APLICACIÓN MÓVIL PARA REALIZAR PEDIDOS A UN RESTAURANTE

FRAN GABARDA ORTIZ

MEMORIA DEL PROYECTO FINAL DEL CICLO SUPERIOR DE DESARROLLO DE APLICACIONES MULTIPLATAFORMA

IES Eduardo Primo Marqués. Curso 2023/24

Tutor individual: Francisco Lliso Bisbal

Índice

1. Introducción.

En las zonas periféricas de las ciudades o en pueblos donde las populares plataformas de reparto de comida a domicilio no ofrecen sus servicios, los restaurantes y negocios de alimentación se ven obligados a contratar sus propios repartidores si desean ofrecer este servicio. Para facilitar esta tarea, se ha desarrollado una aplicación que permite a los clientes realizar pedidos de manera sencilla y eficiente mediante una cuenta de usuario personal. Esto no solo reduce el volumen de llamadas telefónicas que deben gestionar los restaurantes, sino que también agiliza y mejora la precisión de los pedidos para los clientes.

Con esta aplicación, los usuarios pueden seguir un proceso simple e intuitivo para seleccionar los productos que desean recibir. Pueden elegir entre pagar con tarjeta o en efectivo y, si están en casa de un amigo o familiar, pueden seleccionar una dirección alternativa de entrega. Los cocineros reciben una comanda precisa con todos los detalles añadidos por el usuario, lo que facilita entregar exactamente lo que el cliente desea.

Reducir el tiempo de espera es crucial para fomentar y mantener la fidelidad de los clientes, un objetivo fundamental para cualquier negocio. En el caso de los restaurantes, los principales objetivos de esta aplicación son:

- Agilizar los pedidos
- Fidelizar a los clientes
- Mejorar la gestión de productos
- Optimizar la atención al cliente

En resumen, esta aplicación representa una solución innovadora para los restaurantes y negocios de comida que operan en áreas con servicios de entrega a domicilio limitados. Al proporcionar una plataforma intuitiva y fácil de usar, se simplifica el proceso de pedido para los clientes, mientras que los propietarios pueden mejorar la eficiencia operativa y la gestión de productos. Al enfocarse en la agilidad de los pedidos, la fidelización de los clientes y la mejora de la atención al cliente, esta aplicación tiene el potencial de transformar la experiencia de compra de alimentos tanto para los consumidores como para los comerciantes, estableciendo una base sólida para el crecimiento y la prosperidad en la industria gastronómica.

2. Estado del arte.

En este apartado, exploraremos el estado actual del desarrollo de aplicaciones móviles, centrándonos en las tendencias, avances y tecnologías que han moldeado el paisaje de las aplicaciones hasta la fecha.

Comenzaremos explorando la historia y evolución de las aplicaciones móviles, desde sus humildes comienzos hasta su ubicación actual como una fuerza dominante en el mundo digital. Luego, examinaremos las tendencias actuales en el desarrollo de las aplicaciones, incluyendo tecnologías emergentes como la realidad aumentada, la inteligencia artificial y el Internet de las cosas, que están dando forma al futuro de las aplicaciones móviles. Además, analizaremos investigaciones y estudios previos relevantes en el campo del desarrollo, identificando áreas de interés, desafíos y oportunidades para la innovación. Al comprender el estado actual del arte en el desarrollo de aplicaciones móviles estaremos mejor equipados para informar nuestras decisiones de diseño en el desarrollo de nuestra aplicación.

2.1. Historia

La evolución de las aplicaciones móviles ha sido un viaje fascinante que ha transformado la forma en que interactuamos con la tecnología en nuestros dispositivos. En este subapartado exploraremos los hitos clave en la historia de las aplicaciones móviles, desde sus humildes comienzos hasta su ubicación actual como una parte integral de nuestras vidas digitales

2.1.1 Inicio de las aplicaciones móviles

A finales de los años 90, surgieron las primeras aplicaciones móviles, que marcaron el inicio de la revolución en la forma en la que interactuamos con nuestros dispositivos móviles. En esta era pionera, los teléfonos móviles comenzaron a ofrecer más que simplemente la capacidad de realizar y recibir llamadas. Las primeras aplicaciones eran simples pero revolucionarias en su funcionalidad, brindando a los usuarios la capacidad de llevar consigo herramientas esenciales y funcionalidades básicas en sus dispositivos, como las agendas, calculadoras, editores de tonos de llamada y juegos simples. Estas aplicaciones ofrecían entretenimiento y productividad en sus dispositivos, y sentaron las bases para el desarrollo futuro de aplicaciones más avanzadas y sofisticadas.

Veamos de una forma un poco más detallada y visual como eran estas primeras aplicaciones:

- Gestión de contactos y direcciones: Las aplicaciones de gestión de contactos permitían a los usuarios almacenar y organizar sus contactos o direcciones en una agenda digital. Esto elimina la necesidad de llevar una libreta física y proporcionaba acceso rápido a la información de contacto de amigos, familiares y conocidos. En la Ilustración 1, se puede observar la interfaz y funcionamiento de la agenda de direcciones del IBM Simon, conocido como el primer smartphone de la historia según la enciclopedia libre.
- Agendas electrónicas o calendarios: Las agendas electrónicas permitían a los usuarios programar citas, recordatorios y eventos directamente desde sus teléfonos móviles. Esto facilitaba la gestión de horarios y tareas diarias sin la necesidad de llevar una agenda física. Se puede ver en la Ilustración 2 y 3 un ejemplo de la aplicación calendario del Nokia 9000, extraído de su manual de uso.

- Bloc de dibujo: Las pantallas táctiles existen desde antes de la revolución de los dispositivos móviles, IBM aprovechó esta tecnología y consiguió implementar estas pantallas en su Simon, algo que hizo posible tener una aplicación como el bloc de dibujo, destinada para realizar bocetos muy limitados en cuanto a color y tamaño de las líneas, pero funcional. En la Ilustración 4 se observa un extracto del manual de uso del IBM Simon, en el que se muestra la interfaz la aplicación.

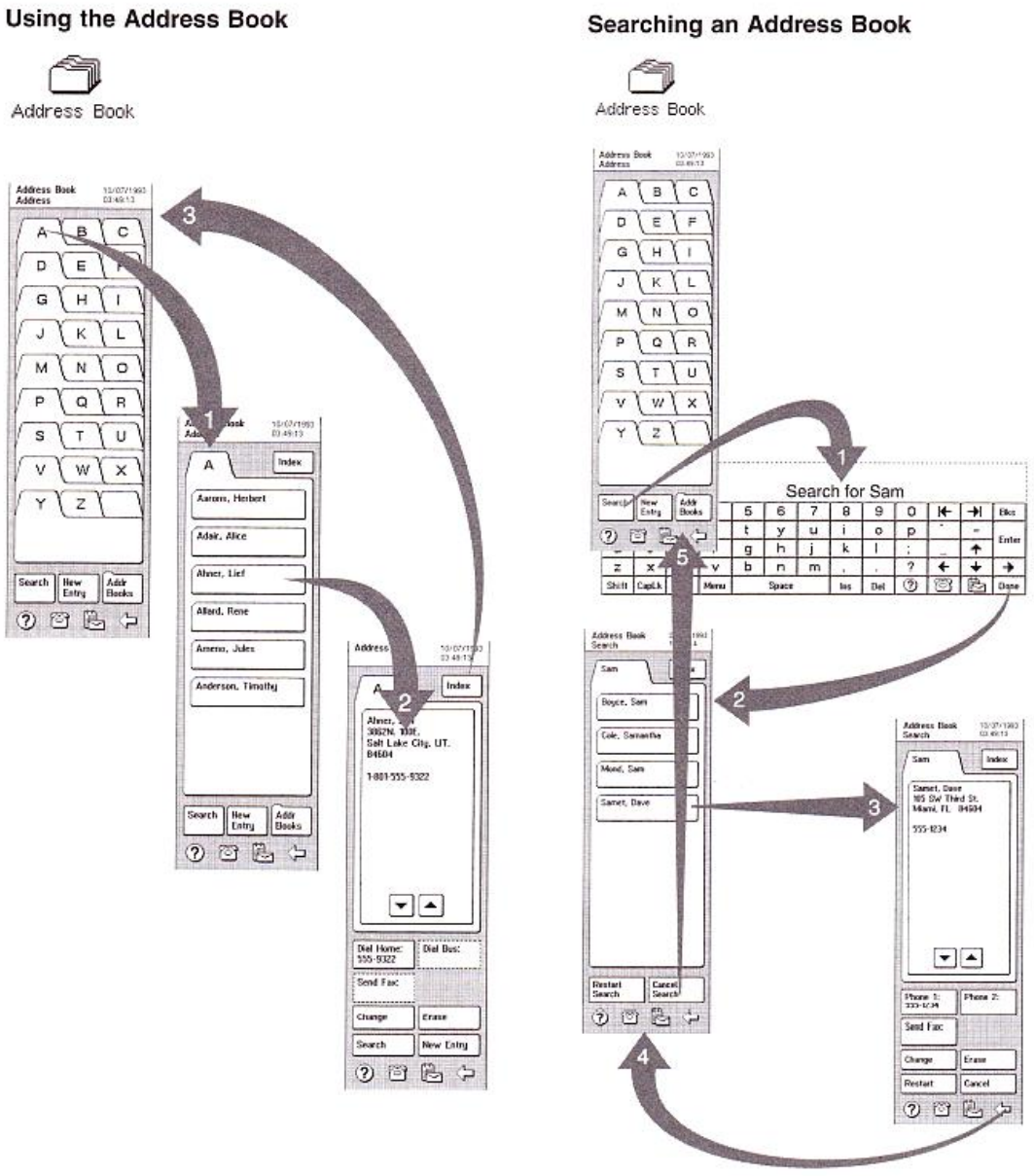


Ilustración 1

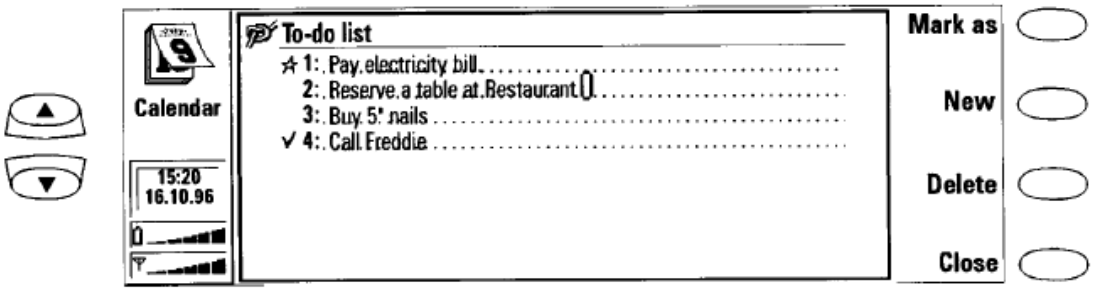


Ilustración 2

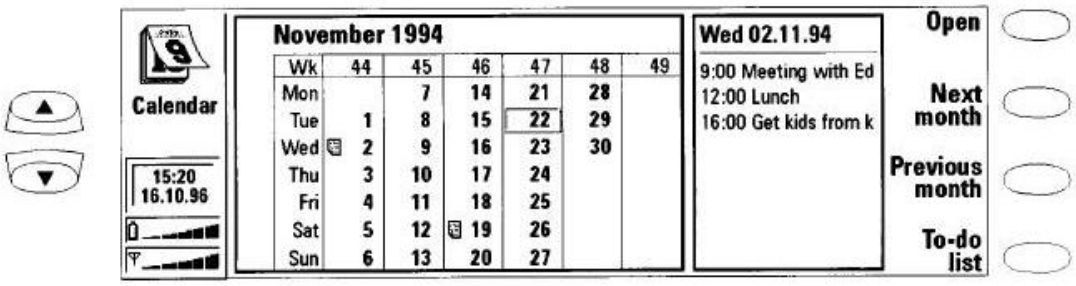


Ilustración 3

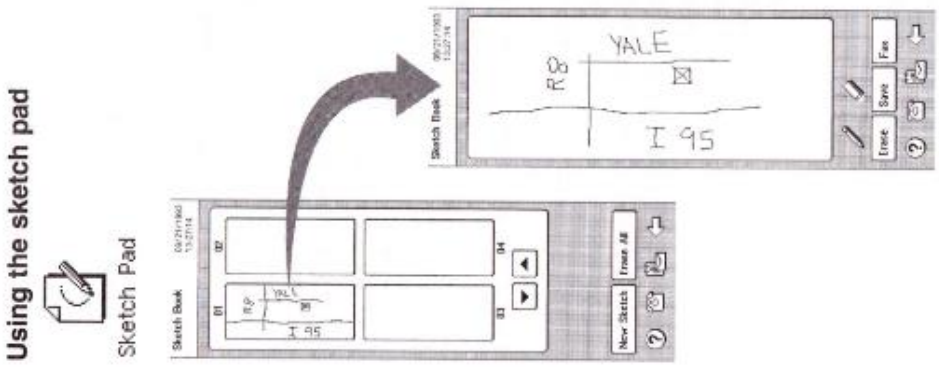


Ilustración 4

2.1.2 Llegada de los juegos móviles

Los juegos móviles han sido una parte integral de la experiencia de los dispositivos desde los primeros días de la telefonía móvil. A medida que la tecnología avanzaba y los dispositivos se volvían más capaces, los juegos experimentaron un auge de popularidad y se convirtieron en una forma de entretenimiento accesible para millones de personas en todo el mundo.

En la década de 1990, los primeros juegos eran simples pero adictivos, diseñados para aprovechar al máximo las limitaciones de hardware de los dispositivos de la época. Uno de los juegos más icónicos de esta época fue Tetris, que debutó en 1984 y se convirtió en uno de los primeros juegos en ser portado a dispositivos móviles. Con su sencilla premisa y su capacidad para mantener a los jugadores enganchados durante horas, Tetris marcó el comienzo de la era de los juegos móviles.

A principio de los 2000 los teléfonos Nokia dominaron el mercado de los dispositivos móviles y trajeron consigo una serie de juegos preinstalados que se convirtieron en clásicos instantáneos. Juegos como Snake, Space Impact y Bounce se convirtieron en los favoritos de los usuarios y ayudaron a popularizar los juegos móviles entre una amplia audiencia.

Los juegos móviles clásicos han desempeñado un papel fundamental en la evolución de la tecnología móvil, transformando nuestros dispositivos en plataformas de entretenimiento portátiles y accesibles para millones de usuarios en todo el mundo. Desde los primeros días de Tetris y Snake hasta los emocionantes avances en gráficos y jugabilidad de juegos más recientes, los juegos móviles han continuado cautivando a audiencias de todas las edades y han demostrado ser una fuerza impulsora en la innovación tecnológica. A medida que miramos hacia el futuro, es evidente que los juegos móviles, inspirando nuevas formas de entretenimiento y conectando a jugadores de todo el mundo.

2.1.3 Auge en los *markets* de aplicaciones

En este subapartado exploraremos un poco como y cuando surgieron los primeros *markets*, o tiendas de aplicaciones, como la App Store de Apple y Google Play Store.

El surgimiento de estas tiendas de aplicaciones está estrechamente ligado al desarrollo de los *smartphones* y al crecimiento de la conectividad móvil, veamos algunos hitos importantes en la historia de los *markets* de aplicaciones.

Primeros dispositivos con capacidad de descarga de aplicaciones (2000s)

A principios de los años 2000, los primeros dispositivos móviles con capacidades de descarga de aplicaciones comenzaron a aparecer en el mercado. Sin embargo, en esta época, la distribución de aplicaciones era a menudo fragmentada y dependía de los fabricantes de dispositivos y operadoras móviles.

Lanzamiento de la App Store de Apple (2008)

Uno de los hitos más importantes en la historia de los *markets* de aplicaciones fue el lanzamiento de la App Store de Apple en julio de 2008. La App Store permitió a los usuarios de iPhone descargar aplicaciones de forma centralizada y segura, lo que transformó radicalmente la forma en que las personas interactuaban con sus dispositivos móviles.

La App Store ofrecía una amplia variedad de aplicaciones, desde juegos hasta utilidades de productividad, y proporcionaba a los desarrolladores una plataforma para distribuir y monetizar sus creaciones. Se puede notar el entusiasmo de esta novedad en la publicación de Txema Marín, donde cuenta que la tienda contaba con más de 160 aplicaciones de ocio y juego, ordenados por categorías.

Lanzamiento de Google Play (anteriormente Android Market)(2008)

Poco después de lanzamiento de la App Store, concretamente en octubre de 2008, Google lanzó su propia tienda de aplicaciones, Android Market. Junto con el sistema operativo Android 1.0, servía como una plataforma centralizada para que los usuarios de dispositivos Android descargaran aplicaciones, juegos y otros contenidos.

En marzo de 2012 Google anunció una reestructuración y renombramiento de sus servicios de entretenimiento digital. Como parte del cambio, el Android Market fue renombrado como Google Play Store, reflejando la expansión de la plataforma para incluir no solo aplicaciones, sino también música, libros, y otros contenidos digitales.

Explosión de la popularidad y crecimiento continuo

Desde el lanzamiento de la App Store y Google Play Store, el número de aplicaciones disponibles ha experimentado un crecimiento exponencial. Millones de aplicaciones cubren una amplia gama de categorías, desde juegos y redes sociales hasta productividad y salud, ofreciendo a los usuarios una variedad sin precedentes de opciones para personalizar y mejorar sus móviles.

Los markets han creado oportunidades significativas para los desarrolladores de aplicaciones para monetizar su trabajo. A través de modelos de negocio como la publicidad, las compras dentro de la aplicación y las suscripciones, los desarrolladores pueden generar ingresos significativos y construir negocios sostenibles entorno a sus aplicaciones. También han implementado medidas de seguridad robustas para proteger a los usuarios contra aplicaciones maliciosas y potencialmente dañinas. Las tiendas de aplicaciones realizan revisiones y verificaciones rigurosas antes de aprobar nuevas aplicaciones para su inclusión en sus plataformas, lo que brinda a los usuarios una mayor confianza al descargar y utilizar aplicaciones.

Las tiendas continúan siendo un centro de innovación, con nuevos tipos de aplicaciones, características y servicios que se lanzan regularmente. Desde aplicaciones de realidad aumentada hasta aplicaciones de inteligencia artificial, los *markets* están en constante evolución para satisfacer las cambiantes necesidades y demandas de los usuarios móviles.

En las ilustraciones 8 y 9 se puede observar la evolución de estas tiendas desde su lanzamiento hasta día de hoy.

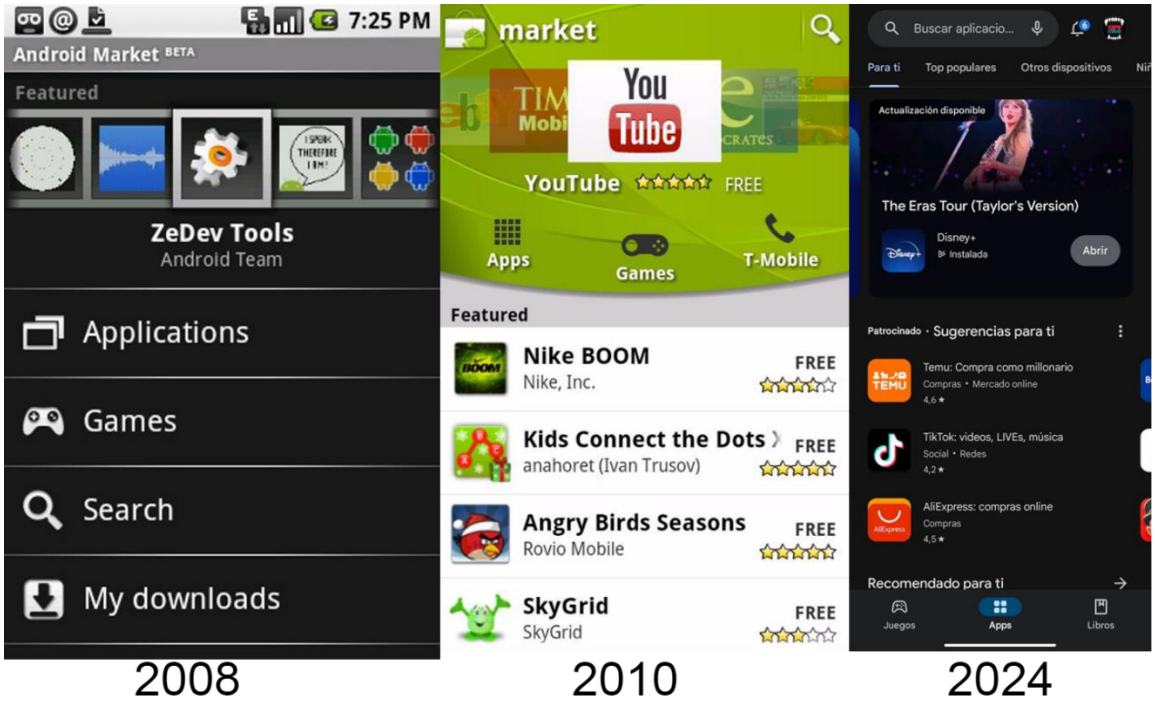


Ilustración 5

Ilustración 6

2.2. Estado actual de las aplicaciones

En el vertiginoso mundo de la tecnología móvil, las aplicaciones desempeñan un papel fundamental en la forma en que interactuamos con el mundo que nos rodea. En el contexto específico de la gestión de pedidos en restaurantes, las aplicaciones móviles han transformado la experiencia tanto para los propietarios de restaurantes como para los clientes.

Este apartado se centra en explorar el estado actual de las aplicaciones móviles relacionadas con el proyecto de desarrollo de una aplicación para la gestión de pedidos en restaurantes. Se analizarán las tendencias y características que conforman el panorama actual de las aplicaciones en este ámbito.

Tendencias actuales en aplicaciones para restaurantes

Entre las principales tendencias actuales, se encuentran:

- Pedidos en línea y entrega a domicilio: Las aplicaciones que permiten a los usuarios realizar pedidos desde sus dispositivos móviles y programar entregas han experimentado un aumento significativo en la demanda, sobre todo desde la pandemia de COVID-19.
- Pedidos sin contacto y pagos móviles: Características como la digitalización de menús, la gestión de pagos y la emisión de recibos electrónicos ayudan a minimizar el contacto físico (imprescindible después de la pandemia) y mejorar la experiencia del cliente.
- Personalización y recomendaciones: La capacidad de adaptar la oferta del restaurante a las preferencias individuales aumenta la satisfacción del cliente y fomenta la lealtad a la marca.
- Integración con las redes sociales y reseñas en línea: Las aplicaciones que incluyen funciones sociales, como compartir fotos de platos y dejar reseñas en línea, permiten a los usuarios interactuar con el restaurante y difundir sus experiencias, lo que aumenta la visibilidad y la reputación del establecimiento.
- Gestión de filas (o colas) y reservas: Estas características permiten a los usuarios evitar largas esperas y asegurar una mesa en el restaurante, mejorando la experiencia general del cliente y haciendo que aumente la demanda.
- Integración con programas de fidelización y cupones: Ofrecer códigos de descuento o recompensar a los usuarios con programas de fidelización aumentan las probabilidades de que el usuario vuelva a adquirir el producto.

Estas tendencias están dando forma al panorama de las aplicaciones para restaurantes, ofreciendo nuevas oportunidades para mejorar la experiencia del cliente y aumentar la eficiencia operativa. Por lo tanto, es imprescindible tenerlas en cuenta para crear una solución que responda a las necesidades cambiantes del mercado y destaque entre la competencia.

Experiencia del usuario en aplicaciones de gestión de pedidos

Una interfaz de usuario intuitiva y fácil de usar es esencial para garantizar una experiencia satisfactoria para el usuario. Según Nielsen Normal Group, una navegación clara y sin complicaciones permite a los usuarios encontrar rápidamente lo que buscan y completar tareas de manera eficiente. Por otro lado, la capacidad de personalizar los pedidos, modificar preferencias y guardar configuraciones favorece

una experiencia más satisfactoria y relevante para cada usuario, tal como se explica en un artículo de UX Magazine.

El servicio de atención al cliente también es muy importante. No es suficiente dar una solución al usuario, debemos asegurarnos de que la adaptación sea exitosa y ofrecer instrucciones claras de uso. Recibir poca o nula información a la hora de solucionar los problemas del usuario puede ser frustrante, tal como vemos en GetApp, una plataforma española para que los usuarios opinen y recomienden *software*.

Por otro lado, la seguridad y la privacidad son aspectos críticos en este contexto, ya que los usuarios confían en que sus datos personales y financieros estarán protegidos durante el proceso de compra. Para garantizar la protección de los datos personales de los usuarios (nombres, direcciones y números de teléfono…) frente a accesos no autorizados y uso indebido, se debe cumplir con las regulaciones de protección de datos vigentes, como el Reglamento General de Protección de Datos (GDPR) en la Unión Europea. También se recomienda que todas las transmisiones de datos entre la aplicación y el servidor se realicen a través de conexiones seguras y cifradas mediante protocolos SSL/TLS. Una política de privacidad transparente es importante para que los usuarios puedan acceder fácilmente a esta información y comprender cómo se manejarán sus datos personales.

Innovación y futuro de la tecnología para restaurantes.

La innovación es una de las claves del éxito en cualquier sector y el de la restauración no es una excepción, así dice uno de los consejos de Makro sobre la digitalización en el sector de la restauración. Estas son algunas de las tendencias que pueden conducir a una mayor eficiencia, satisfacción del cliente y un restaurante más rentable.

- Restaurantes virtuales: Se trata de cocinas virtuales que funcionan exclusivamente en línea, con servicio de entrega a domicilio. Esto reduce los costes y los riesgos iniciales, pero además ofrece a los clientes experiencias gastronómicas únicas al poder especializarse en una cocina o menú concretos. Su auge marca una tendencia innovadora en la industria alimentaria y muestra cómo la tecnología puede transformar y ampliar el concepto tradicional de comer fuera. Algunos ejemplos son Wetaca, Foover o Knoweats.
- Tableros de cocina digitales: El uso de pizarras de cocina digitales es cada vez más frecuente en el sector de la restauración, ya que proporcionan una herramienta innovadora y eficaz para gestionar los pedidos y el inventario. El hecho de mostrar los cargos del personal en tiempo real permite una comunicación rápida y precisa entre los equipos de sala y de servicio. Además, pueden ayudar a reducir errores y retrasos y a reducir el desperdicio y los costes de los alimentos al proporcionar un seguimiento y alertas de inventario en tiempo real.
- Sistema de reserva de mesas online: Estas plataformas digitales permiten reservar mesa sin necesidad de realizar llamadas telefónicas y minimiza el riesgo de error en la comunicación (malentendidos, dobles reservas…). Algunos ejemplos de plataformas que se encargan de realizar estos servicios o los implementan en sus funcionalidades podrían ser GloriaFood, Listae o OpenTable, entre otros.
- Asistente virtual o *chatbot* para restaurantes: Al contar con asistente virtual o *chatbot* con IA (inteligencia artificial), los restaurantes pueden automatizar tareas como tomar pedidos, hacer reservas y responder a preguntas comunes, liberando al personal para que se centre en ofrecer un servicio excepcional.

Además, los *chatbots* proporcionan información de valor para poder hacer un seguimiento de las preferencias y comportamientos de los clientes y adaptar sus ofertas en consecuencia. Son herramientas valiosas para mejorar la satisfacción del cliente y agilizar las operaciones.

- Tecnología futurista de reparto de comida: Desde drones hasta robots, las nuevas formas que surgen de repartir pedidos ofrecen un nivel único e incomparable de comodidad, eficiencia e innovación. Con el rápido avance de la tecnología de vehículos autónomos, es fácil imaginar un mundo en el que las comidas sean entregadas por vehículos no tripulados que puedan comunicarse con el móvil del cliente, avisarle de su llegada e incluso proporcionar un código único para acceder a la comida de forma segura. También los pequeños robots automatizados que llevan la comida a domicilio reducen los tiempos y costes de entrega, mejorando la precisión de los pedidos y la satisfacción del cliente.

Estas son solo algunas de las tecnologías y tendencias emergentes que están moldeando el futuro de las aplicaciones para restaurante. A medida que continuamos innovando y explorando nuevas oportunidades, podemos esperar que las aplicaciones para restaurantes se vuelvan aún más avanzadas, personalizadas y centradas en el cliente.

2.3. Tecnologías utilizadas en el desarrollo de aplicaciones móviles

En este apartado, se adentrará en las tecnologías fundamentales en el desarrollo de aplicaciones móviles. Se explorarán los lenguajes de programación, *frameworks*, herramientas de desarrollo y plataformas de despliegue que han sido cruciales en la creación de experiencias digitales modernas.

2.3.1. Lenguajes de programación

En el mercado de aplicaciones móviles, son dos los sistemas operativos que se enfrentan: Android y iOS. El primero es el más usado a nivel mundial, con una cuota de mercado del 70%, según Statcounter; también lo usan la mayoría de las marcas de telefonía móvil. El segundo es el sistema operativo de Apple que, aunque su cuota de mercado es significativamente inferior, su relevancia es similar.
El sistema operativo por el que se apueste determinará los lenguajes de programación que se deberán aprender.

Lenguajes de programación para aplicaciones Android

Hay que destacar que, de entrada, la programación para Android es mucho más accesible. Programar aplicaciones para este sistema operativo es posible desde cualquier ordenador, Microsoft o MacOS, y existen varios entornos de desarrollo con herramientas que facilitan esta tarea. Asimismo, se abre un amplio abanico de oportunidades a nivel global, ya que 7 de cada 10 smartphones utilizan este sistema operativo. Por lo tanto, si se decide este, estos son los lenguajes más usados habitualmente.

Java

Java es un lenguaje de programación orientado a objetos que sirve para desarrollar todo tipo de aplicaciones web, *mobile,* de sistemas, etc. Su carácter multiplataforma hace que tenga un amplio rango de aplicaciones. Entre estas, destaca el desarrollo de aplicaciones para Android.

De hecho, el mismo sistema operativo Android ha sido desarrollado con la tecnología Java. En este contexto, Java es más que uno de los lenguajes de programación para aplicaciones, se trata de todo un ecosistema de herramientas y técnicas, como la máquina virtual de java, que permite convertir un código desarrollado con este lenguaje en una aplicación nativa del software final.

Kotlin

Kotlin es otro de los lenguajes de programación para aplicaciones Android más usados por los desarrolladores. Es de tipado estático, orientado a objetos y, en varios aspectos, se inspira en Java, pero se trata de un lenguaje más sencillo e intuitivo.

Durante mucho tiempo, Java era el lenguaje de referencia para programar aplicaciones Android, hasta que la aparición de Kotlin en 2016 puso fin a su reinado. Actualmente, conviven ambos como los favoritos del desarrollo *mobile*, aunque la popularidad de Kotlin hizo que Google (dueño de Android) se declarase Kotlin First en 2019, es decir, recomienda programar sus apps, en adelante, exclusivamente con este lenguaje.

Lenguajes de programación para aplicaciones iOS

Paralelamente, si lo que se desea es desarrollar apps de forma nativa para iPhone, iPad y demás, Swift es lo que se debe aprender.

Antes de hablar sobre Swift, hay que mencionar Objective-C. Este era el principal lenguaje de programación para aplicaciones iOS y MacOS, hasta que surgió Swift en 2014. Objective-C se basa en la programación orientada a objetos, siendo dinámico y simple a la vez. Se trata de una de las influencias fundamentales de Swift, por ello, comparten varias características.

Swift es un lenguaje de programación multiparadigma, fuertemente tipado y de código abierto, creado por Apple para programar aplicaciones para sus sistemas operativos. En los últimos años, ha ido sustituyendo paulativamente a Objective-C, convirtiéndose en el lenguaje nativo de iOS, macOS y todos los *softwares* de esta familia tecnológica. Actualmente, basta con dominar Swift como lenguaje para programar para Apple.

2.3.2. *Frameworks* y librerías

Los frameworks y librerías desempeñan un papel fundamental en el desarrollo de aplicaciones móviles, permitiendo a los desarroladores acelerar el proceso de desarrollo, mejorar la eficiencia y crear experiencias de usuario de alta calidad. A continuación, se exploran algunos de los más populares.

- Flutter: Es un framework de código abierto y gratuito de Google que permite crear aplicaciones nativas para Android e iOS con base de código sencilla. Es un kit de desarrollo de software innovador para el desarrollo de aplicaciones multiplataforma. Se distingue por su nueva forma de crear aplicaciones nativas.
- Xamarin: Un marco de desarrollo de aplicaciones multiplataforma alternativo para el desarrollo de aplicaciones Android e iOS. Utiliza C#, por lo que las aplicaciones requieren menos líneas de código. Como resultado, el proceso de codificación es más rápido.
- React Native: Respaldado por Facebook, es un marco de desarrollo accesible que se ha convertido en la opción preferida de los programadores. Facilita el

desarrollo para Android e iOS. Aplicaciones conocidas como Skype, Airbnb, Tesla entre otros son claros ejemplos de aplicaciones que hacen uso de React.
- Ionic: Es un framework que utiliza HTML5 para desarrollar las aplicaiones móviles. Con las tecnologías web, los desarrolladores no necesitan molestarse en aprender sobre Java, Objective, Kotlin, C, etc. Ionic utiliza un marco front-end de AngularJS.

2.3.3. Herramientas de desarrollo

Las herramientas de desarrollo son indispensables para los equipos de desarrollo de aplicaciones móviles, proporcionando un entorno eficiente y colaborativo para crear, depurar y desplegar aplicaciones. En esta sección, se verán una variedad de herramientas utilizadas por los desarrolladores.

- Entornos de desarrollo:
  - Android Studio: IDE oficial para el desarrollo de aplicaciones Android.
  - Xcode: IDE oficial para el desarrollo de aplicaciones iOS y macOS
- Editores de código:
  - Visual Studio Code: Editor de código ligero y altamente personalizable.
  - Sublime Text: Editor de código rápido y eficiente con una gran comunidad de desarrolladores.
- Herramientas de depuración:
  - Android Debug Bridge (ADB): Herramienta de línea de comandos para depurar aplicaciones Android.
  - Xcode Debugger: Herramienta integrada en Xcode para depurar aplicaciones iOS.

2.3.4. Plataformas de despliegue

Las plataformas de despliegue permiten a los desarrolladores llegar a una amplia audiencia de usuarios de manera eficiente. A continuación, se detallan algunas de las principales plataformas de despliegue utilizadas por los desarrolladores.

- Google Play Store: La principal tienda de aplicaciones Android, donde los desarrolladores publican y distribuyen sus aplicaciones móviles para millones de usuarios en todo el mundo.
- Apple App Store: La tienda de aplicaciones oficial para dispositivos iOS, donde se distribuyen aplicaciones para los dispositivos de Apple.
- Firebase: Una plataforma de desarrollo de aplicaciones móviles de Google que ofrece servicios de alojamiento, bases de datos, autenticación de usuarios, análisis, mensajería en la nube y más.
- Microsoft Store: La tienda de aplicaciones de Microsoft para dispositivos Windows, donde los desarrolladores pueden publicar aplicaciones universales para Windows 10 y otros dispositivos Windows.

2.3.5. Herramientas de testing

Las herramientas de testing permiten a los desarrolladores identificar y corregir errores antes de que las aplicaciones sean lanzadas al mercado. A continuación, se ven algunas de las principales herramientas utilizadas por los desarrolladores.

- Appium: Una herramienta de automatización de pruebas de aplicaciones móviles que es compatible tanto con Android como iOS. Permite escribir pruebas en Java, Python, Javascript, etc.
- XCTest: Un framework de pruebas integrado en Xcode para realizar pruebas unitarias y de interfaz de usuario en aplicaciones iOS.
- Espresso: Un framework de pruebas de interfaz de usuario para aplicaciones Android, desarrollado por Google. Permite escribir pruebas en lenguaje Java o Kotlin.

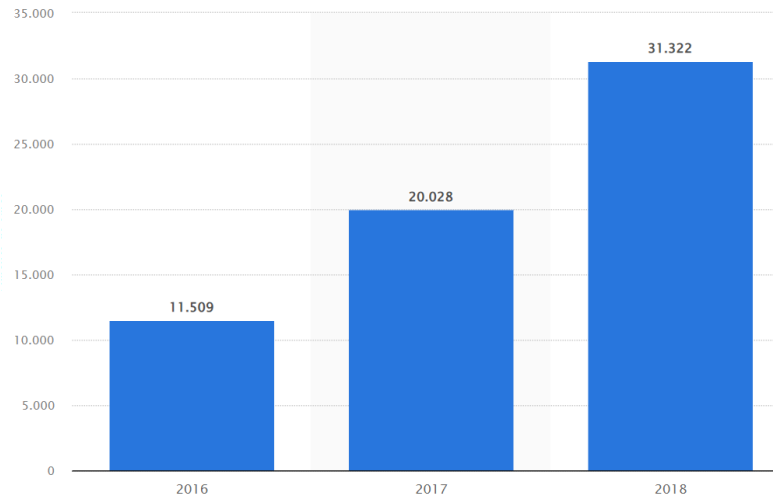
3. Estudio de viabilidad. Método DAFO.

En esta sección, se realizará una evaluación exhaustiva que permita determinar la viabilidad técnica, económica y operativa del proyecto antes de comprometer recursos significativos. Este análisis ayuda a tomar decisiones informadas sobre la viabilidad y el potencial éxito del proyecto antes de avanzar a etapas posteriores de desarrollo y despliegue.

Debilidades	Amenazas
• Dependencia de tecnologías externas • Curva de aprendizaje • Compatibilidad	• Competencia • Evolución tecnológica • Seguridad • Regulaciones
Fortalezas	**Oportunidades**
• Interfaz del usuario intuitiva • Arquitectura escalable • Documentación detallada	• Expansión del mercado • Retroalimentación de usuarios • Marketing digital • Colaboraciones y alianzas

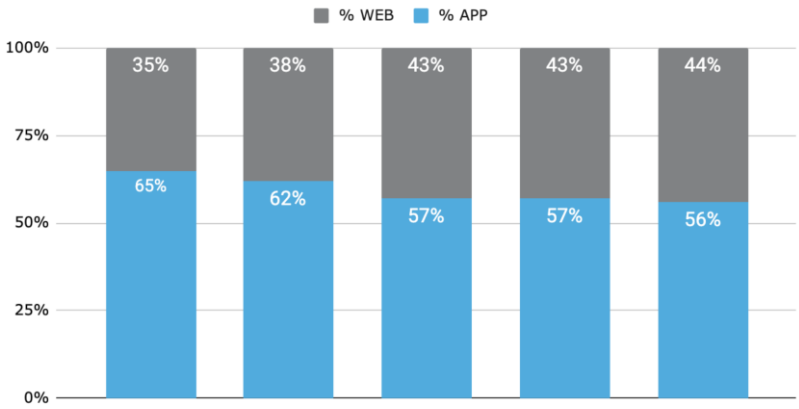
3.1. Estudio de mercado.

Según el último censo disponible, existen en España unas 248.000 empresas dedicadas a la restauración. De ellas, prácticamente dos terceras partes están gestionadas por personas físicas, es decir, autónomos o trabajadores por cuenta propia, con o sin asalariados. En 2018 había en España más de 31.300 presentes en plataformas online de reparto a domicilio o "delivery", según Abigail (2022, rescatado el 26/03/2024).



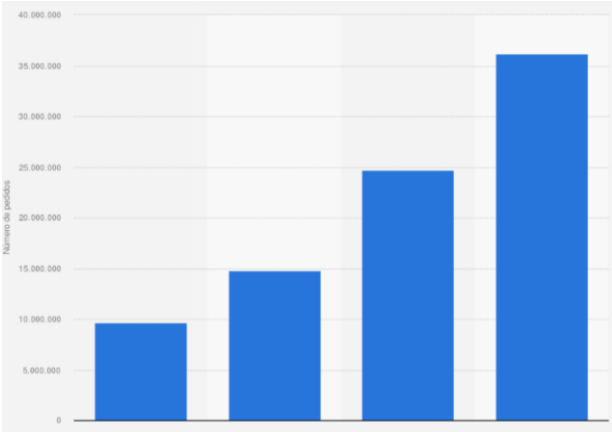
Estadística 1: Número de empresas adheridas a las plataformas online de reparto a domicilio en España entre 2016 y 2018

En los últimos años, casi el 60 % de los pedidos digitales se realizaron a través de aplicaciones móviles. Este hallazgo fue sorprendentemente consistente durante toda la pandemia e incluso en meses más recientes, a medida que las comidas en los restaurantes comenzaron a recuperarse. Este crecimiento no solo refleja una tendencia temporal provocada por la pandemia, sino un cambio en los hábitos de consumo que parece perdurar. (Restaurant Technology News, 2021)



Estadística 2: Comparación entre pedidos realizados desde apps y web.

El mercado de la ordenación de alimentos en línea está experimentando un crecimiento notable, con un aumento del 300% en los pedidos en los últimos años. Esto se atribuye a factores como la conveniencia y la diversidad de opciones ofrecidas por las aplicaciones móviles. Además, se señala que los restaurantes que ofrecen servicios de pedido en línea experimentan un aumento promedio del 30% en los ingresos. Se ofrecen recomendaciones para los propietarios de restaurantes sobre cómo mejorar sus operaciones de pedido en línea y la experiencia del cliente para mantenerse competitivos en este mercado en crecimiento (AppMySite, 2022).



Estadística 3: Número de pedidos en plataformas online de reparto a domicilio en España.

En áreas donde servicios populares como Uber Eats, Glovo y otros similares no están disponibles, existe una oportunidad significativa para la aplicación entre en competencia y llene ese vacío. Esta aplicación local puede ofrecer una ventaja competitiva al centrarse en las necesidades específicas de la comunidad y proporcionar una experiencia personalizada. Además, al operar en un mercado menos saturado, esta aplicación puede encontrar una base de usuarios leales y construir relaciones sólidas con los restaurantes locales. Sin embargo, no se puede subestimar la presencia y el alcance de plataformas como Uber Eats y Glovo en el mercado de la ordenación de alimentos en línea. Estas aplicaciones globales han establecido una fuerte presencia en numerosas ciudades y regiones, ofreciendo una amplia variedad de opciones de restaurantes y una interfaz intuitiva que facilita a los usuarios realizar pedidos en pocos clics.

Estas interfaces son conocidas por su diseño moderno y amigable, que permiten a los usuarios navegar fácilmente por los menús de los restaurantes, realizar pedidos y tener un seguimiento en tiempo real del estado de sus entregas. Además, estas aplicaciones suelen ofrecer funciones adicionales, como la posibilidad de programar pedidos con antelación, seguir recomendaciones personalizadas y recibir notificaciones sobre ofertas y promociones especiales.

Ilustración 9: Pantalla principal Glovo

Ilustración 7: Vista producto Glovo

Ilustración 8: Pantalla principal Uber Eats

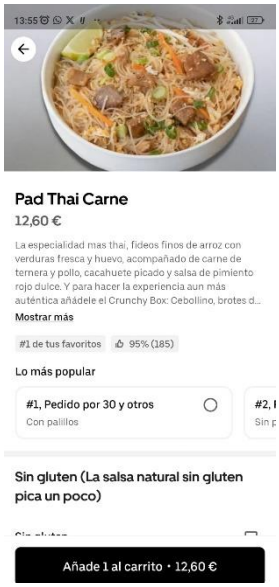


Ilustración 10: Vista producto Uber Eats

Al no tener suficiente información actual sobre la opinión de los posibles usuarios finales, se ha decidido realizar una encuesta para representar de una manera más real el interés que causa tener esta aplicación para realizar pedidos a domicilio. Los objetivos principales de esta encuesta son:

- Evaluar la necesidad de la aplicación.
- Identificar características más valoradas por los usuarios potenciales.
- Determinar la disposición de los restaurantes a adoptar una nueva plataforma.
- Medir la disposición de los consumidores a utilizar una nueva aplicación.

Para ello, se ha utilizado Google Forms, una herramienta que facilita la creación distribución y recopilación de respuestas. La encuesta está dirigida tanto a los consumidores que frecuentan el uso de servicios de entrega de comida a domicilio como a los que no suelen hacer uso de estos servicios. La duración de la encuesta fue de unos diez minutos.

Tras distribuir la encuesta para llegar al máximo número de usuarios posibles, se pudo obtener un total de 71 respuestas, una cifra lo suficiente alta para poder hacer deducciones ya que la población objetivo no es muy grande.

Los resultados preliminares de la encuesta mostraron un interés significativo en la aplicación, con más del 90% de los encuestados indicando que han pedido comida a domicilio en alguna ocasión. Entre estos usuarios, aproximadamente la mitad lo hace al menos una vez al mes, lo que sugiere una demanda estable y recurrente de servicios de entrega de comida a domicilio en la población objetivo.

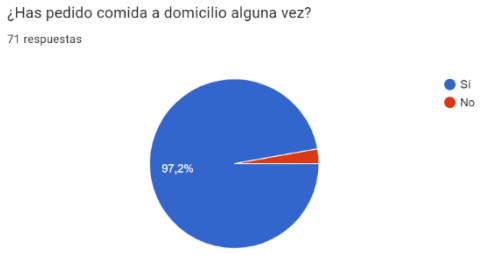


Gráfico de respuestas de la encuesta 2: ¿Has pedido comida a domicilio alguna vez?

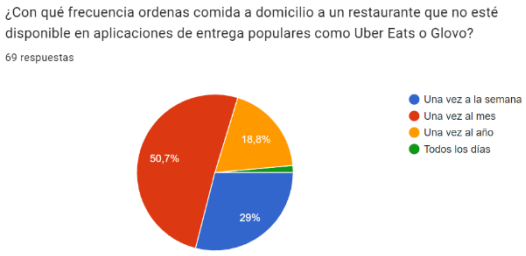


Gráfico de respuestas de la encuesta 1: Frecuencia en los pedidos a domicilio

Se preguntó a los usuarios que habían realizado pedidos anteriormente el método en el que suelen hacerlo, el resultado fue justo lo que se esperaba, con casi el 90% de respuestas indicando que los pedidos se realizan mediante llamada telefónica.

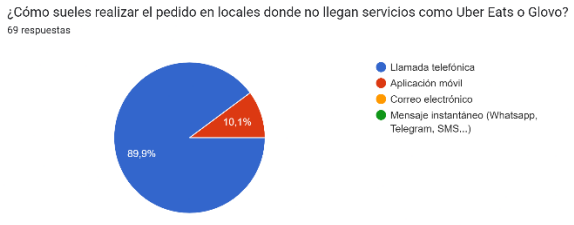


Gráfico de respuestas de la encuesta 3: Método utilizado para realizar los pedidos

Los usuarios hacen uso de el teléfono para realizar sus pedidos en la gran mayoría de los casos, por lo que también se pregunto cual era el motivo por el que se utiliza este método. Los resultados apuntan a una clara necesidad en los negocios, no existe otra

opción en más del 60% de los casos y los usuarios se ven obligados a utilizar esta metodología de ordenar pedidos.

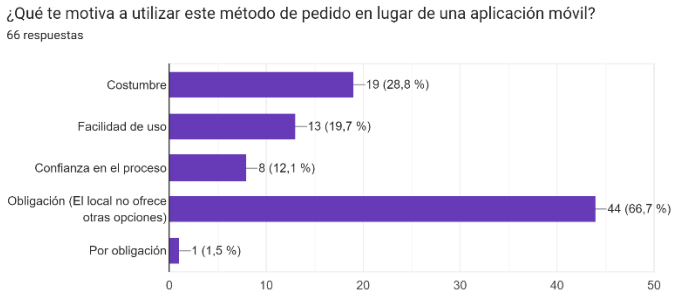


Gráfico de respuestas de la encuesta 4: Razón de usar un método diferente a una aplicación móvil

Muchos aspectos que la aplicación cubre son los que los usuarios necesitan, según las respuestas registradas. Los usuarios comentan que tienen algunas complicaciones al realizar pedidos ya que no pueden ver la carta, o que la comunicación con el teleoperador no es clara, o puede que no cojan el teléfono por tener la línea ocupada.

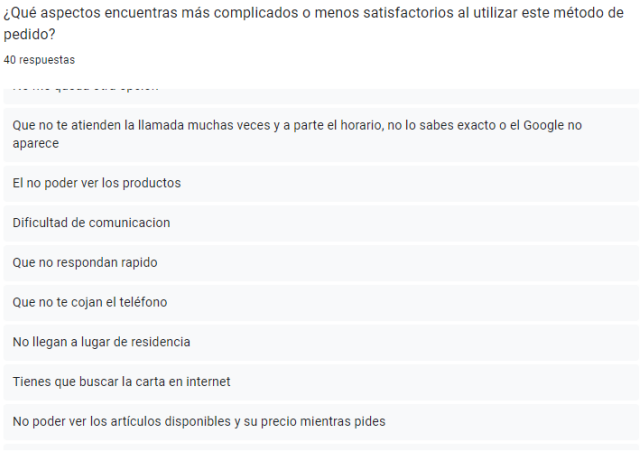


Gráfico de respuestas de la encuesta 5: Aspectos que complican o no satisfacen a los usuarios al utilizar las llamadas telefónicas para realizar los pedidos.

Se realizaron varias preguntas sobre los gustos del usuario, para así poder estudiar una solución que satisfaga sus necesidades. Este tipo de preguntas ayudaron también a determinar con seguridad aspectos de la aplicación que no pueden faltar una vez salga al mercado.

Para concluir la encuesta, se muestra en el Gráfico de respuestas de la encuesta 6 las respuestas obtenidas al proponer a los usuarios utilizar una aplicación móvil para realizar los pedidos en sus restaurantes locales. Los resultados apuntan a que la aplicación tendría un gran éxito en cuanto a descargas por parte de los usuarios, con más del 95% de las respuestas positivas.

Gráfico de respuestas de la encuesta 6: Disposición a probar una aplicación móvil para realizar pedidos en restaurantes locales

En resumen, los resultados de la encuesta revelan un alto nivel de aceptación y disposición por parte de los usuarios para adoptar una aplicación móvil para realizar pedidos en restaurantes locales. La amplia mayoría de los encuestados expresaron interés en probar la aplicación, lo que indica un fuerte potencial de demanda en el mercado objetivo.

Durante el proceso de evaluación, se identificaron diversos aspectos que influyen en la viabilidad y éxito del proyecto. Por ejemplo, la facilidad de acceso a subvenciones para proyectos de digitalización de Pymes proporciona una oportunidad valiosa para que las empresas adopten soluciones tecnológicas como la aplicación móvil propuesta para la gestión de pedidos en restaurantes locales. Este apoyo financiero puede mitigar los costos iniciales de implementación y fomentar una adopción más amplia de la tecnología por parte de los negocios del sector.

El poder de negociación con los clientes es un aspecto crucial para el éxito de la aplicación propuesta en este proyecto de digitalización de Pymes en el sector de la restauración. Dada la naturaleza altamente competitiva del mercado de aplicaciones móviles de pedidos de comida, es fundamental entender cómo este poder puede influir en la adquisición y retención de clientes. En el caso de la aplicación propuesta, su diferenciación radica en su enfoque en nichos de mercado donde no existen plataformas que ofrezcan servicios similares. Esta singularidad proporciona a la aplicación un poder de negociación sólido al ser la primera opción disponible para los consumidores en esas áreas específicas.

La ausencia de competidores directos en estas zonas le otorga a la aplicación una posición privilegiada para captar rápidamente la atención y el interés de los clientes potenciales. Además, al satisfacer una necesidad no cubierta previamente en el mercado, la aplicación puede generar una demanda orgánica y establecerse como la solución preferida para realizar pedidos de comida a domicilio en esas comunidades.

En resumen, el poder de negociación con los clientes y la facilidad para conseguir clientes se ven reforzados por el enfoque de nicho de mercado de la aplicación propuesta. Al dirigirse a áreas donde no existen competidores directos, la aplicación puede establecerse como la opción dominante y captar rápidamente la atención de los consumidores locales.

3.2. Planificación temporal o agenda de trabajo.

A lo largo de doce semanas, desde el 4 de marzo hasta el 24 de mayo, se asignarán 40 horas semanales para llevar a cabo todas las actividades necesarias, incluyendo la

definición de requisitos, diseño del sistema, desarrollo de la aplicación y servidor, pruebas, y la redacción de la memoria del proyecto.

La planificación está estructurada para abarcar todas las fases clave del ciclo de vida del desarrollo de software, garantizando que cada componente del sistema se desarrolle y se pruebe exhaustivamente. Además, se ha reservado tiempo para la redacción de la documentación y la preparación de la presentación final del proyecto. Esta planificación no solo facilitará el seguimiento del progreso del proyecto, sino que también permitirá realizar ajustes necesarios para asegurar la calidad y funcionalidad del sistema final.

Se presenta a continuacion una tabla con la planificacion temporal aproximada para la realizacion del proyecto.

4. Análisis de requisitos.

Se llevará a cabo un análisis de los requisitos necesarios para el diseño y desarrollo de nuestra aplicación para restaurantes. A través de un enfoque meticuloso, se identificarán las funciones clave que la aplicación debe ofrecer, así como los requisitos no funcionales que garantizarán su eficacia, seguridad y usabilidad.

4.1. Objetivos

Deben quedar claros los objetivos del proyecto, como se mencionaba en la introducción del proyecto, hay cuatro objetivos clave que esta aplicación debe de cubrir.

4.1.1. Agilizar los pedidos

El objetivo de agilizar los pedidos busca reducir el tiempo necesario para que los clientes realicen sus pedidos y reciban sus alimentos. Para lograr esto, la aplicación implementará un proceso de pedido eficiente y simplificado. La interfaz de usuario será optimizada para facilitar la navegación y la selección de productos, permitiendo a los clientes encontrar y ordenar sus platos preferidos con rapidez.

4.1.2. Fidelizar a los clientes

La fidelización de los clientes es un aspecto central de la aplicación. Para mantener a los clientes satisfechos y comprometidos a largo plazo, la aplicación ofrecerá una experiencia de usuario excepcional. Esto se logrará a través de una interfaz intuitiva y fácil de usar, junto con la provisión de incentivos atractivos como descuentos y promociones exclusivas. Además, la personalización jugará un papel crucial: la aplicación adaptará las ofertas y recomendaciones según las preferencias y el historial de pedidos de cada usuario. La recopilación constante de comentarios permitirá ajustar y mejorar el servicio de manera continua, asegurando que se cumplan y superen las expectativas de los clientes.

4.1.3. Mejorar la gestión de productos

Para optimizar la gestión de productos, la aplicación implementará varios mecanismos eficaces. Un sistema de gestión de inventario eficiente actualizará automáticamente el stock en tiempo real, evitando así problemas como la venta de productos agotados y mejorando la precisión en los pedidos. La automatización de los procesos de actualización de menús y productos garantizará que la información en la aplicación esté siempre actualizada y refleje cualquier cambio en la disponibilidad o precios de manera inmediata.

4.2. Requisitos funcionales

Aquí se detallan los requisitos funcionales de la aplicación. Estos describen las características y funcionalidades específicas que la aplicación debe cumplir para satisfacer las necesidades de los usuarios y alcanzar los objetivos del negocio.

1. **Inicio de sesión y registro de usuarios:** Se proporcionará al menos una opción de inicio de sesión y registro para facilitar el acceso (correo electrónico y contraseña).
2. **Navegación y búsqueda de productos:** Los usuarios podrán explorar fácilmente el catálogo de productos a través de la navegación por categorías y búsqueda avanzada, mejorando así la experiencia del usuario.
3. **Carrito de compras y comentarios:** Los usuarios podrán agregar productos al carrito de compras y dejar comentarios sobre los productos adquiridos, contribuyendo así a mejorar la experiencia de compra y proporcionar retroalimentación valiosa.
4. **Realizar pedidos:** Los usuarios podrán realizar su pedido en cuanto tengan listo su carro de la compra. Este requisito es imprescindible para que la aplicación tenga sentido.
5. **Funcionalidad multilenguaje:** La aplicación ofrecerá soporte para múltiples idiomas, lo que permitirá llegar a una audiencia global y mejorar la accesibilidad para usuarios de diferentes regiones lingüísticas.

Cada requisito será examinado en detalle, incluyendo su implementación técnica y su impacto en la experiencia del usuario. A través de la cumplimentación de estos requisitos, se espera que la aplicación logre ofrecer una experiencia de compra satisfactoria y fomente la fidelidad de los usuarios.

4.3. Requisitos no funcionales

En esta sección se exploran los aspectos esenciales del diseño de la aplicación, más allá de sus funcionalidades básicas. Estos requisitos, conocidos como requisitos no funcionales, definen cómo la aplicación debe comportarse y adaptarse en diferentes contextos,. Ahora, se detallan los pilares fundamentales del diseño de la aplicación:

1. **Flexibilidad de visualización:** La aplicación debe ser versátil en su presentación, adaptándose de manera fluida a una amplia gama de dispositivos y tamaños de pantalla. Esta capacidad garantiza una experiencia de usuario coherente y atractiva, independientemente del dispositivo utilizado.
2. **Seguridad y confidencialidad:** La protección de los datos del usuario es una prioridad absoluta. La aplicación implementará medidas robustas de seguridad, incluida la encriptación de datos sensibles, el cumplimiento de regulaciones de privacidad y métodos de autenticación solidos para salvaguardar la información del usuario.
3. **Elasticidad y escalabilidad:** Ante el crecimiento potencial del tráfico y la demanda de usuarios, la aplicación debe expandirse sin comprometer su rendimiento. La arquitectura subyacente se diseñará para escalar de manera eficiente, garantizando una experiencia fluida incluso en periodos de alta actividad.

Estos aspectos no solo enriquecen la funcionalidad de la aplicación, sino que también establecen estándares de excelencia en términos de seguridad, adaptabilidad y rendimiento.

4.4. Casos de uso

Los casos de uso son una herramienta fundamental en el análisis y diseño de sistemas de software, permitiendo capturar de manera sistemática los requisitos funcionales del sistema desde la perspectiva de los usuarios. A continuación, se exploran los casos de aplicación móvil, que representan las interacciones entre actores y sistema para lograr objetivos específicos.

Caso de uso	1	Crear cuenta
Actor	Usuario básico	
Objetivo	Requisito funcional 1: Registro de usuarios	
Secuencia	Paso	Acción
	1	El usuario inicia la aplicación y solicita pasar a la pantalla de registro
	2	El usuario rellena el formulario de registro y pulsa en continuar.
	3	El sistema comprueba que los datos sean válidos y envía los datos al servidor, en caso contrario devuelve al usuario al paso 2.
	4	El sistema informa del resultado de la operación.

Caso de uso	2	Iniciar sesión
Actor	Usuario básico	
Objetivo	Requisito funcional 1: Inicio de sesión	
Secuencia	Paso	Acción
	1	El usuario inicia la aplicación y rellena el formulario de inicio de sesión.
	2	El usuario pulsa en el botón de inicio de sesión
	3	El sistema realiza una petición al servidor para comprobar que las credenciales existen y coinciden en la base de datos.
	4	El sistema guarda el token de autenticación y abre la pantalla principal. En caso de error en las credenciales el sistema informa al usuario.

Caso de uso	3	Explorar carta
Actor	Usuario básico	
Objetivo	Requisito funcional 4: Navegación y búsqueda de productos	
Precondición	El usuario debe estar conectado (Haber hecho el proceso de inicio de sesión).	
Secuencia	Paso	Acción
	1	El sistema muestra la pantalla principal
	2	El usuario navega entre la lista de productos y categorías.

Caso de uso	4	Filtrar productos
Actor	Usuario básico	

Objetivo	Requisito funcional 4: Navegación y búsqueda de productos	
Precondición	El usuario debe estar conectado (Haber hecho el proceso de inicio de sesión).	
Secuencia	**Paso**	**Acción**
	1	El sistema muestra la pantalla principal
	2	El usuario hace uso de el botón de búsqueda o del slider de categorías.
	3	El usuario aplica sus términos de búsqueda o pulsa en una categoría.
	4	El sistema muestra los resultados de la búsqueda.

Caso de uso	5	Ver producto en detalle
Actor	Usuario básico	
Objetivo	Requisito funcional 4: Navegación y búsqueda de productos	
Precondición	El usuario debe estar conectado (Haber hecho el proceso de inicio de sesión).	
Secuencia	**Paso**	**Acción**
	1	El usuario pulsa sobre un producto
	2	El sistema abre la pantalla de ver producto
	3	El usuario puede ver una vista detallada del producto.

Caso de uso	6	Personalizar producto
Actor	Usuario básico	
Objetivo	Requisito funcional 4: Navegación y búsqueda de productos	
Precondición	Ver producto en detalle	
Secuencia	**Paso**	**Acción**
	1	El usuario explora entre las diferentes opciones de personalización del producto (añadir o quitar ingredientes, añadir un comentario...)
	2	El usuario pulsa sobre cualquier opción de personalización disponible
	3	El sistema modifica el ingrediente seleccionado

Caso de uso	7	Añadir producto al carrito
Actor	Usuario básico	
Objetivo	Requisito funcional 5: Carrito de compras	
Precondición	Ver producto en detalle / Personalizar producto	
Secuencia	**Paso**	**Acción**
	1	El usuario selecciona la cantidad que desea añadir al carrito
	2	El usuario pulsa sobre el botón “Añadir al carrito”

	3	El sistema añade el producto a una línea del pedido, con la cantidad seleccionada

Caso de uso	8	Realizar pedido
Actor	Usuario básico	
Objetivo	Requisito funcional 6: Realizar pedidos	
Precondición	Añadir producto al carrito	
Secuencia	**Paso**	**Acción**
	1	El usuario pulsa el botón “Tramitar pedido”
	2	El usuario selecciona la dirección donde se hará el envío
	3	El sistema registra el pedido e informa al usuario.

Caso de uso	9	Publicar un comentario
Actor	Usuario básico	
Objetivo	Requisito funcional 4: Navegación y búsqueda de productos	
Precondición	Realizar pedido	
Secuencia	**Paso**	**Acción**
	1	El usuario abre el menú desplegable
	2	El usuario pulsa sobre el botón “Pedidos”
	3	El sistema abre la pantalla de pedidos.
	4	El usuario selecciona el pedido que desea comentar
	5	El sistema abre la pantalla de ver pedido
	6	El usuario pulsa en “Publicar comentario/valoración”
	7	El sistema abre un *dialog* para publicar el comentario/valoración
	8	El usuario escribe el comentario o valora el pedido y pulsa sobre “Publicar”
	9	El sistema registra este comentario/valoración e informa al usuario.

Para complementar esta información, se presenta a continuación el diagrama de casos de uso, que proporciona una visualización simplificada pero completa de todas las interacciones entre los actores y el sistema.

Diagrama 1: Casos de uso

5. Diseño.

En esta sección, se presenta el diseño detallado del sistema, que comprende la arquitectura general, la estructura de datos y los principales componentes que conforman la aplicación. Se abordarán aspectos como el diseño de la base de datos, la interfaz del usuario, la interacción entre los distintos componentes, etc. Se presentarán diagramas y descripciones para facilitar la comprensión del diseño.

5.1. Arquitectura del sistema

La arquitectura del sistema de la aplicación se basa en un modelo cliente-servidor, donde la aplicación móvil actúa como el cliente que se comunica con un servidor centralizado. Este servidor, a su vez, gestiona la lógica empresarial y la interacción con la base de datos.

- Cliente

La aplicación móvil, instalada en dispositivos de usuarios finales, proporciona la interfaz a través de la cual los usuarios interactúan con el sistema. Se encarga de enviar solicitudes al servidor y mostrar la información recibida de manera adecuada para la visualización y la interacción del usuario.

- Servidor

El servidor constituye el núcleo del sistema, actuando como intermediario entre los clientes y la base de datos. Gestiona las solicitudes entrantes de los clientes, procesa la lógica de negocio correspondiente y coordina el acceso a los datos almacenados en la base de datos. Además, se encarga de mantener la seguridad, integridad y el rendimiento del sistema en su conjunto.

- Base de datos

La base de datos almacena de manera persistente la información necesaria para el funcionamiento de la aplicación. El servidor accede a esta base de datos para recuperar y almacenar datos según las solicitudes de los clientes. Se hablará sobre la estructura de esta más adelante.

La comunicación entre el cliente y el servidor, así como entre el servidor y la base de datos, se realiza a través de protocolos de comunicación estándar, como HTTP para las solicitudes web o TCP/IP para la comunicación en red. Este enfoque de arquitectura cliente-servidor proporciona una separación clara de responsabilidades y permite una escalabilidad eficiente a medida que crece la aplicación y se agregan más usuarios y funcionalidades.

En el Diagrama 2, se presenta el diagrama de arquitectura del sistema que ilustra la estructura y las interacciones entre los componentes clave de la aplicación cliente-servidor.

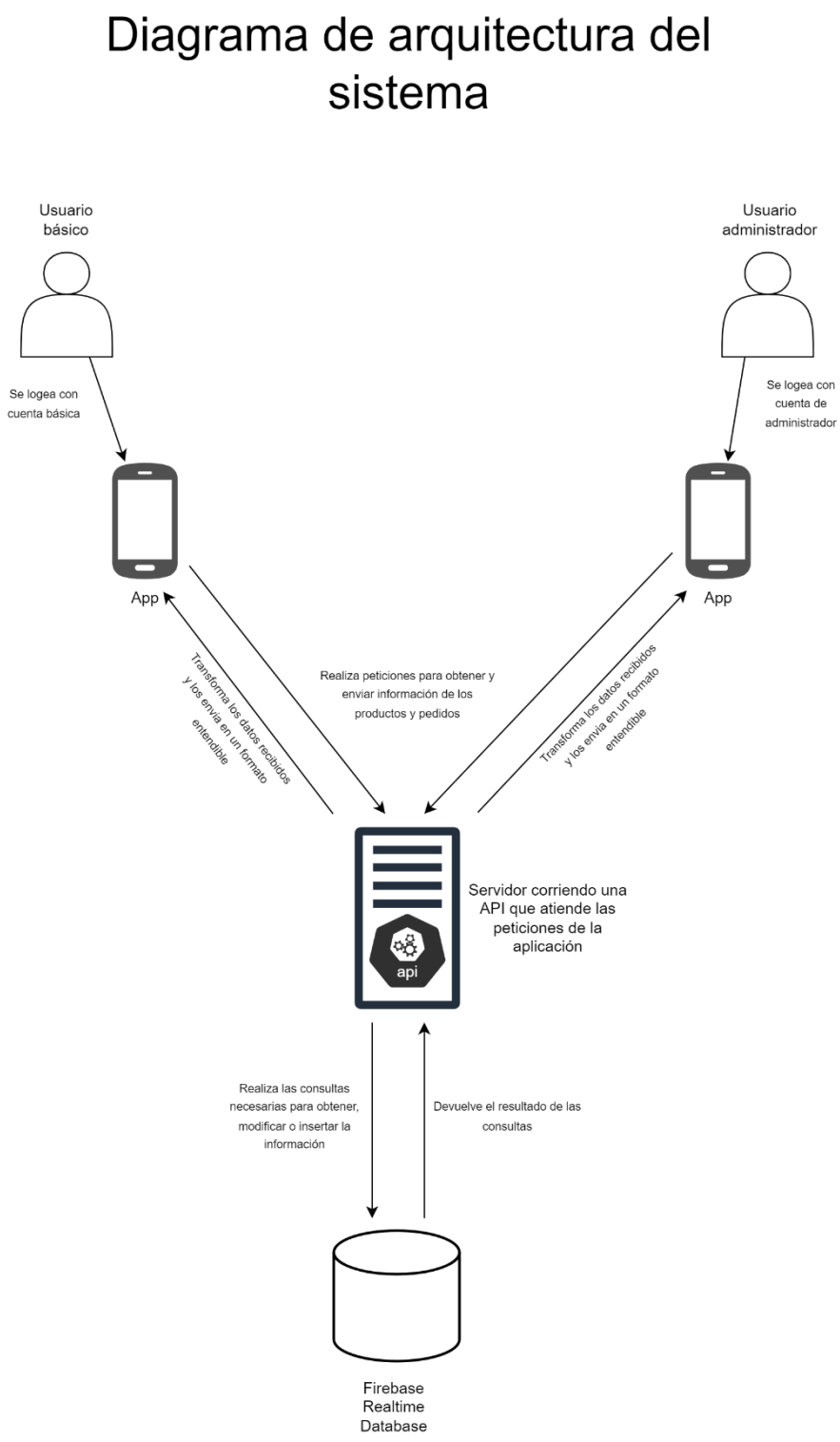


Diagrama 2: Arquitectura del sistema

El diagrama muestra claramente la separación de responsabilidades entre el cliente, el servidor y la base de datos. Las flechas indican la dirección de las comunicaciones, destacando cómo el cliente envía solicitudes al servidor, que a su vez interactúa con la base de datos para recuperar o almacenar información según sea necesario.

5.2. Interfaz del usuario

La interfaz de usuario desempeña un papel fundamental en la experiencia del usuario de nuestra aplicación. En esta sección se puede ver el diseño detallado de la interfaz de usuario, que incluye la disposición de los elementos visuales, la navegación entre pantallas y la interacción del usuario con la aplicación.

Para establecer una base solida para el diseño de la interfaz, se ha decidido realizar la creación de *wireframes* iniciales. Estos bocetos son representaciones esquemáticas de baja fidelidad que muestran la disposición general de los elementos en cada pantalla de la aplicación.

- Pantalla de inicio de sesión

La pantalla de inicio de sesión es la primera pantalla que ven los usuarios al abrir la aplicación. Aquí, los usuarios pueden ingresar sus credenciales para acceder a sus cuentas o registrarse si es necesario.

En el *Wireframe* 1, se pueden observar las tres opciones que el usuario puede realizar. El botón de iniciar sesión envía al usuario a la pantalla principal o *Home* y el de registrar al formulario de registro, finalmente existe la opción de recuperar la contraseña.

- Pantalla de registro

Esta pantalla se abre al pulsar el botón registrar en la pantalla de inicio de sesión. Permite al usuario rellenar el formulario de registro y crear una nueva cuenta. Cuenta con la opción de cancelar el registro y devuelve al usuario a la pantalla de inicio de sesión.

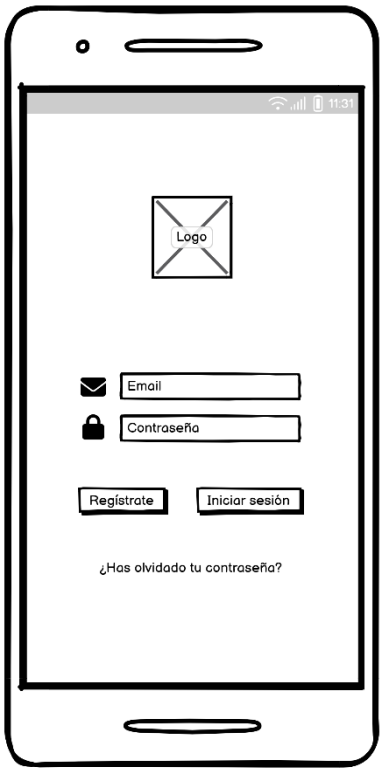
En el Wireframe 2, se puede ver un ejemplo de la pantalla de registro.

- Pantalla de inicio o principal

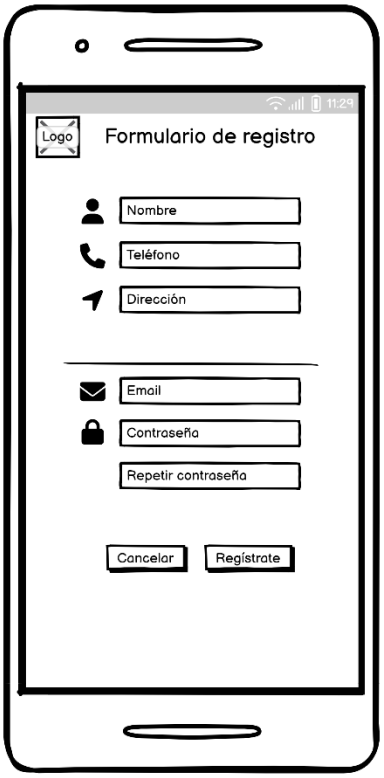
La pantalla de inicio o principal cuenta con varias opciones. Tiene un menú desplegable, el cual se puede desplegar desde el botón superior derecho y muestra tanto el menú de navegación entre las diferentes pantallas de la aplicación, como el carrito de la compra para tramitar un pedido.

En el Wireframe 3 se ve la pantalla principal, donde muestra el catálogo de productos junto a una sección de productos relevantes o recomendados. Contiene una barra de búsqueda para filtrar los productos por término. Por otro lado, en el Wireframe 4 se ve la misma pantalla, pero después de pulsar sobre el botón que desplegaría el menú, de forma que el usuario pueda ver las diferentes opciones y navegar por la aplicación.

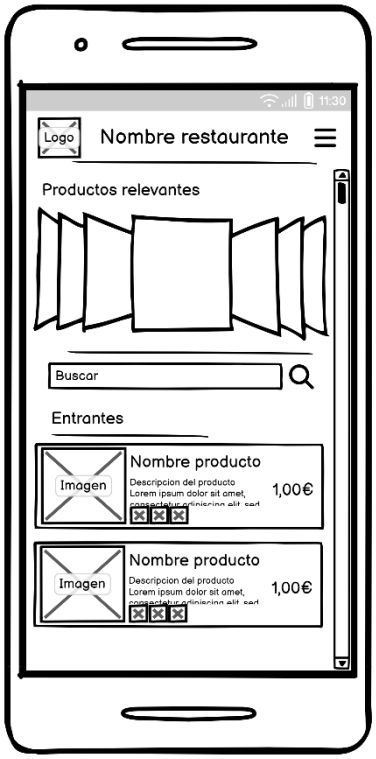
Cabe destacar que en el menú desplegable se ve una opción llamada “Panel administración” que tan solo verán los usuarios de tipo administrador. Se debe recordar también que el menú estará disponible en todas las pantallas una vez iniciada la sesión, de forma que el usuario pueda navegar entre todas las pantallas que la aplicación ofrece.



Wireframe 1: Pantalla de inicio de sesión



Wireframe 2: Pantalla de registro



Wireframe 3: Pantalla principal

Wireframe 4: Menú lateral desplegado

Como se ha comentado, desde el menú de la pantalla principal se podrá acceder a las distintas pantallas de la aplicación, a continuación se muestran las más relevantes, como la vista de un producto en detalle o la lista de pedidos del usuario.

- Pantalla para ver un producto en detalle

La pantalla que permite ver un producto en detalle, es la misma que permite añadir nuestro producto al carrito de la compra. Desde aquí, el usuario tiene más información sobre el producto en relacion a la que se ve en la pantalla de inicio, como la descripcion completa, los alérgenos que contiene (junto al botón de ayuda que mostrará una leyenda) o las imágenes del producto. Al añadir el producto al carrito de la compra, no se redirige al usuario a ninguna pantalla.

En el Wireframe 5 se ve un ejemplo de la vista que nos permite ver un producto en más detalle.

- Pantalla para ver la lista de pedidos del usuario.

En esta vista, el usuario puede ver el historial de pedidos que ha realizado, y entrar a ver en detalle cada uno de ellos. En caso de no haber realizado ningún pedido, la pantalla mostrará un mensaje indicandolo, y un botón que incita al usuario a regresar a la pantalla principal sin tener que desplegar de nuevo el menú. En el Wireframe 6 se puede ver los dos ejemplos de la pantalla de pedidos, tanto en caso de no tener pedidos realizados, como el caso en el que hay varios.

Wireframe 5: Pantalla de vista de un producto

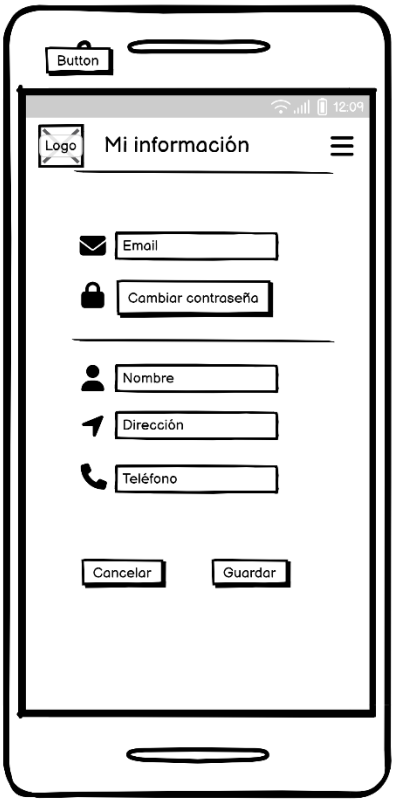
Wireframe 6: Pantalla donde el usuario puede consultar sus pedidos anteriores

El usuario también podrá editar su propia información, por lo que al pulsar la opción “Mi información” del menú desplegable se abrirá su correspondiente pantalla.

- Pantalla de información del usuario

Esta pantalla carga los datos del usuario y los muestra en diferentes cuadros de texto que son editables, por lo que el usuario puede cambiar cualquier campo y posteriormente guardar esta información. También cuenta con un botón para abrir el formulario de cambio de contraseña, en el cual se requiere introducir la contraseña actual, la nueva, y repetir la nueva por seguridad. Tras pulsar el botón de guardar se informa al usuario de el resultado de la operación y se redirige a éste a la pantalla principal en caso de éxito.

En el Wireframe 7 se puede observar la estructura que se ha decidido para la pantalla de información del usuario.



Wireframe 7: Pantalla donde ele usuario puede modificar su información.

Estas son solo algunas de las pantallas principales que componen la interfaz de usuario de la aplicación móvil. Cada pantalla ha sido diseñada con atención al detalle y centrada en las necesidades y expectativas de los usuarios finales.

5.3. Diseño de la base de datos

Este apartado se adentra en el diseño detallado de la base de datos que respalda la aplicación. Se verán aspectos como el modelo de datos, las relaciones entre las diferentes entidades y la seguridad e integridad de datos.

5.3.1. Modelo de datos

En el diseño de la base de datos, se utiliza Firebase Firestore, que adopta un enfoque basado en documentos y permite almacenar datos de manera flexible y escalable. Firebase Firestore, es una base de datos NoSQL en la nube que organiza los datos en colecciones que contienen documentos, y cada documento puede contener campos de valor o incluso subcolecciones anidadas.

Para este modelo de datos, se identifican las entidades principales de la aplicación y se representan como colecciones. Cada entidad se corresponde con una colección y los registros individuales se representan como documentos dentro de esas colecciones.

En el Diagrama 3 podemos observar el diagrama de colecciones de la base de datos de la aplicación. Éste ayudara a la compresion de la estructura de las colecciones y sus documentos.

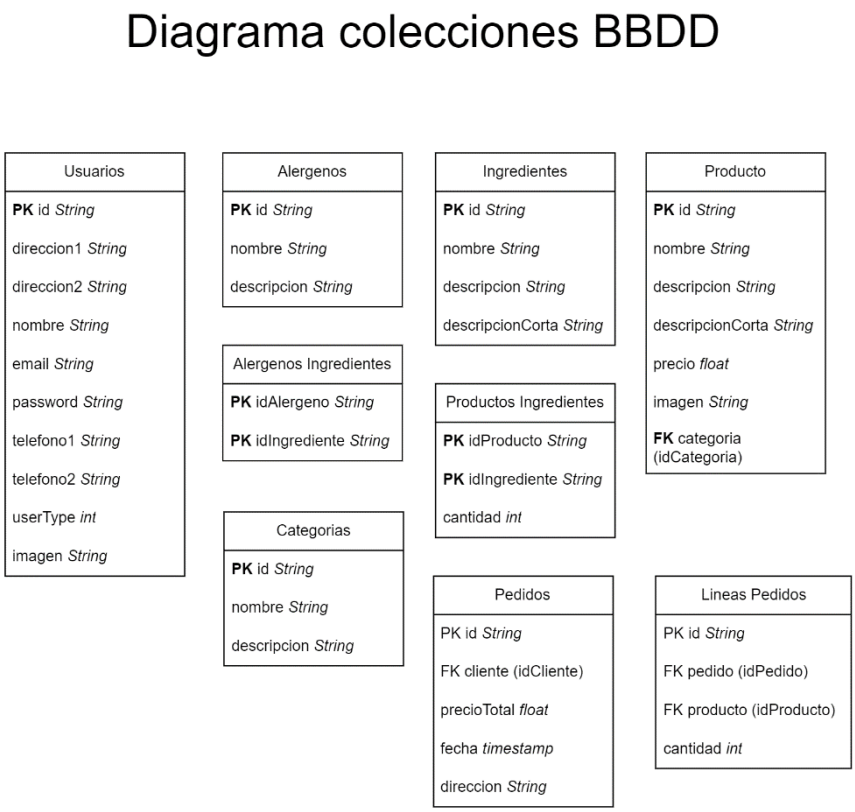


Diagrama 3: Colecciones de la base de datos.

Para representar las relaciones entre las entidades principales de la aplicación, se utiliza un diagrama entidad-relación. Este diagrama proporciona una visualización clara de cómo interactúan las diferentes entidades en la base de datos. Puede verse en el Diagrama 4.

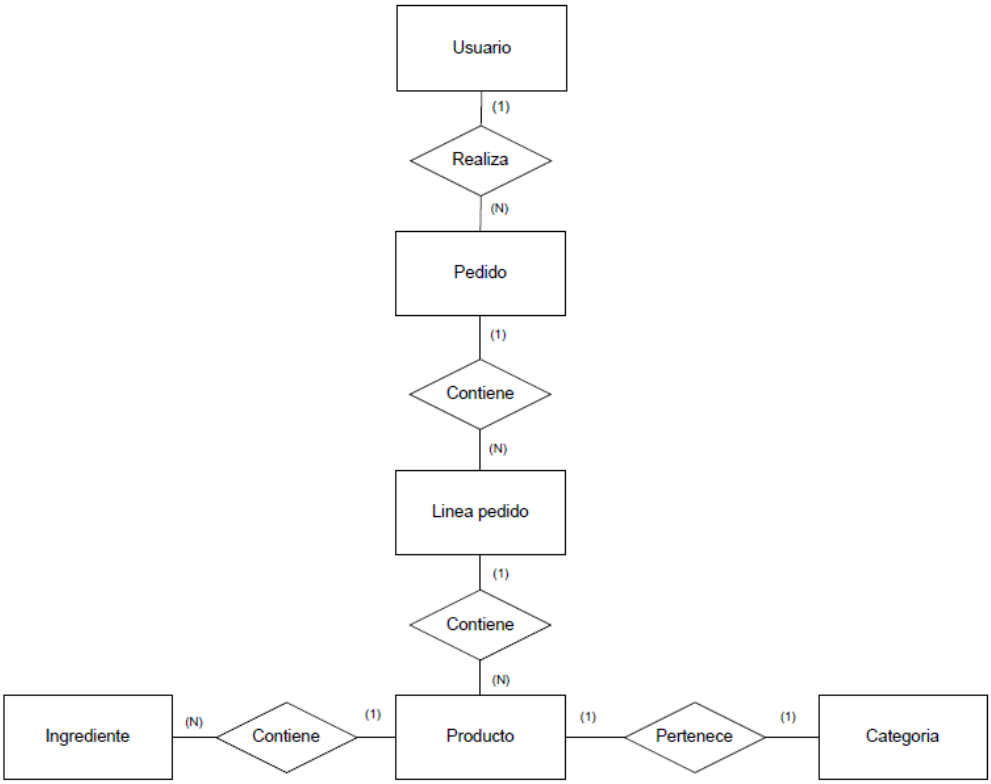


Diagrama 4: Entidad-relación

5.3.2. Seguridad de los datos

La seguridad de los datos es una consideración fundamental en el diseño e implementación de la aplicación. Dado que se está utilizando una arquitectura cliente-servidor, donde la aplicación móvil se comunica con el servidor que a su vez interactúa con la base de datos, se han tenido en cuenta algunas medidas de seguridad para proteger la confidencialidad, integridad y disponibilidad de los datos almacenados.

- Acceso controlado al servidor: En la arquitectura del sistema, la única entidad que tiene acceso directo a la base de datos es el servidor. Esto significa que toda la interacción, como consultas y modificaciones de datos, se realizan a través del servidor. Esta configuración reduce el riesgo de acceso no autorizado a la base de datos y garantiza que todas las operaciones de acceso a los datos se realicen de manera controlada y segura.
- Autenticación y autorización: Se ha implementado un sistema de autenticación robusto en el servidor para verificar la identidad de los usuarios que intentan acceder a la aplicación móvil. Cada usuario debe iniciar sesión con credenciales validadas antes de poder utilizar la aplicación. Además, el servidor verifica los permisos y roles de cada usuario para garantizar que solo tengan acceso a los datos y funciones autorizados.
- Encriptación de datos: Todos los datos transmitidos entre la aplicación móvil y el servidor, así como entre servidor y la base de datos, se encriptan utilizando protocolos de seguridad estándar. Esto protege los datos de posibles intercepciones o manipulaciones durante la transmisión, garantizando la confidencialidad y la integridad de la información.

Con la implementación de estas medidas de seguridad, se asegura el cumplimiento del requisito no funcional de seguridad y confidencialidad.

5.4. Consideraciones de mantenimiento

Las consideraciones de mantenimiento son muy importantes para garantizar la longevidad y la escalabilidad de la aplicación. Al diseñar el sistema, se ha tenido en cuenta varios aspectos que facilitarán su mantenimiento continuo y su evolución futura.

- Modularidad del código: Se ha adoptado un enfoque modular en el desarrollo de la aplicación, dividiendo el código en componentes independientes y reutilizables. Esto facilita la compresión y la modificación del código, así como la incorporación de nuevas funcionalidades sin afectar al resto del sistema.
- Documentación adecuada: Se ha prestado especial atención a la documentación del código y del sistema en su conjunto. Esto incluye comentarios claros y concisos en el código fuente, así como documentación técnica detallada que describe la arquitectura, el funcionamiento y los procedimientos de mantenimiento del sistema. La documentación adecuada facilita la colaboración entre desarrolladores y el diagnóstico y solución de problemas durante el mantenimiento del sistema.
- Facilidad de extensión y actualización: Se han establecido interfaces claras y bien definidas entre los diferentes componentes del sistema, lo que facilita la integración de nuevas funcionalidades y la realización de actualizaciones sin afectar al funcionamiento existente.

Considerar estas medidas de mantenimiento desde las primeras etapas del diseño, asegura que la aplicación sea robusta, escalable y fácil de mantener a lo largo del tiempo.

6. Codificación.

En esta sección se adentra en el proceso de desarrollo y codificación de la aplicación. Se detallan las tecnologías seleccionadas, la documentación interna del código y el manual del usuario para ofrecer una visión completa del trabajo realizado en esta etapa.

6.1. Tecnologías elegidas y su justificación.

Se presentan las tecnologías seleccionadas para el desarrollo de la aplicación, tanto para la aplicación móvil como para el servidor.

- **Lenguaje de programación:**
    - **Aplicación móvil:** Se ha elegido Java como lenguaje de programación para la aplicación móvil debido varias razones, entre ellas, su amplia documentación y comunidad de desarrolladores, así como su portabilidad, seguridad y rendimiento. Java cuenta con características para manejar automáticamente la memoria y el sistema de gestión de permisos, que ayudan a proteger nuestra aplicación contra vulnerabilidades. El hecho de que Java sea compatible con múltiples plataformas, permite que la aplicación funcione en una amplia variedad de dispositivos Android sin necesidad de realizar cambios en el código. Cabe destacar que la documentación y comunidad mencionadas anteriormente, han sido muy importantes en la elección del lenguaje de programación, ya que ayuda mucho a la detección y solución de

problemas que pueden surgir durante el desarrollo.

- **Servidor**: Después de una pequeña investigación, se ha elegido Javascript como lenguaje de programación, ya que es uno de los lenguajes más populares en la web, lo que significa que hay un montón de recursos, bibliotecas y frameworks disponibles para facilitar el desarrollo. Su compatibilidad con Firebase Firestore, la base de datos, y su sintaxis sencilla ayudan a que el desarrollo no tenga que centrarse tanto en entender su complejidad.

- **Base de datos:** Se ha elegido Firebase Firestore, que es una base de datos NoSQL en la nube, algunas de las razones por las que se ha elegido son:

  - **Escalabilidad:** Está diseñado para escalar automáticamente según las necesidades de la aplicación, lo que garantiza un rendimiento consistente incluso en aplicaciones con grandes volúmenes de datos y alto tráfico de usuarios.
  - **Tiempo real:** Proporciona sincronización en tiempo real entre clientes y servidores, lo que permite actualizaciones instantáneas en la aplicación móvil sin esperar actualizaciones periódicas.
  - **Facilidad de uso:** Ofrece una interfaz de usuario intuitiva y una API simple de usar, lo que facilita la configuración y el mantenimiento de la base de datos sin requerir conocimientos especializados en administración de bases de datos.

- **Frameworks y librerias:** Se ha elegido Express.js, un framework de servidor para Node.js, para construir una API RESTful. Express es conocido por su flexibilidad, rendimiento y amplia comunidad de desarrolladores. Facilita la creación de endpoints y el manejo de solicitudes HTTP. Su sencillez lo ha hecho ideal para desarrollar el servidor sin problemas con el nivel de experiencia.
- **Entornos de desarrollo (IDE):**
  - **Aplicación móvil:** Se ha seleccionado Android Studio para el desarrollo de la aplicación móvil, por sus herramientas avanzadas de edición de código, depuración y emulación. Una de las grandes razones de elección es que ya se tenia experiencia con este entorno de desarrollo.
  - **Servidor:** Visual Studio Code (VSCode) ha sido el entorno seleccionado para desarrollar el servidor, debido a su facilidad de uso y amplia gama de extensiones que mejoran la funcionalidad del editor. También se tenía experiencia con este entorno y esto ha sido una de las principales razones de su elección.

Al seleccionar cuidadosamente estas tecnologías, se asegura tener un conjunto sólido y confiable de herramientas para el desarrollo y la operación de la aplicación móvil y su infraestructura de servidor.

6.2. Documentación interna del código.

Este apartado se centra en la documentación interna del código, que abarca tres aspectos principales: la estructura del proyecto, los comentarios en el código y la generación automática de documentación. Estos elementos son fundamentales para garantizar que el código sea claro, comprensible y fácil de mantener para los desarrolladores presentes y futuros.

6.2.1. Estructura del proyecto

Empezando por el servidor, tiene una estructura un poco menos cuidada, pero no por ello menos comprensible. La estructura de un proyecto JavaScript puede variar según las necesidades y la complejidad del proyecto, pero en algunos casos, como este, puede ser bastante simple. A continuación, se presenta una descripción de la estructura básica del proyecto, que consta de dos archivos principales:

- **Archivo "app.js":** Este archivo es el punto de entrada principal del servidor. Aquí se encuentra la lógica principal, incluyendo la creación de la aplicación Express, la definición de las rutas de la API y la configuración de cualquier middleware necesario.
- **Archivo "config.js":** En este archivo se encuentran las variables de configuración utilizadas por el servidor. Puede incluir información como el puerto en el que el servidor escucha las solicitudes, la configuración de la base de datos, las claves de API, etc.

Aunque esta estructura es muy simple, proporciona una base sólida para el desarrollo y la ejecución del servidor. A medida que el proyecto crezca es posible que sea necesario agregar más archivos y directorios para organizar el código de manera más efectiva.

Por otro lado, en el caso de la aplicación móvil, la estructura que sigue se construye a partir de la base que Android Studio proporciona. Cuando se crea un nuevo proyecto, se establece una estructura básica que organiza los archivos y recursos de la aplicación de manera coherrente y fácil de entender.

La estructura tipica de un proyecto en Android Studio incluye directorios como “java” para el código fuente de la aplicacion, “res” para recursos como diseños de interfaz de usuario, imagenes y archivos de valores, y “Gradle Scripts” para los archivos de configuración de ensamblado. Además se pueden encontrar otros directorios como “manifests” para el archivo AndroidManifest.xml y “build” para archivos generados durante el proceso de compilación. En la Ilustación 10 se puede ver un ejemplo de la estructura básica que se genera al crear un nuevo proyecto.

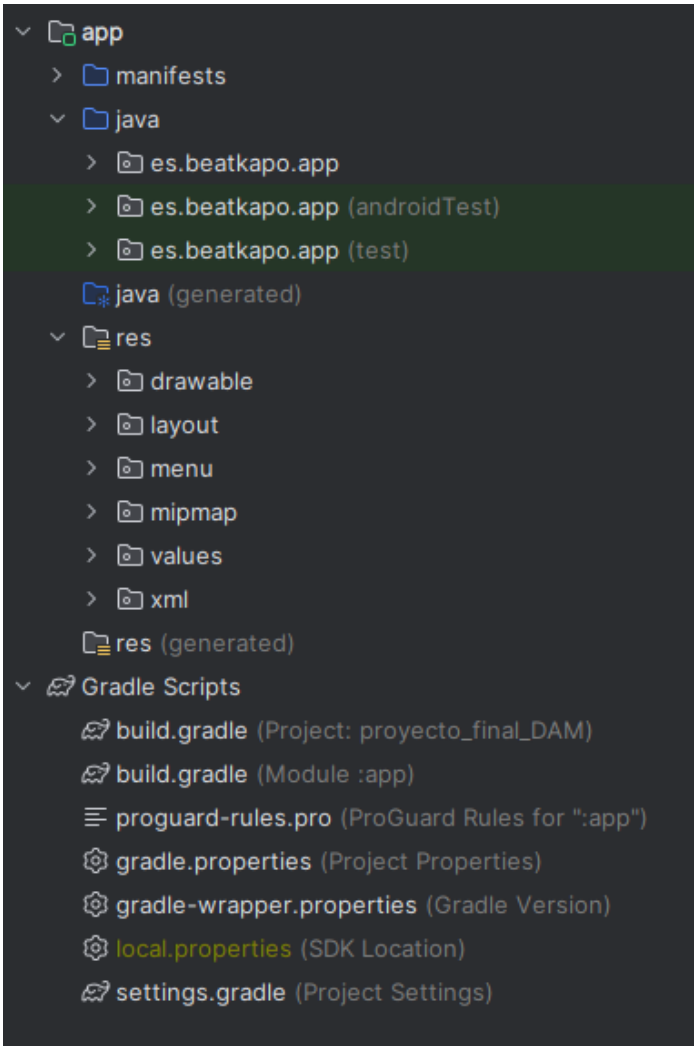


Ilustración 11

Dentro del directorio “java”, Android Studio crea automáticamente un paquete base para la aplicación, en este caso, “es.beatkapo.app”. Aquí es donde se almacenan las clases de las actividades principales, los controladores se organizan a la altura de este paquete.

Además del paquete base, se ha dividido la aplicacion en varios paquetes adicionales:

- **Paquete “adapter”:**
  Este paquete contiene clases de adaptadores que se utilizan para vincular datos a las vistas en las actividades. Los adaptadores son comunes en el desarrollo

de aplicaciones Android para gestionar la presentación de listas de elementos, cuadros de diálogo y otros componentes de la interfaz de usuario.

- **Paquete "model":**
  Aquí se encuentran las clases de modelo que representan los datos de la aplicación. Estas clases son simples POJOs (Plan Old Java Objects) que contienen campos y metodos para acceder y manipular los datos.
- **Paquete "response":**
  Almacena las clases que representan las respuestas de las solicitudes a servicios externos, como respuestas de API REST. Estas clases suelen mapear directamente los datos recibidos en formato JSON a objetos Java para facilitar su manipulación en la aplicación.
- **Paquete "service":**
  En este paquete se encuentran las clases que definen servicios utilizados por la aplicación. Estas clases encapsulan la lógica relacionada con la comunicación con servicios externos o la ejecución de tareas en segundo plano.
- **Paquete "util":**
  Aquí se almacenan clases de utilidad que contienen métodos reutilizables y funciones de ayuda utilizadas en toda la aplicación. Estas clases suelen proporcionar funcionalidades comunes que pueden ser utilizadas en diferentes partes de la aplicación.

Esta estructura modular y organizada del proyecto de la aplicación movil facilita el desarrollo, la comprensión y el mantenimiento del código, lo que permite una mayor eficiencia y escalabilidad a medida que el proyecto crece y evoluciona. En la Ilustración 11 se puede ver un ejemplo más visual de la jerarquía del proyecto.

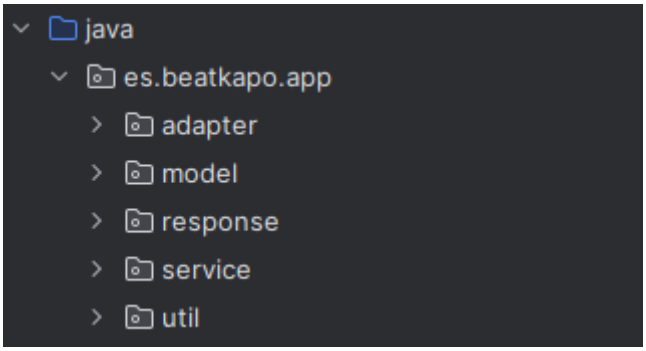
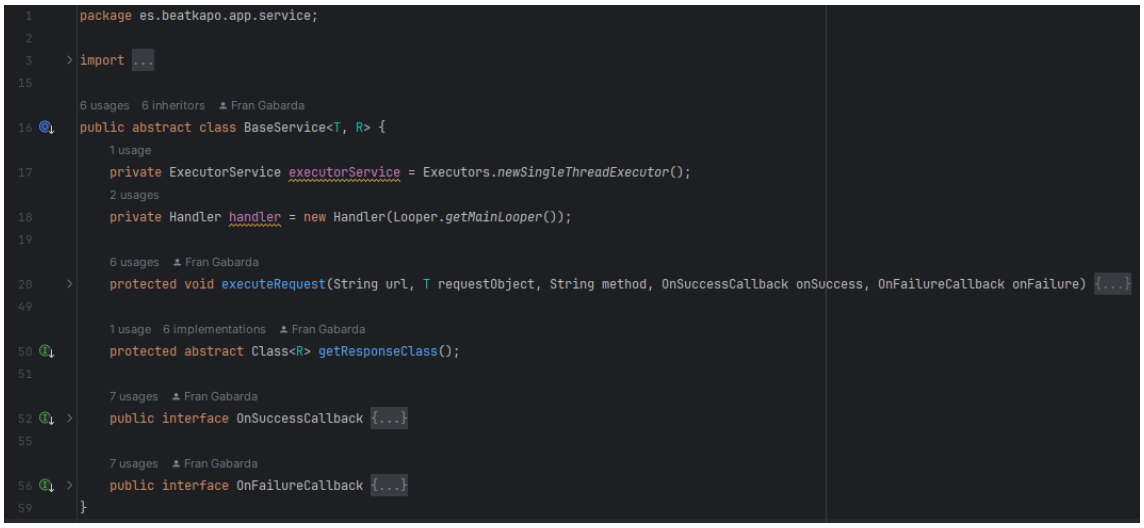


Ilustración 12

6.2.2. Comunicación entre componentes del sistema

En esta sección se detalla como se comunican los diferentes elementos del sistema, centrándose en la comunicación entre la aplicación móvil, el servidor y la base de datos. Se proporcionan ejemplos concretos para ilustrar el flujo de datos en cada etapa de la comunicación.

La comunicación entre la aplicación móvil y el servidor se gestiona a través de servicios que extienden la clase **BaseService**. Esta clase proporciona métodos para realizar solicitudes HTTP de forma asíncrona y manejar las respuestas de manera eficiente.

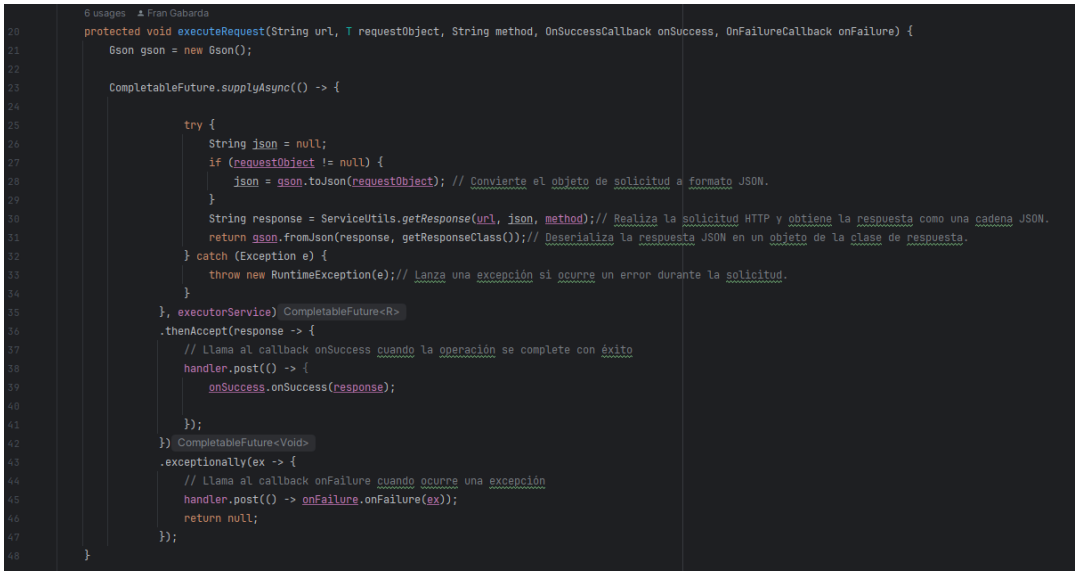


```java
package es.beatkapo.app.service;

import ...

6 usages  6 inheritors  Fran Gabarda
public abstract class BaseService<T, R> {
    1 usage
    private ExecutorService executorService = Executors.newSingleThreadExecutor();
    2 usages
    private Handler handler = new Handler(Looper.getMainLooper());

    6 usages  Fran Gabarda
    protected void executeRequest(String url, T requestObject, String method, OnSuccessCallback onSuccess, OnFailureCallback onFailure) {...}

    1 usage  6 implementations  Fran Gabarda
    protected abstract Class<R> getResponseClass();

    7 usages  Fran Gabarda
    public interface OnSuccessCallback {...}

    7 usages  Fran Gabarda
    public interface OnFailureCallback {...}
}
```

Ilustración 13: Resumen de la clase BaseService

La clase **BaseService** encapsula la lógica común para realizar solicitudes HTTP y manejar las respuestas. Utiliza la biblioteca Gson para serializar y deserializar objetos Java a y desde JSON. Además, se utiliza un **ExecutorService** para ejecutar las solicitudes en un hilo separado y un **Handler** para enviar las respuestas al hilo principal de la interfaz de usuario.

Para realizar una solicitud al servidor, se implementa el método **executeRequest**. Este método toma la URL del servicio, un objeto de solicitud, el método HTTP (GET, POST, etc.), y callbacks para manejar el éxito o el fracaso de la solicitud.



```java
6 usages  Fran Gabarda
protected void executeRequest(String url, T requestObject, String method, OnSuccessCallback onSuccess, OnFailureCallback onFailure) {
    Gson gson = new Gson();

    CompletableFuture.supplyAsync(() -> {

            try {
                String json = null;
                if (requestObject != null) {
                    json = gson.toJson(requestObject); // Convierte el objeto de solicitud a formato JSON.
                }
                String response = ServiceUtils.getResponse(url, json, method);// Realiza la solicitud HTTP y obtiene la respuesta como una cadena JSON.
                return gson.fromJson(response, getResponseClass());// Deserializa la respuesta JSON en un objeto de la clase de respuesta.
            } catch (Exception e) {
                throw new RuntimeException(e);// Lanza una excepción si ocurre un error durante la solicitud.
            }
        }, executorService) CompletableFuture<R>
        .thenAccept(response -> {
            // Llama al callback onSuccess cuando la operación se complete con éxito
            handler.post(() -> {
                onSuccess.onSuccess(response);

            });
        }) CompletableFuture<Void>
        .exceptionally(ex -> {
            // Llama al callback onFailure cuando ocurre una excepción
            handler.post(() -> onFailure.onFailure(ex));
            return null;
        });
}
```

Ilustración 14: Ejemplo del método executeRequest de la clase BaseService

La clase **Response** proporciona una estructura común para las respuestas del servidor. Esta clase encapsula información sobre el éxito o el fracaso de la solicitud, el código de error (si corresponde) y un mensaje descriptivo.

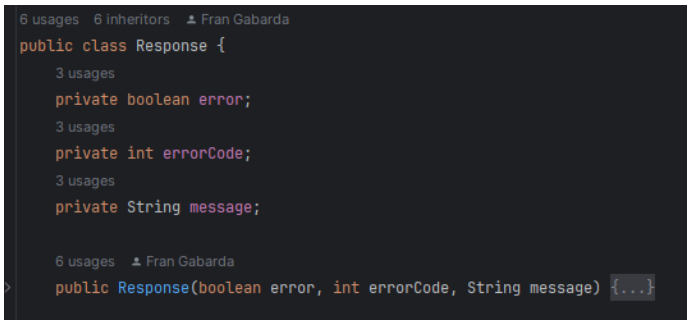


Ilustración 15: Resumen de la clase Response

Las clases de respuesta específicas, como **ProductoResponse**, heredan de la clase **Response** y pueden contener atributos adicionales específicos de la respuesta del servidor.

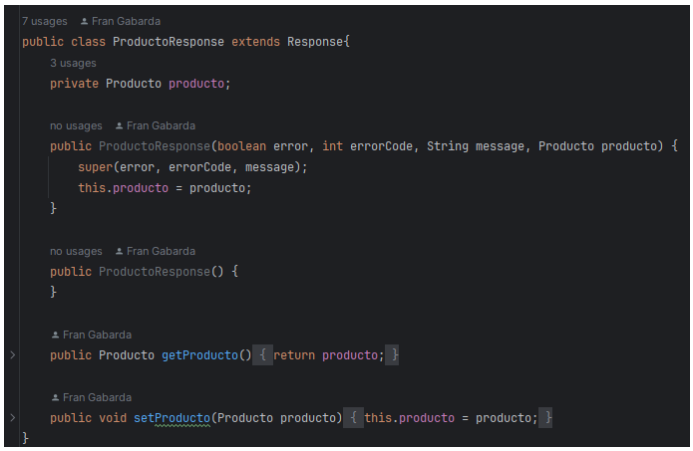


Ilustración 16: Ejemplo de la clase ProductoResponse

Cuando el servidor recibe una solicitud, utiliza rutas definidas en la API para determinar cómo manejarla. Por ejemplo, al recibir una solicitud para iniciar sesión, el servidor verifica las credenciales del usuario en la base de datos y genera un token JWT para autenticar al usuario.

Ilustración 17: Ejemplo de método que genera un token de autenticación

```
async function loginUser(user) {
    try {
        // Comprueba que el usuario existe en la base de datos
        const q = query(collection(db, 'usuarios'), where('email', '==', user.email));
        const querySnapshot = await getDocs(q);
        // Comprueba que la contraseña coincide
        if (querySnapshot.empty) {
            data = { error: true , errorCode: 0 , message: 'Usuario no encontrado' };
            return data;
        }
        const doc = querySnapshot.docs[0];
        const userData = doc.data();
        userData.id = doc.id;
        if (userData.password === user.password) {
            console.log('Usuario con id '+userData.id+' logueado:', userData.email);
            data = { error: false, token: await generateToken(userData) };
        } else {
            data = { error: true, errorCode: 1 ,message: 'Contraseña incorrecta' };
        }
        return data;
    } catch (error) {
        console.error('Error interno al iniciar sesión:', error);
        throw error;
    }
}
```

Ilustración 18: Ejemplo del método que se utiliza para iniciar sesión

Los datos llegan al servidor encriptados, por lo que este solo se encarga de compararlos. Se emplea un proceso de encriptación utilizando el algoritmo de hash SHA-256. Este algoritmo, ampliamente reconocido por su robustez y resistencia a los ataques de fuerza bruta, convierte las contraseñas en una secuencia de bytes cifrada, que luego se almacena en la base de datos del servidor.

El método **encryptPassword** de la clase **Utilidades** se encarga de realizar la encriptación de las contraseñas utilizando el algoritmo SHA-256. Este método toma la contraseña sin cifrar como entrada y devuelve la contraseña encriptada como una cadena hexadecimal.

```
public static String encryptPassword(String password) {
    try {
        // Crear una instancia de MessageDigest con el algoritmo SHA-256
        MessageDigest digest = MessageDigest.getInstance( algorithm: "SHA-256");

        // Obtener el arreglo de bytes de la contraseña
        byte[] hash = digest.digest(password.getBytes());

        // Convertir el arreglo de bytes a una cadena hexadecimal
        StringBuilder hexString = new StringBuilder();
        for (byte b : hash) {
            String hex = Integer.toHexString( i: 0xff & b);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        return null;
    }
}
```

Ilustración 19: Método que se encarga de cifrar las contraseñas

El servidor expone una serie de *endpoints* o rutas que permiten a la aplicación móvil realizar diversas operaciones. Estas rutas reciben las peticiones del cliente y realizan las acciones correspondientes en el servidor. A continuación se presentan algunos ejemplos de *endpoints*:

- Registro de Usuarios: **POST /api/register**

Ilustración 20: Ruta de la API para el registro de usuarios

- Inicio de Sesión: **POST /api/login**

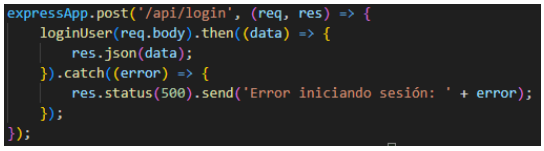


Ilustración 21: Ruta de la API que permite iniciar sesión en el sistema

- Obtener Productos: **GET /api/productos**

En las rutas que no son para el registro e inicio de sesión, se verifica el token para asegurar que el usuario que realiza la peticion está logueado.

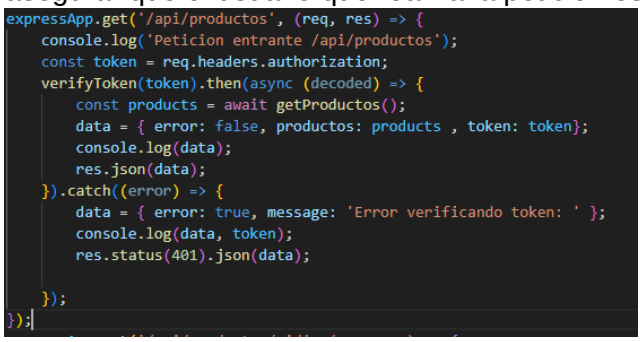


Ilustración 22: Ruta de la API para obtener todos los productos

- Realizar un Pedido: **GET /api/pedidos**

La ruta para obtener los pedidos, verifica el token y devuelve solo los pedidos del usuario que realiza la petición. Llama a las funciones asincronas que se encargan de hacer las consultas a la base de datos, para así tener una mejor claridad en el codigo y poder atender varias peticiones simultáneas.

Ilustración 23: Ruta de la API para obtener los pedidos del usuario.

6.2.3. Comentarios en el código

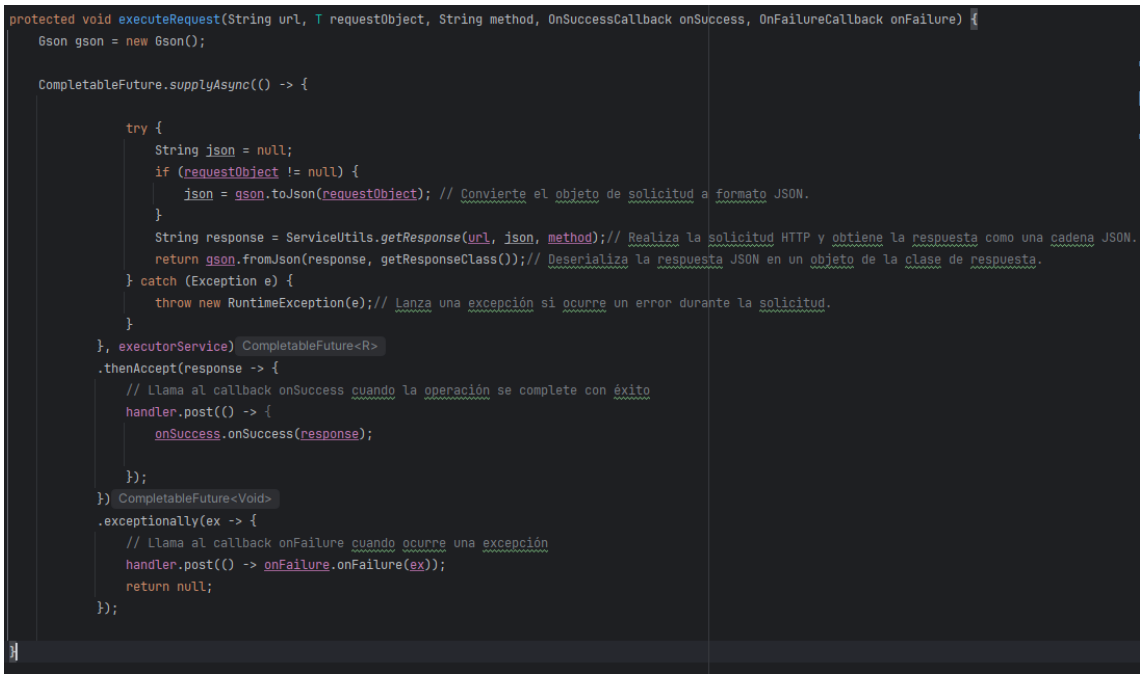
Los comentarios son una práctica fundamental para mejorar la comprensión y mantenibilidad del mismo. En este proyecto, se utilizan dos tipos principales de comentarios: comentarios JavaDoc y comentarios básicos.

- Comentarios JavaDoc:
  Son comentarios estructurados que siguen una convención especifica y pueden ser procesados por herramientas de generación de documentación, como JavaDoc. Estos comentarios se utilizan para documentar clases, métodos, variables y otros elementos del código fuente de manera formal y detallada. Siguen el formato de inicio con "/**" y finalización con "*/", y pueden incluir etiquetas como "@param", "@return" y "@throws" para describir los parámetros, valor de retorno, y excepciones lanzadas por un método, respectivamente. Puede verse un ejemplo en la Ilustración 12.

```
/**
 * Constructor de la clase Usuario.
 *
 * @param id Identificador único del usuario.
 * @param nombre Nombre del usuario.
 * @param apellidos Apellidos del usuario.
 * @param telefono1 Primer número de teléfono del usuario.
 * @param telefono2 Segundo número de teléfono del usuario.
 * @param email Dirección de correo electrónico del usuario.
 * @param password Contraseña del usuario.
 * @param direccion1 Dirección principal del usuario.
 * @param direccion2 Dirección secundaria del usuario.
 * @param tipoUsuario Tipo de usuario (0: Normal, 1: Moderador, 2: Administrador).
 * @param imagen URL de la imagen de perfil del usuario.
 */
```

Ilustración 24: Ejemplo de comentarios JavaDoc

- Comentarios básicos:
  Además de los comentarios JavaDoc, se utilizan comentarios básicos de Java para aclarar partes del código que pueden resultar confusas o necesitan una explicación adicional. Estos comentarios suelen ser más informales y no siguen una convención especifica, simplemente se colocan al lado del código relevante para proporcionar una explicación rápida. Se puede ver un ejemplo en la Ilustración 13.



```
protected void executeRequest(String url, T requestObject, String method, OnSuccessCallback onSuccess, OnFailureCallback onFailure) {
    Gson gson = new Gson();

    CompletableFuture.supplyAsync(() -> {

            try {
                String json = null;
                if (requestObject != null) {
                    json = gson.toJson(requestObject); // Convierte el objeto de solicitud a formato JSON.
                }
                String response = ServiceUtils.getResponse(url, json, method);// Realiza la solicitud HTTP y obtiene la respuesta como una cadena JSON.
                return gson.fromJson(response, getResponseClass());// Deserializa la respuesta JSON en un objeto de la clase de respuesta.
            } catch (Exception e) {
                throw new RuntimeException(e);// Lanza una excepción si ocurre un error durante la solicitud.
            }
        }, executorService) CompletableFuture<R>
        .thenAccept(response -> {
            // Llama al callback onSuccess cuando la operación se complete con éxito
            handler.post(() -> {
                onSuccess.onSuccess(response);

            });
        }) CompletableFuture<Void>
        .exceptionally(ex -> {
            // Llama al callback onFailure cuando ocurre una excepción
            handler.post(() -> onFailure.onFailure(ex));
            return null;
        });

}
```

Ilustración 25

El uso de comentarios mejora la legibilidad, la comprensión y la mantenibilidad del mismo, lo que facilita el trabajo colaborativo y la evolución del proyecto a lo largo del tiempo.

6.2.4. Documentación generada automáticamente

La documentación generada automáticamente es una práctica común en el desarrollo de software que consiste en utilizar herramientas especializadas para extraer información directamente del código fuente y generar documentación estructurada y detallada. En este proyecto, se ha utilizado la herramienta JavaDoc para generar documentación automáticamente a partir de los comentarios incrustados en el código. Esta documentación proporciona una referencia exhaustiva de las clases, métodos, variables y otros elementos del código fuente, incluyendo sus descripciones, parámetros, valores de retorno, excepciones lanzadas y más. Se presenta en formato HTML y puede ser fácilmente navegada y consultada utilizando un navegador web estándar.

La documentación generada automáticamente se encuentra disponible en el directorio “Docs” del repositorio de GitHub del proyecto.

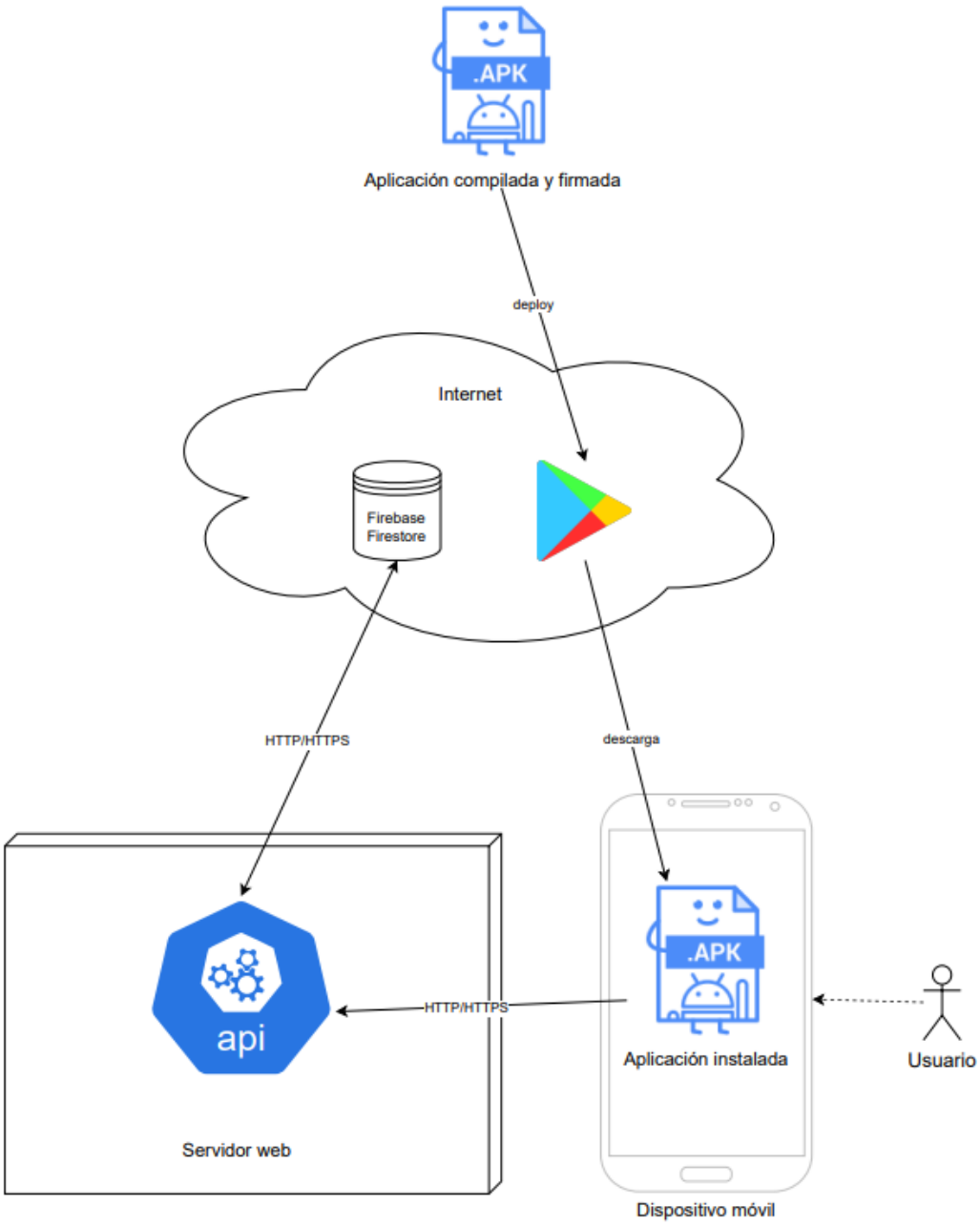
6.3. Manual de usuario.

Referencia al anexo del manual de usuario.

7. Despliegue.

El despliegue de esta aplicación comprende tanto la aplicación móvil desarrollada en Android como el servidor web que la respalda. Se verán los pasos necesarios para llevar la aplicación desde el entorno de desarrollo hasta un entorno de producción, donde esté disponible para su uso por parte de los usuarios finales.

7.1. Diagramas de despliegue.



7.2. Descripción de la instalación o despliegue.

Despliegue de la aplicación móvil

El despliegue de la aplicación móvil implica la compilación y preparación de un archivo APK (Android Package) listo para ser instalado en dispositivos Android. Para poder realizar la compilación, es importante tener un entorno de desarrollo configurado en el sistema, esto incluye tener instalado Android Studio, así como tener configuradas las herramientas y el SDK (Software Development Kit) de Android necesarias para compilar y construir la aplicación. Las opciones que se deben configurar se encuentran en el archivo “build.gradle” del proyecto de Android Studio.

Una vez que el proyecto esté configurado correctamente, se puede compilar la aplicación para generar el archivo APK. Para hacerlo, se debe seleccionar la opción “Build” en la barra de menú de Android Studio y elegir “Build APK”. Esto iniciará el proceso de compilación y empaquetado de la aplicación, generando el archivo APK que contendrá todos los recursos y código necesarios para ejecutar la aplicación en dispositivos Android.

Tras realizar las respectivas pruebas de funcionalidad, se puede ejecutar la aplicación en un emulador o en un dispositivo físico para probar todas las características y funcionalidades de la aplicación y asegurarse así de que no haya errores o problemas de rendimiento. Una vez quede un resultado óptimo en respecto al rendimiento de la aplicación, se debe tener en cuenta el siguiente paso, el firmado del APK. Esto es necesario para garantizar la autenticidad e integridad del archivo APK y es un requisito para distribuir la aplicación a través de la Google Play Store y otras tiendas de aplicaciones. Se puede utilizar Android Studio o la línea de comandos para firmar el APK con el certificado de firma.

Para continuar con el proceso de distribución de la aplicación, se necesita una cuenta de desarrollador de Google Play. Se puede crear una pagando una tarifa única de registro y completando el proceso de registro. Después de crear una cuenta, se debe acceder al Google Play Console, donde se pueden administrar las aplicaciones, configurar la lista de la tienda, monitorear el rendimiento y mucho más. Desde aquí se debe de crear una nueva lista de aplicación proporcionando detalles como nombre de la aplicación, la descripción, las capturas de pantalla, el icono, las categorías a las que pertenece, las palabras clave y cualquier otra información relevante. Esta lista será visible para los usuarios en la Google Play Store.

Después de haber creado la lista, deben configurarse también los detalles de la versión de la aplicación, incluyendo numero de versión, fecha de lanzamiento, y cualquier cambio importante o actualización en la nueva versión de esta. En la sección “Publicación de la versión” del Google Play Console, se puede cargar el APK firmado que se ha generado anteriormente. Hay que asegurarse de seguir las instrucciones y requisitos de Google Play para el tamaño máximo del APK, las políticas de contenido, y cualquier requisito de la plataforma. Tras cargar el archivo, se deben configurar detalles como la disponibilidad de la aplicación en diferentes países y regiones, las opciones de precios y distribución, y cualquier otra configuración especifica de la lista de la aplicación.

Una vez completados los pasos anteriores, se envía la aplicación para revisión y aprobación por parte del equipo de Google Play. La revisión puede llevar algún tiempo, y es importante asegurarse de cumplir con todas las políticas y directrices de Google Play para evitar retrasos o rechazos en la aprobación.

Cuando la aplicación sea aprobada, estará lista para ser publicada en la Google Play Store. Una vez que la aplicación esté publicada, estará disponible para su descarga y uso por parte de los usuarios finales en todo el mundo.

Despliegue del servidor.

El despliegue de la API Node.js es un paso crítico para llevar la aplicación desde el entorno de desarrollo hasta un entorno de producción accesible para los usuarios finales. Este proceso implica configurar y preparar cuidadosamente el entorno de producción para garantizar que la API esté disponible de manera confiable y segura.

El servidor donde se va a desplegar la API debe tener Node.js instalado. Se puede descargar desde la página oficial o utilizando un manejador de versiones como “nvm” para gestionar las múltiples versiones de Node.js. Hay que asegurarse de que todas las dependencias necesarias para la API están especificadas en el archivo “package.json” para poder ser instaladas ejecutando “npm install” en el directorio del proyecto.

Se necesita un servidor web como Nginx o Apache para actuar como intermediario entre el cliente y la API, hay que configurar el servidor para redirigir las solicitudes entrantes al puerto en el que se ejecutará la aplicación Node.js. Entre las diferentes configuraciones, una de las más importantes y que se debe tener en cuenta es el firewall, el cual se debe configurar correctamente para permitir el tráfico entrante y saliente en los puertos que utilice la API. Se pueden utilizar plataformas como AWS (Amazon Web Services) para ahorrar tiempo y trabajo, de hecho, es recomendable su uso para ahorrar también en costes.

Una vez configurado el servidor web, llega la hora de transferir los archivos de la aplicación (código fuente, archivos estáticos, etc.) al servidor. Se pueden usar herramientas como FTP, SCP o Git para hacerlo. Tras esto, ya se pueden instalar las dependencias necesarias para ejecutar la aplicación en un entorno de producción, esto se puede realizar ejecutando “npm install –production”.

Algo que también forma parte de la configuración, es el archivo “config.js” que almacena las claves que necesita para trabajar, por un lado, la “API key” de la base de datos, la cual debe ser reemplazada para una correcta comunicación con la base de datos que se desee utilizar. Por otro lado, también almacena la contraseña de cifrado, la cual se recomienda cambiar por una contraseña segura y que cumpla con los requisitos mínimos, como el número de caracteres, utilizar tanto números como letras, mayúsculas, etc.

Después de hacer los cambios necesarios en el archivo, la aplicación está lista para correr. Para finalmente hacer que la aplicación funcione, se puede ejecutar “node server.js” o utilizar herramientas como PM2 para gestionar el proceso de Node.js y asegurar que se reinicie automáticamente en caso de fallos.

Despliegue de la base de datos.

El primer paso en el despliegue de la base de datos es la creación de un proyecto en Firebase. Para ello, se debe acceder a la consola de Firebase y hacer click en “Añadir proyecto”. Se ingresa el nombre del proyecto y se siguen los pasos de configuración inicial, lo que incluye asociar la cuenta de Google correspondiente. Una vez completada la configuración, se redirige al panel principal del proyecto, donde se pueden gestionar los servicios que ofrece Firebase. Para este caso concreto se ha utilizado Firestore, la base de datos NoSQL de Firebase. Para configurarla, se debe seleccionar “Firestore Database” en el menú lateral izquierdo del panel y hacer clic en “Crear base de datos”. A continuación, se presenta la opción de iniciar en modo de prueba o en modo de producción. Es recomendable comenzar en modo de prueba para facilitar el desarrollo, y posteriormente ajustar las reglas de seguridad para el entorno de producción.
En la pestaña "Reglas" de Firestore, se pueden definir estas reglas utilizando un lenguaje de scripting específico de Firebase. A continuación se presenta un ejemplo

básico de reglas para el desarrollo:

```
rules_version = '2';

service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if true;
    }
  }
}
```

Ilustración 26: Reglas que permiten el acceso de lectura y escritura a cualquier usuario

Estas reglas permiten el acceso de lectura y escritura a cualquier usuario, lo cual es útil durante la fase de desarrollo. Sin embargo, para el entorno de producción, es necesario implementar reglas más estrictas que verifiquen la autenticación y los permisos de los usuarios. En este caso, tan solo debería de tener acceso un usuario, el servidor. A continuación, se muestra un ejemplo de reglas que permite lectura y escritura solo a un usuario:

```
rules_version = '2';

service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth != null && request.auth.uid == 'UID_DEL_USUARIO_AUTORIZADO';
    }
  }
}
```

Ilustración 27: Reglas que permite el acceso de lectura y escritura a un solo usuario

Una vez configurada la base de datos, es necesario integrarla con el servidor, que se encargará de manejar las peticiones a Firestore. Para ello, se debe registrar la aplicación del servidor en Firebase. Desde la sección “Ajustes del proyecto” en el panel de Firebase, se selecciona la opción de añadir una nueva aplicación y se elige la plataforma “Node.js”.

La configuración del servidor para interactuar con Firebase comienza con la instalación del SDK de Firebase para Node.js. Este SDK permite que el servidor realice operaciones de lectura y escritura en Firestore. Para iniciar se debe instalar el paquete “firebase-admin” a través de “npm”.

Luego, se debe inicializar Firebase en el servidor. Esto implica cargar el archivo de clave de servicio, que se obtiene desde la consola de Firebase, y configurar la conexión a Firestore. El siguiente fragmento de código ejemplifica cómo realizar esta inicialización:

```
const { initializeApp } = require('firebase/app');
const { getFirestore, collection, query, where, getDocs, addDoc, getDoc, doc } = require('firebase/firestore');
const { firebaseConfig, secretWord } = require("./config");
const firebaseApp = initializeApp(firebaseConfig);
const db = getFirestore(firebaseApp);
```

Ilustración 28: Ejemplo de inicialización de Firebase en el servidor

El archivo “config.json” contiene la configuración de Firebase y una clave secreta para la autenticación. Con esta configuración, el servidor está listo para interactuar con Firestore.

8. Herramientas de apoyo.

Durante el desarrollo de la aplicación, se utilizaron varias herramientas que facilitaron el trabajo con las tecnologías implementadas.

Visual Studio Code (VSCode): Para el desarrollo del servidor con Node.js, se utilizó Visual Studio Code debido a su versatilidad y extensibilidad. VSCode ofreció varias extensiones que mejoraron la productividad y la calidad del código. La extensión **Node.js Extension Pack** proporcionó herramientas esenciales como el debugger y el IntelliSense para autocompletado de código, mientras que **ESLint** y **Prettier** ayudaron a mantener un código limpio y conforme a las mejores prácticas. La integración nativa con Git facilitó el control de versiones directamente desde el editor, y la terminal integrada permitió ejecutar comandos de Node.js y npm sin salir del entorno de desarrollo. Además, las herramientas de depuración avanzadas de VSCode permitieron configurar breakpoints e inspeccionar variables, lo que fue crucial para la solución de problemas en la aplicación del servidor.

Android Studio: En el desarrollo del cliente, Android Studio fue el IDE elegido. Este entorno proporcionó un completo emulador de Android para pruebas, un editor de diseño visual que facilitó la creación de interfaces de usuario y herramientas de depuración avanzadas que ayudaron a identificar y solucionar errores rápidamente. Además, las herramientas de refactorización y análisis de código de Android Studio permitieron mantener el código limpio y organizado. La gestión de dependencias y la construcción de aplicaciones con Gradle aseguraron un desarrollo más estructurado y modular.

GitHub: Como herramienta de control de versiones, GitHub permitió almacenar el código en repositorios remotos, facilitando la colaboración y el acceso desde cualquier ubicación. Las ramas y pull requests fueron esenciales para el trabajo en equipo, permitiendo integrar cambios de manera ordenada y controlada. La integración con GitHub Actions permitió automatizar pruebas y despliegues, mejorando la eficiencia del desarrollo.

Herramienta en Python para Insertar Datos en Firebase: Durante el desarrollo del proyecto, se programó una herramienta en Python para facilitar la inserción de datos en Firebase desde un archivo Excel. Esta herramienta automatizó el proceso de carga de datos utilizando la librería **pandas** para leer archivos Excel y **firebase-admin** para conectarse a Firebase y cargar los datos. Esta automatización mejoró significativamente la eficiencia y precisión en la gestión de datos, permitiendo una carga rápida y reduciendo errores manuales.

9. Control de versiones

Uso de github y referencia a mi repositorio

10. Pruebas.

11. Conclusiones.

11.1. Conclusiones sobre el trabajo realizado

11.2. Conclusiones personales

11.3. Posibles ampliaciones y mejoras

Sacar una versión de escritorio para los ordenadores del establecimiento Tablero de cocina digital, para dejar de utilizar impresoras y comandas en papel.

Ampliar la aplicación para que cubra otros servicios como las comandas de mesa y reservas en línea.

Dar la opción al usuario a seleccionar sus alérgenos, de manera que no le saldrán productos que los contengan.

Control de reservas:

- El usuario podrá solicitar una reserva.
- El administrador confirmará o denegará la reserva.

Auditorías y registro de actividades: Estaría genial implementar un sistema de registros detallados de todas las actividades realizadas, incluyendo accesos exitosos y fallidos, consultas de datos y modificaciones realizadas.

12. Referencias

12.1. Bibliografía

12.2. Direcciones web

12.3. Artículos, revistas, apuntes, …