic en *Descargar Android Studio Flamingo*. Cuando esté descargado, lo ejecutaremos e instalaremos. Es una instalación sencilla, podemos darle clic a *Siguiente* todo rato hasta instalarlo.

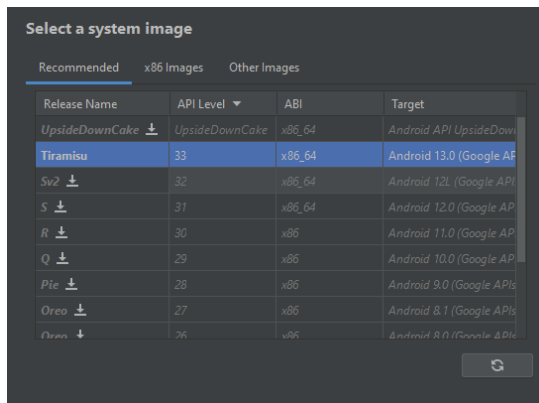
Con Android Studio ya instalado, debemos abrir y elegir/crear un proyecto. A continuación, iremos a *Tools* → *Device Manager*.



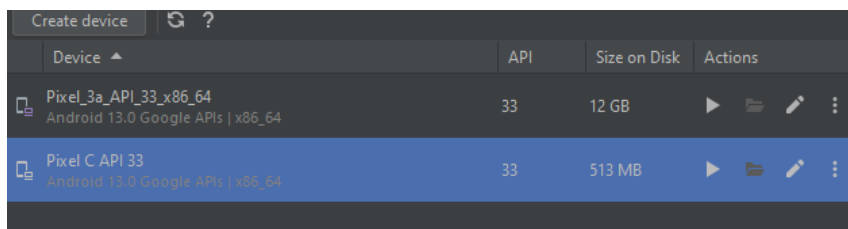
Cuando le cliquemos en él, se nos abrirá a la derecha un panel en el que podremos observar todos los dispositivos creados. Para crear uno, tendremos que darle al botón *Create Device*. Se nos abrirá una ventana en la que podremos elegir que tipo de dispositivo queremos y sus características. En mi caso, he elegido el móvil Pixel 6 Pro, aunque podemos elegir cualquier tipo o, si no nos convence ningún móvil predeterminado, podremos crearlo nosotros dándole a *New Hardware Profile*.



En cualquier caso, cuando tengamos elegido nuestro dispositivo, le daremos a *Next* y daremos paso a elegir qué sistema operativo elegimos. En mi caso, elegí la versión *Tiramisu*.



Le daremos a *Next* y a *Finish*, aunque podremos hacer algún cambio más antes de finalizar, como cambiar el nombre o la orientación de dispositivo cuando lo ejecutemos. Según mi criterio, podríamos crear otro dispositivo, como una Tablet, para observar cómo cambia el diseño entre estos dispositivos, pero dejo al criterio del lector si crea o no otro dispositivo. El apartado de *Device Manager* quedaría tal que así.



Si, además de ejecutar el proyecto en máquinas virtuales, queremos ejecutar en un dispositivo físico, debemos hacer lo siguiente. Antes que nada, quiero aclarar que el ejemplo que voy a mostrar es para dispositivos Android.

Para comenzar esta configuración, debemos acceder a los ajustes de nuestro dispositivo. Ya dentro, buscaremos las opciones de desarrollador y, en ellas, activaremos la depuración y la instalación por USB.



Tras realizar este cambio, debemos dirigirnos a nuestro ordenador. Abriremos un navegador y buscaremos el driver de nuestro móvil. Si es un Xiaomi, puedes encontrar los drivers de tu móvil en <https://xiaomidriver.com>. Los instalaremos y reiniciaremos el equipo. Teniendo ya los drivers y las máquinas virtuales instaladas, podremos ejecutar y probar nuestro código.

4.2.4. Otras aplicaciones

Las otras dos aplicaciones que han sido utilizadas durante el desarrollo de este proyecto son *Postman* y *MongoDB Compass*.

La primera de estas dos herramientas, *Postman*, es una herramienta que nos permite hacer peticiones HTTP para comprobar el funcionamiento de ciertas rutas. Esta herramienta ha sido utilizada para comprobar si las rutas creadas en nuestra API funcionan correctamente. Para descargar *Postman* para *Windows*, debemos dirigirnos al siguiente enlace: https://www.postman.com/downloads/?utm_source=postman-home. La instalación es como la que hemos hecho anteriormente, por lo que no voy a comentarlo.

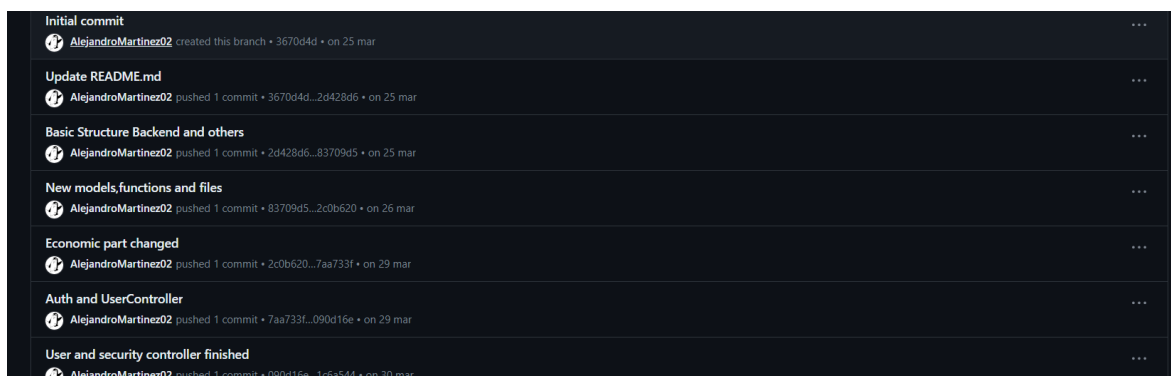


Por otro lado, *MongoDB Compass* es una herramienta que usaremos para distintos propósitos, como pueden ser visualizar cuántos documentos tenemos guardados en cierta colección o para insertar datos. El enlace para descargar el ejecutable es <https://www.mongodb.com/try/download/community>. Tras instalarlo, ya tendremos acceso a *MongoDB* y *MongoDB Compass*.

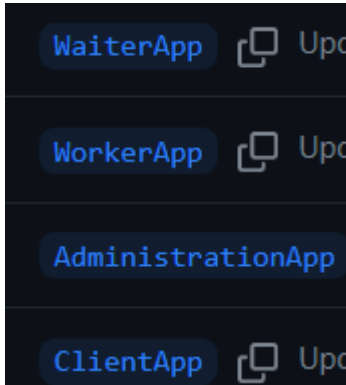


4.3. Sistema de control de versiones

Para el control de versiones, la plataforma que se ha utilizado es *GitHub*. En un principio, la idea general era crear una rama desde la rama principal e ir creando las distintas aplicaciones por separado en su rama correspondiente. Muchas veces la práctica es diferente a la teoría, en este caso fue así. Al comienzo de este proyecto no apliqué esta teoría o idea de una manera eficaz, como se puede ver en la siguiente captura.



Todos estos commits son realizados directamente a la rama principal, lo cual no es lo ideal. A mediados del proyecto, sí que se cambió de manera de actuar y se crearon y dividieron las aplicaciones en las siguientes ramas.

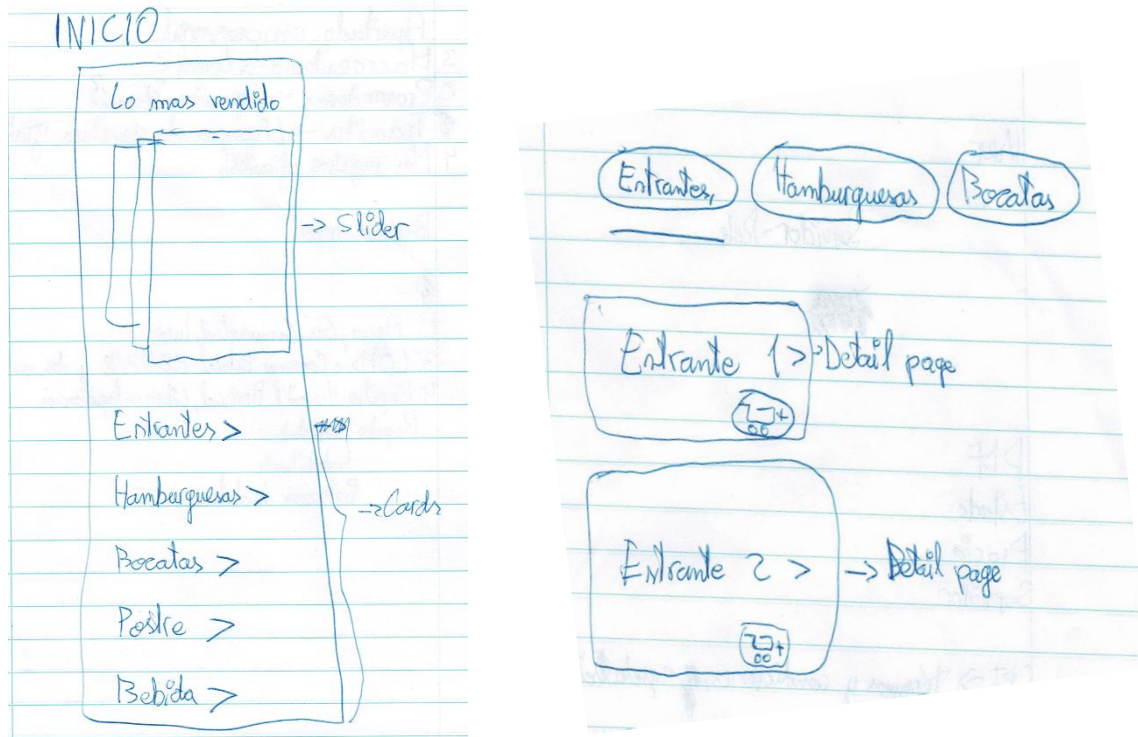


El control de versiones se ha gestionado con la aplicación de escritorio GitHub Desktop y en el navegador web, ambas opciones son realmente buenas.

5. Implementación: Prototipado, codificación y validación

5.1.1. Prototipado inicial

El prototipado de un proyecto se podría definir como un boceto que permite crear una idea general de cómo sería una pantalla, una acción o un componente. En mi caso, realicé unos bocetos a mano al principio del proyecto sobre el funcionamiento principal de la aplicación del cliente, los cuales son los siguientes.



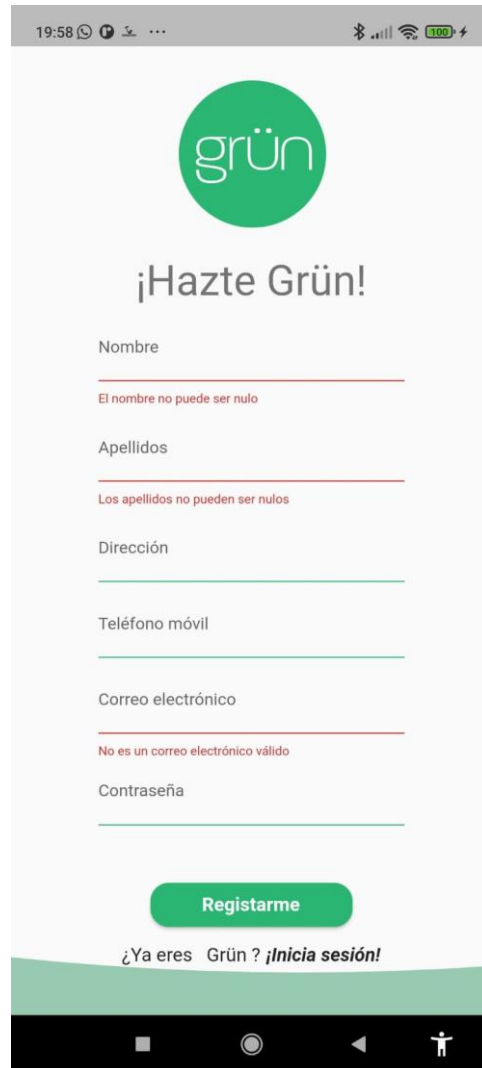
Además de estos prototipos hechos a mano, me inspiré y saqué ideas de distintas aplicaciones conocidas, como puede ser *Burger King* o *Glovo*. Como veremos más adelante, estos primeros diseños no han sido modificados a rasgos generales.

En cambio, para las aplicaciones, no realicé ningún tipo de prototipo por distintos detalles. Por ejemplo, uno de los motivos por lo que no hice un prototipo de la aplicación de administración es debido a ya tener una idea clara de cómo desarrollarlo, dado que el año pasado y este, en el proyecto intermodular, hice aplicaciones administradoras. Con referencia a la aplicación del camarero, comparte una parte del prototipo con la aplicación cliente, por lo que era innecesario hacer otro. Por último, con la aplicación de los cocineros sí podría haber hecho un prototipo, dado que no tenía ninguna experiencia previa desarrollando una aplicación así. En cambio, no lo hice porque, a mi parecer, era una interfaz sencilla y clara, por lo que no veía necesario.

5.1.2. Primeras aproximaciones

Aunque de algunas aplicaciones y/o pantallas no tengamos un prototipo, sí que ha habido muchos cambios durante el desarrollo de las aplicaciones. En este apartado, me gustaría mostrar qué diseño tenías ciertas pantallas, por qué decidí cambiarlas y qué diseño final se les ha dado.

La primera captura sobre una pequeña aproximación es sobre el inicio de sesión y el registro.

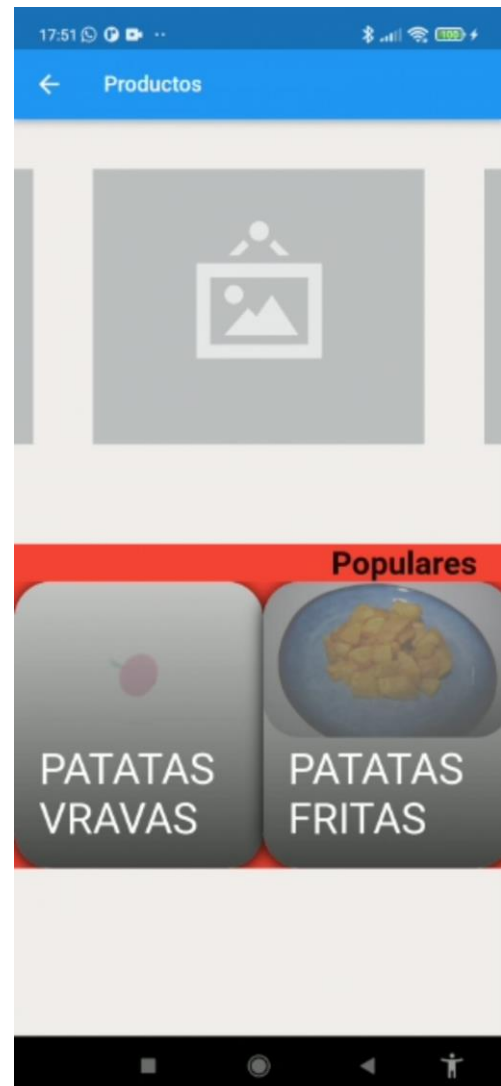


Estos diseños no me parecieron totalmente correctos, sobretodo el del inicio de sesión, por lo que cambié los diseños y quedaron tal que así.



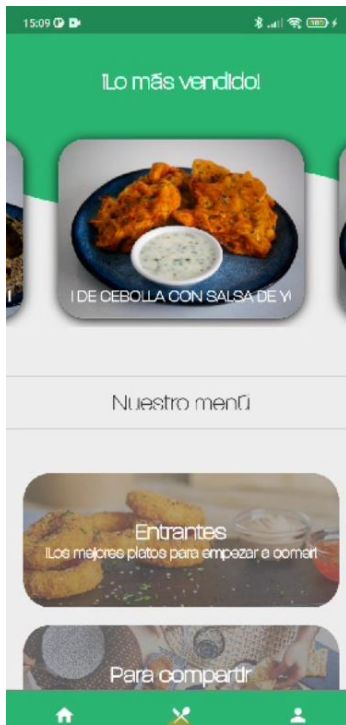
De esta manera, creo que queda un diseño mucho más serio y señorial a mi parecer. Además, con el fondo en movimiento crea la sensación de una pantalla moderna y puede llegar a atraer a los nuevos usuarios.

Por otro lado, tenemos la pantalla principal del programa, la cuál es la que más cambios ha sufrido desde su diseño inicial hasta el final, desde cambios en los componentes hasta cambios en la distribución de estos mismos. Para comenzar la parte principal, toca mostrar los primeros diseños de algunos componentes.



En la primera foto, podemos observar componente que, en un principio, sería el que mostraría los productos recomendados. Si observamos la segunda captura, vemos que ese producto ha cambiado al componente que está actualmente, el cuál muestra los productos recomendados o, mejor dicho, los menos vendidos. Lo último que me gustaría comentar sobre estas capturas es sobre el otro componente que vemos en la segunda captura de pantalla, el cuál iba a mostrar los productos más vendidos, pero se cambió por otro componente que, a mi parecer, le otorga a la aplicación una información y una funcionalidad mucho más completa que este primer componente.

En esta captura podemos observar un diseño mucho más fiel al diseño final, en el cual



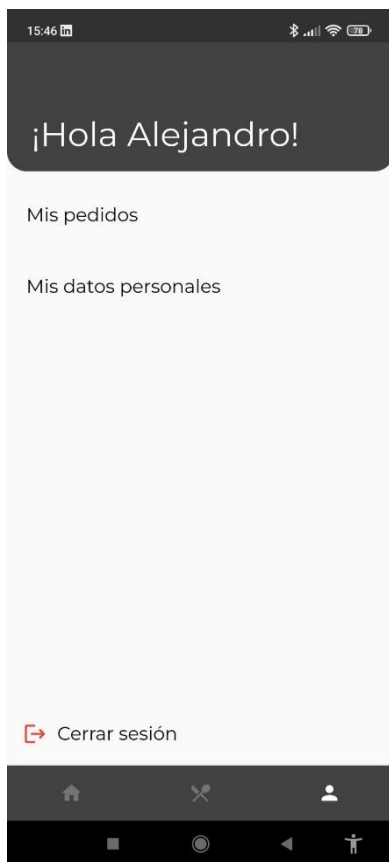
podemos ver el componente que nos muestra los productos más vendidos, aunque esto se cambió para poder promocionar aquellos productos que no son tan vendidos. Además, vemos uno de los componentes finales de esta pantalla, que es el listado de categorías del restaurante. En un deslizable vertical, se muestran de manera dinámica las categorías creadas con una imagen, un título y su correspondiente descripción. La idea de este componente es crear un acceso rápido a todas las categorías, eliminando así la necesidad de que el usuario se mueva de pantallas para dirigirse a la categoría que él quiera. Por último, se puede observar el componente que menos ha cambiado, tanto en diseño como en funcionalidad, que es la barra de navegación inferior. Esta únicamente ha cambiado el color, dado que en un principio la mayoría de componentes que

tuvieran un color de fondo se les asignaría el color principal, pero este fue cambiado a uno más oscuro para dar una sensación seria y formal, como pudimos mencionar en el diseño del inicio de sesión. Aquí podemos observar cómo es el resultado final.



Por último, la última parte que quiero mencionar en este apartado es la ventana del usuario. En un principio quería que fuese al estilo de la aplicación Glovo, un estilo de *drawer* en el que podremos encontrar todo lo relacionado con el usuario, pero no me terminó de convencer este diseño y opté por el actual, que en un principio fue como la foto que tenemos a la derecha.

Cabe aclarar que se ve algo oscuro debido a que es una captura de pantalla, pero a rasgos generales este era el diseño principal, el cual no distancia mucho del diseño final, el cual es el siguiente.



5.2. Aspectos relevantes de la codificación y validación.

En la próxima sección vamos a analizar el proyecto de manera más profunda y técnica, viendo más detenidamente cada pantalla, qué funcionalidades tiene y partes del código que sean interesantes.

5.2.1. API REST

Comenzamos a analizar la codificación y validación del proyecto con la parte menos gráfica del mismo, el *backend*. Esta API REST tiene una estructura separada en componentes o módulos, de la misma manera que tenemos divididas las aplicaciones de *Flutter*. Además de los componentes, tiene distintos servicios y *middlewares*, como autenticación o validación de datos, que complementan los *endpoints*, los cuáles vamos a comentar algo más adelante. Antes de comenzar con los *endpoints*, me gustaría resaltar un fragmento de código para poder observar como el backend administra las peticiones HTTP y, además, como se crea un *socket server*.

Este primer fragmento de código es sobre la administración de las peticiones HTTP.

```
app.use('/api/users', userRouter)
  .use('/api/workers', workersRoutes)
  .use('/api/dishes', dishesRoutes)
  .use('/api/categories', categoryRoutes)
  .use('/api/orders', orderRoutes)
  .use('/api/', securityRoutes)
  .use('/api/tables', tableRoutes)
```

Como se puede observar, dividimos cada recurso del restaurante en un *router* con sus distintos *endpoints*.

Este segundo fragmento coincide con la creación del *socket server*.

```
const server = require('http').createServer(app)
module.exports = require('socket.io')(server)
server.listen(3000)
```

```
const io = require('../..../app')

const UpdateTables = (table) => {
  io.emit('UpdateTables', table)
}

const NewOrder = (order) => {
  io.emit('NewOrder', order)
}
```

```
const UpdateOrder = (id) => {  
  io.emit(id)  
}
```

La primera parte del código es la sencilla creación del servidor. En la segunda parte se puede observar las funciones que utilizamos para enviar a las aplicaciones y actualizar ciertos recursos.

Otro recurso, o mejor dicho servicio, que me gustaría destacar es el servicio de respuesta hacía el frontend. De esta manera, se mantienen centralizadas todas las respuestas, pudiendo tener un modelo de respuestas para todos los *endpoints*.

```
const RESPONSE_500 = (res) => res.status(500).send({ "status": 500,  
  "data": "An internal error has occurred" })  
  
const RESPONSE_401 = (res) => res.status(401).send({ "status": 401,  
  "data": "Unauthorized" })  
  
const RESPONSE_403 = (res) => res.status(403).send({ "status": 403,  
  "data": "Forbidden" })  
  
const RESPONSE_404 = (res) => res.status(404).send({ "status": 404,  
  "data": "Not Found" })  
  
const RESPONSE_200 = (res) => res.status(200).send({ "status": 200,  
  "data": "OK" })  
  
const RESPONSE_201 = (res, data) => res.status(201).send({ "status":  
  201, "data": data })  
  
const RESPONSE_TOKEN = (res, data) => res.status(200).send({ status:  
  data.status, data: data.token })  
  
const RESPONSE_LOGIN_WORKER = (res, data) =>  
  res.status(data.status).send({ status: data.status, data: data.data,  
    worker: data.worker })
```

Dado que todos los recursos comparten la misma estructura, a continuación, se mostrará el flujo de una petición HTTP que contenga la mayoría de servicios creados, tanto la autorización, validación y controlador de errores.

Para este ejemplo, se va a utilizar la petición *POST localhost:8080/api/dishes/*. El primero paso que se realiza cuando una petición llega al servidor es analizar el *path* de la ruta para saber a que router enviar dicha petición. En este caso, irá al *router* de los platos.

En él, se encuentran los siguientes *endpoints*.

ROUTER

```
.get('/', AUTH, CONTROLLER.GetOne)
.get('/all', AUTH, CONTROLLER.GetAll)
.get('/bycategory', AUTH, CONTROLLER.GetByCategory)
.post('/', AUTH, VALIDATOR(), CHECKERRORS, CONTROLLER.Create)
.patch('/', AUTH, VALIDATOR(), CHECKERRORS, CONTROLLER.Update)
.delete('/', AUTH, CONTROLLER.Delete)
```

Como se puede observar, nuestra petición HTTP coincide con uno de los *endpoints*. Así pues, vamos a fragmentar las partes de este *endpoint* y analizarlas por separado.

5.2.1.1. Autorización

El primer *middleware* que se encuentra la petición es el de autorización. En la función que veremos a continuación, se obtendrá el *Json Web Token*, en el caso de que exista, y se determinará si es válido o no.

```
if (!req.headers.authorization) {
  return RESPONSE_MANAGER.RESPONSE_401(res)
}
const token = req.headers.authorization.split(" ")[1]
SERVICE.decodeToken(token)
  .then(response => {
    req.user = response;
    next();
  })
  .catch(() => RESPONSE_MANAGER.RESPONSE_401(res))
```

Además, la función `decodeToken` es la siguiente.

```
const decoded = new Promise((resolve, reject) => {
  try {
    const payload = jwt.decode(token,
process.env.SECRET_TOKEN)
    resolve(payload)
  } catch (error) {
    reject({ status: 500, "data": "Expired token or invalid
token" })
  }
})
```

El módulo JWT nos permite descryptar el token y, junto al secret token, determina si es válido o no, proporcionando así una respuesta u otra.

5.2.1.2. Validación

No en todas las peticiones HTTP están protegidas por una validación, aunque la mayoría las que conlleven la creación o actualización de un objeto guardado en la base de datos la tienen. En este caso, estamos creando un nuevo producto, por lo que existe una validación.

Gracias a *express-validator* podemos crear estas validaciones y rechazar las peticiones si no la cumplen. La creación de productos tiene la siguiente validación.

```
check('name')
  .trim()
  .notEmpty()
  .custom(async (name) => {
    if (await DISH.find({ name: name }).length == 0)
throw new Error()
  }),

check('ingredients')
  .notEmpty(),

check('category')
  .notEmpty()
  .isMongoId(),

check('cost')
  .notEmpty(),

check('description')
  .notEmpty()
```

Hay ciertos atributos que solamente es necesario que no estén vacíos, pero existen otros, como la categoría, que es necesario que cumplan un filtro, concretamente que sean un *id* de *MongoDB*. Además, con el atributo *name*, se hace una llamada a la base de datos para comprobar que este nombre sea único.

Esta validación también se realiza a la hora de crear los productos en la base de datos, pero para mejorar la seguridad de nuestra base de datos y del *backend* se crean este tipo de validaciones para rechazarlas lo antes posible. Lo recomendado, también, es crear ciertos filtros en el *frontend*, para generar así otra capa de seguridad.

5.2.1.3. Comprobación de errores

En caso o no de superar la validación, el siguiente *middleware* al que accede nuestra petición es a la comprobación de errores. El anterior *middleware*, la validación, devuelve un *array* que contiene los errores, en caso de tenerlos, y se lo envía al siguiente *middleware*, que es el actual. En él, con una simple función se comprueba que el arreglo de errores está vacío.

En caso de estar vacío, se redirige la petición a la función principal del *endpoint*. En el caso contrario, mediante el servicio de respuesta mencionado en secciones anteriores, se

devuelve cierta respuesta al *frontend*. La función que determina esto, aunque no es extremadamente destacable, es el siguiente.

```
const { validationResult } = require('express-validator');
const RESPONSE_MANAGER =
  require('../services/response/ResponseManager')

const checkErrors = (req, res, next) => {
  let errors = validationResult(req)
  if (!errors.isEmpty()) {
    console.log(errors)
    return RESPONSE_MANAGER.RESPONSE_404(res)
  }
  return next();
}
```

5.2.1.4. Función del *endpoint*

Llegados a este punto, se podría decir que la petición es prácticamente válida. En algunos casos, aún quedaría cierto tipo de validación para realizar ciertas peticiones. En este caso, se ha determinado que este *endpoint* solo puede ser satisfactoriamente realizado si se cumplen dos condiciones: ser trabajador y tener el rol “Admin”. En cualquier otro caso, se rechaza dicha petición con el siguiente código.

```
if (req.user.rol !== "Admin") return RESPONSE_MANAGER.RESPONSE_403(res)
```

En caso de ser trabajador y tener dicho rol, se ejecutará el siguiente código.

```
const { body } = req
delete body._id
await SERVICE.Create(body)
  .then((response) => RESPONSE_MANAGER.RESPONSE_201(res,
response))
  .catch(() => RESPONSE_MANAGER.RESPONSE_500(res))
```

Obtenemos el *body*, el cual contiene toda la información sobre el nuevo plato, y se lo pasamos por parámetro a otro servicio, el cual es el encargado de trabajador con los modelos de la base de datos. El código de la siguiente función es la siguiente.

```
const Create = async (dish) => {
  return await new DISH(dish).save()
}
```

En caso de tener un dato inválido, salta un error, el cual es administrado por el bloque try-catch de la anterior función. Dependiendo si se guarda correctamente o no, se devolverá una u otra respuesta y la petición HTTP habrá acabado.

5.2.2. Aplicación cliente

Antes de empezar cada aplicación, es necesario mencionar para qué plataformas está desarrollado y si existe algún filtro para los dispositivos. En el caso de las aplicaciones del cliente, esta es exclusiva de Android, tanto móviles como tablets. Cabe mencionar la posibilidad de ejecutarse en IOS, lo cual es posible, pero no recomendado, dado que IOS tiene un diseño muy concreto y no está implementado en esta aplicación, es un diseño basado y orientado a Android, pero no quita que pueda utilizarse en IOS.

5.2.2.1. Splash Screen

Antes de comenzar con la codificación de esta aplicación, me gustaría comentar que en este apartado se mencionarán a las ventanas de inicio y del inicio de sesión, pero en las siguientes secciones no se comentará. Esto es debido a que en todas las aplicaciones están ambos módulos, pero la diferencia entre aplicaciones es mínima, cambiando el modo de autorizarse y el *endpoint* al cual hacer la petición.

Para comenzar, hablaremos de la primera ventana que vemos nada más iniciar la aplicación, la cuál es la que tenemos a la derecha. A rasgos generales, tenemos tres componentes: Una imagen, un texto y un *CircularProgressBar*. En la imagen no se puede detectar, pero tanto la imagen como el texto tienen una animación. El código de estos dos componentes es el siguiente.

```

AnimatedBuilder(
  animation: controller,
  child: Scaffold(
    body: Container(
      height: double.infinity,
      width: double.infinity,
      color: Colors.white,
      child: _splashContent(
        size: size,
        controller: controller,
        primaryColor:
primaryColor))),
  builder: (BuildContext context, Widget? child) {
    return Opacity(
      opacity: opacity.value,
      child: Transform.scale(scale: scale.value, child: child),
    );
  },
)

```



En el fragmento de código anterior se puede observar una parte del código, en el cuál se hace la animación de agrandar el logo del restaurante. En la siguiente parte del código concuerda con la parte de la animación del texto.

```

FadeTransition(
    opacity: Tween(begin: 0.0, end:
1.0).animate(CurvedAnimation(
    parent: controller,
    curve: const Interval(0.5, 0.8, curve:
Curves.easeInExpo))),
    child: const Text(Constants.slogan,
    textAlign: TextAlign.center,
    style: TextStyle(
        fontSize: 40,
        fontFamily: 'Paralucen',
        fontWeight: FontWeight.bold,
        color: Constants.secondaryColor,
    )),
),

```

En cuanto a la parte gráfica, no hay nada más que comentar, dado que el *CircularProgressBar* es un componente simple, nada destacable en esto.

Si nos pasamos a la parte funcional, esta pantalla nos permite realizar dos funciones:

Comprobar si existe un token en el dispositivo y, en el caso que sea así, hacer una petición a la API para comprobar si es un token válido.

Dependiendo del resultado de las anteriores comprobaciones, la aplicación redirigirá al usuario a una ventana, como puede ser el login en caso de no tener token o tenga un token inválido, o directamente a la pantalla principal. El código de la función es tan sencillo como esto.

```

try {
    _isLoading = true;
    notifyListeners();
    final checkToken = await CheckTokenUseCase.checkToken();
    isLoading = false;
    notifyListeners();
    return checkToken;
} catch (ex) {
    isLoading = false;
    notifyListeners();
    return false;
}

```

Me gustaría comentar dos matices en este fragmento de código. El primero es destacar el uso de la función *notifyListeners()*, función que comunica a los componentes que están

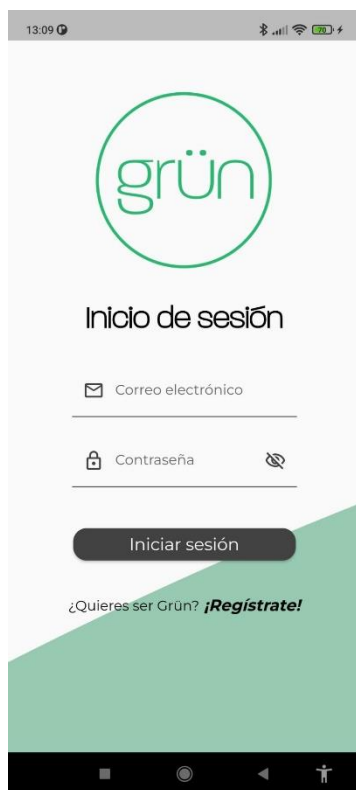
enlazados con alguna variable del *Provider* que el valor ha cambiado, de esta manera el componente se reconstruirá para actualizarse y estar a la par con los valores. Esto lo comento ahora para cuando se repita esta función, se sepa el porqué está ahí. El otro matiz que me gustaría comentar es sobre la función que llamamos para obtener el token. En ella, recurrimos al paquete *Flutter Secure Storage*, nombrado en anteriores secciones, para obtener de manera segura dicho token y comprobar, mediante la petición HTTP nombrada líneas atrás, si el token es válido. Para no repetir código en un futuro, muestro ahora cómo es una petición HTTP, concretamente un *GET*, en el siguiente fragmento de código.

```
final url = Uri.http(_baseUrl, '/api/checkToken');
return (await http.get(url, headers: {'authorization': 'Bearer $token'}))
    .body;
```

Esta petición es una de las más sencilla que se pueden hacer. En el caso de realizar otro tipo de peticiones HTTP, como pueden ser *POST*, *PATH* o *DELETE*, mostraré el código.

5.2.2.2. Login Screen

Teniendo en cuenta el flujo de la aplicación que tendría la primera vez que ejecutamos la aplicación, la anterior pantalla nos debería dirigir a la pantalla para iniciar sesión/registrarnos.



A nuestra izquierda podemos ver el inicio de sesión. Se puede observar un inicio de sesión normal y corriente con un botón para registrarnos. En el caso del inicio de sesión, está programado para validar tanto el correo electrónico como la contraseña a tiempo real, es decir, cada vez que ocurre un cambio en esos *textbox*. Además, en el cuadro de texto de la contraseña podemos cambiar la visión de esta pulsando el icono del ojo. Por último, al darle al botón de inicio de sesión, comprobará de nuevo si estos dos campos son correctos y, en caso afirmativo, hará la petición *POST* hacia la API.

A partir de aquí, pueden ocurrir distintos casos. El mejor de estos casos es haber escrito un correo y una contraseña válida, por lo que nos daría acceso a la pantalla principal. El segundo de los casos es habernos equivocado en la contraseña y/o el correo electrónico. El último de los casos, y el peor, es haber escrito la contraseña mal en varios intentos, por lo que la cuenta se bloqueará por unos cuantos minutos y no podremos iniciar sesión de ninguna manera con esas credenciales.

5.2.2.3. Register Screen

En el caso de no tener una cuenta registrada en el sistema, podremos hacer clic en el texto que dice *¡Regístrate!* y acceder a la pantalla del registro.

Como podemos observar en la zona lateral derecha del documento, el registro tiene un diseño tal que así. Cada uno tiene alguna validación, como que en el nombre y los apellidos deba poner al menos un carácter, o que en el teléfono móvil y el correo sean válidos. De la misma manera, la contraseña también tiene un patrón de validación, teniendo que poner al menos:

- Una letra mayúscula.
- Una letra minúscula.
- Un número.
- Un carácter especial.

Cuando todos los campos sean válidos, se realizará la petición HTTP hacia la API, para comprobar si todos los datos son correctos y, en caso afirmativo, guardar estos datos en la base de datos. La petición para realizar el registro, la cuál es un *POST*, es así.

```
final url = Uri.http(_baseUrl,
  '/api/register');

final response = await http.post(url,
  body: user.toJson(), headers: {'content-type':
  'application/json'});
return RegisterResponse.fromJson(response.body);
```

Como podemos observar, para realizar una petición *POST* debemos seguir esta estructura, creando un *String* del objeto y aplicando un encabezado a la petición para declarar el tipo del *body*. Para aclarar esta petición, mostraré también el código que se utilizar para convertir un objeto en *String*, como es este caso, o en un formato *json*, con el cuál no es necesario declarar el tipo de *body*.

```
String toRawJson() => json.encode(toJson());
```

```
Map<String, dynamic> toJson() => {  
    "name": name,  
    "lastname": lastname,  
    "email": email,  
    "phone_number": phoneNumber,  
    "address": address,  
    "password": password,  
};
```

Para concluir el apartado del registro, y también del inicio de sesión, quiero recalcar que en ambas pantallas, al realizar satisfactoriamente sus funciones, obtenemos un token que se guarda de manera segura en el dispositivo.

5.2.2.4. Main Screen – Home

Continuaremos con la pantalla principal, la cual se dividirá en tres partes: *Home*, *Menu* y *Profile*. En esta primera sección, como el título indica, nos centramos en el funcionamiento y en el diseño de la primera parte, el *home*.

Esta pantalla la vamos a dividir en dos sectores: el encabezado y las categorías. El primero de ellos, el encabezado, consta del deslizable automático situado en la parte superior de la pantalla. Como podemos observar a la derecha del documento, se aprecian distintos componentes, uno de ellos complejo, que es el cuadro donde se muestra el producto. Si dividimos este componente, podremos observar los siguientes componentes simples.



- Un *Container*, que almacena todos los demás componentes simples
- Una imagen, la cual es cargada dinámicamente y, una vez guardada, se almacena en la memoria caché
- Un texto que indica el nombre del producto. En caso de que el nombre sea muy extenso, este componente pasa a ser un autodeslizable, el cual permite al usuario leer cómodamente el texto.

El deslizable automático es generado por el siguiente código.

```
CarouselSlider.builder(
  options: CarouselOptions(
    viewportFraction: size.width < 600 ? 0.80 : 0.55,
    autoPlay: true,
    autoPlayCurve: Curves.fastOutSlowIn,
    autoPlayAnimationDuration:
      const Duration(milliseconds: 2500),
    autoPlayInterval: const Duration(seconds: 5)),
  itemCount: products.length,
  itemBuilder: (_, index, __) {
    return Padding(
      padding: const EdgeInsets.symmetric(vertical:
15),
      child: ProductCard(product: products[index]),
    );
  }
),
```

Y el componente compuesto es generado por este otro código.

```
GestureDetector(
  onTap: () {
    menuProvider.newOrderLine =
      OrderLineDTO(cost: product.cost, count: 1, product:
product);
    Navigator.push(
      context,
      CreateRoutes.slideFadeIn(
        direccion: const Offset(1, 0),
        screen: ProductScreen(product: product)));
  },
  child: Container(
    width: size.width < 600 ? 275 : 350,
    decoration: _cardDecoration(),
    child: Hero(
      tag: product.id!,
      child: Stack(children: [
        BackgroundImage(
          photo: product.photo,
        ),
        _cardTitle(
          title: product.name,
        )
      ]),
    ),
  ),
);
```

Es necesario explicar distintas partes del código para entenderlo a la perfección. El primer detalle que creo que es necesario comentar es sobre el componente *GestureDetector*. Este nos permite crear componentes interactivos da igual su tipo. De esta manera, no solo los botones pueden ser clicados, movidos, etc., si no que cualquier tipo de componente, o mejor dicho widget, puede serlo. Se utiliza para que, una vez el cliente entre en la aplicación, pueda seleccionar uno de los platos recomendados y se dirija directamente a la pantalla de detalles de ese propio producto. Sobre esta ventana, la de detalles, se entrará en detalle en el siguiente apartado.

Otra parte del código que es necesario explicar es sobre el *widget Hero*. Este *widget* permite realizar una transición bastante fluida y bonita que se podrá apreciar cuando se hagan las pruebas de funcionamiento.

Retomando el tema principal de esta sección, nos vamos a enfocar en el segundo segmento de esta pantalla, las categorías. Como se puede ver, es un listado con las categorías que tiene el restaurante con una pequeña descripción de cada una. Esta sección de esta pantalla es un pequeño acceso rápido para el usuario. De esta manera, puede ver sin cambiar de pantalla que categorías hay y dirigirse a una de ellas rápidamente. El código es muy parecido al del deslizable que hay en la parte superior de esta pantalla, por lo que no se adjuntará ningún tipo de código en este caso. Antes de finalizar, cabe recalcar que estas categorías cargan de manera dinámica, por lo que el administrador podrá crear, cambiar o eliminar categorías sin que la aplicación del cliente tenga que ser actualizada.



5.2.2.5. Main Screen – Menú

El próximo apartado es uno de los más complejo de los tres que existen en esta pantalla principal. En las siguientes líneas comentaremos los componentes que se involucran en esta pestaña, como otras dos vistas que están conectadas con lo relacionado con los productos y los pedidos. Además, se explicarán ciertas curiosidades y detalles centrados en los productos, algo más alejado de la programación, pero que aun así es sobre el proyecto.



Como se puede ver en la captura de pantalla que tenemos a uno de nuestros lados, se podrían sacar tres grandes widgets: la lista de categorías, el producto y el botón del carrito de compra.

Siguiendo un orden vertical, el primer *widget* que se encuentra el usuario es la lista de categorías. De la misma manera que se obtienen las categorías de manera dinámica en la anterior pantalla, aquí es más de lo mismo. Con un deslizable horizontal, el usuario podrá elegir que categoría elegir e irán cargando los productos según clique en las categorías. Existen dos escenarios al clicar una categoría:

Si esa categoría ya ha intentado cargar algún dato, esté o no vacía de productos, no realizará otra petición HTTP por mucho que se pulse la categoría. Esta funcionalidad evita un exceso de peticiones a la API REST y que una de las dos partes termine teniendo un error.

La otra solución es totalmente la contraria. La categoría nunca ha sido cargada, por lo que realizará una petición y obtendrá todos los productos que tengan esa categoría.

En el caso de que una categoría no tenga ningún tipo de producto, ya sea porque está vacía o debido a un error, saldrá una ventana de error, la cuál es la siguiente. Aunque en la captura, claramente, no se aprecie, es una foto animada.



Sobre el código, se puede llegar a destacar dos funcionalidades, como son el deslizable horizontal y la carga de datos. El deslizable horizontal tiene el siguiente código.

```
ScrollablePositionedList.separated(
  itemScrollController: menuProvider.categoryController,
  scrollDirection: Axis.horizontal,
  itemCount: menuProvider.categories.length,
  initialScrollIndex: menuProvider.actualCategory,
  physics: const ClampingScrollPhysics(),
  itemBuilder: (BuildContext context, int index) {
    return GestureDetector(
      onTap: () {
        menuProvider.changeActualPage(page: index);
        menuProvider.categoryController.scrollTo(
          alignment: 0.5,
          index: index,
          duration: const Duration(milliseconds: 200));
      },
      child: Container(
        padding: const EdgeInsets.all(10),
        decoration: BoxDecoration(
          color: menuProvider.actualCategory == index
            ? Constants.secondaryColor
            : primaryColor),
        child: Center(
          child: AutoSizeText(
            menuProvider.categories[index].name,
            style:
              Theme.of(context).textTheme.bodyLarge!.copyWith(
                fontSize: size.width < 600 ? 22 :
24,
          ),
        ),
      ),
    ),
  ),
),
```

```

    ),
    ),
);

```

Un punto que me gustaría aclarar de este código es el *itemScrollController*, el cual permite saber la posición en la que nos encontramos y permite mover el deslizable a nuestra conveniencia.

Sobre la funcionalidad de la obtención de datos, funciona de la siguiente manera.

```

Map<String, List<ProductDTO>> categoryProducts = {};
List<CategoryDTO> categories = [];

Future loadProducts() async {
  if
(categoryProducts[categories[_actualCategory].name]!.isEmpty)
return;

  categoryProducts[categories[_actualCategory].name] =
    await GetProductsByCategoryUseCase.getProductsByCategory(
      category: categories[_actualCategory].name);
}

```

Antes de explicar el código, es necesario explicar que un mapa es un objeto que nos permite guardar valores con una clave, similar a los diccionarios de *C#* o *Python*. Mediante este mapa, guardamos listas de objetos con el nombre de la categoría como clave de cada entrada de datos. Con un simple *if* comprobamos si esa entrada de datos ya guarda una lista de objetos y, en caso negativo, hacemos la petición para obtenerlos.

El siguiente *widget* que el usuario observará, en el orden anteriormente nombrado, serán los productos. De manera general, el usuario verá cinco partes: una imagen del producto, un título, una descripción, el precio y un símbolo de más. Este diseño está pensado para que el usuario pueda agregar de manera eficaz y rápida, mediante el botón con el símbolo de adición, dicho producto al pedido. Si quiere observar de manera más detenida el producto, podrá hacer clic y verá otra pestaña con datos más concretos.



En la pantalla detallada del producto, el usuario podrá observar de manera más meticulosa todo lo relacionado con el producto como la imagen, ingredientes, descripción, etc. Además, desde esta ventana podremos añadir la cantidad que deseemos de dicho producto. Cuando añadamos este producto al carrito, pulsando el botón, añadirá el producto y la cantidad indicada al carrito y llevará a la anterior pantalla al usuario para que pueda seguir añadiendo productos. El código de esta vista es el siguiente.

```

ListView(
  physics: const BouncingScrollPhysics(),
  children: [
    ProductImage(photo: product.photo, id: product.id),
    Container(
      padding: EdgeInsets.symmetric(
        horizontal: size.width * 0.05, vertical: size.height * 0.02),
      child: Column(children: [
        FadeInRightBig(
          duration: Constants.componentAnimationDuration,
          child: Column(
            children: [
              Container(
                constraints: BoxConstraints(
                  maxHeight: 50, maxWidth: size.width),
                child: ProductTitle(name: product.name)),
              const Divider(
                thickness: 2,
                color: Colors.white,
              ),
              ProductCategory(category: product.category),
              const Divider(
                thickness: 2,
                color: Colors.white,
              ),
              ProductDescription(description: product.description),
              ProductIngredients(ingredients: product.ingredients),
            ],
          )),
      ]),
    ),
  ],
  bottomNavigationBar: ProductBottomBar(cost: product.cost),

```



Algo que se debe mencionar sobre este código es la abstracción de los *widgets* y la estructura limpia que tiene, permitiendo así una lectura de código mucho más cómoda. Aquellos componentes que empiecen por *Product* son esas abstracciones que se han realizado, moviendo estos *widgets* a otros archivos separados, para poder tener un código legible.

El último *widget* que deberíamos mencionar es el botón flotante del carrito, el cual va cambiando según la cantidad de productos que haya en el carrito, así el usuario podrá conocer durante la elección de los productos cuántos lleva añadidos. Si el



usuario da click sobre él, navegará a otra ventana en la que tendremos el resumen del pedido con su coste total. Añadido a esto, en él podremos editar la cantidad de productos que queremos, eliminando así los productos que el usuario haya añadido de manera involuntaria. No creo que sea necesario comentarlo, pero cabe mencionar que si el usuario pulsa los botones de substracción y adición, situados en la parte inferior de cada producto, actualizarán al momento tanto la cantidad del producto, situado a la izquierda del producto, como el coste total.

El código que me gustaría destacar de esta pantalla, y en el estado actual, sería la validación y la funcionalidad que tienen los botones de adición y substracción.

```
void removeProductCount({required OrderLineDTO orderLine}) {
    if (orderLine.count - 1 == 0) {
        order.orderLines.removeWhere((element) => element == orderLine);
    } else {
        orderLine.count--;

        orderLine.cost -= orderLine.product.cost;
    }
    order.totalCost -= orderLine.product.cost;
    productCount -= 1;
    notifyListeners();
}

void addProductCount({required OrderLineDTO orderLine}) {
    orderLine.count++;
    orderLine.cost += orderLine.product.cost;
    order.totalCost += orderLine.product.cost;
    productCount += 1;
    notifyListeners();
}
```

```
}
```

No es ningún código extraordinario, pero cumple su función perfectamente y debe ser digno de mostrar.

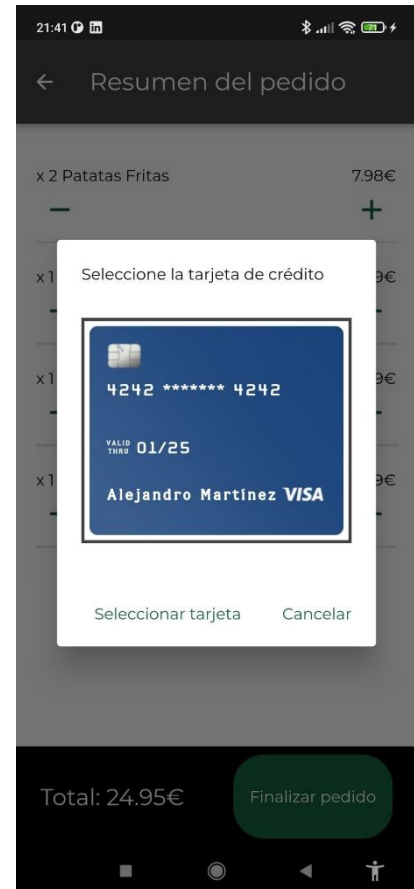
La última vista, con su correspondiente funcionalidad, que me gustaría mostrar es la ventana emergente que aparece a la hora de completar el pedido. En ella, el usuario podrá elegir entre las tarjetas de crédito que tenga guardadas en ese momento. En un principio, se iba a hacer una pasarela de pago en su totalidad, accediendo a la API de *Stripe*, la plataforma de pago, y recibiendo una respuesta según los datos enviados, todo esto con datos de prueba. Por desgracia, la librería de *Stripe* en *Flutter* no ha funcionado como debería, por lo que ha sido descartada esa opción. De todas formas, se ha mantenido la idea de poder seleccionar una tarjeta, añadido a mano en el código, para que simule el pago con tarjeta. Como este proyecto realmente es para el crecimiento y desarrollo de un servidor, en un futuro se implementará esta pasarela de pago para tener una aplicación lo más cercana a una real.

Cuando el usuario selecciona una de las tarjetas, de un total de tres, pulsará el botón y enviará una petición a la API para añadir ese pedido. En caso favorable, aparecerá una ventana emergente comunicando al usuario que puede ver el estado del pedido en el apartado “Mis pedidos” de la vista del perfil, la cual comentaremos en la siguiente sección.

Para enseñar algo de código de esta vista, se procederá a enseñar la manera en la que el programa procesa la respuesta del servidor, mostrando así una notificación u otra. Este estilo de respuesta es verdaderamente usado en muchas partes de esta aplicación y de las demás, por lo que creo más que correcto mencionarla.

```
void _checkResponse({required int status, required BuildContext context}) {
  switch (status) {
    case 200:
      Navigator.pop(context);
      _showDialog(child: const CreatedOrderNotification());

      break;
    case 500:
      Navigator.pop(context);
      _showDialog(child: const ErrorCreatingOrderNotification());
  }
}
```



```

        break;
        default:
    }
}

```

Este código recibe el código de petición de la respuesta y lo transforma en una nueva ventana emergente.

5.2.2.6. Main Screen – Profile

Esta es la última parte de la aplicación del cliente. En ella podremos encontrar tres funcionalidades importantes, las cuales son la vista de los pedidos, la edición de los datos personales y la contraseña y el cerrado de sesión.

El primero de todos es la vista de los pedidos. Básicamente, es un historial de tus propios pedidos en los que se incluyen los que estén en proceso. En la primera pantalla podrás ver



de manera muy general y resumida datos importantes del pedido, como es la fecha y hora del pedido, el coste total y el estado del mismo. Cabe mencionar que, dependiendo del estado del pedido, el texto del estado tendrá un color u otro. Si clicamos en uno de los pedidos, podremos observar un resumen más amplio del pedido, mostrando los productos como si del menú se tratase.

Además, se puede observar la cantidad de cada producto a la derecha del mismo y, en la parte inferior de la vista, veremos el coste total y el estado del pedido, de la misma manera que se podía observar en la pantalla anterior.



El código que se quiere destacar sobre esta vista no es una funcionalidad que se haya podido ver a simple vista o se haya comentado. Se está hablando de la implementación de un socket, el cual permite estar escuchando continuamente un evento, concretamente emitido por el servidor, para que un cambio de estado de un pedido sea comunicado de manera inmediata al usuario. De esta manera, el usuario podrá estar atento del pedido y saber cuándo está en entrega. El código para escuchar dicho evento y actualizar el pedido es el siguiente.

```
void createSocket(OrderDTO order) {
    _socket.on(order.id, (data) {
        order.state = 'Finalizado';
        _socket.off(order.id);
        notifyListeners();
    });
}
```

Con este código, cada vez que el usuario crea un pedido se crea una escucha para recibir ese dato. Cuando este recibe el mensaje, el pedido cambia de estado y cierra esta escucha para no consumir recursos de más.

La segunda parte de esta pantalla es referente a la edición de los datos personales. Cuando el usuario entre en esta nueva ventana, podrá elegir entre editar sus datos personales, como se puede ver en la captura de la derecha, como

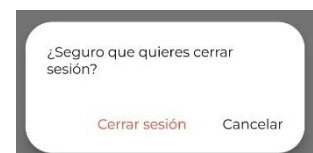


la contraseña, como se puede ver en una captura a la izquierda. Se ha decidido separar los datos personales de la contraseña para que el usuario, por equivocación, cambie la contraseña cuando realmente quería cambiar

otro parámetro. Sobre la validación de estos campos, la parte de los datos personales tiene exactamente la misma validación que tiene el registro. En cambio, el apartado de la contraseña, como se puede observar, es algo distinto, dado que es requerida la antigua contraseña para introducir una nueva. Desde esta aplicación, nada más se comprueba que la contraseña antigua no sea nula y que la nueva cumpla con los requerimientos mencionados en la parte del registro. La función de comprobar si la antigua contraseña es correcta es ejecutada por el servidor, por lo que esta aplicación solo se encargará de mostrar si se ha cambiado correctamente o no.



Por último, pero no por ello menos importante, está la opción de cerrar sesión. Es una simple ventana emergente que avisa y pregunta al usuario si desea realmente cerrar sesión. Esta función, aparte de devolvernos al inicio de sesión, borra totalmente el token almacenado en el dispositivo, para que, en caso de reiniciar la aplicación, no cargue directamente a la pantalla principal y tenga que realizar sí o sí el inicio de sesión.



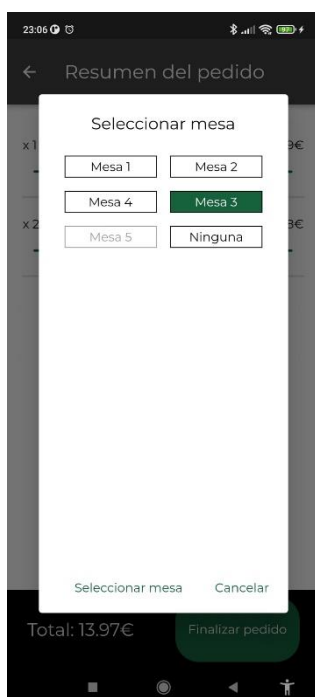
5.2.3. Aplicación Trabajador – Camarero

La aplicación para los camareros está desarrollada para ser utilizada en tres tipos distintos de dispositivos. Está desarrollado para dispositivos *Android*, tanto tabletas como móviles, y para *Windows*. Como se ha mencionado en la anterior aplicación, esta también podría ser ejecutada en *IOS* y *MacOS*, pero no es seguro que funcionen correctamente, por lo que lo recomendado es que se utilice en los dispositivos nombrados anteriormente. Cabe destacar que, para cierto tipo de dispositivos, normalmente móviles, se ha bloqueado la opción de poder girar la cámara, previniendo así posibles errores por tamaño. En cambio, en la mayoría de tabletas, se puede ejecutar esta aplicación tanto en vertical como en horizontal.

Con respecto a las aplicaciones de los trabajadores, han sido divididas en 2 para la simplificación de estas y que sean lo menos pesadas e independientes posibles. En este punto se tratará una de ellas, concretamente la de los camareros de barra/mesa.

Este apartado no será muy extenso debido a que la funcionalidad principal de esta aplicación ha sido explicada en el anterior punto de manera detallada, por lo que en este solo se comentará dos funcionalidades muy importantes y exclusivas de esta aplicación. Para concretar más cuál es la funcionalidad comentada anteriormente es toda la sección “Menú” de la aplicación de usuarios. A la hora de desarrollar, se analizó el diseño que se creó en esta aplicación y se llegó a la conclusión de que es un buen diseño para los camareros también, por lo que la implementación de este será completamente igual a la del usuario.

La primera funcionalidad exclusiva de esta aplicación está implementada a la hora de



cerrar un pedido. Si se observa la captura que está situada en el lateral del texto, podremos ver de fondo una pantalla utilizada en la aplicación de los usuarios. Cuando el camarero pulse el botón para finalizar el pedido, en vez de aparecer una tarjeta para pagar, saldrá la mesa la que va dirigida este pedido. En el caso de que sea un pedido para llevar, el camarero elegirá la opción de *Ninguna*. Además, si se observa más a fondo, se podrá observar que la mesa nº5 está en gris, dando la sensación de estar deshabilitada. En esta parte de la aplicación hay un socket escuchando un evento, de la misma manera que se hizo en la aplicación de los usuarios. En este socket, la aplicación estará escuchando los cambios que se realicen sobre las mesas, por lo que si otro camarero elige la mesa nº1 y finaliza el pedido mientras

este camarero está cerrando el pedido, le aparecerá la mesa 1 como deshabilitada. Esto está programado de esta manera para evitar equivocaciones sobre la creación de varios pedidos para la misma mesa.

La segunda funcionalidad exclusiva de esta aplicación es la creación e impresión de los pedidos realizados. El camarero, cuando seleccione la mesa y pulse el botón, se le abrirá un lector de PDF en el que se habrá creado una factura del pedido, pudiendo imprimir el ticket al instante. La impresión debería ser automática y en una impresora térmica de 8cm de ancho



Avinguda Rei Jaume I, 32
Teléfono: 695811734
Sei Vegan, Sei Grün
www.grun.es

Pedido 20235291137
Pedido a recoger/domicilio
Atendido/a por: Alex
Fecha: 29-5-2023 11:37

Producto	Unid.	Precio	Total
Patatas	1	3.99	3.99
Bhaji de cebolla	1	5.99	5.99
Batata	1	4.99	4.99

Coste total: 14.97 EUR

como en los restaurantes tradicionales, pero a falta de una impresora de estas características, se ha decidido crear de esta manera. Un ejemplo de factura sería la siguiente. Se ha intentado programar el diseño de esta factura lo más cercano a la realidad, poniendo datos del restaurante, quien atendió al comensal y los datos del pedido.

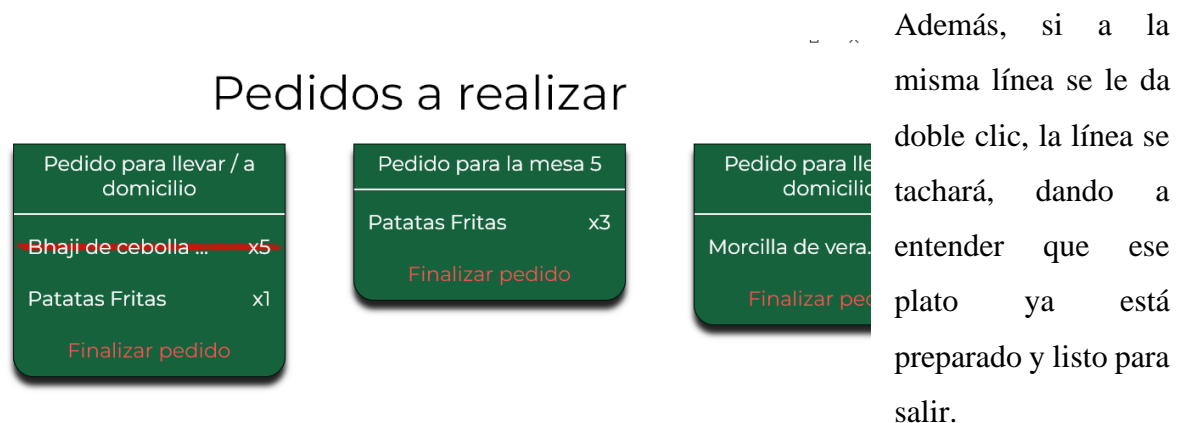
5.2.4. Aplicación Trabajador – Cocinero

La aplicación enfocada en el trabajo en la cocina está desarrollada para dispositivos con el S.O *Windows*, aunque también puede funcionar en *Linux* y *MacOS*, y en *Android*, aunque está configurado para que la *Play Store* solo recomiende esta aplicación para unos dispositivos que cumplan unas características específicas de pantalla.

Este apartado también es relativamente corto, dado que tiene una funcionalidad reducida para que los cocineros no tengan que estar muy centrados en ella. Se ha creado de esta manera para que sea sencilla y permita a los cocineros estar atentos a lo que de verdad lo necesita, la comida.

Por ello, esta aplicación consta de una única pantalla, excluyendo el inicio de sesión, en la que se visualizan los pedidos con sus detalles. Como se puede ver en la captura, es un deslizable horizontal en el que se muestran todos los pedidos activos.

Hay una funcionalidad en esta aplicación orientada para los nuevos cocineros del restaurante o por si a alguno le surge una duda. Si se le da clic a la línea del producto, podremos ver los ingredientes que lleva dicho plato para poder realizarlo correctamente.



Además, el botón de finalizar, como indica su nombre, cierra el pedido y lo marca como finalizado, moviendo todos los otros pedidos hacia la izquierda.

La última funcionalidad que es necesario comentar es, otra vez, la creación de un socket. Esto permite obtener, sin necesidad de recargar la pantalla ni darle a un botón, los nuevos pedidos, ya sean desde la aplicación de los clientes como la de los camareros.

Todas estas funciones están pensadas para facilitarle el trabajo al cocinero desde un punto de vista de alguien que nunca ha estado en una cocina profesional trabajando. Para un resultado 100% acorde al mundo laboral, en un futuro se contactará con cocineros para que otorguen un *feedback* de la aplicación y así mejorarla lo máximo posible.


5.2.5. Aplicación administradora.

La última de las aplicaciones que han sido creadas está enfocada para el uso del dueño del restaurante. En ella, se pueden gestionar distintos recursos del restaurante, como los platos, los trabajadores o los pedidos.

Esta aplicación está desarrollada y desplegada en *Windows* y dispositivos *Android*. Es necesario mencionar que esta aplicación está configurada para que, en el caso que se publique en la *Play Store*, no sea recomendada para ciertos tipos de dispositivos que no cumplan ciertas características de tamaño. De esta manera, se restringe el uso de esta aplicación a dispositivos móviles, los cuales suelen tener una pantalla algo pequeña.

Por último, cabe destacar que existe una diferencia entre la aplicación desarrollada en *Windows* y en *Android*. En *Windows*, el usuario administrador podrá subir y cambiar las fotos de los productos, mientras que en *Android* no estará esta función.

Para comenzar, en esta aplicación existe un patrón que se repite, y es a la hora de mostrar, editar y borrar la información de cada recurso. Se ha utilizado una librería de la empresa *Syncfusion* que facilita mucho estas acciones. Debido a esta facilidad, el patrón utilizado es similar en todos los recursos, cambiando los datos y las funciones utilizadas. Por ejemplo, si



Actualizar usuario

Nombre
Alejandro

Apellidos
Martin

Correo electrónico
martinezmorilloalejandro@gmr

Teléfono móvil
695811734

Dirección
Avenida 2, 32, 5drch

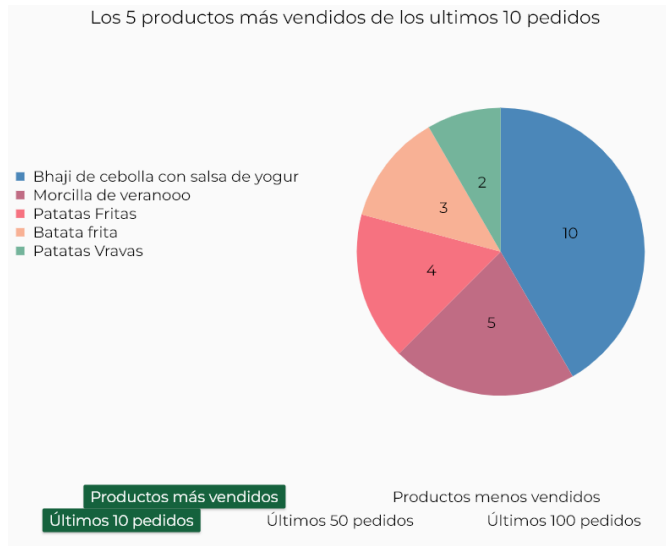
Actualizar usuario Cancelar

se selecciona a los usuarios o los pedidos, estos no tienen las mismas funcionalidades. Por un lado, se puede editar o borrar el usuario, según crea el administrador conveniente. Como se puede observar, se pueden cambiar todos los datos del usuario, exceptuando la contraseña, dado que es un dato sensible y creemos que debería ser cambiado únicamente por el usuario.

Por otro lado, podemos borrar pedidos y visualizar un resumen del mismo, viendo así los productos que lleva ese pedido y sus detalles. Además, el usuario administrador puede visualizar las ganancias brutas de los pedidos, es decir, la suma de todos los pagos de los pedidos.

Panel de administración de los pedidos				
Fecha de creación	%	Coste total	%	Estado
23-5-2023 20:59		34,93		Finalizado
23-5-2023 21:58		34,93		Finalizado
23-5-2023 22:24		34,93		Finalizado
24-5-2023 15:30		34,93		Finalizado
24-5-2023 22:21		34,93		Finalizado
Ganancias brutas: 1671.66€				

Antes de analizar el componente común de todos los recursos, me gustaría analizar otro componente de *Syncfusion*, concretamente el gráfico. En este componente, el usuario administrador podrá observar los datos de los platos que mejor y peor se han vendido. De esta manera podrá pensar mejoras sobre cómo vender estos productos. Como se puede ver



en la captura, el usuario administrador podrá ver la cantidad de estos productos, además de poder cambiar el filtro del gráfico, pudiendo elegir la cantidad de pedidos a analizar y si quiere ver lo más vendido o lo menos. El código de este gráfico es el siguiente.

```
return SfCircularChart(
  title: ChartTitle(
    text: Constants.getTitle(graphicsProvider.isBestSelected,
      graphicsProvider.currentNumOrders.abs()),
    textStyle: bodyLarge),
  legend: Legend(
    iconHeight: 40,
    iconWidth: 40,
    isVisible: true,
    position: LegendPosition.left,
    overflowMode: LegendItemOverflowMode.wrap,
    legendItemBuilder:
      (String name, dynamic series, dynamic point, int index) {
        return Row(
          mainAxisAlignment: MainAxisAlignment.min,
          children: [
            Container(
              color: (point as ChartPoint).color, height: 10, width: 10),
            const SizedBox(width: 10),
            Text(
              name,
              style: bodyLarge,
            ),
          ],
        );
      },
  series: <CircularSeries>[
    PieSeries<ProductGraphicsDTO, String>(
      dataSource: graphicsProvider
```

```

        .numProducts[graphicsProvider.currentNumOrders.toString()],
        dataLabelMapper: (ProductGraphicsDTO data, index) =>
            data.count.toString(),
        xValueMapper: (ProductGraphicsDTO data, _) => data.product.name,
        yValueMapper: (ProductGraphicsDTO data, _) => data.count,
        dataLabelSettings:
            DataLabelSettings(isVisible: true, textStyle: bodyLarge))
    ],
);

```

Por último, esta vez sí, se va a analizar el componente común en la mayoría de recursos.

Foto	Nombre	Descripción	Precio	Ingredientes	Categoría
	Patatas Vrav...	Un plato e...	3.99€	Patatas, sa...	Entrantes
	Patatas Fr...	Un simple...	2.99€	Patatas, A...	Entrantes
	Babagano...	Plato liba...	4.99€	Babagano...	Para com...
	Bhaji de c...	Entrante i...	3.99€	Bhaji(Ceb...	Entrantes
	Batata frita	Batata, ta...	3.99€	Batata, sal...	Entrantes
Añadir plato					

Como se puede observar, en la captura se detecta los distintos campos que tiene este recurso, los productos que se venden, junto a los valores registrados en la base de datos. En el caso de la descripción, por ejemplo, es un texto largo que no se logra leer correctamente, por lo que se elaboró una función para que,

al hacer click en esa celda, se pueda ver correctamente el valor. Aparece una ventana emergente con el nombre de la columna junto a su valor, incluso fotos.

Nombre

Patatas Vravas

Foto



El código de este *datagrid* es el siguiente.

```

SfDataGrid(
    onQueryRowHeight: (details) =>
        details.rowIndex == 0 ?
size.height * 0.07 : size.height * 0.13,
    verticalScrollPhysics: const
ClampingScrollPhysics(),
    horizontalScrollPhysics: const ClampingScrollPhysics(),
    onCellTap: (details) => details.rowColumnIndex.rowIndex - 1 < 0
        ? null
        : cellTapFunction(context, details, dishesSource),
    allowSwiping: true,
    columnWidthMode: ColumnWidthMode.fill,
    highlightRowOnHover: true,
    source: dishesSource,

```

```

        columns: dataGridColumns,
        startSwipeActionsBuilder: (context, dataGridRow, rowIndex) =>
            UpdateDish(rowIndex: rowIndex),
        endSwipeActionsBuilder: (context, dataGridRow, rowIndex) => DeleteDish(
            bodyLarge: bodyLarge,
            rowIndex: rowIndex,
        ),
        footerFrozenRowCount: 1,
        footer: const AddDish(),
    ),
);
}

```

Hay ciertos *widgets* personalizados, como *UpdateDish* o *DeleteDish*, que añaden la funcionalidad de la ventana emergente para añadir, actualizar o borrar un plato.

Además, si el usuario administrador desliza hacia un lado u otro, este tendrá la opción de editar, si desliza hacia la derecha, o de borrar, si desliza hacia la izquierda. En la opción de actualizar, existen también validaciones para que, por ejemplo, haya nombres únicos, en el caso de los platos, o un correo electrónico, en el caso de los trabajadores. En la captura de la derecha, se puede apreciar que podemos cambiar todos los datos de los platos, incluso cambiar la imagen, función exclusiva de los sistemas de sobremesa. El código de esta ventana emergente es algo extraño, pero se va a destacar el código que es necesario para obtener la imagen y subirla a nuestra API de imágenes, *Cloudinary*.

Este código es para obtener la imagen.

```

const XTypeGroup typeGroup =
    XTypeGroup(label: 'images', extensions: ['jpg', 'png', 'jpeg']);
_file = await openFile(acceptedTypeGroups: [typeGroup]);

```

Este código permite abrir una ventana emergente con el explorador de archivos que permite elegir un único archivo, el cuál debe ajustarse a las extensiones seleccionadas.

Actualizar plato


Nombre
Patatas Vravas

Descripción
Un plato español mítico adaptado a la dieta vegana. Unas patatas fritas en

Precio
3.99

Ingredientes
Patatas, salsa vrava(Cebolla,ajo tierno, caldo vegetal, pimenton,

Entrantes

 Cambiar imagen

Actualizar plato Cancelar

Por otro lado, tenemos el código de la subida de archivo a *Cloudinary*.

```
final cloudinary = Cloudinary.basic(cloudName: Constants.cloudName);  
final response = await cloudinary.unsignedUploadResource(  
  CloudinaryUploadResource(  
    uploadPreset: Constants.uploadPreset,  
    filePath: file.path,  
    resourceType: CloudinaryResourceType.image,  
    folder: Constants.folderCloudName));  
return response.secureUrl!;
```

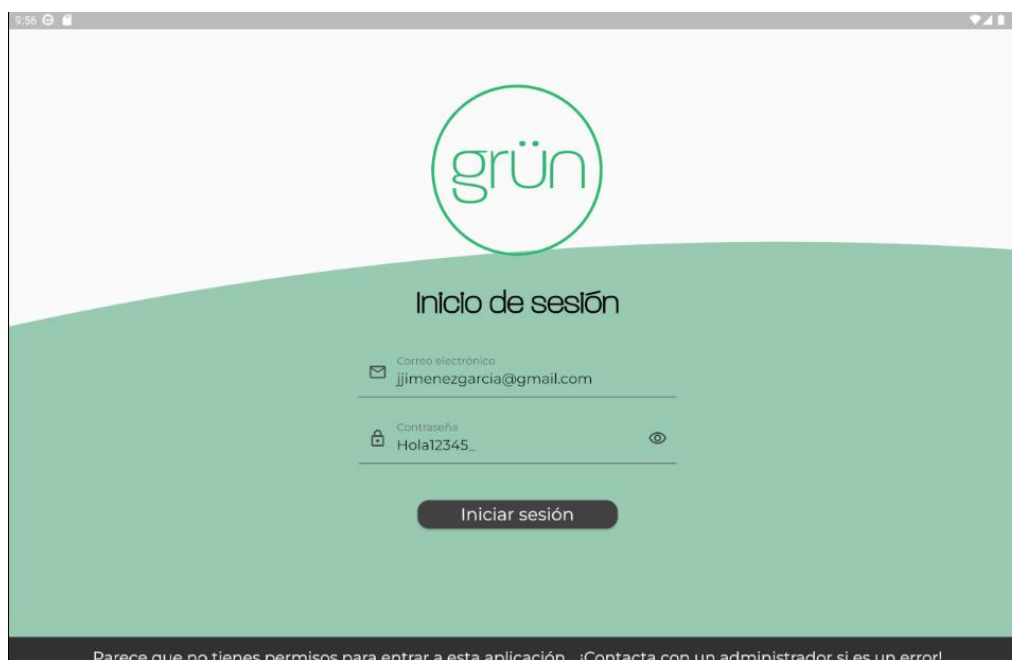
De esta petición poco hay que saber, dado que es especializada en este servicio, por lo que no creo que sea algo necesario entenderlo del todo.

5.3. Pruebas de funcionamiento y accesibilidad

En el próximo apartado se realizarán cierto tipo de pruebas y casos de uso en todas las aplicaciones para observar el funcionamiento de las mismas.

5.3.1. Aplicación administrativa

La primera aplicación en la que se harán dichas pruebas va a ser en la que está enfocada en el uso administrativo. Comenzando por el inicio de sesión, el usuario debe poner un correo y una contraseña. En caso de poner unas credenciales con un rol distinto al de administrador, le aparecerá un mensaje al usuario como el siguiente.



Además, también existe la posibilidad de introducir una contraseña errónea, la cual también está administrada para que aparezca un mensaje.

Cuando el inicio de sesión es correcto, la vista cambiará y le mostrará al usuario el panel de administración de los usuarios. En dicho panel, se pueden editar ciertos atributos de los usuarios, como el nombre o el correo, exceptuando la contraseña. Esto es debido a que se cree que si se puede cambiar la contraseña del usuario puede crearse una brecha de seguridad.

Nombre	Apellidos	Correo electrónico	Teléfono móvil	Dirección
Alejandro	Martin	martinezmorillo...	695811734	Avenida Rei Jau...
Sofia	Garcia	sgarcia@gmail.c...	695571243	Avenida 2

Si se observa la siguiente captura, se podrá ver que hay datos que no pueden verse al completo. Para solucionar este problema, en todas las tablas de datos se podrá hacer clic en la celda deseada y se abrirá una ventana que mostrará el dato seleccionado.

Correo electrónico

martinezmorilloalejandro@gmail.com

La ventana de edición del usuario es la siguiente. En esta ventana, existen distintos escenarios y sus correspondientes respuestas. La más simple es intentar actualizar al usuario sin editar ningún cambio, lo cual le devolverá al administrador un mensaje diciendo que edite al usuario para poder actualizarlo.

Debes realizar algún cambio para poder actualizar el usuario

En caso de cambiar algún dato, todos ellos tienen cierto tipo de validación, como que el correo sea uno válido o que el nombre debe no debe ser nulo.

Si todo es correcto y se ha cambiado algún dato, el programa enviará una petición HTTP al *backend* y este verificará estos datos de nuevo y, sea correcto o no, devolverá una respuesta que será transmitida al administrador.

¡Usuario actualizado correctamente!

En este panel, también se puede eliminar al usuario, mostrando una ventana de confirmación. En caso de darle a *Borrar usuario*, este

Actualizar usuario

Nombre
Alejandro

Apellidos
Martin

Correo electrónico
martinezmorilloalejandro@gmail.c

Teléfono móvil
695811734

Dirección
Avenida 2, 32, 5drch

Actualizar usuario Cancelar

será eliminado de la base de datos. Cabe destacar que la opción de borrado está en todos los recursos, por lo que no se volverá a repetir que está disponible esta opción.

Si el usuario observa la parte izquierda de la aplicación, verá una lista deslizable con todos los recursos y opciones con las que puede tratar. Si va en orden, la siguiente opción serían los platos.

Foto	Nombre	Descripción	Precio	Ingredientes	Categoría
	Patatas Vravas	Un plato esp...	3.99€	Patatas, sals...	Entrantes
	Patatas Fritas	Un simple pl...	2.99€	Patatas, AOV...	Entrantes
	Babaganous...	Plato libanés...	4.99€	Babaganous...	Para compar...

Al igual que en los usuarios, se podrá hacer clic en cualquier celda para ver su valor completamente. Si queremos editarlo, al igual que en con los usuarios, se deberá deslizar hacia la derecha para visualizar el botón de editado. Al hacer clic, tendremos una ventana con estas características. De la misma manera que en con los usuarios se debe realizar algún cambio para que se pueda actualizar, con los platos pasa exactamente lo mismo. Además, los campos en los que se puede introducir algún valor están sujetos a validaciones. Cada uno tiene ciertas características, como que el precio solamente se pueden introducir números o la lista desplegable de las

Entrantes
Para compartir
Hamburguesas
Bocatas
Postres
Salsas
Bebidas

categorías. Añadido a todo esto, y a diferencia de los usuarios, el administrador podrá agregar un plato y contendrá los mismos parámetros. En Windows,

sumado a todo esto, se podrá actualizar y/o añadir una foto al plato.

Actualizar plato

Nombre
Patatas Vravas

Descripción
Un plato español mítico adaptado a la dieta vegana. Unas patatas fritas en AOVE

Precio
3.99

Ingredientes
Patatas, salsa vrava(Cebolla,ajo tierno, caldo vegetal, pimenton, cayena),

Entrantes ▼

Actualizar plato Cancelar

El siguiente recurso sería las categorías, pero creo que no es necesario mostrar el funcionamiento de este recurso dado que es similar, ya sea creando, actualizando o

eliminando una categoría ni hay ningún componente a destacar. Por la misma razón, las mesas tampoco serán mostradas.

El cuarto recurso al que puede acceder el usuario administrador es a los trabajadores. Aquí, como en los anteriores recursos, podrá crear, actualizar y eliminar trabajadores.

Panel de administración de los trabajadores						
Nombre	Apellidos	Correo electrónico	Teléfono móvil	Edad	Emparejado	Rol
Marc	Jiménez G...	mjimenez...	675581231	23	Empareja...	Waiter
Jose	Jiménez G...	jjimenezg...	685911734	23	Casado	Chef
Añadir trabajador						

Los parámetros en este recurso son algo distintos, teniendo diferentes entradas de valores a la hora de introducir datos. Como se puede observar en las siguiente capturas,

Actualizar trabajador

Nombre
Marc

Apellidos
Jiménez García

Correo electrónico
mjimenezgarcia@gmail.com

Teléfono móvil
675581231

Salario
1400

Contraseña

Emparejado ▼
abril 2000

DOM.	LUN.	MAR.	MIÉ.	JUE.	VIE.	SÁB.
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

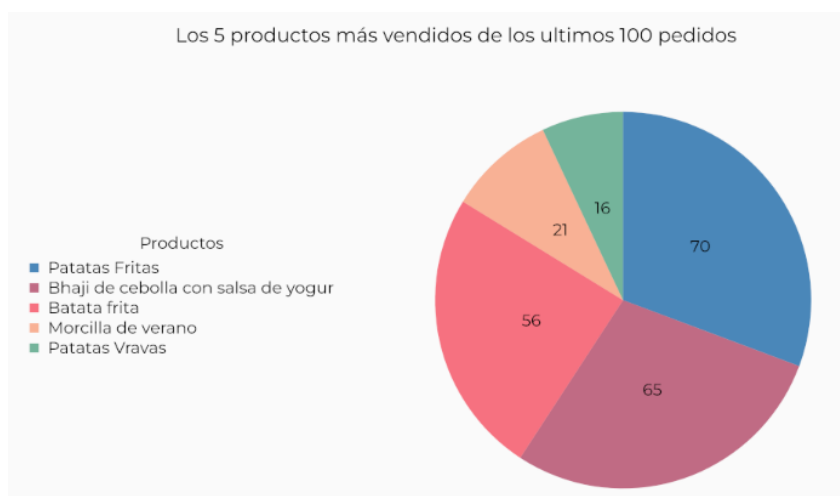
tenemos algunas entradas de textos ya vistas, como son las del nombre, apellidos, etc. y las que determinan el estado civil y el rol del trabajador, que son listas desplegables. La entrada de datos que es nueva es el calendario. En él, se podrá seleccionar la fecha de nacimiento del trabajador. Esta entrada de datos calcula, con la mayor precisión posible, la fecha mínima para que el trabajador pueda trabajar, siendo los 18 años la edad mínima y la edad máxima, siendo la de 65 años. Si el usuario administrador intenta seleccionar una fecha anterior/posterior a la edad máxima/mínima, no se seleccionará y marcará todas las fechas que no estén en el intervalo como nulas.

La penúltima opción a la que el administrador puede acceder es a los pedidos. A diferencia de los anteriores recursos, los pedidos no pueden ser editados por el administrador. Esta decisión se tomó porque se pensó que la aplicación administradora debe ser algo complementario al restaurante, sin tener que intervenir en la cadena de trabajo del mismo. Por ello, en vez de actualizar el pedido, se ha diseñado una función para observar y saber que productos cada pedido. Un pedido, al deslizar a la derecha y hacer clic en el botón con el icono del ojo, se verá lo siguiente.

Resumen del pedido

Nombre del producto	Cantidad	Coste
Patatas Vravas	1	3.99€
Patatas Fritas	1	2.99€
Campero de poyo	2	19.98€

La última opción a la que puede acceder el administrador son los gráficos. Estos gráficos muestran las ventas de cierto número de pedidos, que pueden ser 10, 50 o 100 y los cinco productos más/menos vendidos, según elija el usuario. Además, en el gráfico aparecerá la cantidad del producto. Por ejemplo, si selecciona los cinco productos más vendidos de los último 100 pedidos, saldría este gráfico.



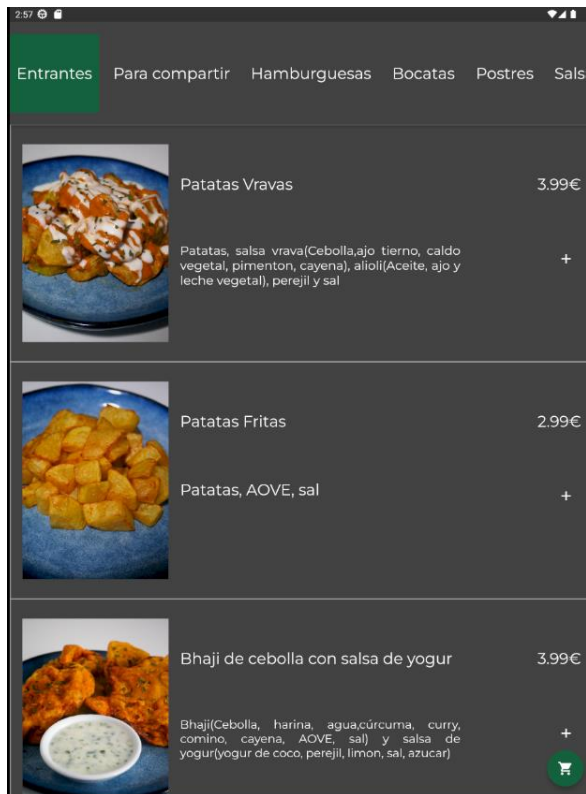
5.3.2. Aplicación camarero

La idea de este apartado y los siguientes será crear pedidos, tanto por la parte del camarero como por la parte del cliente, y posteriormente ver cómo funciona la aplicación de los cocineros. En este primer apartado de los 3 que quedan, y como indica el título, nos vamos a enfocar en la aplicación del camarero.

El inicio de sesión de esta aplicación es como el de la aplicación administradora, por lo que si intentamos iniciar sesión con un usuario que no tiene el rol de camarero, no podrá iniciar sesión. Además, aunque no se haya mencionado antes, si el trabajador se equivoca escribiendo la contraseña varias veces de manera consecutiva, la cuenta se bloqueará durante 4 minutos.

Si se ha hecho el inicio de sesión correctamente, la aplicación llevará al usuario a esta nueva vista.





Si volvemos a centrarnos en la aplicación, si el cocinero tiene que añadir más de un producto, en vez de darle al símbolo de adición, puede hacer clic en el producto para ver con mayor detalle los productos. Además, si es un camarero de mesa y desea enseñar a los comensales la foto del plato para que tengan una mejor idea de este, si pulsan la foto aparecerá una ventana emergente en la que podrán hacer zoom.

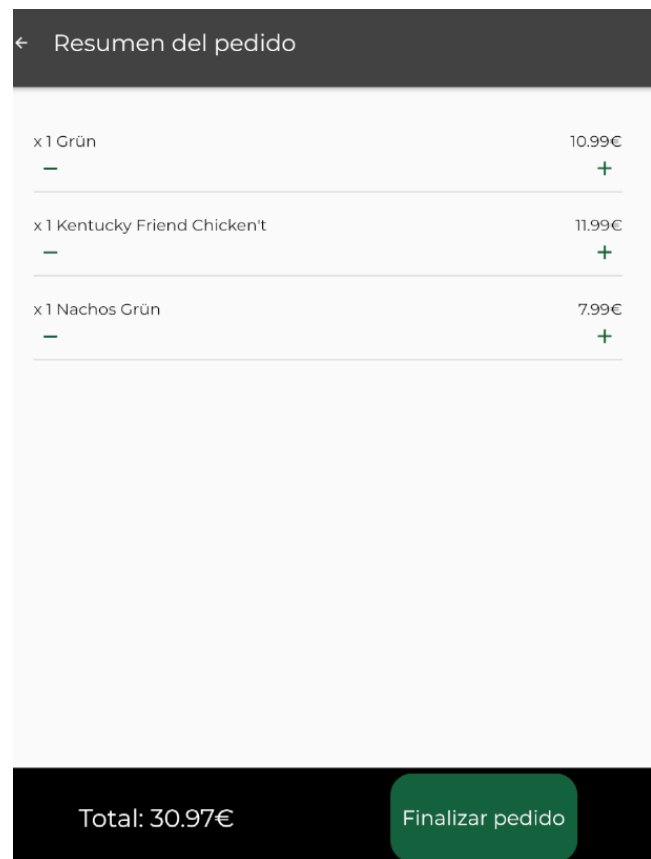
Esta pantalla permite al camarero añadir productos al pedido según el cliente quiera. Al correr el programa en una pantalla grande, se pueden observar todas las categorías, pero en pantallas más pequeñas, quedaría tal que así, pudiendo deslizar de izquierda a derecha.

Además, todo el diseño es adaptable a la pantalla, como en todas las aplicaciones. Estas características son notables si nos fijamos en el tamaño de los productos, la imagen adaptable o la colocación de los componentes.



Cuando el camarero ya ha añadido todos los productos que los comensales desean, deberá hacer clic al botón del carrito de compra. Esto abrirá una nueva vista en la que veremos un resumen del pedido, en el cuál podrá terminar el pedido, además de añadir más cantidad de ciertos productos o eliminarlos del pedido. Además, puede observarse el precio total del pedido.

El próximo paso que debería hacer el camarero es finalizar el pedido, lo cual abre una ventana emergente y le permite elegir a qué mesa va dirigida el pedido. En el caso de no dirigirse a ninguna mesa, también existe la opción de Ninguna. La siguiente captura es lo que vería el camarero para seleccionar la mesa.



Seleccionar mesa

Mesa 1	Mesa 2
Mesa 3	Mesa 4
Mesa 5	Ninguna

Seleccionar mesa Cancelar

Esta pantalla, además, consta de un socket que permite saber en tiempo real si alguna mesa está ocupada. Tras finalizar el pedido eligiendo una mesa, al camarero se le abrirá una pequeña ventana externa a la aplicación para imprimir el ticket. Esto se hace de esta manera dado que no contamos con una impresora térmica para realizar estas pruebas, pero en un

caso real se imprimiría directamente el ticket. Un ejemplo de ticket es el siguiente, el cual ha sido generado con la anterior comanda.



Avinguda Rei Jaume I, 32
Teléfono: 695811734
Sei Vegan, Sei Grün
www.grun.es

Pedido 2023681527

Mesa: 5

Atendido/a por: Alex

Fecha: 8-6-2023 15:27

Producto	Unid.	Precio	Total
Gr	1	10.99	10.99
Kentucky Fri	1	11.99	11.99
Nacho	1	7.99	7.99

Coste total: 30.97 EUR

5.3.3. Aplicación cliente

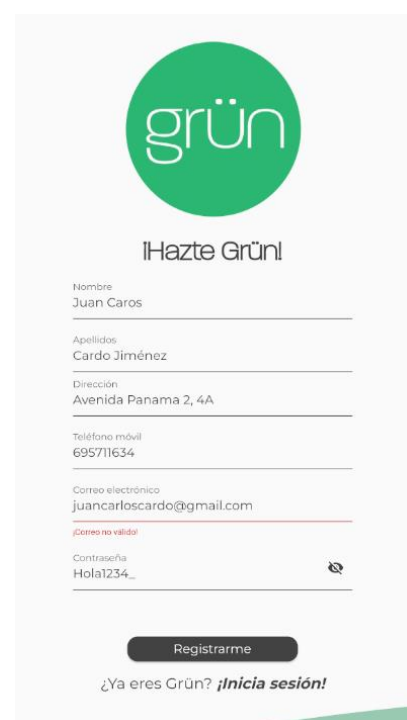
En esta sección vamos a realizar distintos casos de uso de la aplicación. Lo primero que vamos a hacer es registrarnos y editar nuestros datos una vez creada la cuenta. Lo segundo que haremos será iniciar sesión y crear un pedido, aunque solo será la parte del pago, dado que la creación del pedido es igual que en la aplicación de los camareros.

5.3.3.1. Registro y edición de datos

Esta parte se realizará en un dispositivo móvil con una pantalla no excesivamente grande. El primer paso que el usuario hará será abrir la aplicación y esperar que se inicie. Cuando cargue el inicio de sesión, podrá ver lo siguiente.



En esta vista podemos observar un texto en la zona inferior de la pantalla. Si el usuario le da click, aparecerá una nueva vista en la que tendrá que rellenar ciertas entradas de datos para poder crear la nueva cuenta. Si todos los datos son válidos, se enviará al usuario a la página principal y el nuevo usuario.

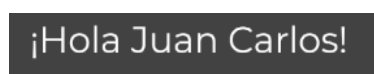


A continuación, el usuario se dirigirá al apartado del usuario, situado en la zona inferior derecha. La pantalla tendrá un aspecto como este.



Si el usuario se dirige a *Mis datos personales*, podrá cambiar tanto la contraseña de la cuenta como otros datos. En este caso, se va a cambiar los datos personales. La pantalla para visualizar y editar los datos es la siguiente. El usuario va a cambiar el nombre a “Juan Carlos”, dado que lo escribió mal a la hora de registrarse. Aquí, al igual que en el registro, existen ciertas validaciones para la entrada de datos. En el caso de no cambiar ningún dato e intentar actualizar el usuario, este recibirá un mensaje diciéndole que cambie algún dato.

Al actualizar el nombre y volver a la pestaña principal del usuario, el encabezado será tal que así.



Actualiza tu perfil

Nombre
Juan Caros

Apellidos
Cardo Jiménez

Dirección
Avenida Panama 2, 4A

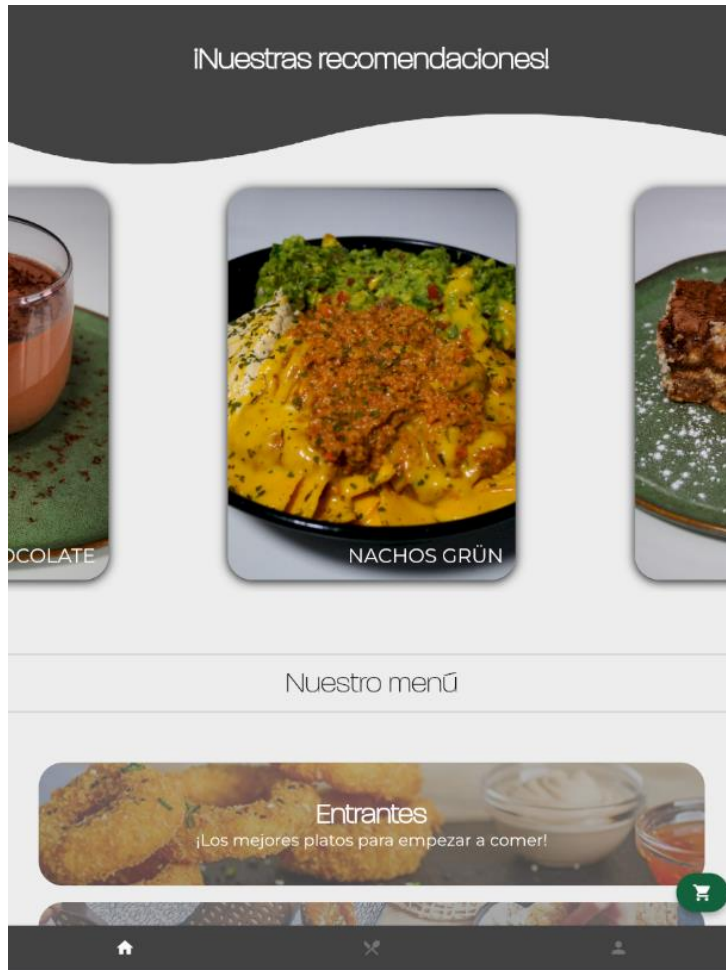
Teléfono móvil
695711634

Correo electrónico
juancarloscardo@gmail.com

Actualizar usuario

5.3.3.2. Inicio de sesión y creación de pedido

Cuando el usuario abra la aplicación, deberá iniciar sesión en el caso de no haberla iniciado anteriormente. Debe de tener cuidado, debido a que, si se equivoca en la contraseña, su cuenta se bloqueará durante 4 minutos. Cuando el inicio sesión sea correcto, al usuario se le enviará a la página principal de la app. La siguiente captura es la página de la que



hablamos. Se pueden observar dos componentes principales, el primero recomienda los platos menos vendidos y, por otro lado, las categorías del menú. Si se clic en uno de los productos recomendados, abrirá una pestaña con los detalles del producto, muy parecido a la de los camareros. En el otro caso, de clicar en una de las categorías, moverá al usuario a la categoría seleccionada. El usuario tendrá que usar estas características de la misma manera que el camarero, por lo que no creo que sea necesario mostrar el uso de estas pantallas.

Cuando el pedido esté completo, el usuario se dirigirá al resumen del pedido, haciendo clic al botón con el icono del carrito de la compra. A diferencia del camarero, el cliente no podrá elegir la mesa del pedido, pero deberá elegir una tarjeta de crédito guardada. Cuando sea seleccionada, como en la siguiente captura, el usuario finalizará el pedido y tendrá que esperar.

Seleccione la tarjeta de crédito



Seleccionar tarjeta Cancelar

Cuando finaliza el pedido, el usuario recibe el siguiente mensaje

¡Pedido enviado! Revisa el apartado de "Mis pedidos" en tu perfil para ver como va el pedido

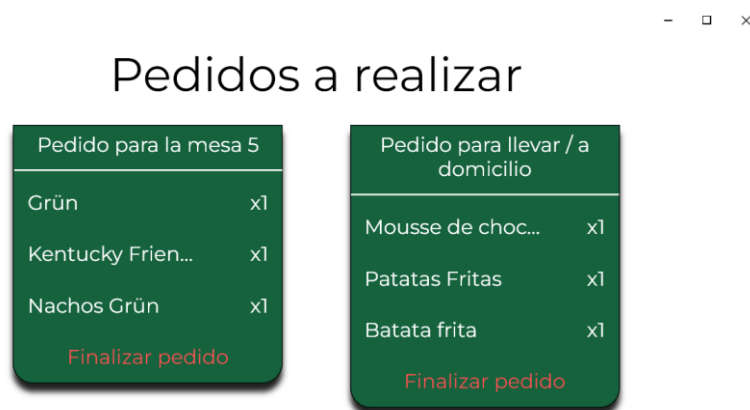
¡Entendido!

Si se dirige a tal apartado, podrá observar a tiempo real cuando su pedido cambia de estado.

5.3.4. Aplicación cocinero.

La última aplicación que queda por mostrar es la del cocinero, la cual tiene cierto tipo de conexión con las dos anteriores, pero se explicará en su momento. El inicio de sesión de esta aplicación es exactamente igual que la de los camareros y la del administrador, por lo que es innecesario comentarla.

Cuando el cocinero tenga la sesión iniciada, tendrá una única pantalla en la que se muestran los pedidos.

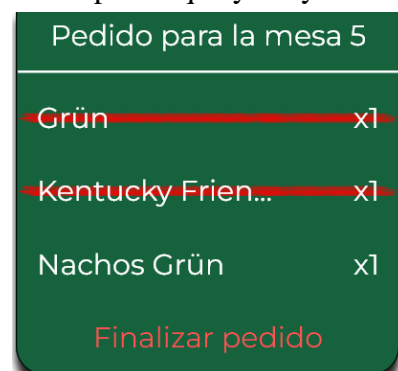


En esta pantalla, hay un total de tres funcionalidades. La primera de ellas está enfocada en el uso de cocineros novatos o que no saben la receta. Si se le da clic a uno de los platos, saldrá en pantalla los ingredientes de este plato para poder realizarlo correctamente.

Ingredientes

Filete de soja, salsa vrava(ajo tierno, cebolla tierna, caldo de verduras, harina y cayena), queso vegano, lechuga y tomate

La segunda funcionalidad disponible es al hacer doble clic en uno de los productos, este es tachado. Esto sirve para tener un orden en la cocina y tachar los platos que ya hayan sido preparados.



Por último, el cocinero podrá finalizar el pedido, actualizando en tiempo real el pedido. Tanto la mesa utilizada, si es el caso, como el estado del pedido será actualizado y cambiará en tiempo real en las respectivas aplicaciones que estén escuchando dicho evento. Por lo tanto, si un cocinero cierra el pedido, los camareros verán como esa mesa dejará de estar usada, en caso de estar vinculada al pedido, y el usuario podrá ver como cambia el estado de su pedido.

6. Despliegue del proyecto

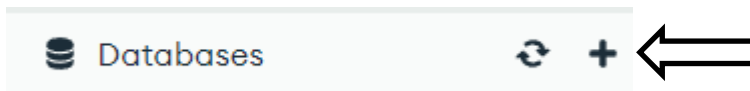
A continuación, se va a explicar cómo preparar todo para poder hacer pruebas desde casa. Lo primero de todo, debemos observar que los siguientes archivos y carpetas existan:

- Una carpeta llamada *BBDD* en la que hay archivos con la extensión *json*.
- Dos carpetas llamadas *Frontend* y *Backend*, en las que dentro contienen cinco proyectos en total, habiendo un único proyecto en *Backend* y los demás en *Frontend*.

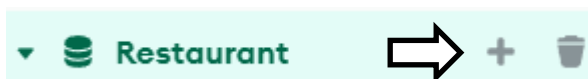
Si se tienen todas estas carpetas y archivos, *Visual Studio Code* junto a las máquinas virtuales configuradas y *MongoDB Compass*, explicadas en el punto 4, se puede pasar al próximo paso.

6.1. Ejecución de los proyectos

Antes de ejecutar los proyectos, debemos obtener cierto dato. Lo primero que se debe de hacer es abrir una nueva línea de comandos, escribir *ipconfig* y obtener la dirección IP del equipo. Además, también debemos añadir los datos a nuestra base de datos. Debemos abrir *MongoDB Compass* y conectarnos con nuestra base de datos local. Cuando hayamos conectado correctamente, debemos crear una nueva base de datos llamada *Restaurant*. En caso de no saber cómo crearlo, debemos darle al siguiente botón.



Debemos crear una colección por cada archivo que haya en la carpeta *BBDD*. Si no sabemos cómo crear una colección, debemos darle clic al siguiente botón.

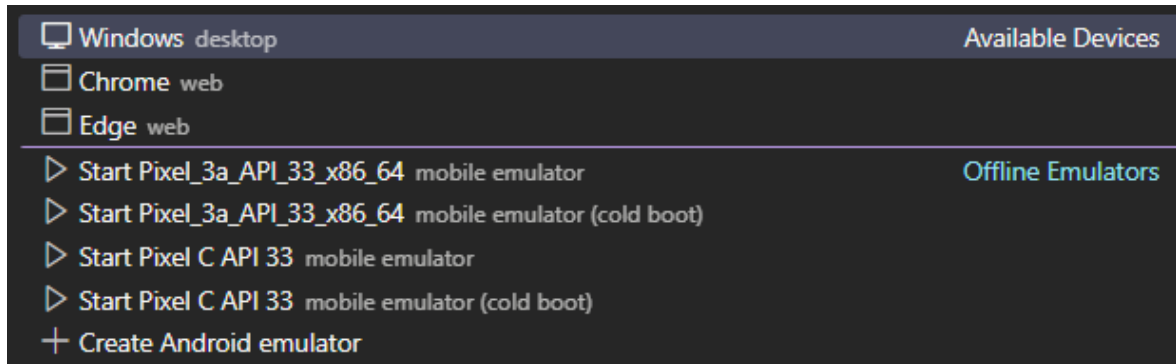


El próximo paso debe ser añadir los datos. Haciendo clic en la colección que elijamos, nos saldrán todos los datos de esta colección. Para añadir datos mediante un archivo, podemos darle clic al botón que dice *Add data* y, posteriormente, a *Import from a file*. Seleccionamos el archivo con el mismo nombre de la colección. Repetiremos estos pasos con todas las colecciones. Cuando lo hagamos, tendremos todos los datos añadidos.

El primer proyecto que debemos ejecutar es el que está ubicado en la carpeta *Backend*, es decir, la API. Con *Visual Studio Code*, debemos abrir esta carpeta para poder ejecutarlo. Teniendo este proyecto abierto, debemos dirigirnos al menú de la zona superior y hacer clic en *Terminal* ➡ *New terminal*. Escribiremos *npm start* para obtener todas las dependencias del proyecto. Después, ejecutaremos *npm run start* para ejecutar nuestro

backend. Se puede ejecutar también el código `npm run dev` para no tener que reiniciar el servidor en caso de fallo, aunque no puede pasar esto.

Con los datos añadidos y la API desplegada, llega el turno de desplegar las aplicaciones. Realizaré el ejemplo con un único proyecto, dado que solo tendremos que repetir en los demás. Debemos abrir la carpeta del proyecto con *Visual Studio Code*. Lo primero que debemos hacer es hacer pulsar *F1* y buscar *Flutter: Select device*. Nos saldrán los dispositivos disponibles, además de las máquinas virtuales. En mi caso es la siguiente lista.



Debemos iniciar la máquina virtual que deseemos, en mi caso le daré a *Start Pixel_3a_API_33_x86_64*.

Ya iniciado, debemos abrir una terminal en *VSCode*, como en el anterior paso. Escribiremos `flutter devices` para obtener el nombre de todos los dispositivos disponibles. Nos saldrá el nombre de nuestra máquina virtual, en mi caso aparece lo siguiente.

```

sdk gphone x86 64 (mobile) • emulator-5554 • android-x64 • Android 13 (API 33) (emulator)
Windows (desktop)         • windows • windows-x64 • Microsoft Windows [Version 10.0.19045]
Chrome (web)              • chrome • web-javascript • Google Chrome 114.0.5735.110
Edge (web)                • edge • web-javascript • Microsoft Edge 114.0.1823.37

```

Copiaremos el nombre `emulator-5554`. Nos dirigimos a la carpeta `.vscode` y abriremos el archivo `launch.json`. Deberá tener el siguiente formato. Debemos cambiar el nombre copiado anteriormente en el apartado `"deviceId"` del bloque llamado `"Android-VM"`. En caso de ser un dispositivo físico, lo cambiaremos en el bloque llamada `"Flutter-Android"`.


El último detalle que debemos de hacer es cambiar la dirección IP escrita en los proyectos de *Flutter*. Debemos dirigirnos a la carpeta `lib` ➡ `utils` ➡ `constants.dart`. Pulsaremos `Control + F`, o podemos buscarlo de manera manualmente, y buscaremos

```

{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Flutter-Android",
      "request": "launch",
      "type": "dart",
      "deviceId": "M2101K6G"
    },
    {
      "name": "Flutter-Tablet",
      "request": "launch",
      "type": "dart",
      "deviceId": "SM T870"
    },
    {
      "name": "Android-VM",
      "request": "launch",
      "type": "dart",
      "deviceId": "emulator-5556"
    },
    {
      "name": "Tablet-VM",
      "request": "launch",
      "type": "dart",
      "deviceId": "emulator-5554"
    },
    {
      "name": "Windows",
      "request": "launch",
      "type": "dart",
      "deviceId": "Windows"
    },
  ]
}

```


httpAPI. En esta variable y la siguiente llamada *httpSocket*, en caso de existir, debemos cambiar la que ya está puesta por nuestra IP.

Si todo esto se ha realizado correctamente, el proyecto deberá ejecutarse correctamente si pulsamos *F5* o si pulsamos en el siguiente botón  y seleccionamos una manera de ejecutarlo.

Aquí muestro algunas credenciales para iniciar sesión:

Aplicación Cliente:

Correo: martinezmorilloalejandro@gmail.com

Contraseña: Hola1234_

Aplicación Administradora:

Correo: martinezmorilloalejandro@gmail.com

Contraseña: Hola1234_

Aplicación Camareros:

Correo: mjimenezgarcia@gmail.com

Contraseña: Hola1234_

Aplicación Cocineros:

Correo: jjimenezgarcia@gmail.com

Contraseña: Hola1234_

En el caso de que alguna contraseña no funcione correctamente, la contraseña a introducir será Hola12345_

7. Conclusión del proyecto

Para concluir esta memoria y el proyecto, me gustaría comentar ciertas cosas del desarrollo del mismo. En términos generales, ha sido un desarrollo más que satisfactorio y enriqueciente. Aprender *Flutter* ha sido una experiencia muy buena, de hecho he descubierto un lenguaje de programación, y un *framework*, que me gustan mucho. Además, al aprender *Flutter*, he ido conociendo otro tipos de *frameworks* y lenguajes que quiero empezar a estudiar.

Por otro lado, estoy muy orgulloso del resultado final y de haber cumplido todos los objetivos, tanto obligatorios como opcionales, y de haber añadido ciertos detalles que no tenía pensado en un principio, como el gráfico de la aplicación administradora. También estoy orgulloso de ver cómo he afrontado los problemas que han ido saliendo durante el desarrollo, tanto los relacionados con el proyecto como por problemas externos.

8. Bibliografía

8.1. Apartado económico

Datos de economía y demografía en España [en línea] [consultado el 4 de abril de 2023]

Disponible en: <https://datosmacro.expansion.com/paises/espana>.

España. Convenio colectivo estatal, de 22 de febrero de 2018, de empresas de consultoría y estudios de mercado y de la opinión pública.

Disponible en: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2018-3156.

España. Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales.

Disponible en: <https://www.boe.es/buscar/pdf/2018/BOE-A-2018-16673-consolidado.pdf>.

España. Ley 37/1992, de 28 de diciembre, del Impuesto sobre el Valor Añadido.

Disponible en: https://noticias.juridicas.com/base_datos/Fiscal/l37-1992.html.

España. Ley 27/2014, de 27 de noviembre, del Impuesto sobre Sociedades.

Disponible en: <https://www.boe.es/buscar/act.php?id=BOE-A-2014-12328>.

España. Real Decreto Legislativo 1/2010, de 2 de julio, por el que se aprueba el texto refundido de la Ley de Sociedades de Capital.

Disponible en: <https://www.boe.es/buscar/act.php?id=BOE-A-2010-10544>.

8.2. Extensiones VSCode

Extensión *Awesome Flutter Snippets* [software]. Versión 4.0.0

Disponible en: <https://marketplace.visualstudio.com/items?itemName=Nash.awesome-flutter-snippets>

Extensión *Better Comments* [software]. Versión 3.0.2

Disponible en: <https://marketplace.visualstudio.com/items?itemName=aaron-bond.better-comments>

Extensión *TODO Tree* [software]. Versión 0.0.226

Disponible en:
<https://marketplace.visualstudio.com/items?itemName=Gruntfuggly.todo-tree>

Extensión *Dart* [software]. Versión 3.67.20230601

Disponible en:
<https://marketplace.visualstudio.com/items?itemName=Dart-Code.dart-code>

Extensión *Flutter* [software]. Versión 3.67.20230601

Disponible en:
<https://marketplace.visualstudio.com/items?itemName=Dart-Code.flutter>

Extensión *EsLint* [software]. Versión 2.4.1

Disponible en:
<https://marketplace.visualstudio.com/items?itemName=dbaumer.vscode-eslint>

Extensión *Dracula Theme* [software]. Versión 2.24.2

Disponible en:
<https://marketplace.visualstudio.com/items?itemName=dracula-theme.theme-dracula>

Extensión *Material Icon Theme* [software]. Versión 4.28.0

Disponible en:

<https://marketplace.visualstudio.com/items?itemName=PKief.material-icon-theme>

Extensión *Error Lens* [software]. Versión 3.11.1

Disponible en:

<https://marketplace.visualstudio.com/items?itemName=usernamehw.errorlens>

Extensión *Image Preview* [software]. Versión 0.30.0

Disponible en:

<https://marketplace.visualstudio.com/items?itemName=kisstkondoros.vscode-gutter-preview>

Extensión *Indent Rainbow* [software]. Versión 8.3.1

Disponible en:

<https://marketplace.visualstudio.com/items?itemName=oderwat.indent-rainbow>

Extensión *Pubspec Assist* [software]. Versión 2.3.2

Disponible en:

<https://marketplace.visualstudio.com/items?itemName=jeroen-meijer.pubspec-assist>

Extensión *Path Intellisense* [software]. Versión 2.8.4

Disponible en:

<https://marketplace.visualstudio.com/items?itemName=christian-kohler.path-intellisense>

8.3. Paquetes Flutter

Librería *another_flushbar* [software]. Versión 1.12.30.

Disponible en: https://pub.dev/packages/another_flushbar

Librería *auto_size_text* [software]. Versión 3.0.0

Disponible en: https://pub.dev/packages/auto_size_text

Librería *Marquee*[software]. Versión 2.2.3

Disponible en: <https://pub.dev/packages/marquee>

Librería *bitsdojo_window* [software]. Versión 0.1.5

Disponible en: https://pub.dev/packages/bitsdojo_window

Librería *file_selector*[software]. Versión 0.9.3

Disponible en: https://pub.dev/packages/file_selector

Librería *cloudinary_sdk* [software]. Versión 5.0.0+1

Disponible en: https://pub.dev/packages/cloudinary_sdk

Librería *cached_network_image* [software]. Versión 3.2.3

Disponible en: https://pub.dev/packages/cached_network_image

Librería *carousel_slider* [software]. Versión 4.2.1

Disponible en: https://pub.dev/packages/carousel_slider

Librería *animate_do* [software]. Versión 3.0.2

Disponible en: https://pub.dev/packages/animate_do

Librería *simple_animations* [software]. Versión 5.0.1

Disponible en: https://pub.dev/packages/simple_animations

Librería *provider* [software]. Versión 6.0.5

Disponible en: <https://pub.dev/packages/provider>

Librería *http* [software]. Versión 1.0.0

Disponible en: <https://pub.dev/packages/http>

Librería *flutter_secure_storage* [software]. Versión 8.0.0

Disponible en: https://pub.dev/packages/flutter_secure_storage

Librería *syncfusion_flutter_charts* [software]. Versión 21.2.9

Disponible en: https://pub.dev/packages/syncfusion_flutter_charts

Librería *syncfusion_flutter_datagrid* [software]. Versión 21.2.9

Disponible en: https://pub.dev/packages/syncfusion_flutter_datagrid

Librería *syncfusion_flutter_datapicker* [software]. Versión 21.2.9

Disponible en: https://pub.dev/packages/syncfusion_flutter_datapicker

Librería *flutter_locations* [software]. Versión 0.1.12

Disponible en: https://pub.dev/packages/flutter_localization

Librería *scrollable_positioned_list* [software]. Versión 0.3.8

Disponible en: https://pub.dev/packages/scrollable_positioned_list

Librería *font_awesome_flutter* [software]. Versión 10.4.0

Disponible en: https://pub.dev/packages/font_awesome_flutter

Librería *flutter_credit_card* [software]. Versión 3.0.6

Disponible en: https://pub.dev/packages/flutter_credit_card

Librería *path_provider* [software]. Versión 2.0.15

Disponible en: https://pub.dev/packages/path_provider

Librería *pdf* [software]. Versión 3.10.4

Disponible en: <https://pub.dev/packages/pdf>

Librería *printing* [software]. Versión 5.11.0.

Disponible en: <https://pub.dev/packages/printing>

8.4. Otras referencias

MongoDBCompass [software]. Versión 1.37.0.

Disponible en: <https://www.mongodb.com>.

Syncfusion [en línea]

Disponible en: <https://www.syncfusion.com>

Flaticon [en línea]

Disponible en: <https://www.flaticon.es>

Easings [en línea]

Disponible en: <https://easings.net/en>

Medium [en línea]

Disponible en: <https://medium.com>

Flutter campus [en línea]

Disponible en: <https://www.fluttercampus.com/>

Dribbble [en línea]

Disponible en: <https://dribbble.com>

Stack overflow [en línea]

Disponible en: <https://stackoverflow.com>

Node.js [en línea]

Disponible en: <https://nodejs.org>

GitHub [en línea]

Disponible en: <https://github.com>

Youtube [en línea]

Disponible en: <https://www.youtube.com>