Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Aplicación Móvil para la Gestión de Tickets y Facturas mediante el Uso de LLMs

Autor: Darío Elguezabal Selioutin

Tutor(a): Alejandro Rodríguez González

Madrid, Enero 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Aplicación Móvil para la Gestión de Tickets y Facturas mediante el Uso de LLMs
Enero 2025

Autor: Darío Elguezabal Selioutin

Tutor:
Alejandro Rodríguez González
Lenguajes y sistemas informáticos e ingeniería de software
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Actualmente, la gestión de tickets de compra presenta varios desafíos, incluso en entornos digitalizados. Muchas personas recurren a tomar fotografías de sus tickets como solución para conservarlos y evitar su pérdida, pero este método también tiene limitaciones importantes. Las imágenes suelen almacenarse en galerías desorganizadas, mezcladas con otras fotos personales, lo que dificulta encontrar un ticket en particular cuando se necesita. Además, este enfoque no permite extraer ni aprovechar los datos contenidos en los recibos de manera estructurada, como fechas o montos, lo que limita su utilidad para el control financiero.

MyTickets busca resolver estas problemáticas al ofrecer una alternativa inteligente y eficiente para gestionar tickets. En lugar de simplemente guardar imágenes, la aplicación utiliza tecnologías avanzadas, como APIs de modelos de lenguaje (LLMs), para extraer automáticamente los datos clave de los tickets y almacenarlos de manera organizada en una base de datos local (SQLite). Esto permite a los usuarios acceder fácilmente a información relevante, como el importe total, productos o fechas, sin necesidad de buscar manualmente en galerías de fotos o transcribir datos.

Además, MyTickets incluye funcionalidades que van más allá del almacenamiento básico, como la posibilidad de editar, filtrar y eliminar tickets según las necesidades del usuario. Esto no solo ahorra tiempo, sino que también mejora significativamente la capacidad de realizar un seguimiento detallado de los gastos.

Con esta herramienta, MyTickets transforma la gestión de tickets en un proceso intuitivo y organizado, eliminando la frustración de los métodos tradicionales y permitiendo a los usuarios optimizar su tiempo y control financiero de manera práctica y moderna.

Abstract

Nowadays, managing purchase receipts presents several challenges, even in digitalized environments. Many people resort to taking photos of their receipts as a way to preserve them and avoid losing them, but this method also has significant limitations. Images are often stored in disorganized galleries, mixed with personal photos, making it difficult to find a specific receipt when needed. Furthermore, this approach does not allow the extraction or structured use of the information contained in the receipts, such as dates or amounts, limiting their usefulness for financial tracking.

MyTickets aims to address these issues by offering a smart and efficient alternative for managing receipts. Instead of simply saving images, the app leverages advanced technologies like APIs powered by Large Language Models (LLMs) to automatically extract key data from receipts and store it in an organized manner in a local database (SQLite). This enables users to easily access relevant information, such as total amounts, products or dates, without the need to manually search through photo galleries or transcribe data.

Additionally, MyTickets includes features that go beyond basic storage, such as the ability to edit, filter, and delete receipts according to user needs. This not only saves time but also significantly enhances the ability to track expenses in detail.

With this tool, MyTickets transforms receipt management into an intuitive and organized process, eliminating the frustrations of traditional methods and allowing users to optimize their time and financial control in a practical and modern way.

Tabla de contenidos

Índice de Ilustraciones

1 Introducción

En un mundo cada vez más digitalizado, la gestión de tickets y facturas sigue presentando desafíos significativos para los usuarios. Aunque la mayoría de las personas conservan estos documentos mediante fotografías en sus dispositivos móviles, esta solución presenta limitaciones importantes. Las imágenes suelen quedar desorganizadas entre otras fotos personales, dificultando la búsqueda de información específica cuando es necesaria. Además, este enfoque no permite aprovechar el potencial de los datos contenidos en los tickets, como fechas, importes o detalles fiscales, de manera estructurada y eficiente.

Este Trabajo Fin de Grado tiene como objetivo desarrollar una solución innovadora: una aplicación móvil que transforme la manera en que los usuarios gestionan sus tickets y facturas. La propuesta, denominada MyTickets, aprovecha tecnologías avanzadas como los Modelos de Lenguaje de Gran Escala (LLMs), específicamente Gemini, para procesar y extraer automáticamente información clave de los tickets. Estos datos se almacenan de forma organizada en una base de datos local, permitiendo al usuario acceder fácilmente a información relevante, como el importe total, fechas de vencimiento de garantías y detalles de compras.

El desarrollo de este proyecto también incluye la implementación de funcionalidades prácticas como la edición, filtrado y eliminación de tickets, diseñadas para facilitar la gestión y el control financiero de los usuarios. Asimismo, la aplicación ha sido diseñada utilizando herramientas modernas como Kotlin y Jetpack Compose, que representan un cambio significativo respecto a las tecnologías tradicionales de desarrollo móvil, aportando eficiencia y flexibilidad al proceso.

En este contexto, el presente documento detalla el proceso de desarrollo de MyTickets, desde la identificación de las necesidades de los usuarios hasta la implementación técnica y los resultados obtenidos. A lo largo del trabajo, se busca no solo ofrecer una solución técnica eficiente, sino también generar un impacto positivo en la vida cotidiana de los usuarios, simplificando una tarea que, aunque cotidiana, puede resultar engorrosa y consumir tiempo innecesario.

Con este proyecto, no solo se aborda un problema común de manera innovadora, sino que también se demuestra cómo las tecnologías actuales pueden aplicarse para mejorar aspectos prácticos del día a día. MyTickets representa un esfuerzo por combinar usabilidad, funcionalidad y tecnología de vanguardia en una solución accesible para cualquier usuario.

1.1 Objetivos

La lista de objetivos del presente proyecto es la siguiente:

Para el desarrollo de la aplicación, se han definido una serie de pasos clave que estructuran el proceso y garantizan la implementación de las funcionalidades requeridas. En primer lugar, se implementó una función que permite al usuario capturar imágenes de los tickets con la cámara del dispositivo o cargar imágenes desde la galería. Esta funcionalidad es esencial para iniciar el proceso de digitalización de los tickets físicos.

Una vez obtenida la imagen del ticket, se desarrolló una funcionalidad que utiliza un modelo de lenguaje (LLM) para extraer los datos relevantes en formato JSON. Esta etapa es crucial para automatizar la obtención de información, como el nombre del restaurante, la fecha, los productos comprados y los precios. Posteriormente, los datos extraídos se presentan al usuario a través de una interfaz clara, permitiéndole revisar y, si es necesario, editar la información antes de guardarla.

Para el almacenamiento de los tickets procesados, se utilizó una base de datos en la nube mediante Firebase Firestore. Esta elección garantiza que los datos se mantengan seguros, accesibles desde cualquier dispositivo y organizados de manera eficiente. Además, se incorporaron opciones avanzadas de búsqueda y filtrado, que permiten al usuario localizar tickets específicos basándose en criterios como la fecha o el restaurante.

Otra funcionalidad importante añadida es la posibilidad de exportar los tickets almacenados en formato CSV, ofreciendo al usuario una manera sencilla de gestionar sus datos fuera de la aplicación. Este enfoque refuerza la versatilidad de la herramienta, adaptándose a diferentes necesidades de los usuarios.

La interfaz de usuario, diseñada con Jetpack Compose, proporciona una experiencia intuitiva y amigable. Esto asegura que los usuarios puedan navegar fácilmente entre las distintas secciones de la aplicación y acceder a todas las funcionalidades de manera fluida.

Finalmente, se llevaron a cabo diversas validaciones y pruebas para garantizar el correcto funcionamiento de la aplicación bajo diferentes condiciones. Asimismo, se elaboró una documentación detallada del proyecto, describiendo cada una de las funcionalidades, el diseño técnico y las decisiones tomadas durante el desarrollo. Esta documentación no solo sirve como referencia para el trabajo, sino que también permite futuras mejoras o ampliaciones del sistema.

2 Contexto

En este capítulo se presentará el contexto en el que se enmarca el desarrollo de este proyecto. Se analizará el perfil de los posibles usuarios de la aplicación, identificando sus necesidades y los motivos que podrían impulsarlos a utilizar esta solución. Asimismo, se describirá cómo se espera que interactúen con la aplicación y qué beneficios les aportará.

Este apartado también servirá para examinar las soluciones actualmente disponibles en el mercado, evaluando sus fortalezas y debilidades, así como aquellas características que podrían resultar relevantes para incorporar en el diseño de nuestra propuesta.

Por último, se llevará a cabo un breve estudio de mercado con el objetivo de determinar la viabilidad económica del proyecto en caso de que se plantee su comercialización para obtener rentabilidad.

2.1 Necesidades del Usuario

Antes de comenzar con el desarrollo de la aplicación, es imprescindible identificar al usuario objetivo y comprender sus necesidades específicas. Esto permitirá establecer unos requisitos claros y funcionalidades definidas desde el inicio, minimizando la necesidad de realizar modificaciones durante el proceso de desarrollo, lo que podría generar retrasos y aumentar los costos.

El público principal al que va dirigida esta aplicación es cualquier persona interesada en organizar y gestionar sus tickets de restauración de manera eficiente. La aplicación está diseñada para un uso personal o doméstico, sin enfocarse en entornos empresariales que generan grandes volúmenes de datos. Aunque en versiones futuras podría estudiarse una adaptación para un público comercial, en esta fase inicial se prioriza su aplicación en el ámbito individual.

Partiendo de este contexto, se han identificado las siguientes necesidades clave:

- **Digitalización rápida de tickets:** Los usuarios requieren un método ágil para registrar sus tickets, ya sea mediante la captura de imágenes con la cámara o seleccionando fotografías almacenadas en la galería del dispositivo.

- **Extracción automatizada de información relevante:** Es fundamental que la aplicación utilice tecnologías avanzadas, como la API de Gemini, para procesar las imágenes y extraer datos esenciales en formato JSON. Entre estos datos se incluyen el nombre del restaurante, la lista de productos adquiridos, los precios y las fechas.

- **Gestión y organización personalizada:** Los usuarios necesitan herramientas para gestionar sus tickets de manera flexible. Esto abarca la capacidad de editar, eliminar y filtrar los registros en función de criterios como la fecha, el producto o el restaurante.

- **Almacenamiento en la nube seguro y accesible:** Para garantizar la disponibilidad y protección de los datos, los tickets se almacenarán en

Firebase, lo que permitirá el acceso desde cualquier dispositivo vinculado a la cuenta del usuario.

- **Exportación en formatos útiles:** La opción de descargar los tickets en formato CSV proporcionará a los usuarios la posibilidad de utilizar sus datos en otras plataformas o aplicaciones de gestión personal.

- **Diseño intuitivo y fácil de usar:** La interfaz de la aplicación debe ser visualmente atractiva, fácil de navegar y accesible para cualquier nivel de experiencia tecnológica, garantizando una experiencia de usuario fluida.

Con estas funcionalidades, la aplicación pretende ser una herramienta eficaz y versátil para el control y la gestión de tickets de restauración, ofreciendo una solución práctica para organizar información financiera y optimizar el tiempo del usuario.

2.2 Soluciones Existentes

Antes de comenzar con el desarrollo de la aplicación, se llevó a cabo un trabajo de investigación sobre las soluciones existentes en el mercado que abordan problemas similares. Este análisis permitió identificar las funcionalidades clave ofrecidas por estas herramientas, así como sus ventajas y desventajas desde la perspectiva del usuario. La investigación tuvo como objetivo obtener una visión clara de las tendencias actuales y las expectativas de los usuarios, para así incorporar ideas útiles y evitar posibles limitaciones detectadas en estas soluciones. A continuación, se presentan los resultados obtenidos de este análisis.

2.2.1 Zoho Expense

Zoho Expense es una aplicación diseñada para facilitar la gestión de gastos y viajes, ofreciendo una solución eficiente tanto para individuos como para pequeñas organizaciones. La aplicación permite a los usuarios escanear recibos de manera sencilla mediante la función Autoscan, que captura los datos de los tickets y los organiza automáticamente en registros de gastos. Además, incluye herramientas como el seguimiento de kilometraje, utilizando un rastreador GPS integrado para calcular los costos asociados a viajes.

Una de las características destacadas de Zoho Expense es su capacidad para vincular tarjetas de crédito personales y corporativas, lo que permite importar transacciones automáticamente y convertirlas en registros de gastos con facilidad. También simplifica la gestión de anticipos, ya que permite registrar anticipos en efectivo y aplicarlos directamente a los informes de gastos, ajustando los totales automáticamente.

La aplicación está diseñada para optimizar el flujo de trabajo relacionado con la aprobación y reembolso de gastos. Los gerentes pueden revisar y aprobar informes de gastos y solicitudes de viaje con un solo toque, gracias a un sistema

de notificaciones instantáneas que mantiene a los usuarios informados sobre el estado de sus solicitudes. Además, Zoho Expense ofrece herramientas analíticas que proporcionan información rápida y detallada sobre los gastos, ayudando a los usuarios a tomar decisiones informadas sobre su gestión financiera. [1]

Ilustración 1: Logo Zoho Expense

2.2.2 Okticket

Okticket es una aplicación móvil diseñada para facilitar la gestión de gastos profesionales. Permite a los usuarios digitalizar tickets y facturas mediante una simple fotografía, eliminando la necesidad de conservar documentos en papel. La aplicación extrae automáticamente información clave, como fechas e importes, y clasifica los gastos en categorías predefinidas, como comidas, hoteles o gasolina.

Además, Okticket está homologada por la Agencia Tributaria, garantizando la validez legal de los documentos digitalizados. Sin embargo, es importante destacar que la aplicación no desglosa ni procesa detalles específicos de los productos adquiridos; su enfoque se centra en la gestión general de los gastos profesionales.

Ilustración 2: Logo okTicket

2.3 Nuestra Propuesta - MyTickets

Tras estudiar las necesidades de los usuarios y evaluar las soluciones disponibles en el mercado, hemos diseñado nuestra propia propuesta: "MyTickets", una aplicación que tiene como propósito simplificar la organización y almacenamiento de tickets de compra de manera práctica y automatizada. El nombre refleja el enfoque en la personalización y facilidad de uso para el usuario.

“MyTickets” se centra en permitir a los usuarios digitalizar y gestionar sus tickets de forma eficiente. La aplicación extrae automáticamente información clave, como el nombre del restaurante, el CIF, la fecha, la hora, los productos adquiridos y los precios (con y sin IVA), y almacena todos estos datos en una base de datos en la nube. Además, ofrece funcionalidades avanzadas como la posibilidad de filtrar tickets por diferentes criterios y exportarlos a un archivo CSV para un manejo externo.

Un aspecto innovador de “MyTickets” es el uso de tecnologías de reconocimiento de imágenes mediante la API de Gemini. Gracias a esto, el usuario puede procesar un ticket simplemente tomando una foto con la cámara de su dispositivo o eligiendo una imagen desde la galería. La aplicación analiza automáticamente la imagen y extrae los datos relevantes, minimizando el esfuerzo manual. No obstante, el usuario mantiene el control, pudiendo verificar y modificar los datos extraídos para garantizar su precisión.

La aplicación también incluye características clave como el inicio de sesión con Google, la eliminación de cuenta junto con todos los datos asociados, y la sincronización en la nube para asegurar que la información esté siempre accesible desde cualquier dispositivo. Con estas funcionalidades, “MyTickets” busca no solo digitalizar la gestión de tickets, sino ofrecer al usuario una herramienta intuitiva y confiable para mejorar su experiencia diaria.

Ilustración 3: Logo MyTickets

3 Entorno de Desarrollo

En este apartado se describirán los recursos, herramientas, metodologías y tecnologías que se han empleado durante el proceso de desarrollo, y que forman parte integral de la construcción final de la aplicación. Estas herramientas han sido seleccionadas cuidadosamente para garantizar una implementación eficiente y un producto de calidad.

3.1 Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para crear aplicaciones Android, desarrollado por Google. Ofrece herramientas que facilitan el diseño, la codificación y las pruebas de aplicaciones en esta plataforma.

Ilustración 5: Logo de Android Studio

Ilustración 4: Logo de Jetpack Compose

¿Por qué Android Studio es adecuado para este proyecto?

1. **Soporte oficial y actualizaciones constantes:** Al ser la herramienta oficial de Google, garantiza compatibilidad con las últimas versiones de Android y acceso a nuevas funcionalidades.
2. **Integración con Jetpack Compose:** Nuestro proyecto utiliza Jetpack Compose para una interfaz moderna. Android Studio ofrece soporte nativo para esta herramienta, facilitando su implementación. [4]
3. **Herramientas integradas:** Incluye emuladores, depuradores y analizadores de rendimiento, permitiendo gestionar todo el ciclo de desarrollo en un solo lugar.
4. **Facilidad de integración con Firebase:** La aplicación requiere almacenamiento en la nube, y Android Studio simplifica la configuración e integración con Firebase.
5. **Comunidad activa y recursos disponibles:** Su amplia adopción asegura una comunidad vibrante, facilitando el acceso a documentación, tutoriales y soporte.

En resumen, Android Studio proporciona un entorno completo y eficiente para el desarrollo de aplicaciones Android, alineándose perfectamente con las necesidades de nuestro proyecto. [2]

3.2 Lenguaje de Programación

Kotlin es un lenguaje de programación moderno, desarrollado por JetBrains, que se ejecuta sobre la Máquina Virtual de Java (JVM). Es interoperable con Java, lo que significa que puede integrarse fácilmente con proyectos existentes en dicho lenguaje. Kotlin fue adoptado oficialmente por Google como lenguaje principal para el desarrollo de aplicaciones Android en 2017.

Ilustración 6: Logo de Kotlin

¿Por qué se eligió Kotlin para este proyecto?

1. **Soporte oficial para Android:** Kotlin cuenta con el respaldo oficial de Google, lo que garantiza una integración completa con Android Studio y herramientas específicas para facilitar el desarrollo en esta plataforma.
2. **Código más conciso y legible:** Una de las principales ventajas de Kotlin es su capacidad para reducir la cantidad de código necesario, lo que resulta en una mayor legibilidad y menor propensión a errores.
3. **Mayor seguridad:** Kotlin incorpora características como la gestión de nulabilidad, que ayuda a evitar errores comunes como las excepciones de puntero nulo (NullPointerException), mejorando la estabilidad de la aplicación.
4. **Compatibilidad con Java:** Al ser totalmente interoperable con Java, Kotlin permite aprovechar bibliotecas y marcos ya existentes, lo que facilita la integración de funcionalidades avanzadas.
5. **Productividad del desarrollador:** Las características modernas de Kotlin, como las funciones de extensión y las expresiones lambda, agilizan el proceso de desarrollo, permitiendo implementar soluciones más rápidamente.
6. **Comunidad activa y en crecimiento:** La creciente popularidad de Kotlin asegura una amplia disponibilidad de recursos, documentación y foros para resolver dudas o problemas durante el desarrollo.

En definitiva, Kotlin ha sido elegido como el lenguaje de programación para este proyecto debido a sus múltiples ventajas, entre ellas su simplicidad, seguridad y soporte nativo en Android Studio. Estas características no solo garantizan un desarrollo más eficiente, sino que también aseguran que la aplicación sea robusta y fácil de mantener a largo plazo. [3]

3.3 Almacenamiento de Datos

Firebase es una plataforma de desarrollo de aplicaciones móviles y web proporcionada por Google. Ofrece una variedad de herramientas y servicios que facilitan la creación, el despliegue y la gestión de aplicaciones de alta calidad.

Ilustración 7: Logo de Firebase

¿Por qué se escogió Firebase para este proyecto?

1. **Almacenamiento en la nube eficiente:** Firebase proporciona soluciones de base de datos en tiempo real y almacenamiento en la nube, lo que permite gestionar y sincronizar datos de manera efectiva entre los usuarios y sus dispositivos.
2. **Autenticación de usuarios simplificada:** Ofrece servicios de autenticación que facilitan la gestión de usuarios, permitiendo iniciar sesión mediante correo electrónico, redes sociales y otros métodos, garantizando una experiencia de usuario segura y fluida.
3. **Integración con herramientas de análisis:** Firebase se integra con Google Analytics, proporcionando métricas detalladas sobre el comportamiento de los usuarios, lo que es esencial para mejorar y adaptar la aplicación según las necesidades detectadas.
4. **Escalabilidad y seguridad:** Firebase está diseñado para escalar según las necesidades de la aplicación, ofreciendo infraestructura segura y gestionada por Google, lo que garantiza la protección de los datos y la disponibilidad del servicio.

En resumen, Firebase proporciona un conjunto completo de herramientas que facilitan el desarrollo y la gestión de aplicaciones móviles, alineándose perfectamente con los requisitos de nuestro proyecto y permitiendo una implementación eficiente y segura. [5]

3.4 Control de Versiones

Para el control de versiones, he decidido utilizar GitHub porque ofrece una forma sencilla y confiable de gestionar y realizar un seguimiento del código fuente del proyecto. GitHub permite mantener un historial detallado de los cambios realizados, lo que resulta esencial para identificar y corregir errores, así como para revertir modificaciones si es necesario.

Aunque este proyecto ha sido desarrollado de forma individual, GitHub proporciona herramientas que han sido útiles para organizar el trabajo, como la gestión de ramas para implementar nuevas funcionalidades de forma aislada y segura antes de fusionarlas con la versión principal del proyecto.

Además, al estar basado en la nube, GitHub asegura que el repositorio esté siempre accesible y respaldado, ofreciendo tranquilidad frente a posibles pérdidas de datos. Su integración con Android Studio ha simplificado el proceso de sincronización de cambios y commits, mejorando significativamente mi flujo de trabajo.

Para simplificar el flujo de trabajo, solo se empleó la rama “main”, donde se subieron las versiones actualizadas del código. Si bien este enfoque fue suficiente para un desarrollo individual, el uso de ramas adicionales habría permitido un mayor control y organización del proyecto. Por ejemplo, podrían haberse creado ramas específicas para el desarrollo de nuevas funcionalidades o para solucionar errores, manteniendo la rama principal exclusivamente para versiones estables del código.

Ilustración 8: Logo de Github

En definitiva, GitHub ha sido una herramienta clave para mantener un control efectivo del proyecto, asegurando la organización y la seguridad del código fuente durante todas las fases del desarrollo.

3.5 Metodología de Desarrollo

El modelo escogido ha sido “en cascada” para el desarrollo de esta aplicación porque se adapta perfectamente a las necesidades del proyecto. Desde el principio, los requisitos estaban bien definidos y no se esperaba realizar cambios significativos durante el desarrollo. Este enfoque lineal permitió avanzar de manera estructurada por cada fase del proyecto: primero el análisis, luego el diseño, la implementación y finalmente las pruebas.

Aunque hubo un pequeño ajuste durante la implementación (cambiar el almacenamiento local por Firebase en la nube), este no alteró el flujo general del desarrollo y se integró sin problemas. Al ser el único desarrollador, el modelo en cascada facilitó la tarea de organizar mejor el trabajo y mantener un control claro sobre cada etapa. Es una metodología sencilla y efectiva para proyectos con objetivos claros como este.

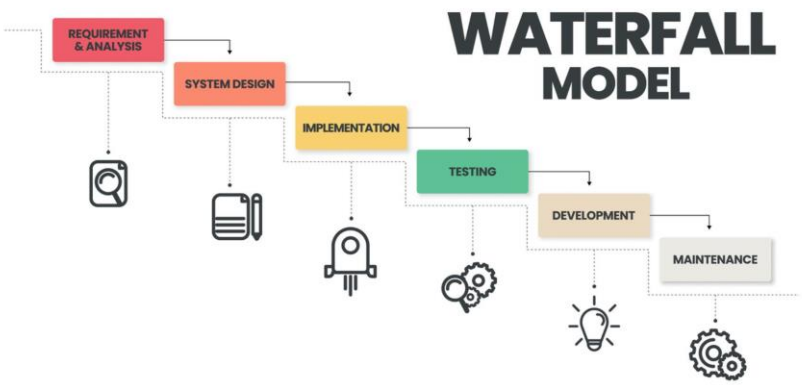


Ilustración 9: Modelo en Cascada

4 Análisis de Requisitos

La aplicación debe cumplir una serie de funcionalidades que se describen a continuación para satisfacer las necesidades del usuario:

Autenticación con cuenta de Google:

La aplicación debe permitir al usuario autenticarse utilizando su cuenta de Google para acceder a la aplicación.

Cerrar sesión:

La aplicación debe permitir al usuario cerrar sesión, asegurándose de que no se guarde ninguna información personal en el dispositivo tras el cierre.

Acceso a cámara y galería:

La aplicación debe permitir al usuario acceder a la cámara del dispositivo para capturar imágenes de tickets en tiempo real.

La aplicación debe permitir al usuario seleccionar imágenes de tickets almacenadas previamente en la galería del dispositivo.

Envío a la API del LLM:

La aplicación debe enviar las imágenes capturadas o seleccionadas a la API del LLM (como ChatGPT o Gemini) para su procesamiento.

Extracción de datos en formato JSON:

La aplicación debe extraer información relevante del ticket en formato JSON, incluyendo datos como restaurante, CIF, fecha y hora, productos adquiridos, precio sin IVA, IVA y precio total con IVA.

Mostrar y editar datos extraídos:

La aplicación debe mostrar al usuario los datos extraídos del ticket para que pueda revisarlos y compararlos con el ticket original.

La aplicación debe permitir al usuario editar cualquier dato extraído en caso de que haya errores en el procesamiento.

Almacenamiento de tickets:

La aplicación debe guardar los tickets en formato JSON en una base de datos en la nube utilizando Firebase.

Acceso y filtrado de tickets guardados:

La aplicación debe permitir al usuario acceder a los tickets almacenados en la base de datos.

La aplicación debe proporcionar opciones para filtrar los tickets por criterios como fecha, restaurante o CIF.

Edición o eliminación de tickets:

La aplicación debe permitir al usuario editar la información de los tickets previamente guardados.

La aplicación debe permitir al usuario eliminar tickets existentes de la base de datos.

Exportación de tickets en formato CSV:

La aplicación debe permitir al usuario descargar los tickets almacenados en formato CSV para su uso externo o análisis.

Eliminar cuenta:

La aplicación debe permitir al usuario eliminar su cuenta, lo que incluirá la eliminación de todos los tickets y datos asociados almacenados en la base de datos.

Los requisitos definidos para cumplir las funcionalidades descritas anteriormente son los siguientes:

RQ01

- **Nombre:** Autenticación con cuenta de Google
- **Prioridad:** Alta
- **Descripción:** La aplicación debe permitir al usuario iniciar sesión utilizando su cuenta de Google.
- **Entradas:** Credenciales de la cuenta de Google del usuario.
- **Proceso:** Validación y autenticación a través del servicio de Google Sign-In.
- **Salidas:** Acceso a la aplicación con los datos del usuario autenticado.

RQ02

- **Nombre:** Cerrar sesión
- **Prioridad:** Alta
- **Descripción:** La aplicación debe permitir al usuario cerrar sesión, garantizando que no se guarde ninguna información personal en el dispositivo tras el cierre.
- **Entradas:** Solicitud de cierre de sesión del usuario.
- **Proceso:** Eliminación de los datos de sesión del dispositivo.
- **Salidas:** Confirmación de cierre de sesión y retorno a la pantalla de inicio.

RQ03

- **Nombre:** Acceso a la cámara
- **Prioridad:** Alta
- **Descripción:** La aplicación debe permitir al usuario acceder a la cámara del dispositivo para capturar imágenes de tickets en tiempo real.
- **Entradas:** Permiso de acceso a la cámara y captura de imagen.
- **Proceso:** Activación de la cámara y captura de la imagen.
- **Salidas:** Imagen capturada lista para ser procesada.

RQ04

- **Nombre:** Acceso a la galería
- **Prioridad:** Media
- **Descripción:** La aplicación debe permitir al usuario seleccionar imágenes de tickets almacenadas previamente en la galería del dispositivo.
- **Entradas:** Selección de una imagen desde la galería.
- **Proceso:** Obtención de la imagen seleccionada.
- **Salidas:** Imagen seleccionada lista para ser procesada.

RQ05

- **Nombre:** Envío de imagen a la API del LLM
- **Prioridad:** Alta
- **Descripción:** La aplicación debe enviar las imágenes capturadas o seleccionadas a la API del LLM para su procesamiento.
- **Entradas:** Imagen del ticket capturada o seleccionada.
- **Proceso:** Envío de la imagen a la API para su análisis.
- **Salidas:** Datos del ticket extraídos en formato JSON.

RQ06

- **Nombre:** Extracción de datos en formato JSON
- **Prioridad:** Alta
- **Descripción:** La aplicación debe extraer información relevante del ticket en formato JSON, incluyendo datos como restaurante, CIF, fecha y hora, productos adquiridos, precios, IVA y total.
- **Entradas:** Imagen del ticket procesada por la API del LLM.
- **Proceso:** Análisis de la imagen y conversión de la información a formato JSON.
- **Salidas:** Datos relevantes del ticket en formato JSON.

RQ07

- **Nombre:** Mostrar datos extraídos
- **Prioridad:** Alta
- **Descripción:** La aplicación debe mostrar al usuario los datos extraídos del ticket para que pueda revisarlos y compararlos con el ticket original.
- **Entradas:** Datos del ticket en formato JSON.
- **Proceso:** Presentación visual de los datos extraídos.
- **Salidas:** Vista de los datos extraídos para su revisión.

RQ08

- **Nombre:** Edición de datos extraídos
- **Prioridad:** Media
- **Descripción:** La aplicación debe permitir al usuario editar cualquier dato extraído en caso de errores en el procesamiento.
- **Entradas:** Datos extraídos y correcciones realizadas por el usuario.
- **Proceso:** Actualización de los datos según las correcciones realizadas.
- **Salidas:** Datos corregidos listos para ser almacenados.

RQ09

- **Nombre:** Almacenamiento de tickets en Firebase
- **Prioridad:** Alta
- **Descripción:** La aplicación debe guardar los tickets en formato JSON en una base de datos en la nube utilizando Firebase.
- **Entradas:** Datos del ticket en formato JSON.
- **Proceso:** Envío de los datos a Firebase para su almacenamiento.
- **Salidas:** Confirmación de almacenamiento exitoso en la nube.

RQ10

- **Nombre:** Acceso a tickets almacenados
- **Prioridad:** Alta
- **Descripción:** La aplicación debe permitir al usuario acceder a los tickets almacenados en la base de datos.
- **Entradas:** Solicitud del usuario para acceder a sus tickets.
- **Proceso:** Recuperación de los datos almacenados en Firebase.
- **Salidas:** Lista de tickets disponibles para su visualización.

RQ11

- **Nombre:** Filtrado de tickets
- **Prioridad:** Media
- **Descripción:** La aplicación debe proporcionar opciones para filtrar los tickets por criterios como fecha, restaurante o CIF.
- **Entradas:** Criterios de filtrado seleccionados por el usuario.
- **Proceso:** Aplicación de filtros a los datos almacenados.
- **Salidas:** Lista de tickets filtrados según los criterios especificados.

RQ12

- **Nombre:** Edición de tickets almacenados
- **Prioridad:** Media
- **Descripción:** La aplicación debe permitir al usuario editar la información de los tickets previamente guardados.
- **Entradas:** Datos del ticket y cambios realizados por el usuario.
- **Proceso:** Actualización de los datos en la base de datos.
- **Salidas:** Confirmación de que los datos han sido actualizados correctamente.

RQ13

- **Nombre:** Eliminación de tickets almacenados
- **Prioridad:** Media
- **Descripción:** La aplicación debe permitir al usuario eliminar tickets existentes de la base de datos.
- **Entradas:** Solicitud del usuario para eliminar un ticket específico.
- **Proceso:** Eliminación de los datos seleccionados en Firebase.
- **Salidas:** Confirmación de que el ticket ha sido eliminado.

RQ14

- **Nombre:** Exportación de tickets en formato CSV
- **Prioridad:** Baja
- **Descripción:** La aplicación debe permitir al usuario descargar los tickets almacenados en formato CSV para su uso externo.
- **Entradas:** Solicitud del usuario para exportar tickets.
- **Proceso:** Generación de un archivo CSV con los datos almacenados.
- **Salidas:** Archivo CSV descargado en el dispositivo del usuario.

RQ15

- **Nombre:** Eliminación de cuenta
- **Prioridad:** Alta
- **Descripción:** La aplicación debe permitir al usuario eliminar su cuenta, incluyendo la eliminación de todos los datos asociados en la base de datos.
- **Entradas:** Solicitud del usuario para eliminar su cuenta.
- **Proceso:** Eliminación de los datos del usuario y los tickets asociados en Firebase.
- **Salidas:** Confirmación de que la cuenta y los datos han sido eliminados.

5 Diseño

En esta fase del proyecto, se detalla el proceso de planificación y estructuración de la aplicación, abordando desde la conceptualización inicial hasta la implementación técnica de cada funcionalidad. El objetivo principal de esta etapa es definir cómo se organizan y conectan los diferentes componentes de la aplicación para cumplir con los requisitos previamente establecidos.

Se presentará un diseño claro y estructurado que incluye la arquitectura de la aplicación, los diagramas de flujo de trabajo y la descripción de las principales interfaces de usuario. Además, se explicará paso a paso cómo se implementaron las funcionalidades clave, desde la autenticación con Google hasta el almacenamiento y manejo de tickets en la nube, asegurando que cada decisión técnica esté alineada con los objetivos del proyecto y garantizando una experiencia de usuario óptima.

5.1 Arquitectura de la Aplicación

En este apartado se mostrará un diagrama de arquitectura detallado de cómo el usuario interactúa con la aplicación y los diferentes componentes involucrados como la base de datos de Firebase o la Api de Gemini.

Ilustración 10: Logo de Android

5.1.1 Diagrama de Arquitectura

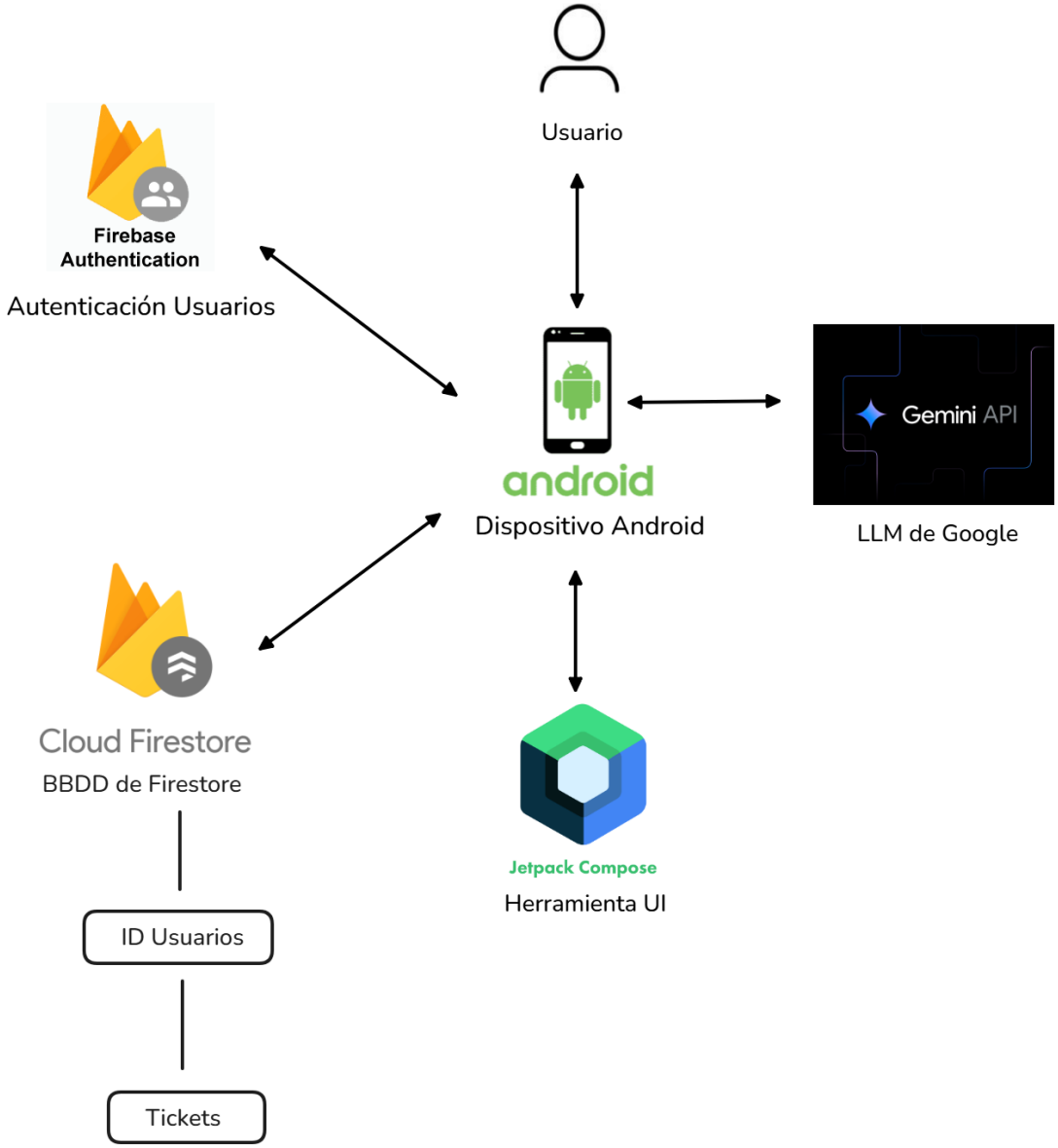


Ilustración 11: Diagrama de Arquitectura

5.1.2 Patrón de Diseño

En este proyecto, se ha optado por utilizar el patrón de diseño MVVM (Model-View-ViewModel). Este modelo es ampliamente reconocido en el desarrollo de aplicaciones Android debido a su capacidad para separar las responsabilidades de los diferentes componentes, facilitando la mantenibilidad, la escalabilidad y las pruebas del código.

Ventaja del modelo MVVM:

1. **Compatibilidad con Jetpack Compose:**
   Este patrón funciona perfectamente con **Jetpack Compose**, la herramienta utilizada para construir la interfaz de usuario de la aplicación. MVVM aprovecha la naturaleza reactiva de Compose para sincronizar automáticamente los datos entre la lógica de negocio y la UI.
2. **Separación de responsabilidades:**
   La lógica de negocio, el manejo de datos y la representación de la interfaz están claramente separados, lo que evita que el código de la UI se mezcle con la lógica de la aplicación.
3. **Reactividad y flujo de datos:**
   La aplicación requiere actualizaciones constantes de datos provenientes de Firebase y la API del LLM. MVVM permite gestionar estos datos de forma eficiente y actualizar la interfaz sin intervención manual del desarrollador.
4. **Facilidad para pruebas:**
   Gracias a la separación entre la UI y la lógica de negocio, el **ViewModel** se puede probar de manera independiente, lo que mejora la calidad del desarrollo.

Funcionamiento de MVVM en la aplicación

1. **Model (M):**

   El Model representa la capa de datos de la aplicación.

   En esta app, incluye la interacción con los servicios de Firebase (Authentication y Firestore) y la API del LLM (Gemini). [6]

   Ejemplo: Consultar la base de datos Firestore para recuperar los tickets almacenados o enviar una imagen a la API del LLM para procesarla.

2. **ViewModel (VM):**

   El ViewModel actúa como intermediario entre el Model y la View.

   Gestiona los datos y la lógica de negocio para proporcionar un estado actualizado a la interfaz de usuario.

   Ejemplo: Envía las imágenes a la API del LLM, recibe los datos procesados y los transforma para que sean presentados de forma amigable en la UI. También gestiona el estado de autenticación del usuario.

3. **View (V):**

   La View corresponde a la interfaz de usuario (UI) diseñada con Jetpack Compose.

   Su función es mostrar los datos al usuario y capturar las interacciones.

   Ejemplo: Mostrar los tickets almacenados, permitir al usuario filtrarlos por fecha o editar un ticket en tiempo real.

5.2 Diagramas de Flujo

En este apartado, los diagramas de flujo se utilizan para describir los procesos clave de la aplicación, como la autenticación de usuarios, el procesamiento y almacenamiento de tickets, la gestión de datos en la nube y la exportación de información. Estos diagramas ofrecen una perspectiva detallada de las secuencias lógicas y de las decisiones involucradas en cada funcionalidad, lo que ayuda a garantizar que los requisitos sean implementados correctamente.

A continuación, se presentan los diagramas de flujo principales, que ilustran el funcionamiento interno de la aplicación y su interacción con los servicios externos, como Firebase y la API del LLM.

5.2.1 Autenticación con Google

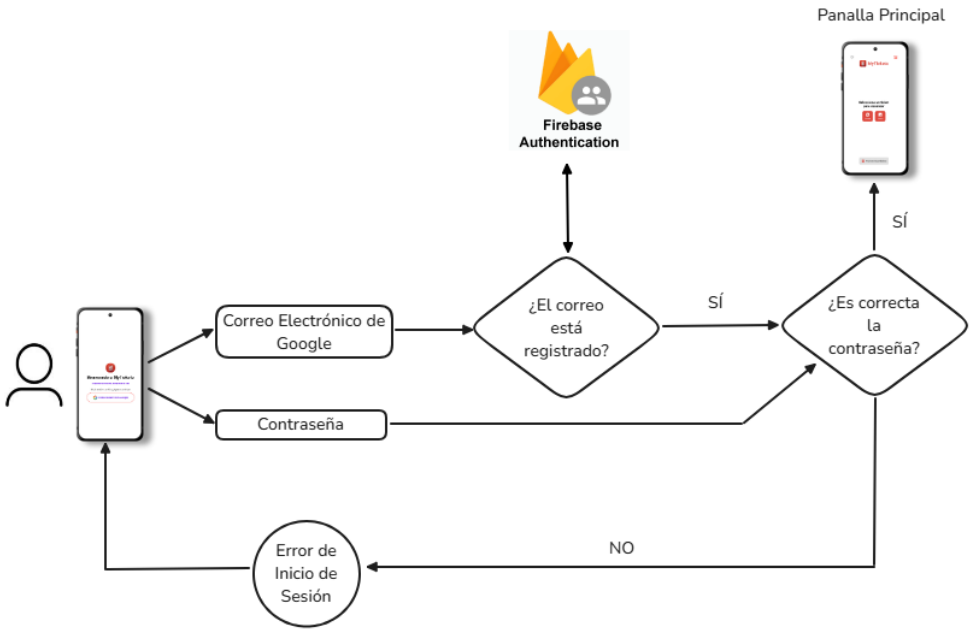


Ilustración 12: Diagrama de Autenticación con Google

5.2.2 Procesamiento de Tickets

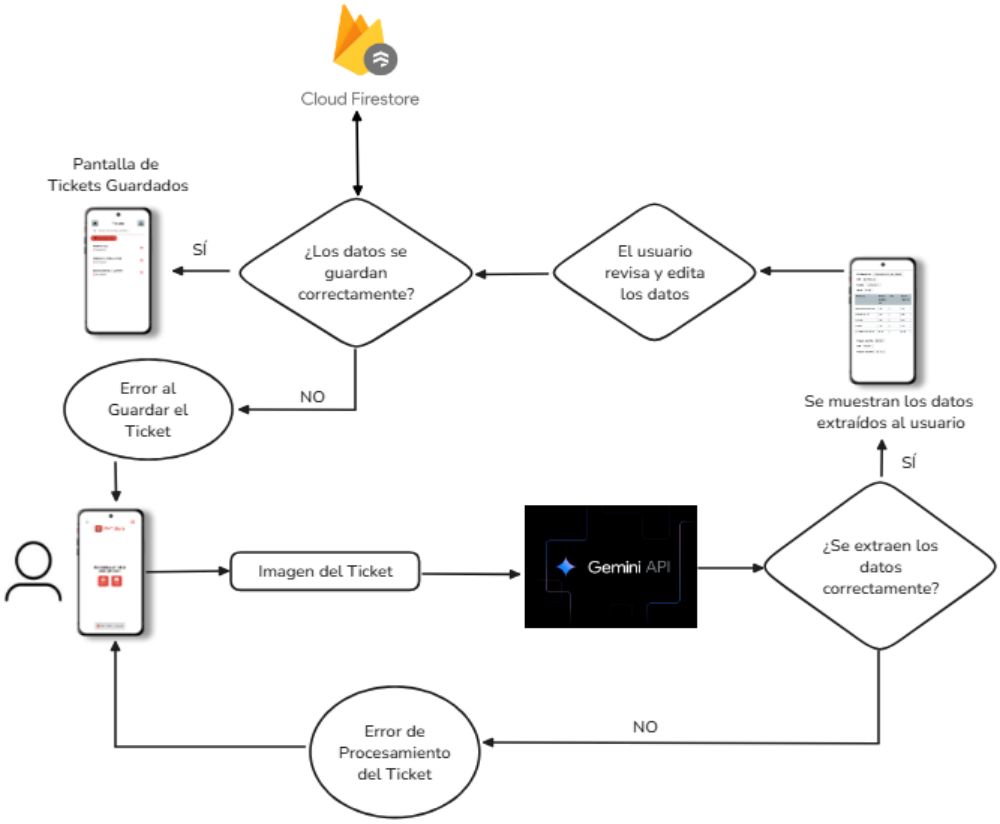


Ilustración 13: Diagrama de Procesamiento de Tickets

5.2.3 Gestión de Tickets

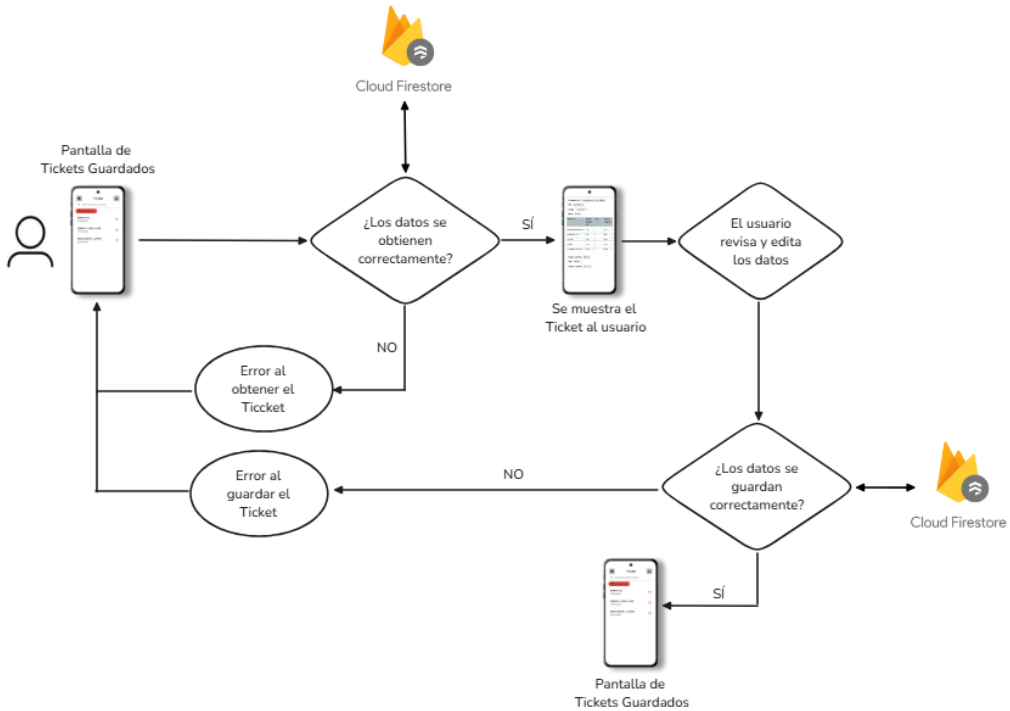


Ilustración 14: Diagrama de Gestión de Tickets

5.3 Diseño de la Base de Datos

El diseño de la base de datos es un aspecto fundamental para garantizar el correcto almacenamiento, acceso y manipulación de los datos en la aplicación. Durante la fase inicial del proyecto, se consideró utilizar una base de datos local con SQLite debido a su simplicidad y facilidad de uso en aplicaciones móviles. Sin embargo, a medida que el proyecto avanzó, se decidió migrar a Firebase Firestore, una base de datos en la nube basada en tecnología NoSQL, por las siguientes ventajas:

- **Escalabilidad:** Firebase Firestore permite manejar grandes cantidades de datos y usuarios sin necesidad de configuraciones complejas.
- **Acceso en tiempo real:** Las actualizaciones en la base de datos se sincronizan automáticamente entre los dispositivos conectados.
- **Integración con Firebase Authentication:** Ofrece un sistema seguro para gestionar el acceso a los datos mediante autenticación.
- **Almacenamiento en la nube:** Elimina la dependencia del almacenamiento local, permitiendo acceder a los datos desde cualquier dispositivo.
- **Flexibilidad NoSQL:** La estructura basada en colecciones y documentos es más adaptable a cambios futuros en los datos o funcionalidades de la aplicación.

Por estas razones, Firebase Firestore resultó ser la opción más adecuada para los objetivos y requisitos de la aplicación.

5.3.1 Estructura de Firebase: Organización de Colecciones y Documentos

En Firebase Firestore, los datos se organizan en colecciones (grupos de documentos) y documentos (estructuras de datos que contienen campos clave-valor). La estructura diseñada para esta aplicación es la siguiente:

users (Colección):

ID del Usuario (Documento):

Almacena el email del usuario.

Subcolecciones:

tickets (Colección):

ID del Ticket (Documento):

Contiene los datos extraídos de cada ticket procesado por la aplicación (restaurante, cif, fecha…)

Subcolecciones:

items (Colección):

ID del item (Documento):

Contiene los elementos asociados a cada producto (item, cantidad, precio…)

Ejemplo de jerarquía de datos:

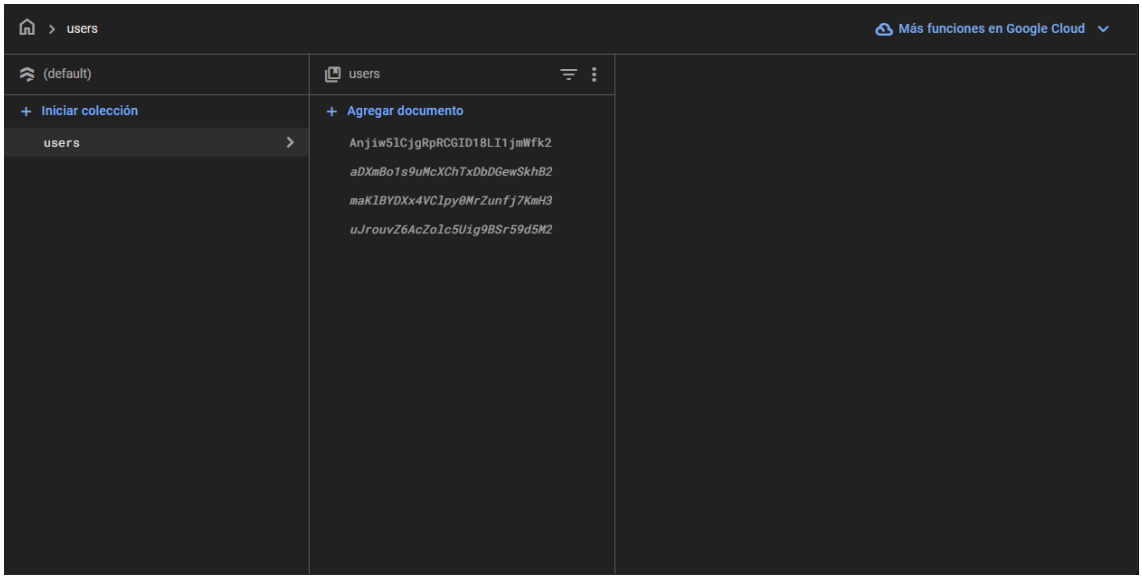


Ilustración 15: Ids de Usuario

En esta pantalla podemos ver los ID aleatorios generados para cada usuario que ha iniciado sesión en la aplicación.

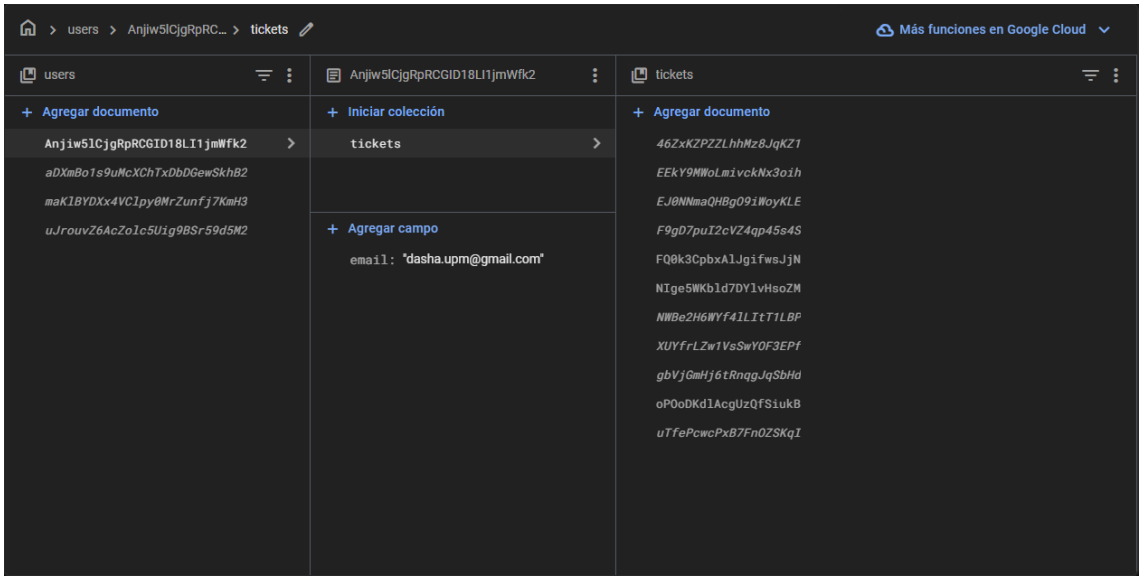


Ilustración 16: Ids de Tickets

En esta pantalla observamos la colección “tickets” que contiene múltiples documentos asociados a cada uno de los tickets almacenados en la cuenta y con un ID generado aleatoriamente para cada uno de ellos, además del campo “email” donde se guarda el correo electrónico de la cuenta de Google asociada.

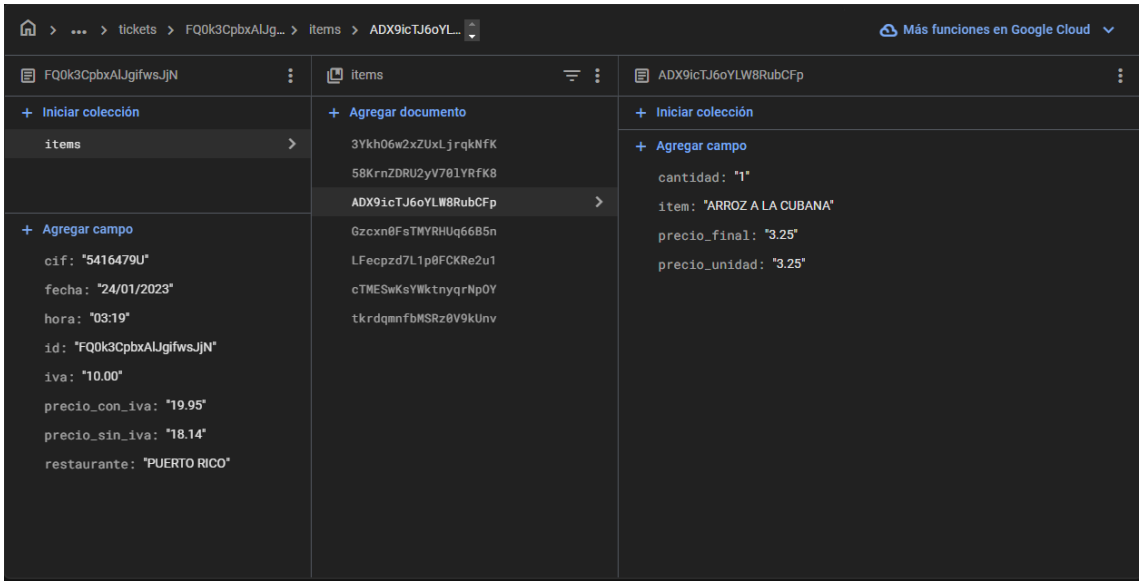


Ilustración 17: Ids de Productos

Por último, en esta pantalla podemos ver los campos asociados al documento del ticket en cuestión (“restaurante: Puerto Rico”, “fecha: 24/01/2023”, etc), además de una subcolección de “items” con sus IDs aleatorios, que alberga los diferentes productos que se han extraído, junto con la cantidad de los mismos, el precio unitario y el precio final.

5.3.2 Reglas de Seguridad en Firestore

Las reglas de seguridad en Firestore son un conjunto de instrucciones que determinan quién puede acceder, modificar o eliminar documentos y colecciones en la base de datos. Estas reglas son fundamentales para proteger los datos de los usuarios y evitar accesos no autorizados.

Firestore permite definir reglas detalladas basadas en:

1. **Autenticación:** Controla el acceso según el estado de autenticación del usuario.
2. **Propiedad de los datos:** Permite que cada usuario acceda únicamente a los datos que le pertenecen.

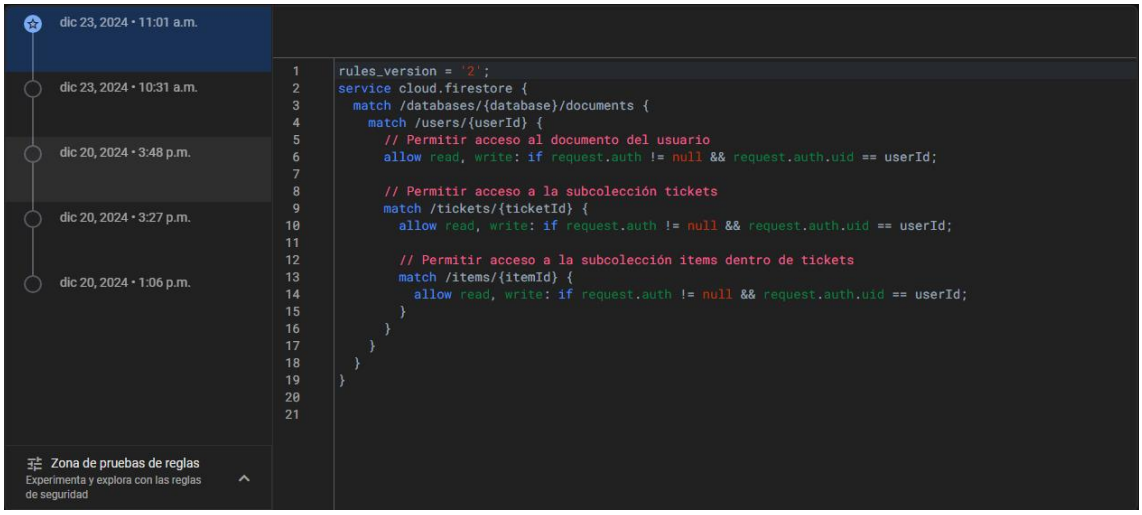


Ilustración 18: Reglas Firebase

En la imagen proporcionada podemos ver las reglas definidas para nuestra aplicación, analicemos paso a paso cada una de ellas:

```
match /databases/{database}/documents {
```

Ilustración 19: Reglas 1

Esta regla abarca todas las colecciones y documentos dentro de la base de datos Firestore.

```
match /users/{userId} {
  // Permitir acceso al documento del usuario
  allow read, write: if request.auth != null && request.auth.uid == userId;
```

Ilustración 20: Reglas 2

Permite que cada usuario autenticado acceda únicamente a su propio documento en la colección `users`.

Condiciones:

- request.auth != null: Verifica que el usuario está autenticado.
- request.auth.uid == userId: Garantiza que el usuario solo accede a los datos asociados con su uid (identificador único asignado por Firebase Authentication).

```
// Permitir acceso a la subcolección tickets
match /tickets/{ticketId} {
  allow read, write: if request.auth != null && request.auth.uid == userId;
```

Ilustración 21: Reglas 3

Permite que un usuario acceda a sus propios tickets dentro de la subcolección `tickets`.

Condiciones:

- Igual que en el documento del usuario, verifica que el usuario está autenticado y que el `userId` coincide con el `uid` del usuario autenticado.

```
// Permitir acceso a la subcolección items dentro de tickets
match /items/{itemId} {
  allow read, write: if request.auth != null && request.auth.uid == userId;
}
```

Ilustración 22: Reglas 4

Extiende la lógica de seguridad a una subcolección más específica (`items`) que pueda contener información detallada de cada ticket (por ejemplo, lista de productos comprados en el ticket).

Condiciones:

- Solo el usuario autenticado con el `uid` correspondiente puede leer o escribir en esta subcolección.

Puntos Clave de Seguridad:

1. **Acceso Condicional:** Todas las reglas dependen de que el usuario esté autenticado (request.auth != null) y de que su uid coincida con el identificador del documento o subcolección.
2. **Propiedad Exclusiva de Datos:** Cada usuario solo puede acceder a su propia información. Los datos de un usuario están aislados de otros usuarios.
3. **Jerarquía de Subcolecciones:** Las reglas aseguran que la seguridad se aplique no solo a los documentos principales, sino también a las subcolecciones (tickets y items).

5.4 Interfaz de Usuario (UI)

5.4.1 Diagrama de Vistas

El objetivo de este esquema es representar las posibles rutas que puede tomar el usuario en la aplicación, destacando cómo las diferentes pantallas están conectadas entre sí. Esto ayudará a visualizar la experiencia del usuario y a estructurar la navegación de la aplicación.

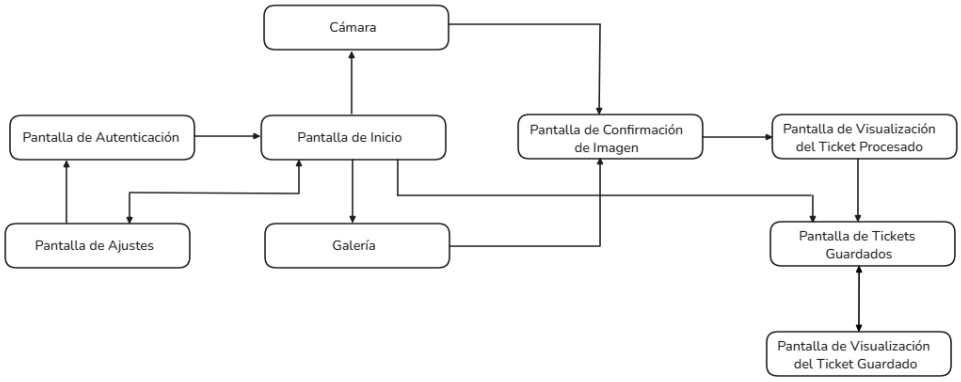


Ilustración 23: Diagrama de Vistas

5.4.2 Diseño de las Pantallas

El diseño de las pantallas de MyTickets ha sido concebido para garantizar una experiencia de usuario intuitiva y eficiente, priorizando la claridad visual y la simplicidad en la navegación. Este apartado describe las decisiones de diseño tomadas y las principales características de las pantallas implementadas.

5.4.2.1 Pantalla de Autenticación

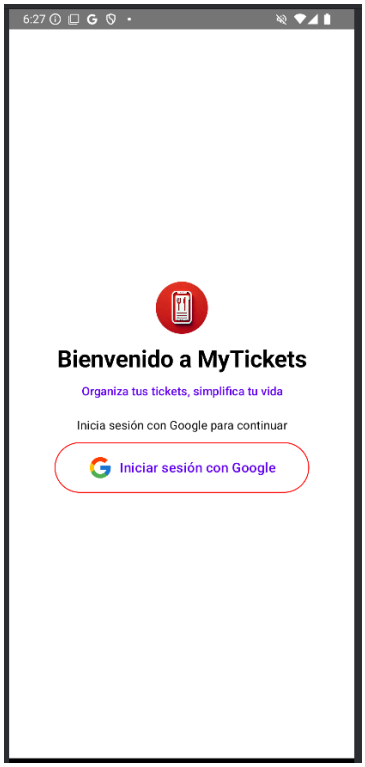


Ilustración 25: Pantalla Autenticación

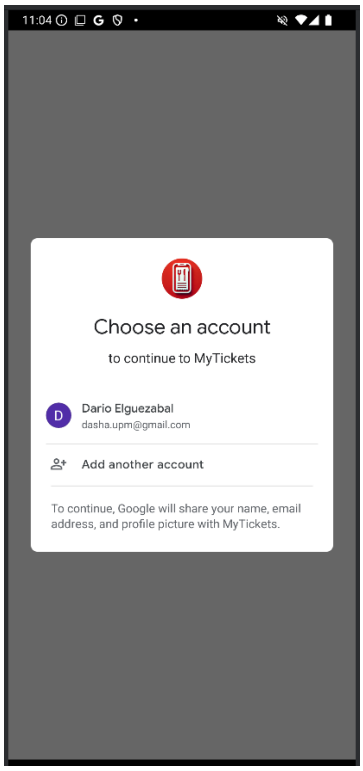


Ilustración 24: Pantalla de Inicio de Google

La Pantalla de Autenticación es la primera vista que se presenta al usuario al abrir la aplicación. Su objetivo principal es permitir al usuario iniciar sesión mediante su cuenta de Google para acceder al resto de funcionalidades.

En la parte superior se encuentra el icono de la aplicación de “MyTickets”, junto con un texto de bienvenida "Bienvenido a MyTickets", acompañado de una breve descripción: "Organiza tus tickets, simplifica tu vida". Esto introduce al usuario al propósito de la aplicación.

Justo en el centro de la pantalla, se muestra un botón con el texto "Iniciar sesión con Google", acompañado del logotipo de Google. Este botón es el único elemento interactivo de la pantalla, lo que facilita la acción inicial del usuario.

5.4.2.2 Pantalla de Inicio

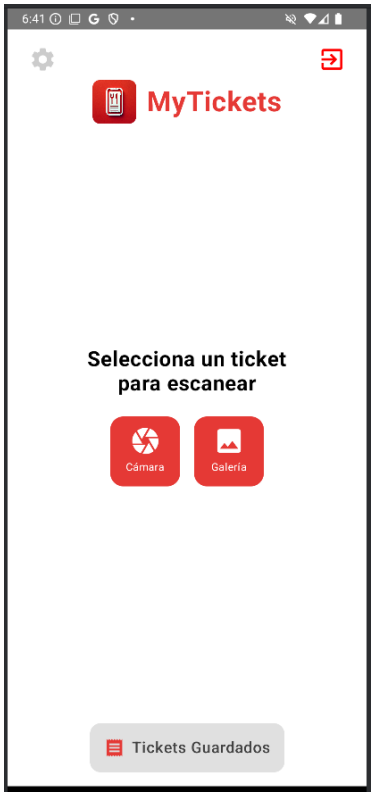


Ilustración 26: Pantalla de Inicio

Ilustración 28: Ejemplo Cámara

Ilustración 27: Ejemplo Galería

La Pantalla de Inicio es la pantalla principal de acceso a la aplicación, es la primera que se le mostrará al usuario al ingresar a la aplicación, una vez haya iniciado sesión con la cuenta de Google.

En esta pantalla poder observar distintos elementos interactivos, entre ellos:

Icono de cámara: permite escanear imágenes a través de la misma.

Icono de galería: permite seleccionar una imagen de preferencia.

Botón de tickets guardados: permite acceder a la pantalla donde se despliegan los tickets almacenados en la nube.

Botón de ajustes: permite acceder a una pantalla de configuración.

Botón de salida: permite cerrar la sesión de la cuenta actual, lleva a la Pantalla de Autenticación.

5.4.2.3 Pantalla de Tickets Guardados

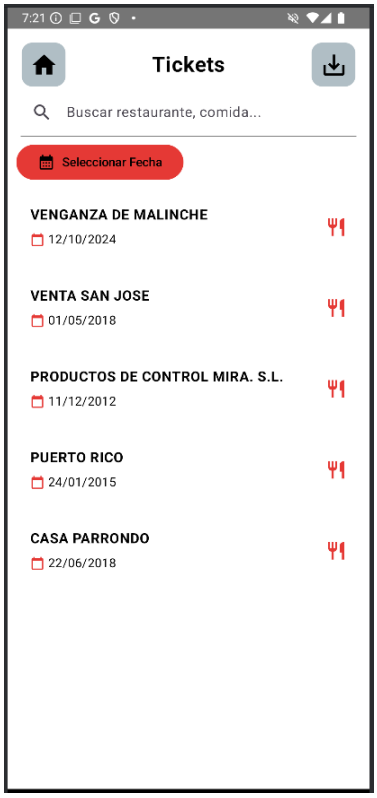


Ilustración 29: Pantalla Tickets

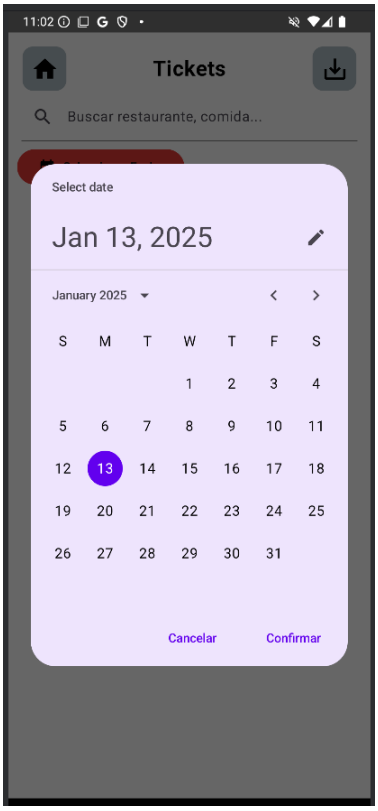


Ilustración 30: Pantalla Calendario

La Pantalla de Tickets Guardados es a la que se accede a través del botón descrito en el apartado anterior, o bien después de guardar un ticket procesado.

En esta pantalla existen diferentes componentes a tener en cuenta:

Barra de búsqueda: permite filtrar tickets a través de diversos campos (restaurante, comida, precio…), apareciendo en pantalla los tickets que coinciden con estas restricciones.

Botón de Home: permite retroceder a la Pantalla de Inicio.

Botón de Seleccionar Fecha: despliega un calendario interactivo para que el usuario filtre los tickets de un día concreto.

Botón de Descarga: permite descargar en formato .csv los tickets guardados.

Lista de tickets guardados: despliega una lista con todos los tickets guardados, siendo cada uno de estos "clickable" y accediendo a la Pantalla de Visualización del Ticket Guardado, que mostraremos a continuación.

5.4.2.4 Pantalla de Visualización del Ticket Guardado

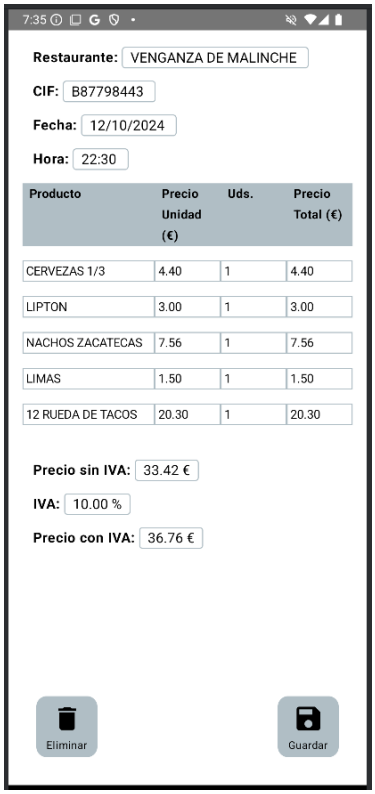


Ilustración 31: Pantalla Ticket Guardado

La pantalla de Visualización del Ticket Guardado muestra los datos extraídos por el LLM de Gemini, en un formato elegante y legible, donde se han seleccionado los elementos de mayor importancia pertenecientes a la imagen del ticket.

En esta pantalla todos los datos extraídos son editables (Restaurante, CIF, precios, productos…). La Inteligencia Artificial de Gemini es muy potente y poco propensa a errores, pero en caso de existir alguna discrepancia con algún dato real, el usuario posee la opción de corregirlo.

Además, observamos dos botones:

Botón de Eliminar: permite eliminar el ticket de la base de datos, lleva a la Pantalla de Tickets Guardados.

Botón de Guardar: permite guardar/actualizar los datos que se presentan tras la edición/visualización del ticket, lleva a la Pantalla de Tickets Guardados.

5.4.2.5 Pantalla de Visualización del Ticket Procesado

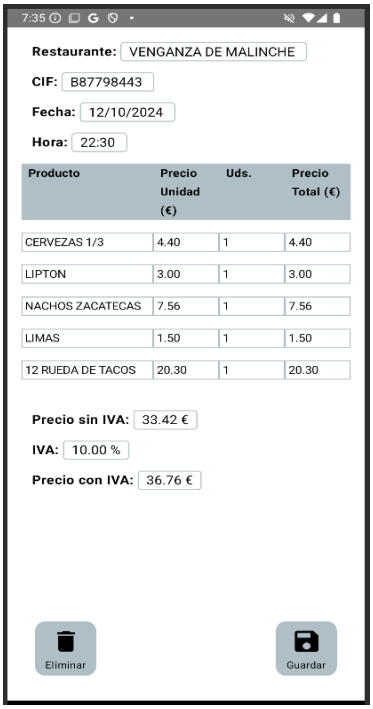


Ilustración 32: Pantalla Ticket Procesado

La pantalla de Visualización del Ticket Procesado es, en esencia, una réplica de la pantalla anterior, pero con sutiles diferencias.

Esta pantalla es únicamente accesible tras el procesado de una nueva imagen. Esto implica que, a diferencia de la pantalla anterior, al pulsar el botón de guardar se añade un nuevo ticket a la base de datos, en lugar de actualizarlo. Posteriormente, lleva a la Pantalla de Tickets Guardados.

El botón de eliminar, en este caso, es el encargado de descartar la operación de guardado del ticket, volviendo así a la Pantalla de Inicio.

5.4.2.6 Pantalla de Confirmación de Imagen

La Pantalla de Confirmación de Imagen es la que se obtiene tras la captura de una imagen con el uso de la cámara o la selección de la misma a través de la galería.

Esta pantalla muestra una previsualización de la imagen seleccionada para que el usuario confirme que ésta es la correcta para el procesado.

Además, observamos dos botones:

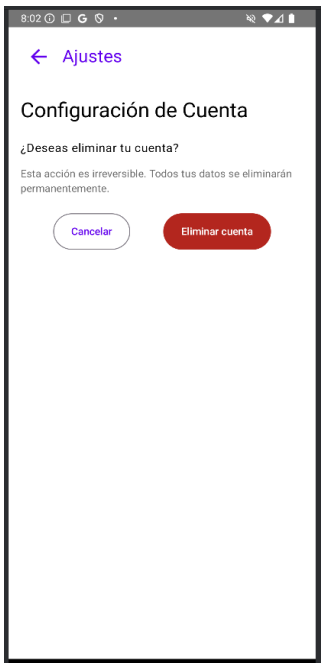
Botón de No: cancela el proceso y devuelve al usuario a la Pantalla de Inicio.

Botón de Sí: confirma voluntad del usuario de querer seguir adelante con el proceso.

Al pulsar este botón, se envía la imagen elegida junto con un prompt personalizado al LLM de Gemini para posteriormente, obtener la respuesta del modelo y procesarla.

Ilustración 33: Pantalla Confirmación

5.4.2.7 Pantalla de Ajustes



La Pantalla de Ajustes es accesible desde el icono de ajustes de la Pantalla de Inicio.

Esta pantalla otorga la posibilidad al usuario de eliminar su cuenta de la base de datos junto con todos los tickets asociados a la misma.

Se observan 3 elementos interactivos en esta pantalla:

Botón de Cancelar: permite cancelar el proceso de eliminación de la cuenta y devuelve al usuario a la Pantalla de Inicio.

Botón de Eliminar Cuenta: permite eliminar la cuenta junto con sus datos de forma permanente.

Botón de Atrás (Flecha): no realiza ningún cambio y devuelve al usuario a la Pantalla de Inicio.

Ilustración 34: Pantalla de Ajustes

6 Implementación

El proyecto se ha subido de forma gradual a la plataforma de Github. A continuación proporciono el link:

https://github.com/darioes04/tfg

6.1 Preparación del Entorno

La preparación del entorno de desarrollo es una etapa clave para garantizar que el proyecto se pueda implementar de manera eficiente. En esta sección, se describe cómo se configuró el entorno y las herramientas necesarias para llevar a cabo el desarrollo de la aplicación.

Android Studio:

- Es el entorno de desarrollo integrado (IDE) utilizado para construir la aplicación.
- Incluye soporte nativo para Kotlin, Jetpack Compose, y Firebase, lo que facilita el desarrollo de aplicaciones modernas para Android.

Kotlin:

- Es el lenguaje de programación elegido para el desarrollo de la aplicación debido a su integración nativa con Android y su compatibilidad con herramientas como Jetpack Compose.

Jetpack Compose:

- Herramienta de UI moderna para Android que permite construir interfaces de usuario de manera declarativa.
- Utilizada para diseñar las pantallas de la aplicación, aprovechando su simplicidad y eficiencia.

Firebase:

- Servicios integrados en la aplicación:

  Firebase Authentication: Para la autenticación de usuarios mediante Google.

  Firebase Firestore: Para el almacenamiento en la nube de tickets y datos del usuario.

GitHub:

- Se utilizó como sistema de control de versiones, subiendo los cambios al repositorio principal en la rama `main` para mantener el código organizado y realizar un seguimiento de las actualizaciones.

Configuración Inicial

1. Creación del Proyecto:

Se creó un nuevo proyecto en Android Studio seleccionando Kotlin como lenguaje principal y habilitando Jetpack Compose.

Durante la configuración, se eligió una compatibilidad mínima con dispositivos Android 14 (API Level 29) para garantizar una cobertura amplia de usuarios.

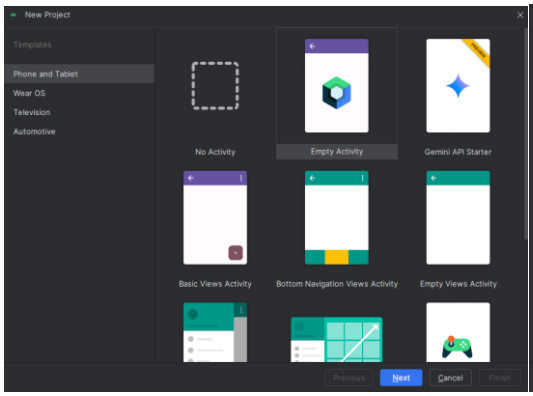


Ilustración 35: Creación Proyecto

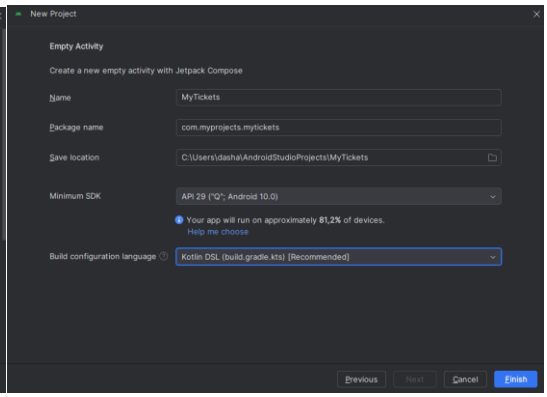


Ilustración 36: Versión Android

2. Integración de Firebase:

Conexión del Proyecto:

El proyecto de Android se vinculó a Firebase mediante Firebase Console. Se añadieron los servicios de Authentication y Firestore configurando el archivo `google-services.json` proporcionado por Firebase.

Reglas de Seguridad:

Se definieron las reglas iniciales en Firestore para garantizar que solo los usuarios autenticados pudieran acceder a los datos.

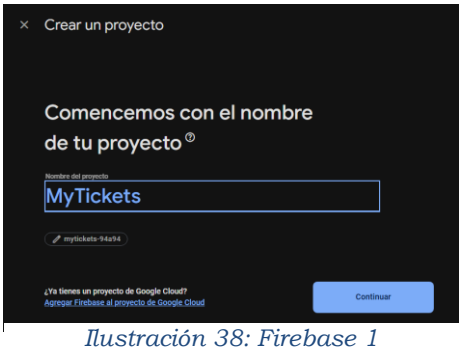


Ilustración 38: Firebase 1

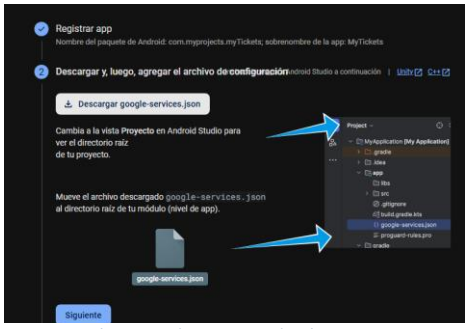


Ilustración 37: Firebase 2

3. Instalación de Dependencias:

Se añadieron las siguientes dependencias al archivo build.gradle para integrar las herramientas y librerías necesarias:

Ilustración 39: Dependencias Firebase

4. Configuración de GitHub:

Se inicializó un repositorio en GitHub para almacenar el código de la aplicación.

La rama `main` se utilizó para realizar commits de las versiones estables del proyecto.

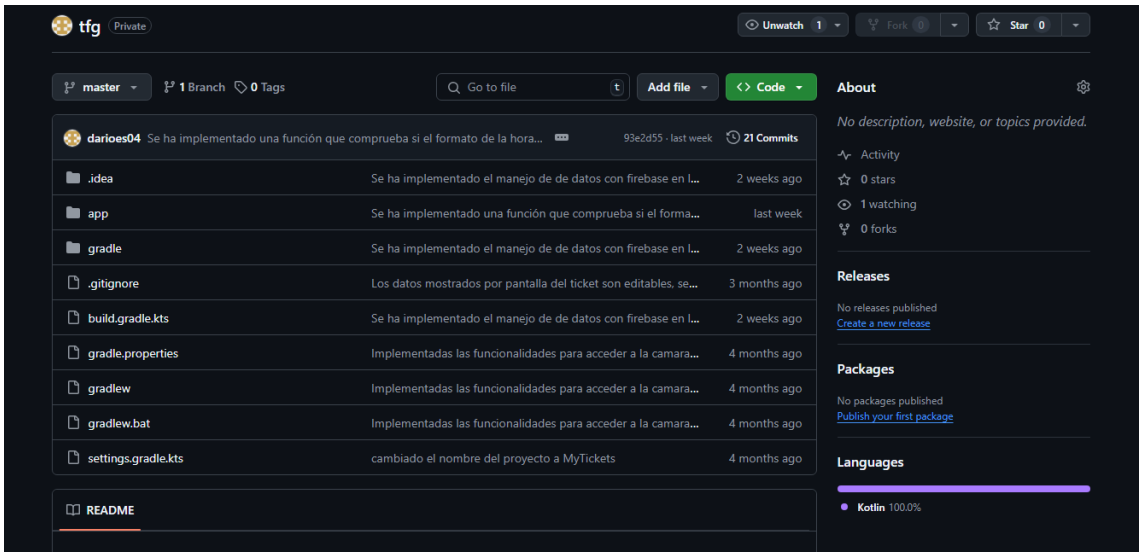


Ilustración 40: Proyecto de Github

6.2 Implementación de Funcionalidades y Estructura

6.2.1 Estructura de Archivos

La aplicación ha sido desarrollada siguiendo una estructura de archivos modular y organizada, diseñada para facilitar la escalabilidad, el mantenimiento y la comprensión del proyecto. Cada funcionalidad principal de la aplicación ha sido separada en paquetes específicos, agrupando los archivos según su responsabilidad. A continuación, se describe la estructura general:

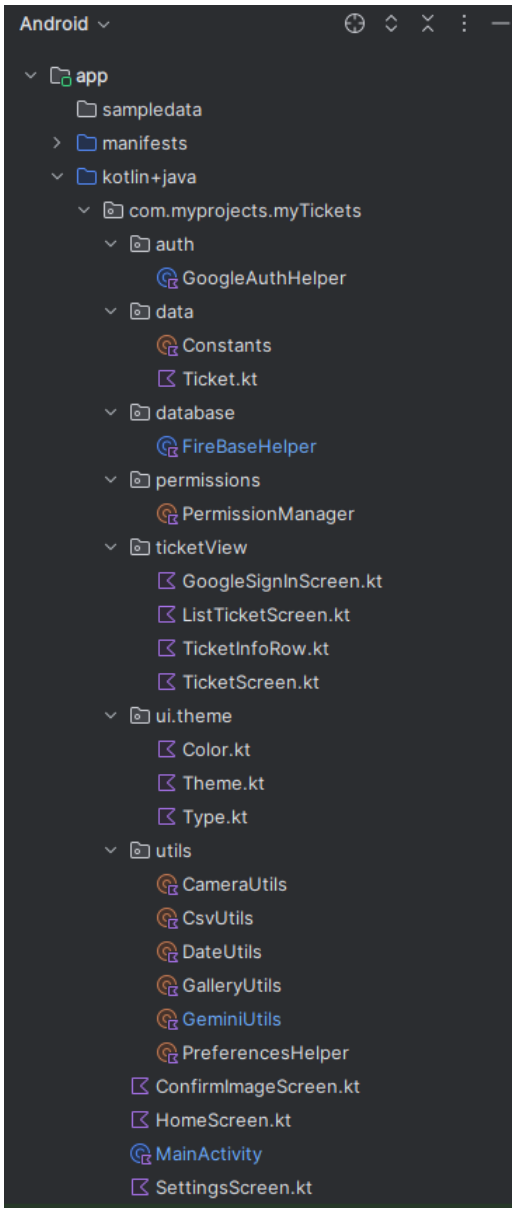


Ilustración 41:Estructura Archivos

Descripción General de los Paquetes:

1. **auth:** contiene las clases relacionadas con la autenticación, como GoogleAuthHelper, que maneja la autenticación mediante cuentas de Google.
2. **data:** agrupa las definiciones de datos utilizados en la aplicación, como las clases Ticket y Constants, que representan la estructura del ticket y las constantes globales.
3. **database:** incluye la clase FireBaseHelper, que gestiona todas las operaciones relacionadas con Firebase, como guardar, actualizar y eliminar tickets.
4. **permissions:** contiene PermissionManager, responsable de manejar los permisos necesarios para acceder a recursos del dispositivo, como la cámara y la galería.
5. **ticketView:** este paquete contiene las pantallas principales relacionadas con la gestión de tickets, como:

    - GoogleSignInScreen.kt: Pantalla de inicio de sesión.

    - ListTicketScreen.kt: Pantalla para listar los tickets almacenados.

    - TicketScreen.kt: Pantalla para visualizar los detalles de un ticket.

6. **ui.theme:** maneja los estilos y configuraciones visuales de la aplicación, como colores, tipografía y temas.
7. **utils:** agrupa utilidades reutilizables que facilitan diversas operaciones en la aplicación, como:

    - CameraUtils y GalleryUtils: Para gestionar la captura de fotos y la selección de imágenes.

    - GeminiUtils: Para procesar imágenes y extraer datos mediante la API de Gemini.

    - CsvUtils: Para exportar tickets en formato CSV.

    - DateUtils: Para validar y formatear fechas.

    - PreferencesHelper: Para gestionar configuraciones y preferencias de usuario.

8. **Archivos Principales:**

    - MainActivity.kt: Punto de entrada de la aplicación que orquesta la navegación y las interacciones entre pantallas.

    - HomeScreen.kt y SettingsScreen.kt: Pantallas principales que ofrecen accesos a las funcionalidades clave y ajustes de la aplicación.

6.2.2 Archivos Relevantes e Implementaciones

A continuación se describe el funcionamiento de los métodos y componentes responsables del funcionamiento de la aplicación.

6.2.2.1 MainActivity.kt

1. Navegación entre pantallas

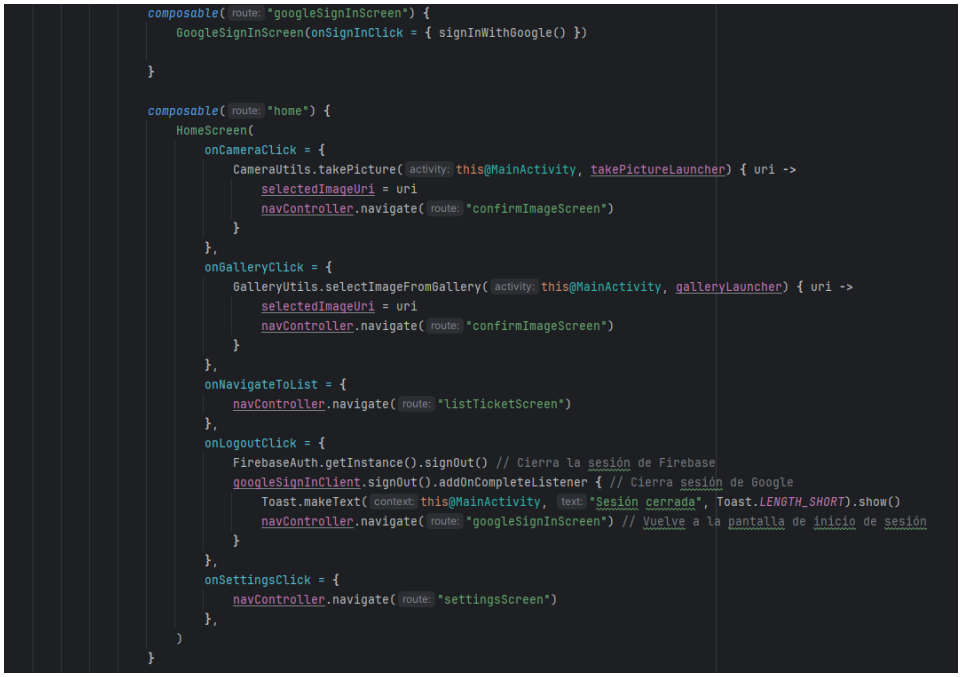
La navegación entre pantallas es un componente esencial a la hora de desarrollar una aplicación en Android. Navigation es la biblioteca que facilita la implementación de la navegación entre diferentes pantallas (o destinos) dentro de una aplicación Android. Proporciona una estructura declarativa para definir las rutas, gestionar el estado del historial de navegación y manejar transiciones entre pantallas. Esto simplifica el desarrollo y asegura una experiencia de usuario fluida y coherente.

En este proyecto, se ha utilizado la biblioteca Navigation Compose, que es una extensión de la biblioteca Navigation específicamente diseñada para aplicaciones que utilizan Jetpack Compose. Esta herramienta permite declarar las rutas de navegación como composables, integrándose de manera nativa con la arquitectura moderna de Android.

El NavHost se configura en la clase MainActivity, donde se define el punto de entrada (pantalla inicial) y se conectan las distintas pantallas de la aplicación mediante destinos (composable). Esto garantiza una navegación ordenada y permite manejar estados como la autenticación del usuario o eventos como el regreso a pantallas previas.

```
NavHost(
    navController = navController,
    startDestination = if (isUserLoggedIn) "home" else "googleSignInScreen"
) {
```

Ilustración 42: NavHost



```
composable( route: "googleSignInScreen") {
    GoogleSignInScreen(onSignInClick = { signInWithGoogle() })

}

composable( route: "home") {
    HomeScreen(
        onCameraClick = {
            CameraUtils.takePicture( activity: this@MainActivity, takePictureLauncher) { uri ->
                selectedImageUri = uri
                navController.navigate( route: "confirmImageScreen")
            }
        },
        onGalleryClick = {
            GalleryUtils.selectImageFromGallery( activity: this@MainActivity, galleryLauncher) { uri ->
                selectedImageUri = uri
                navController.navigate( route: "confirmImageScreen")
            }
        },
        onNavigateToList = {
            navController.navigate( route: "listTicketScreen")
        },
        onLogoutClick = {
            FirebaseAuth.getInstance().signOut() // Cierra la sesión de Firebase
            googleSignInClient.signOut().addOnCompleteListener { // Cierra sesión de Google
                Toast.makeText( context: this@MainActivity,  text: "Sesión cerrada", Toast.LENGTH_SHORT).show()
                navController.navigate( route: "googleSignInScreen") // Vuelve a la pantalla de inicio de sesión
            }
        },
        onSettingsClick = {
            navController.navigate( route: "settingsScreen")
        },
    )
}
```

Ilustración 43: Ejemplos Screens

En esta imagen, podemos ver dos pantallas “composable” que llaman a sus respectivas funciones donde se implementa el diseño de cada una. En este caso, “googleSignInScreen” se trata de la Pantalla de Autenticación y “home” la Pantalla de Inicio.

2. Autenticación con Google

La autenticación se implementa mediante Firebase Authentication con Google.

Al iniciar la aplicación, se verifica si el usuario ya está autenticado. Si no lo está, se muestra la Pantalla de Autenticación.

```
NavHost(
    navController = navController,
    startDestination = if (isUserLoggedIn) "home" else "googleSignInScreen"
) {
    composable( route: "googleSignInScreen") {
        GoogleSignInScreen(onSignInClick = { signInWithGoogle() })

    }
```

Ilustración 44: Navegación Inicial

Se autentica el usuario en Firebase con las credenciales de Google y navega a la pantalla principal si el proceso es exitoso gracias a la función “firebaseAuthWithGoogle”.

```
// Autentica el usuario en Firebase
private fun firebaseAuthWithGoogle(idToken: String) {
    val credential = GoogleAuthProvider.getCredential(idToken, null)
    FirebaseAuth.getInstance().signInWithCredential(credential)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                // Inicio de sesión exitoso
                Toast.makeText( context: this,  text: "Inicio de sesión exitoso", Toast.LENGTH_SHORT).show()

                // Guardar el email del usuario en Firestore
                lifecycleScope.launch {
                    try {
                        firebaseHelper.saveUserEmailToFirestore()
                        Log.d( tag: "Firestore",  msg: "Email del usuario guardado correctamente")
                    } catch (e: Exception) {
                        Log.e( tag: "Firestore",  msg: "Error al guardar el email: ${e.message}")
                    }
                }
                // Navegar a la pantalla principal
                navController.navigate( route: "home")
            } else {
                // Manejo del error
                Toast.makeText( context: this,  text: "Error al autenticar con Firebase", Toast.LENGTH_SHORT).show()
            }
        }
}
```

Ilustración 45: Método Autenticación

Si el inicio de sesión es exitoso, el email del usuario se guarda en Firestore y se navega a la Pantalla de Inicio (“home”).

3. Selección y Procesamiento de Imágenes

Como se ha explicado en el punto 5.4.2 sobre el diseño de las pantallas, el usuario tiene la posibilidad de realizar una fotografía con la cámara de su dispositivo, o bien puede escoger una imagen de la galería.

A continuación se detalla la parte del código encargada de manejar la captura de fotos y selección de imágenes desde la galería, respectivamente.

```
private fun setupActivityResultLaunchers() {
    takePictureLauncher = registerForActivityResult(ActivityResultContracts.TakePicture()) { success ->
        if (success) {
            selectedImageUri = CameraUtils.photoUri
            if (selectedImageUri != null) {
                Toast.makeText( context: this,  text: "Foto capturada", Toast.LENGTH_SHORT).show()
                navController.navigate( route: "confirmImageScreen") // Usa el NavController global
            } else {
                Toast.makeText( context: this,  text: "Error al capturar la foto", Toast.LENGTH_SHORT).show()
            }
        }
    }

    galleryLauncher = registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { result ->
        val imageUri: Uri? = result.data?.data
        selectedImageUri = imageUri
        if (selectedImageUri != null) {
            Toast.makeText( context: this,  text: "Imagen seleccionada", Toast.LENGTH_SHORT).show()
            navController.navigate( route: "confirmImageScreen") // Usa el NavController global
        } else {
            Toast.makeText( context: this,  text: "No se ha seleccionado ninguna imagen", Toast.LENGTH_SHORT).show()
        }
    }
}
```

Ilustración 46: Funcionamiento Cámara/Galería

Tras la selección, se guarda el Uri de la imagen en la variable "selectedImageUri", para posteriormente procesarla con la función "processImage".

```
private fun processImage(imageUri: Uri?, navController: NavHostController) {
    GeminiUtils.processImageWithGemini( context: this, imageUri) { apiResponse ->
        if (isCancelled) {
            isLoading = false
            return@processImageWithGemini
        }

        if (apiResponse.startsWith( prefix: "Error")) {
            Toast.makeText( context: this, apiResponse, Toast.LENGTH_SHORT).show()
        } else {
            val ticket = parseTicketJson(apiResponse)
            ticketState.value = ticket
            isLoading = false
            navController.navigate( route: "ticketScreen")
        }
    }
}
```

Ilustración 47: Método ProcessImage

El método "processImage" es responsable de tomar una imagen representada como un URI, convertirla en un formato procesable, y enviarla a la API de Gemini para extraer información estructurada en formato JSON.

Seguidamente, la API devuelve los datos procesados en formato JSON en la variable "apiResponse". Este JSON contiene toda la información relevante extraída del ticket. Si ocurre un error durante el procesamiento, se devuelve un mensaje de error al usuario.

Por último se deserializa el JSON obtenido con el método parseTicketJson para convertirlo en un objeto de tipo "Ticket", lo que será de gran utilidad a la hora de tratar los datos y mostrarlos en las diferentes pantallas.

"Ticket" es una estructura de datos que contiene los siguientes atributos:

```kotlin
data class Ticket(
    val id: String = "",
    val restaurante: String,
    val cif: String,
    val fecha: String,
    val hora: String,
    val items: MutableList<Producto>,
    val precioSinIva: String,
    val iva: String,
    val precioConIva: String
)

data class Producto(
    var item: String,
    var cantidad: String,
    var precioUnidad: String,
    var precioFinal: String
)
```

Ilustración 48: Objeto Ticket

4. Visualización de Datos en la Pantalla “TicketScreen”

La pantalla "TicketScreen" está diseñada para mostrar al usuario los datos extraídos de un ticket, permitiéndole revisarlos y realizar acciones adicionales como guardar, editar o eliminar el ticket.

Para llevar esto a cabo, la función recibe entre otros argumentos el objeto de tipo “Ticket” que alberga los datos obtenidos de forma estructurada para su posterior visualización.

```kotlin
composable( route: "ticketScreen") {
    ticketState.value?.let { ticket ->
        TicketScreen(
            ticket = ticket,
            onConfirmClick = { updatedTicket ->
                lifecycleScope.launch {
                    try {
                        firebaseHelper.addTicket(updatedTicket)
                        Toast.makeText(
                            context: this@MainActivity,
                            text: "Ticket guardado correctamente",
                            Toast.LENGTH_SHORT
                        ).show()
                        navController.navigate( route: "listTicketScreen")
                    } catch (e: Exception) {
                        Toast.makeText(
                            context: this@MainActivity,
                            text: "Error al guardar el ticket: ${e.message}",
                            Toast.LENGTH_SHORT
                        ).show()
                    }
                }
            },
            onClickDelete = {
                Toast.makeText( context: this@MainActivity, text: "Ticket cancelado", Toast.LENGTH_SHORT).show()
                navController.navigate( route: "home") {
                    popUpTo( route: "home") { inclusive = true } // Elimina pantallas previas del historial
                }
            }
        )
    }
}
```

Ilustración 49: Función Pantalla de Visualización de Tickets

Como podemos observar en el código, el usuario que se encuentra en la pantalla "TicketScreen" tiene dos opciones:

OnconfirmClick: se activa al presionar el botón de guardar (Apartado 5.4.2.5). Contiene la lógica para almacenar un nuevo ticket llamando al método "addTicket" perteneciente a FirebaseHelper y posteriormente navegar a la pantalla "listTicketScreen" o Pantalla de Tickets Guardados.

OnCancelClick: se activa al presionar el botón de eliminar (5.4.2.5). Contiene la lógica para descartar el ticket y navegar a la pantalla de "home".

6.2.2.2 FirebaseHelper.kt

La clase FirebaseHelper actúa como un intermediario entre la aplicación y Firebase Firestore, proporcionando métodos específicos para gestionar los datos de los usuarios y sus tickets. Su principal objetivo es abstraer la complejidad de las operaciones con la base de datos, simplificando la interacción desde otras partes de la aplicación.

Principales Funcionalidades

1. **Agregar Tickets:**

   Permite guardar un ticket en Firestore, asociado al usuario autenticado mediante la llamada del método "addTicket".

   Los tickets se almacenan en una subcolección llamada tickets dentro del documento del usuario, y cada producto del ticket se guarda en una subcolección items.

2. **Obtener Tickets:**

   Recupera todos los tickets almacenados para el usuario actual mediante la llamada del método "getTickets".

   Cada ticket incluye su lista de productos, que también se obtienen de la subcolección items.

3. **Actualizar Tickets:**

   Permite modificar los datos de un ticket existente mediante la llamada del método "updateTicket"

   Actualiza tanto la información general del ticket como los productos asociados.

```
suspend fun updateTicket(ticket: Ticket) {
    val userId = getUserId() ?: throw IllegalStateException("Usuario no autenticado")
    val db = FirebaseFirestore.getInstance()

    // Referencia al documento específico
    val ticketRef = db.collection( collectionPath: "users")
        .document(userId)
        .collection( collectionPath: "tickets")
        .document(ticket.id)

    val ticketData = hashMapOf(
        "id" to ticket.id,
        "restaurante" to ticket.restaurante,
        "cif" to ticket.cif,
        "fecha" to ticket.fecha,
        "hora" to ticket.hora,
        "precio_sin_iva" to ticket.precioSinIva,
        "precio_con_iva" to ticket.precioConIva,
        "iva" to ticket.iva
    )

    // Actualizar datos principales
    ticketRef.set(ticketData).await()

    // Actualizar subcolección 'items'
    val itemsCollection = ticketRef.collection( collectionPath: "items")
    val batch = db.batch()

    // Eliminar los productos anteriores
    val existingItems = itemsCollection.get().await()
    for (item in existingItems) {
        batch.delete(item.reference)
    }
```

Ilustración 50: Método Actualizar Tickets

4. **Eliminar Tickets:**

   Permite eliminar un ticket específico de Firestore, incluyendo los productos de su subcolección items mediante la llamada del método "deleteTicket"

```
suspend fun deleteTicket(ticketId: String) {
    val userId = getUserId() ?: throw IllegalStateException("Usuario no autenticado")
    db.collection( collectionPath: "users").document(userId).collection( collectionPath: "tickets").document(ticketId).delete().await()
}
```

Ilustración 51: Método Eliminar Ticket

5. **Guardar Email del Usuario:**

   Almacena el correo electrónico del usuario autenticado en el documento principal asociado al usuario mediante la llamada del método "saveUserEmailToFirestore"

6. **Eliminar Cuenta y Datos Asociados:**

   Elimina todos los tickets del usuario junto con su documento principal en Firestore mediante la llamada del método "deleteAccountAndTickets"

   Finalmente, elimina la cuenta del usuario en Firebase Authentication.

6.2.2.3 GeminiUtils.kt

La clase GeminiUtils es una utilidad diseñada para gestionar la interacción entre la aplicación y la API de Gemini. Su principal objetivo es procesar imágenes capturadas o seleccionadas por el usuario y enviar la información a la API para obtener un JSON estructurado con los datos del ticket.

Principales Funcionalidades

1. **Conversión de URI a Bitmap:**

   Convierte un URI (representación de la ubicación de una imagen en el dispositivo) en un objeto Bitmap mediante el método “uriToBitmap”.

   Esta conversión es esencial porque la API de Gemini requiere imágenes en formato Bitmap para su procesamiento.

```
private fun uriToBitmap(context: Context, uri: Uri): Bitmap? {
    return try {
        context.contentResolver.openInputStream(uri).use { stream ->
            BitmapFactory.decodeStream(stream) // Carga el Bitmap directamente sin optimización
        }
    } catch (e: Exception) {
        Log.e( tag: "uriToBitmap",  msg: "Error decoding stream: ${e.localizedMessage}")
        null
    }
}
```

Ilustración 52: Método uriToBitMap

2. **Configuración del Modelo Generativo:**

   Se define un esquema (schema) que detalla la estructura de los datos que se espera recibir del ticket.

   Este esquema incluye:

   - Nombre del restaurante, CIF, fecha y hora.

   - Lista de productos con cantidad, precio unitario y precio total.

   - Totales con y sin IVA, y el porcentaje de IVA aplicado.

   La configuración también especifica que la respuesta de la API debe estar en formato JSON.

```
fun processImageWithGemini(context: Context, uri: Uri?, onResult: (String) -> Unit) {
    if (uri == null) {
        onResult("Error: URI es nula.")
        return
    }

    val config = generationConfig {
        temperature = 1f
        responseMimeType = "application/json"
        responseSchema = Schema(
            name = "ticket",
            description = "ticket de restaurante",
            type = FunctionType.OBJECT,
            properties = mapOf(
                "restaurante" to Schema(
                    name = "restaurante",
                    description = "Nombre del restaurante en mayusculas",
                    type = FunctionType.STRING,
                    nullable = false
                ),
                "cif" to Schema(
                    name = "cif",
                    description = "Código de identificación fiscal del restaurante",
                    type = FunctionType.STRING,
                    nullable = false
                ),
                "fecha" to Schema(
                    name = "fecha",
                    description = "Fecha del ticket (DD/MM/YYYY)",
                    type = FunctionType.STRING,
                    nullable = false
                ),
                "hora" to Schema(
                    name = "hora",
                    description = "Hora del ticket (HH:MM), no deben indicarse los segundos",
                    type = FunctionType.STRING,
                    nullable = false
                ),
```

Ilustración 53: Método Procesamiento con gemini

3. **Procesamiento de la Imagen:**

   La imagen convertida a Bitmap se combina con un texto guía (prompt) y se envía a la API de Gemini.

   La API utiliza el modelo configurado para analizar la imagen y extraer los datos relevantes según el esquema definido.

```
val generativeModel = GenerativeModel(
    modelName = "gemini-1.5-flash-002",
    apiKey = BuildConfig.API_KEY,
    generationConfig = config


)


val prompt = Constants.PROMPT

CoroutineScope(Dispatchers.IO).launch {
    try {
        val image1: Bitmap? = uriToBitmap(context, uri)

        if (image1 == null) {
            withContext(Dispatchers.Main) {
                onResult("Error: No se pudo convertir la imagen.")
            }
            return@launch
        }

        val inputContent = content {
            image(image1)
            text(prompt)
        }

        val res = generativeModel.generateContent(inputContent).text
```

Ilustración 54: GenerativeModel

4. **Recepción de la Respuesta:**

   La API devuelve una respuesta en formato JSON que contiene los datos estructurados del ticket.

   Si ocurre algún error durante el procesamiento, se captura y se devuelve un mensaje de error al usuario.

```
val res = generativeModel.generateContent(inputContent).text

val responseContent = res ?: "Respuesta vacía de la API"

withContext(Dispatchers.Main) {
    onResult(responseContent)
}
```

Ilustración 55: Respuesta Api

6.3 Pruebas de las Funcionalidades

El proceso de pruebas es una etapa esencial en el desarrollo de la aplicación, ya que garantiza que todas las funcionalidades implementadas operen correctamente y que la experiencia del usuario sea fluida. A continuación, se describen las pruebas realizadas para verificar cada funcionalidad principal de la aplicación.

1. Pruebas de Autenticación

- **Funcionalidad Probada:** Inicio de sesión con Google.
- **Objetivo:** Verificar que el usuario pueda autenticarse correctamente mediante Firebase Authentication utilizando su cuenta de Google.
- **Pruebas Realizadas:**

  - Inicio de sesión con credenciales válidas: El usuario es redirigido a la pantalla principal.

  - Inicio de sesión con credenciales inválidas: Se muestra un mensaje de error.

  - Cierre de sesión: El usuario es redirigido a la pantalla de inicio de sesión, y su información no permanece en el dispositivo.

- **Resultado:** La autenticación funciona correctamente, y los errores son manejados adecuadamente.

2. Pruebas de Captura y Selección de Imágenes

- **Funcionalidad Probada:** Capturar imágenes con la cámara y seleccionar imágenes desde la galería.
- **Objetivo:** Asegurarse de que el usuario pueda capturar o seleccionar imágenes para su procesamiento.
- **Pruebas Realizadas:**

  - Captura de imagen con la cámara: La imagen se guarda correctamente y se redirige a la pantalla de confirmación.

- Selección de imagen desde la galería: La imagen seleccionada se carga y se redirige a la pantalla de confirmación.

- Cancelación de ambas acciones: El flujo regresa a la pantalla principal sin errores.

- **Resultado:** Ambas funcionalidades funcionan sin problemas en diferentes dispositivos.

3. Pruebas de Procesamiento de Imágenes

- **Funcionalidad Probada:** Extracción de datos del ticket mediante la API de Gemini.
- **Objetivo:** Verificar que la imagen seleccionada o capturada pueda procesarse correctamente y que los datos relevantes se extraigan en formato JSON.
- **Pruebas Realizadas:**

  - Procesar imágenes con buena calidad: Los datos son extraídos correctamente.

  - Procesar imágenes borrosas o de baja calidad: Se muestra un mensaje de error adecuado.

  - Imágenes con datos incompletos: Se extraen los datos disponibles, y se notifica al usuario de los campos faltantes.

- **Resultado:** El procesamiento es exitoso para imágenes de buena calidad, y los errores son manejados correctamente.

4. Pruebas de Gestión de Tickets

- **Funcionalidades Probadas:** Guardar, editar, eliminar y listar tickets.
- **Objetivo:** Asegurarse de que los tickets puedan ser gestionados correctamente en la base de datos.
- **Pruebas Realizadas:**

  - Guardar un ticket con datos válidos: El ticket se almacena correctamente en Firebase.

  - Editar un ticket existente: Los cambios realizados se actualizan en la base de datos.

  - Eliminar un ticket: El ticket y sus productos asociados se eliminan correctamente.

  - Listar tickets almacenados: Se muestran correctamente los tickets del usuario autenticado.

- **Resultado:** Todas las operaciones funcionan correctamente, incluyendo la sincronización con Firebase.

5. Pruebas de Exportación de Datos

- **Funcionalidad Probada:** Exportar tickets en formato CSV.
- **Objetivo:** Verificar que el usuario pueda descargar los datos de sus tickets para su uso externo.
- **Pruebas Realizadas:**

  - Exportar múltiples tickets: El archivo CSV se genera correctamente y contiene toda la información esperada.

  - Exportar sin tickets: Se notifica al usuario que no hay datos para exportar.

- **Resultado:** Los archivos CSV se generan correctamente y contienen la información estructurada.

6. Pruebas de Configuración

- **Funcionalidad Probada:** Eliminación de cuenta y datos asociados.
- **Objetivo:** Asegurarse de que el usuario pueda eliminar su cuenta y todos sus datos de manera segura.
- **Pruebas Realizadas:**

  - Eliminar la cuenta con tickets asociados: Todos los tickets y datos se eliminan de Firebase.

  - Eliminar la cuenta sin tickets: El proceso se completa sin errores.

- **Resultado:** La eliminación de cuentas funciona correctamente, y los datos asociados se borran de manera segura.

6.4 Desafíos y Soluciones

Durante el desarrollo de este proyecto, me enfrenté a una serie de desafíos que requirieron aprendizaje y adaptabilidad. A continuación, detallo los principales retos encontrados y las soluciones que implementé para superarlos:

Uso del lenguaje de programación Kotlin

Uno de los principales desafíos fue utilizar Kotlin como lenguaje de programación, ya que nunca antes lo había utilizado. Sin embargo, gracias a sus similitudes con Java, con el cual tenía experiencia previa, la adaptación fue relativamente sencilla. La curva de aprendizaje inicial fue superada mediante la consulta de documentación oficial, tutoriales en línea y la práctica constante, lo que me permitió aprovechar las características modernas y concisas de Kotlin.

Introducción a Jetpack Compose

Otro reto significativo fue el uso de Jetpack Compose para construir la interfaz de usuario, ya que mi experiencia previa se basaba en el uso de vistas tradicionales en XML. La transición a este nuevo paradigma de desarrollo fue desafiante al principio, pero la documentación clara y numerosos ejemplos disponibles en línea facilitaron el proceso de aprendizaje. Jetpack Compose me permitió diseñar interfaces más dinámicas y declarativas, lo que, a pesar de ser novedoso, se tradujo en una experiencia de desarrollo más fluida.

Pruebas con APIs de procesamiento de lenguaje natural

Inicialmente, probé la integración con la API de ChatGPT para implementar funcionalidades relacionadas con el procesamiento de lenguaje natural. Sin embargo, los resultados obtenidos no cumplían completamente con las expectativas del proyecto, además de que cada petición implicaba un costo. Esto me llevó a explorar alternativas, como Gemini, que ofrecía una integración más efectiva y acorde a las necesidades del proyecto.

Integración con un LLM como Gemini

La integración de un modelo de lenguaje de gran escala (LLM) como Gemini fue uno de los desafíos más complejos debido a mi inexperiencia previa en la conexión de sistemas de inteligencia artificial con aplicaciones móviles. Fue necesario entender los conceptos básicos de las APIs y el flujo de datos entre la aplicación y el modelo. Superé este reto investigando guías, consultando foros de soporte técnico y realizando pruebas iterativas hasta conseguir una integración exitosa.

Uso de Firebase

La incorporación de Firebase en el proyecto fue un desafío importante, ya que nunca antes había trabajado con esta plataforma. Fue necesario aprender a gestionar la autenticación, la base de datos en tiempo real y otras funcionalidades ofrecidas por Firebase. Para superar esta dificultad, recurrí a la documentación oficial de Firebase, cursos en línea y ejemplos prácticos. Esto me permitió dominar los conceptos clave y utilizarlos efectivamente dentro del proyecto.

Para concluir, a pesar de los desafíos, estos obstáculos representaron oportunidades de aprendizaje y mejora continua. La experiencia adquirida no solo me permitió desarrollar una aplicación más robusta y funcional, sino que también amplió mis habilidades técnicas, fortaleciendo mi perfil como desarrollador.

7 Análisis de Impacto

El impacto de este proyecto radica en la simplificación y optimización de una tarea cotidiana que, aunque parece menor, consume tiempo y esfuerzo: la gestión de tickets de restauración. Esta aplicación permite digitalizar, almacenar y organizar tickets de manera eficiente, eliminando la necesidad de acumular papeles físicos y facilitando el acceso rápido a información importante.

Para los usuarios, la capacidad de procesar los tickets directamente desde la cámara del dispositivo representa un avance significativo en términos de comodidad y practicidad. En lugar de buscar y clasificar manualmente cada ticket, la aplicación se encarga de interpretar los datos relevantes, como fechas, precios e información del restaurante, ahorrando tiempo y reduciendo errores.

Además, al incorporar un modelo de inteligencia artificial como Gemini, el sistema mejora su capacidad para identificar y estructurar información con precisión, adaptándose incluso a tickets con formatos variados o poco claros. Esto no solo agiliza el proceso, sino que también lo hace más accesible para personas que no tienen conocimientos técnicos avanzados.

La integración con servicios en la nube como Firebase garantiza que los datos estén seguros y disponibles desde cualquier dispositivo, lo que aporta tranquilidad al usuario. Ya no es necesario preocuparse por perder un ticket físico o extraviar información importante, ya que todo está respaldado de manera automática y puede ser recuperado en cualquier momento.

En un contexto donde el tiempo es un recurso valioso, esta herramienta puede tener un impacto positivo directo en la vida diaria de los usuarios al ofrecerles una solución rápida, moderna y confiable para gestionar sus tickets. Esto no solo beneficia a personas individuales, sino también a pequeños negocios o trabajadores autónomos que necesitan llevar un control más eficiente de sus gastos y facturas para fines administrativos o fiscales.

En resumen, este proyecto se traduce en una mejora tangible en la forma en que los usuarios gestionan información, ofreciendo una experiencia más cómoda, segura y tecnológica que puede integrarse fácilmente en sus rutinas diarias.

8 Conclusión

En este último capítulo me gustaría reflexionar sobre los logros y aprendizajes que me llevo de este proyecto. Haber desarrollado una aplicación móvil desde cero, integrando tecnologías como Kotlin, Jetpack Compose y un modelo de lenguaje como Gemini, es algo que, al inicio de este camino, veía como un desafío enorme, pero que hoy puedo mirar con orgullo. Antes de este proyecto, no tenía experiencia en muchas de estas herramientas, pero el proceso me ha demostrado que con dedicación y constancia es posible enfrentarse a lo desconocido y crecer a partir de ello.

Cada etapa del desarrollo fue un reto, desde la implementación de Firebase hasta la adaptación a Jetpack Compose tras años de trabajar con XML. A pesar de las dificultades, el aprendizaje obtenido ha sido invaluable. Cada decisión tomada y cada problema resuelto reflejan el esfuerzo invertido en hacer que esta aplicación no solo sea funcional, sino también una herramienta útil para las personas. Este proyecto no solo me ha ayudado a mejorar mis habilidades técnicas, sino que también ha reforzado mi confianza en mi capacidad para aprender y adaptarme.

Lo más gratificante de este proyecto es el impacto que puede tener para los usuarios. Saber que esta aplicación facilita una tarea cotidiana, como la gestión de tickets de restauración, y que lo hace de manera moderna, eficiente y segura, le da un propósito tangible al trabajo realizado. Mi objetivo desde el principio fue crear algo que pudiera simplificar la vida de las personas, y creo que esta aplicación logra justamente eso.

Me llevo de este proyecto no solo el conocimiento técnico, sino también la paciencia, la creatividad y la resiliencia que vienen con trabajar en un proyecto tan complejo y, en muchos momentos, solitario. Mirando hacia atrás, estoy satisfecho con lo que he logrado, pero también veo áreas en las que puedo mejorar y crecer. Este proyecto ha sido una experiencia transformadora que marca un antes y un después en mi camino como desarrollador.

9 Bibliografía

[1] Zoho: Soluciones de Software Empresarial
URL: https://www.zoho.com

[2] Android Studio: Herramientas de desarrollo para aplicaciones Android
URL: https://developer.android.com/studio

[3] Kotlin para Android: Guía para principiantes
URL: https://developer.android.com/kotlin/learn

[4] Jetpack Compose: Documentación oficial para crear interfaces de usuario
URL: https://developer.android.com/develop/ui/compose/documentation

[5] Firebase: Soluciones en la nube para aplicaciones modernas
URL: https://firebase.google.com

[6] Gemini API: Documentación para integrar modelos de lenguaje en aplicaciones
URL: https://ai.google.dev/gemini-api/docs

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Tue Jan 14 23:48:31 CET 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)