

DESARROLLO DE APLICACIONES MULTIPLATAFORMA



Aplicación
BUDGET BUDDY

The title is centered on a light pink background featuring three overlapping watercolor circles in shades of pink, red, and orange. The word "Aplicación" is written in a black, flowing cursive script above "BUDGET BUDDY", which is in a bold, black, sans-serif font.

CELIA GARCÍA SÁNCHEZ
I.E.S PUNTA DEL VERDE

ÍNDICE

ÍNDICE.....	2
1. Introducción.....	5
1.1 Introducción del proyecto.....	5
1.2 Propósito del proyecto.....	5
1.3 Objetivos del proyecto.....	6
1.4 Alcance del proyecto.....	6
1.5 Público objetivo.....	7
1.6 Coste del proyecto.....	7
1.6.1 Costes de desarrollo.....	8
1.6.2 Costes de implantación.....	8
2. Análisis del sistema.....	8
2.1 Introducción.....	8
2.2 Análisis de requisitos.....	9
2.2.1 Requisitos.....	9
2.2.2 Requisitos funcionales.....	9
2.2.2 Requisitos No Funcionales.....	10
2.3 Casos de uso.....	10
2.3.1 Diagramas de casos de uso.....	10
Diagrama de Casos de Uso.....	10
Autenticación y cuenta.....	10
Gestión de transacciones.....	11
Gestión de metas de ahorro.....	11
Chatbot de asistencia financiera.....	11
Navegación general.....	11
2.3.2 Narrativas de casos de uso.....	12
CU02 – Iniciar sesión con email y contraseña.....	13
CU03 – Iniciar sesión con Google.....	14
CU04 – Recuperar contraseña.....	15
CU05 – Cerrar sesión.....	16
CU06 – Eliminar cuenta.....	16
CU07 – Resetear datos de la cuenta (dinero, transacciones, etc.).....	17
CU08 – Actualizar información de usuario (nombre, dinero, etc.).....	18
CU09 – Consultar lista de transacciones.....	18
CU10 – Eliminar transacción.....	19
CU11 – Ver resumen de transacciones (gráficos/estadísticas).....	20
CU14 – Ver categorías de gasto.....	20
CU15 – Añadir nueva categoría.....	20
CU16 – Crear meta de ahorro.....	21
CU17 – Consultar lista de metas.....	22

CU18 – Eliminar meta.....	22
CU19 – Cambiar estado de la meta (de "proceso" a "completada").....	22
CU20 – Consultar al chatbot cualquier pregunta financiera.....	23
CU21 – Consultar al chatbot sobre tus transacciones.....	23
CU22 – Consultar al chatbot sobre tus metas.....	24
CU23 – Recibir sugerencias personalizadas del chatbot (según contexto).....	24
CU24 – Navegar entre pantallas principales (Inicio, Transacciones, Metas, Mi Cuenta, Chatbot).....	25
CU25 – Acceder al historial financiero desde el menú.....	26
3. Diseño del sistema.....	26
3.1 Introducción.....	26
3.2 Diagrama de clases.....	27
Modelo de Datos (model).....	30
Lógica de Presentación (viewmodel).....	30
Lógica de Negocio (repository).....	31
Interfaz de Usuario (ui.screen).....	31
Componentes (ui.components).....	31
Utilidades (utils).....	32
3.3 Diseño de la base de datos.....	32
3.3.1 Diseño lógico.....	32
3.3.2 Diseño relacional.....	33
3.4 Diseño de la interfaz de usuario.....	34
Estructura general y navegación.....	35
Componentes reutilizables.....	36
Pantallas principales.....	36
Elementos emergentes (Popups).....	39
Estética general.....	40
3.5 Adaptación a dispositivos.....	40
4. Implementación.....	43
4.1 Introducción.....	43
4.2 Arquitectura utilizada.....	43
Estructura general.....	43
Ventajas de esta arquitectura.....	43
4.3 Lenguajes y tecnologías empleadas.....	44
Lenguajes de programación.....	44
Tecnologías y frameworks.....	44
4.4 Herramientas de desarrollo.....	45
4.5 Dependencias y librerías utilizadas.....	46
4.6 Gestión de permisos.....	47
Permiso utilizado:.....	47
4.7 Estructura del proyecto.....	47

4.8 Codificación.....	48
4.8.1 Navegación entre pantallas.....	48
4.8.2 Inicio de sesión y autenticación con Google.....	49
4.8.3 Restablecer contraseña por email.....	51
4.8.4 Añadir transacciones.....	51
4.8.5 Visualización de gráfico agrupado (BarraAgrupadaGrafico).....	52
4.8.6 Cálculo del progreso de metas.....	52
4.8.7 Generación de PDF resumen.....	53
4.8.8 Chatbot financiero con contexto.....	53
5. Pruebas de software.....	54
5.1 Introducción.....	54
5.2 Técnicas de prueba.....	54
5.2.1 Pruebas de caja blanca.....	55
5.2.2 Pruebas de caja negra.....	55
5.3 Pruebas funcionales en dispositivos.....	56
5.4 Pruebas de rendimiento.....	64
5.5 Registro y análisis de errores.....	64
6. Conclusiones.....	65
6.1 Valoración del proyecto.....	65
6.2 Dificultades encontradas.....	66
6.3 Mejoras y ampliaciones futuras.....	66
7. Bibliografía y referencias.....	67
7.1 Fuentes utilizadas.....	67
Anexos.....	68
Anexo A: Manual de instalación.....	68
Requisitos previos.....	69
Pasos de instalación.....	69
Anexo B: Manual de usuario.....	70
Pantalla de Inicio (Splash).....	70
Login.....	70
Registro.....	71
Home.....	71
Transacciones.....	72
Metas.....	72
Chatbot.....	72
Mi Cuenta.....	73

1. Introducción

1.1 Introducción del proyecto

Este proyecto consiste en el desarrollo de una aplicación móvil Android enfocada en la gestión financiera personal. Su objetivo principal es brindar a los usuarios una plataforma intuitiva para registrar, visualizar y analizar sus ingresos, gastos y metas económicas.

La aplicación ha sido construida utilizando el framework Jetpack Compose, lo cual permite una interfaz moderna y adaptable. Entre las funcionalidades principales se encuentran el registro de usuarios, visualización de transacciones, manejo de metas financieras, y un sistema de navegación fluido entre las diferentes pantallas.

El proyecto está orientado a usuarios que desean llevar un control eficiente de sus finanzas personales desde su dispositivo móvil, brindando una experiencia de usuario amigable, con elementos visuales interactivos y diseño centrado en la simplicidad y claridad.



1.2 Propósito del proyecto

Su propósito es brindar a los usuarios una herramienta digital que les permita tomar el control de sus finanzas personales de manera simple, eficiente y accesible. A través de esta aplicación móvil, los usuarios podrán registrar sus ingresos y gastos, definir metas financieras, y visualizar el estado de su economía mediante gráficos y resúmenes claros y comprensibles.

Más allá de ser una simple herramienta de registro, esta aplicación tiene un enfoque educativo: busca ayudar a las personas a desarrollar una mayor conciencia y comprensión de sus hábitos financieros. Al utilizarla de forma constante, los usuarios podrán identificar patrones de consumo, corregir malos hábitos y tomar decisiones más informadas con respecto a su dinero. De esta manera, no solo controlan sus finanzas, sino que también adquieren conocimientos valiosos sobre el manejo del dinero, el ahorro y la planificación financiera.

El objetivo final es motivar al usuario a adoptar un estilo de vida más responsable y consciente en lo económico. Ahorrar ya no será una tarea complicada o aburrida, sino un proceso interactivo y gratificante, guiado por una aplicación amigable, moderna y adaptada a sus

necesidades. En un mundo donde la estabilidad financiera es clave para la tranquilidad y el bienestar, esta aplicación se convierte en una aliada esencial para quienes desean mejorar su relación con el dinero y construir un futuro más sólido.

1.3 Objetivos del proyecto

Objetivo General:

Desarrollar una aplicación móvil para el sistema operativo Android que permita a los usuarios gestionar sus finanzas personales de forma intuitiva y eficiente, facilitando el registro de transacciones, el seguimiento de metas económicas y promoviendo el aprendizaje y la conciencia financiera.

Objetivos Específicos:

- Diseñar una interfaz amigable e interactiva que permita al usuario registrar ingresos y gastos de manera rápida y sencilla.
- Implementar un sistema de visualización de datos que muestre resúmenes y gráficos del estado financiero del usuario.
- Desarrollar funcionalidades que permitan establecer y monitorear metas de ahorro u objetivos financieros específicos.
- Integrar un chatbot basado en inteligencia artificial que ofrezca asistencia personalizada, resuelva dudas y brinde recomendaciones financieras en tiempo real.
- Asegurar que la aplicación funcione correctamente en dispositivos Android, ofreciendo una experiencia fluida y sin errores técnicos.
- Fomentar la adopción del uso regular de la aplicación como una herramienta de apoyo para el ahorro y la toma de decisiones financieras responsables.

1.4 Alcance del proyecto

Este proyecto contempla el desarrollo de una solución funcional centrada en cubrir las necesidades básicas de un usuario interesado en organizar y monitorear su economía personal a través de su dispositivo móvil.

El alcance incluye la construcción de las pantallas necesarias para la navegación dentro de la aplicación, la implementación de un sistema de interacción conversacional mediante inteligencia artificial, y la creación de un entorno gráfico que priorice la facilidad de uso. También se considera la validación del correcto funcionamiento general del sistema, enfocándose en la estabilidad, usabilidad y respuesta de la interfaz.

No se contemplan en esta fase inicial funciones avanzadas como la vinculación con servicios bancarios externos, almacenamiento en la nube, sincronización entre múltiples dispositivos o un sistema de notificaciones automatizadas. Tampoco se abordará la monetización de la aplicación ni su publicación en plataformas oficiales como Google Play Store.

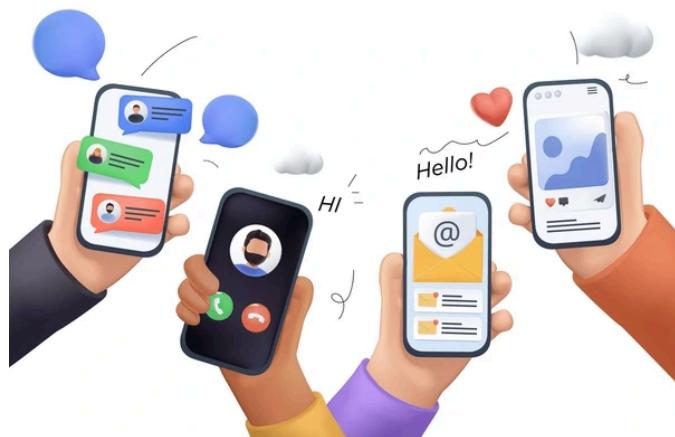
Este alcance está diseñado para entregar una versión funcional, centrada en el núcleo esencial de la propuesta, permitiendo evaluar su efectividad y aceptación por parte de los usuarios antes de futuras expansiones.

1.5 Público objetivo

El público objetivo de este proyecto está compuesto por personas que desean mejorar el control y la administración de sus finanzas personales, independientemente de su nivel de conocimiento financiero. Se dirige principalmente a jóvenes adultos, estudiantes universitarios, trabajadores independientes y profesionales que buscan una herramienta sencilla pero funcional para registrar sus ingresos y gastos, establecer metas económicas y adquirir mejores hábitos financieros.

La aplicación también está pensada para usuarios que valoran la tecnología como medio de aprendizaje y autoorganización, y que prefieren soluciones móviles accesibles desde cualquier lugar y en cualquier momento. Gracias a su interfaz intuitiva y su asistente conversacional basado en inteligencia artificial, incluso personas con poca experiencia en el uso de aplicaciones financieras pueden beneficiarse de su uso.

En resumen, este proyecto busca atender a un público amplio que comparte el interés por organizar sus finanzas, tomar decisiones más conscientes sobre el dinero y desarrollar una mayor educación financiera mediante el uso de una herramienta digital moderna y accesible.



1.6 Coste del proyecto

La estimación del coste y esfuerzo del software y/o hardware requerido para la realización del sistema se ha dividido en dos subapartados: los costes asociados al desarrollo de la solución, y los necesarios para su puesta en funcionamiento.

1.6.1 Costes de desarrollo

Costes para la realización del sistema. Se dividen en **Costes Recursos Informáticos** (costes hardware y software) y **Costes de personal**.

- **Costes de Recursos Informáticos:**

Para el desarrollo del proyecto se utilizaron herramientas gratuitas como Android Studio y Firebase (plan básico), por lo que no fue necesario adquirir licencias de software ni hardware adicional. El equipo de desarrollo consistió en un ordenador personal con sistema operativo Windows, por lo que el coste económico en recursos informáticos ha sido nulo.

- **Costes de Personal:**

Al tratarse de un Trabajo de Fin de Grado (TFG), el desarrollo ha sido realizado de manera individual y sin compensación económica. Sin embargo, se estima una dedicación aproximada de más de 200 horas, lo que refleja una inversión significativa de tiempo y esfuerzo personal en el diseño, implementación y pruebas del sistema.

1.6.2 Costes de implantación

Costes para poner en funcionamiento el sistema. Resumen del coste total de ponerlo en funcionamiento.

En esta fase, no se ha incurrido en gastos económicos directos. No se ha realizado registro en Google Play ni se ha utilizado infraestructura de pago, ya que el sistema se ejecuta de manera local en un dispositivo Android personal. Todos los recursos necesarios para la implantación han sido gratuitos, por lo que el coste de esta etapa también se considera **nulo**.

2. Análisis del sistema

2.1 Introducción

En este apartado se presenta el análisis del sistema correspondiente a mi aplicación móvil, cuyo objetivo principal es facilitar la gestión de finanzas personales desde un dispositivo Android. Este análisis permite comprender cómo se estructura la solución, qué componentes la integran y cómo interactúan entre sí para ofrecer al usuario una experiencia funcional, intuitiva y útil.

El sistema se compone de diversas pantallas que permiten al usuario registrar ingresos y gastos, establecer metas económicas, consultar un historial financiero y recibir asistencia mediante un chatbot integrado basado en inteligencia artificial. A través de este análisis, se identifican los principales requerimientos del sistema, tanto funcionales como no funcionales, así como los actores implicados y los distintos casos de uso que definen el comportamiento de la aplicación.

El propósito de esta sección es establecer las bases técnicas y conceptuales sobre las cuales se desarrolló el proyecto, asegurando que la solución implementada responde adecuadamente a las necesidades del público objetivo y cumple con los objetivos planteados en las etapas anteriores.

2.2 Análisis de requisitos

2.2.1 Requisitos

El análisis de requisitos tiene como objetivo identificar y describir todas las funcionalidades y características que debe cumplir el sistema para satisfacer las necesidades del usuario. Este proceso es esencial para garantizar que el desarrollo se alinee con los objetivos planteados y se cubran correctamente las expectativas del proyecto. Los requisitos se dividen en funcionales y no funcionales.

2.2.2 Requisitos funcionales

Los requisitos funcionales describen las acciones específicas que el sistema debe ser capaz de realizar. Para esta aplicación de gestión financiera, los principales requisitos funcionales son:

- El sistema debe permitir al usuario registrar ingresos y gastos de forma manual.
- El sistema debe permitir visualizar un historial de transacciones.
- El sistema debe ofrecer gráficos o resúmenes que representen la información financiera registrada.
- El usuario debe poder crear, editar y eliminar metas de ahorro personalizadas.
- El sistema debe incluir un chatbot capaz de responder preguntas básicas sobre el uso de la app y ofrecer sugerencias financieras generales.
- El sistema debe permitir la navegación entre diferentes secciones como "Inicio", "Mi cuenta", "Transacciones", "Metas", entre otras.
- El sistema debe permitir la autenticación del usuario (registro e inicio de sesión).

2.2.2 Requisitos No Funcionales

Los requisitos no funcionales definen criterios de calidad del sistema, como rendimiento, diseño, usabilidad o restricciones técnicas. En este proyecto, se consideran los siguientes:

- La aplicación debe estar disponible para dispositivos con sistema operativo Android.
- La interfaz debe ser intuitiva, clara y fácil de usar para todo tipo de usuarios.
- El tiempo de respuesta entre pantallas debe ser inferior a 2 segundos.
- La aplicación debe funcionar sin conexión a internet para el registro básico de transacciones.
- El diseño debe ser adaptable a distintas resoluciones y tamaños de pantalla.
- El sistema debe garantizar la integridad de los datos registrados durante su uso local.

2.3 Casos de uso

2.3.1 Diagramas de casos de uso

Los casos de uso permiten representar de forma clara y estructurada las funcionalidades principales del sistema desde el punto de vista del usuario. A través de ellos, se describen las interacciones entre los actores y el sistema, estableciendo qué tareas pueden realizar y cómo fluye la información.

A continuación, se presentan los principales casos de uso identificados en la aplicación:

Diagrama de Casos de Uso

Autenticación y cuenta

- **CU01** – Registrarse con email y contraseña
- **CU02** – Iniciar sesión con email y contraseña
- **CU03** – Iniciar sesión con Google
- **CU04** – Recuperar contraseña
- **CU05** – Cerrar sesión

- **CU06** – Eliminar cuenta
- **CU07** – Resetear datos de la cuenta (dinero, transacciones, etc.)
- **CU08** – Actualizar información de usuario (nombre, dinero, etc.)

Gestión de transacciones

- **CU09** – Añadir ingreso/gasto (transacción)
- **CU10** – Consultar lista de transacciones
- **CU12** – Eliminar transacción (posible si se gestiona en la interfaz)
- **CU13** – Ver resumen de transacciones (gráficos/estadísticas)
- **CU14** – Ver categorías de gasto
- **CU15** – Añadir nueva categoría

Gestión de metas de ahorro

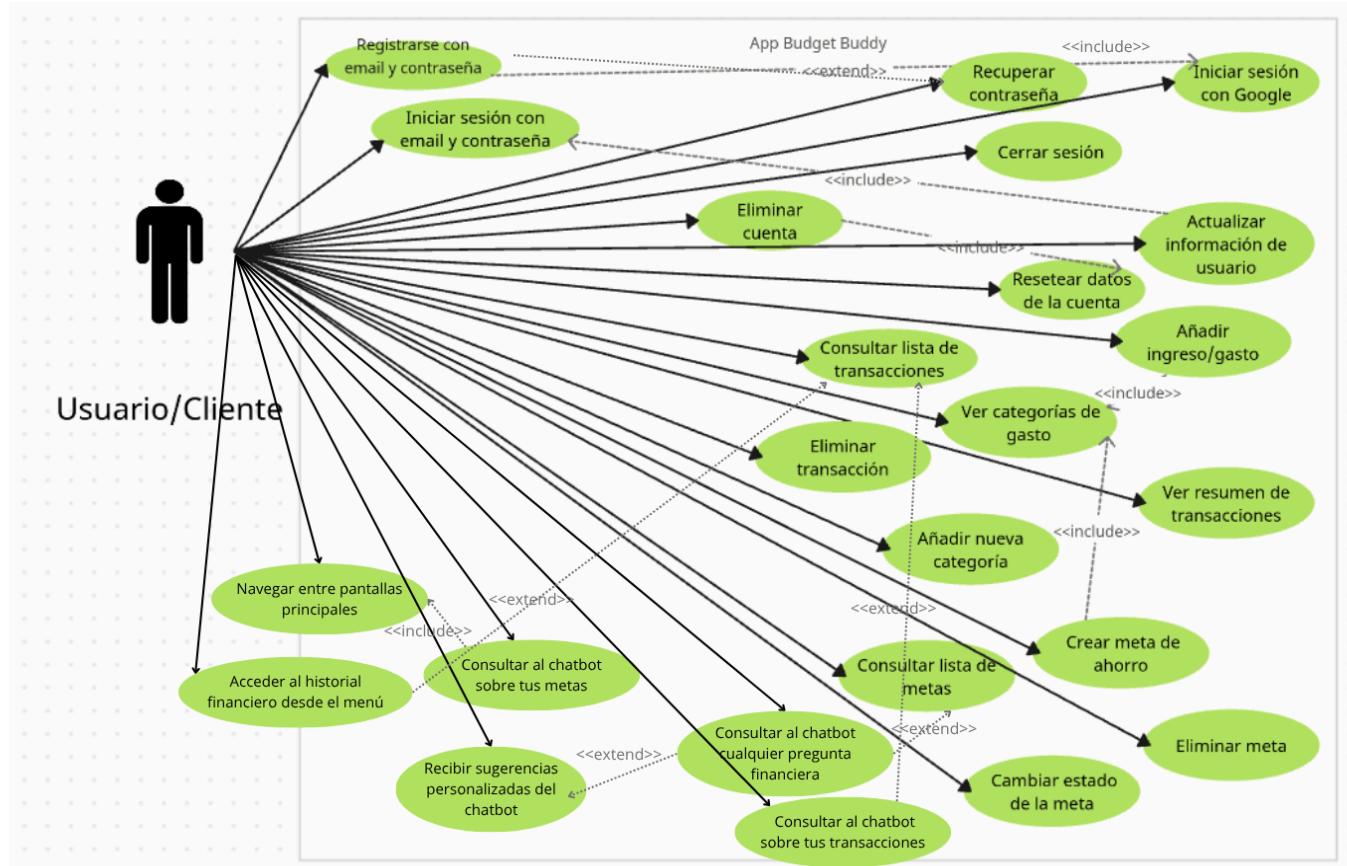
- **CU16** – Crear meta de ahorro
- **CU17** – Consultar lista de metas
- **CU18** – Eliminar meta
- **CU19** – Cambiar estado de la meta (por ejemplo, de "en proceso" a "completada")

Chatbot de asistencia financiera

- **CU20** – Consultar al chatbot cualquier pregunta financiera
- **CU21** – Consultar al chatbot sobre tus transacciones
- **CU22** – Consultar al chatbot sobre tus metas
- **CU23** – Recibir sugerencias personalizadas del chatbot (según contexto)

Navegación general

- **CU24** – Navegar entre pantallas principales (Inicio, Transacciones, Metas, Mi Cuenta, Chatbot)
- **CU25** – Acceder al historial financiero desde el menú



2.3.2 Narrativas de casos de uso

CU01 – Registrarse con email y contraseña

Actor Principal: Usuario

Condiciones de entrada: El usuario no debe tener una cuenta registrada previamente en el sistema.

Flujo de eventos:

1. El usuario selecciona la opción “¿No tienes cuenta? Regístrate” desde la pantalla de login.
2. El sistema redirige al formulario de registro.
3. El usuario introduce su nombre completo, nombre de usuario, correo electrónico, contraseña y confirmación de contraseña.
4. El sistema valida que todos los campos están completos y cumplen con el formato (nombre válido, nombre de usuario único, correo válido, contraseñas coincidentes).
5. a. El sistema crea una cuenta con los datos ingresados y almacena la información en Firebase Authentication y Firestore.
6. a. El sistema muestra un mensaje de éxito e inicia sesión automáticamente.
7. a. El usuario es redirigido a la pantalla principal (Home).

Camino alternativo:

5. b. Alguno de los campos está vacío o no cumple el formato.
6. b. El sistema muestra un mensaje de error indicando qué campo es incorrecto y permanece en la pantalla de registro.

Condiciones de salida:

El usuario queda registrado en la aplicación y redirigido a la pantalla principal.

CU02 – Iniciar sesión con email y contraseña

Actor Principal: Usuario

Condiciones de entrada: El usuario debe estar previamente registrado en el sistema.

Flujo de eventos:

1. El usuario abre la aplicación y se encuentra en la pantalla de login.
2. Introduce su correo electrónico o nombre de usuario y la contraseña.
3. El usuario pulsa el botón “Iniciar sesión”.

4. El sistema verifica que los datos ingresados son correctos y coincide con un usuario registrado en Firebase Authentication.
5. a. El sistema obtiene los datos del usuario desde Firestore.
6. a. El sistema inicia la sesión y redirige al usuario a la pantalla principal.

Camino alternativo:

4. b. Los datos introducidos no son correctos.
5. b. El sistema muestra un mensaje de error indicando que las credenciales son inválidas y permite volver a intentarlo.

Condiciones de salida:

El usuario accede correctamente a la aplicación con su cuenta registrada.

CU03 – Iniciar sesión con Google

Actor Principal: Usuario

Condiciones de entrada: El usuario debe tener una cuenta de Google activa y conexión a internet.

Flujo de eventos:

1. En la pantalla de login, el usuario pulsa el botón “Iniciar sesión con Google”.
2. El sistema lanza el intent de autenticación con Google utilizando GoogleSignInClient.
3. El usuario selecciona su cuenta de Google y acepta los permisos requeridos.
4. El sistema recibe el token de autenticación y lo valida con Firebase.
5. a. Si es la primera vez que inicia sesión, el sistema crea automáticamente su perfil en Firestore (nombre, email, UID).
6. a. El sistema inicia sesión y redirige al usuario a la pantalla principal.

Camino alternativo:

3. b. El usuario cancela el proceso de selección de cuenta.
4. b. El sistema no realiza ninguna acción y permanece en la pantalla de login.

Condiciones de salida:

El usuario accede correctamente a la aplicación mediante su cuenta de Google.

CU04 – Recuperar contraseña**Actor Principal:** Usuario**Condiciones de entrada:** El usuario debe haber registrado previamente su dirección de correo electrónico en la aplicación.**Flujo de eventos:**

1. El usuario pulsa en el enlace “He olvidado mi contraseña” desde la pantalla de login.
2. El sistema muestra un cuadro de diálogo solicitando su correo electrónico.
3. El usuario introduce su correo y confirma.
4. El sistema envía un correo electrónico a la dirección indicada con un enlace de recuperación mediante Firebase Authentication.
5. El sistema muestra un mensaje indicando que el correo fue enviado correctamente.

Camino alternativo:

3. b. El usuario cancela la acción → El sistema cierra el cuadro de diálogo y permanece en la pantalla de login.
4. b. El correo introducido no es válido o no está registrado → El sistema muestra un mensaje de error.

Condiciones de salida:

El usuario recibe un email con instrucciones para restablecer su contraseña.

CU05 – Cerrar sesión

Actor Principal: Usuario

Condiciones de entrada: El usuario debe estar autenticado y con sesión activa en la aplicación.

Flujo de eventos:

1. Desde el menú lateral o desde la pantalla “Mi cuenta”, el usuario pulsa el botón “Cerrar sesión”.
2. El sistema cierra la sesión actual utilizando FirebaseAuth.signOut().
3. El sistema redirige automáticamente al usuario a la pantalla de login.

Camino alternativo:

Ninguno relevante, ya que la acción es directa.

Condiciones de salida:

El usuario queda desautenticado y debe iniciar sesión nuevamente para acceder a la app.

CU06 – Eliminar cuenta

Actor Principal: Usuario

Condiciones de entrada: El usuario debe estar autenticado y con sesión activa.

Flujo de eventos:

1. Desde la pantalla “Mi cuenta”, el usuario pulsa el botón “Eliminar cuenta”.
2. El sistema muestra un diálogo de confirmación.
3. El usuario confirma su intención de eliminar su cuenta.
4. El sistema elimina su documento en la colección “usuarios” de Firestore.
5. El sistema elimina todas las subcolecciones relacionadas: transacciones, categorías, metas.
6. El sistema elimina la cuenta en Firebase Authentication (currentUser.delete()).
7. El sistema redirige al usuario a la pantalla de login.

Camino alternativo:

3. b. El usuario cancela la acción → El sistema cierra el diálogo y permanece en la pantalla de cuenta.

Condiciones de salida:

Todos los datos del usuario son eliminados de la base de datos y el sistema. La cuenta queda completamente eliminada.

CU07 – Resetear datos de la cuenta (dinero, transacciones, etc.)**Actor Principal:** Usuario**Condiciones de entrada:** El usuario debe estar autenticado y tener sesión activa.**Flujo de eventos:**

1. Desde la pantalla “Mi cuenta”, el usuario pulsa el botón “Resetear cuenta”.
2. El sistema solicita confirmación mediante un diálogo.
3. El usuario confirma su intención de resetear.
4. El sistema actualiza el campo dineroTotal del usuario en Firestore a 0.
5. El sistema accede a las subcolecciones del usuario:
 - Elimina todos los documentos de la colección categorías.
 - Elimina todos los documentos de la colección transacciones.
6. El sistema recarga las categorías desde la vista.
7. El sistema muestra la pantalla actualizada con los valores reseteados.

Camino alternativo:

3. b. El usuario cancela → El sistema cierra el diálogo y no realiza ninguna modificación.

Condiciones de salida:

La cuenta del usuario queda limpia de transacciones y categorías, y el total de dinero se reinicia a cero.

CU08 – Actualizar información de usuario (nombre, dinero, etc.)

Actor Principal: Usuario

Condiciones de entrada: El usuario debe estar autenticado y con sesión activa.

Flujo de eventos:

1. El usuario accede a la pantalla “Mi cuenta”.
2. El usuario modifica los campos de nombre completo, nombre de usuario y/o dinero total.
3. El usuario pulsa el botón “Guardar cambios” o “Actualizar dinero”.
4. El sistema valida los datos introducidos.
5. Si son válidos, actualiza el documento del usuario en Firestore con los nuevos valores.
6. El sistema muestra un mensaje o estado de éxito.

Camino alternativo:

4. b. Los campos están vacíos o el dinero introducido no es numérico → El sistema muestra un error y no actualiza nada.

Condiciones de salida:

La información del usuario queda actualizada y visible en la interfaz.

CU09 – Consultar lista de transacciones

Actor Principal: Usuario

Condiciones de entrada: El usuario debe haber iniciado sesión.

Flujo de eventos:

1. El usuario accede al menú lateral y selecciona “Transacciones”.
2. El sistema filtra las transacciones según el tipo (Ahorro o Gasto).
3. Se muestran en una lista con tarjetas visuales.
4. Cada transacción incluye título, cantidad, categoría y fecha.

5. El usuario puede hacer clic para ver los detalles en un AlertDialog.

Condiciones de salida:

El usuario visualiza sus transacciones almacenadas con información clara y accesible.

CU10 – Eliminar transacción

Actor Principal: Usuario

Condiciones de entrada: El usuario debe haber iniciado sesión y tener transacciones registradas.

Flujo de eventos:

1. En la lista de transacciones, el usuario pulsa el ícono de “más opciones”.
2. Selecciona “Eliminar transacción”.
3. El sistema solicita confirmación mediante un AlertDialog.
4. El usuario confirma la acción.
5. El sistema elimina el documento en Firestore correspondiente a esa transacción.
6. El sistema actualiza el dinero total del usuario.

Camino alternativo:

4. b. El usuario cancela → El sistema cierra el diálogo sin modificar nada.

Condiciones de salida:

La transacción desaparece de la lista y la base de datos; el dinero total se ajusta correctamente.

CU11 – Ver resumen de transacciones (gráficos/estadísticas)

Actor Principal: Usuario

Condiciones de entrada: Usuario autenticado con transacciones registradas.

Flujo de eventos:

1. El usuario accede a la pantalla principal (Home).
2. El sistema agrupa las transacciones por categoría y tipo.
3. Se genera un gráfico de barras agrupadas (BarraAgrupadaGrafico) para visualizar ingresos y gastos por categoría.
4. También se calcula un resumen de totales e indicadores (dinero disponible, metas, progreso).
5. El usuario puede visualizar estos gráficos en tiempo real y tomar decisiones.

Condiciones de salida:

El usuario obtiene un resumen visual claro de su situación financiera actual.

CU14 – Ver categorías de gasto

Actor Principal: Usuario

Condiciones de entrada: Usuario autenticado.

Flujo de eventos:

1. El usuario accede al formulario para añadir una nueva transacción o meta.
2. El sistema carga desde Firestore las categorías existentes dentro del documento del usuario.
3. Se muestran en un desplegable (ExposedDropdownMenu) para seleccionar.

Condiciones de salida:

Las categorías disponibles son visibles y seleccionables por el usuario.

CU15 – Añadir nueva categoría

Actor Principal: Usuario

Condiciones de entrada: Usuario autenticado.

Flujo de eventos:

1. En el formulario de nueva transacción, el usuario selecciona “ Añadir nueva categoría”.
2. Se habilita un campo de texto para introducir el nombre.
3. El usuario escribe el nombre de la nueva categoría.
4. Al confirmar la transacción, el sistema añade esa categoría a la colección categorias en Firestore.

Condiciones de salida:

La nueva categoría queda guardada y disponible en futuras selecciones.

CU16 – Crear meta de ahorro

Actor Principal: Usuario

Condiciones de entrada: Usuario autenticado y con categorías disponibles.

Flujo de eventos:

1. El usuario pulsa el botón flotante "+" en la pantalla de metas.
2. Se abre PopupMeta con campos: categoría, tipo (Gasto/Ahorro), cantidad y fecha límite.
3. El usuario rellena los datos y pulsa “ Guardar”.
4. El sistema valida los datos, genera un objeto Meta, y lo guarda en Firestore.
5. Se actualiza la lista de metas del usuario.

Camino alternativo:

4.b. Si la fecha no es válida o faltan campos, se muestra un mensaje de error.

Condiciones de salida:

La meta queda almacenada en Firestore y visible en la lista de metas.

CU17 – Consultar lista de metas

Actor Principal: Usuario

Condiciones de entrada: Usuario autenticado.

Flujo de eventos:

1. El usuario accede a la pantalla "Metas".
2. El sistema obtiene todas las metas del usuario desde Firestore.
3. Se filtran por estado (en proceso) y se muestran en tarjetas con datos relevantes.

Condiciones de salida:

El usuario visualiza sus metas actuales, sus cantidades, tipos y fechas límite.

CU18 – Eliminar meta

Actor Principal: Usuario

Condiciones de entrada: Usuario autenticado y con metas en estado "Proceso".

Flujo de eventos:

1. El usuario pulsa el botón de "más opciones" en la tarjeta de una meta.
2. Selecciona "Eliminar".
3. El sistema elimina la meta de Firestore.
4. Se actualiza automáticamente la lista de metas.

Condiciones de salida:

La meta desaparece del sistema.

CU19 – Cambiar estado de la meta (de "proceso" a "completada")

Actor Principal: Usuario

Condiciones de entrada: Usuario autenticado con metas activas.

Flujo de eventos:

1. El usuario accede a la lista de metas.
2. En una tarjeta de meta, abre el menú y selecciona "Marcar como completada".

3. El sistema actualiza el campo estado en Firestore a "Completada".
4. La meta deja de mostrarse en las metas en proceso.

Condiciones de salida:

La meta pasa a estado "completada" y se actualiza su progreso.

CU20 – Consultar al chatbot cualquier pregunta financiera

Actor Principal: Usuario**Condiciones de entrada:** El usuario debe estar autenticado y acceder a la pantalla de chatbot.**Flujo de eventos:**

1. El usuario accede a la pantalla "Chatbot".
2. Escribe una pregunta financiera general, como: "¿Qué es el interés compuesto?"
3. El sistema envía la pregunta al ChatViewModel.
4. Este la reenvía al backend del chatbot (ChatGPT o LLM embebido).
5. Se recibe una respuesta y se muestra en la interfaz.

Condiciones de salida:

El usuario recibe una respuesta útil y clara a su pregunta financiera.

CU21 – Consultar al chatbot sobre tus transacciones

Actor Principal: Usuario**Condiciones de entrada:** Usuario autenticado, con transacciones registradas.**Flujo de eventos:**

1. El usuario accede al chatbot y formula una consulta contextual, por ejemplo:
"¿Cuánto he gastado este mes en comida?"
2. El sistema obtiene las transacciones del usuario desde Firestore.

3. El ChatViewModel agrega esa información como contexto a la petición.
4. El chatbot analiza y responde con un resumen específico, filtrando por categoría y fecha.

Condiciones de salida:

El usuario obtiene un análisis financiero personalizado de sus gastos.

CU22 – Consultar al chatbot sobre tus metas**Actor Principal:** Usuario**Condiciones de entrada:** Usuario autenticado y con metas activas.**Flujo de eventos:**

1. El usuario accede al chatbot y pregunta:
“¿Cómo van mis metas de ahorro?”
2. El MetaViewModel proporciona las metas actuales desde Firestore.
3. El ChatViewModel añade estos datos al contexto y los envía al chatbot.
4. El chatbot responde con un resumen del estado, progreso y recomendaciones.

Condiciones de salida:

El usuario visualiza un informe textual generado por IA sobre sus metas.

CU23 – Recibir sugerencias personalizadas del chatbot (según contexto)**Actor Principal:** Usuario**Condiciones de entrada:** Usuario autenticado, con datos financieros en el sistema.**Flujo de eventos:**

1. El usuario pregunta al chatbot algo como:
“¿Algún consejo para reducir mis gastos?”
2. El sistema analiza:

- Categorías con más gasto.
 - Progreso de metas.
 - Dinero disponible.
3. El ChatViewModel empaqueta el contexto y lo envía al modelo.
 4. El chatbot responde con sugerencias personalizadas y relevantes:
 - “Reduce tus gastos en ocio en un 20%.”
 - “Estás cerca de completar tu meta de ‘Vacaciones’.”

Condiciones de salida:

El usuario recibe consejos adaptados a su situación financiera actual.

CU24 – Navegar entre pantallas principales (Inicio, Transacciones, Metas, Mi Cuenta, Chatbot)

Actor Principal: Usuario

Condiciones de entrada: Usuario autenticado dentro de la app.

Flujo de eventos:

1. El usuario despliega el menú lateral (AppDrawer) desde cualquier pantalla.
2. El sistema muestra las opciones principales: Inicio, Transacciones, Metas, Mi Cuenta y Chatbot.
3. El usuario selecciona una de las opciones.
4. El NavController navega a la pantalla correspondiente mediante rutas definidas en AppNavGraph.
 - Por ejemplo: navController.navigate("miCuenta")
5. La pantalla seleccionada se renderiza y se carga el ViewModel asociado si aplica.

Condiciones de salida:

El sistema muestra correctamente la nueva pantalla con los datos del usuario.

CU25 – Acceder al historial financiero desde el menú

Actor Principal: Usuario

Condiciones de entrada: Usuario autenticado, con transacciones registradas.

Flujo de eventos:

1. El usuario accede al menú lateral y selecciona “Transacciones”.
2. El sistema pregunta si quiere ver ingresos o gastos (por ruta: "transacciones/{tipo}").
3. Se accede a la pantalla TransaccionesScreen con el tipo seleccionado.
4. El sistema, mediante TransactionViewModel, carga el historial filtrado por tipo desde Firestore.
5. Se muestra una lista con tarjetas por transacción, permitiendo ver detalles, eliminar, etc.

Condiciones de salida:

El usuario visualiza su historial financiero (ingresos o gastos) con estilo visual neón y puede gestionarlo.

3. Diseño del sistema

3.1 Introducción

En el siguiente apartado se detallará cómo se ha llevado a cabo el diseño de la aplicación, teniendo como punto de partida el análisis de requisitos previamente establecido. Este diseño busca dar respuesta a las necesidades funcionales y no funcionales del sistema, proponiendo una estructura lógica, eficiente y coherente con los objetivos del proyecto.

Para ello, se han considerado diferentes aspectos esenciales del sistema, tales como la estructura de datos, la lógica de navegación entre pantallas, la arquitectura general de la aplicación y el diseño visual enfocado en la usabilidad.

Se desarrollará una solución lógica basada en el entorno Android, haciendo uso del lenguaje Kotlin y Jetpack Compose como framework principal. El sistema se divide en tres grandes áreas de diseño:

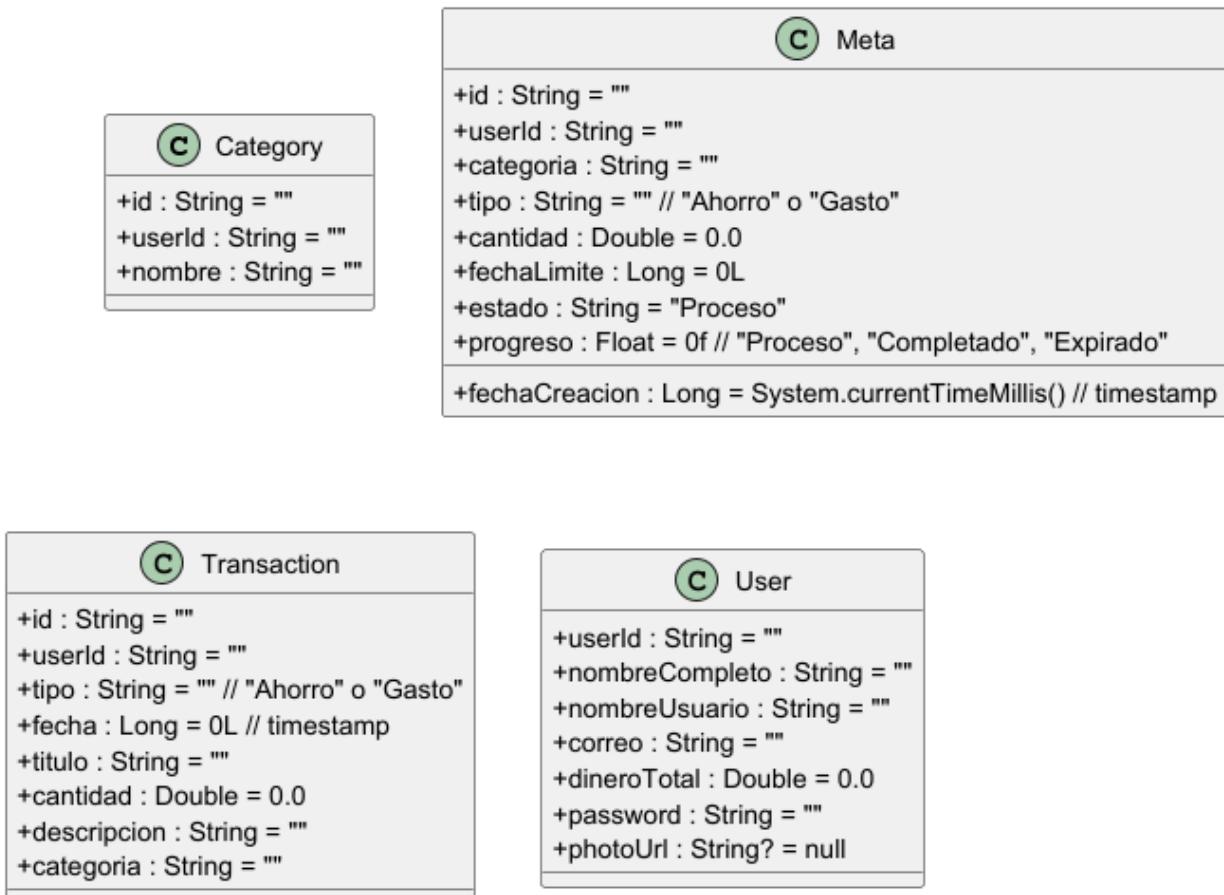
- **Diagrama de clases**, donde se representa la estructura de las entidades del sistema y su relación.

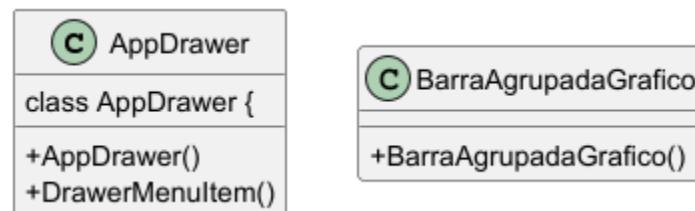
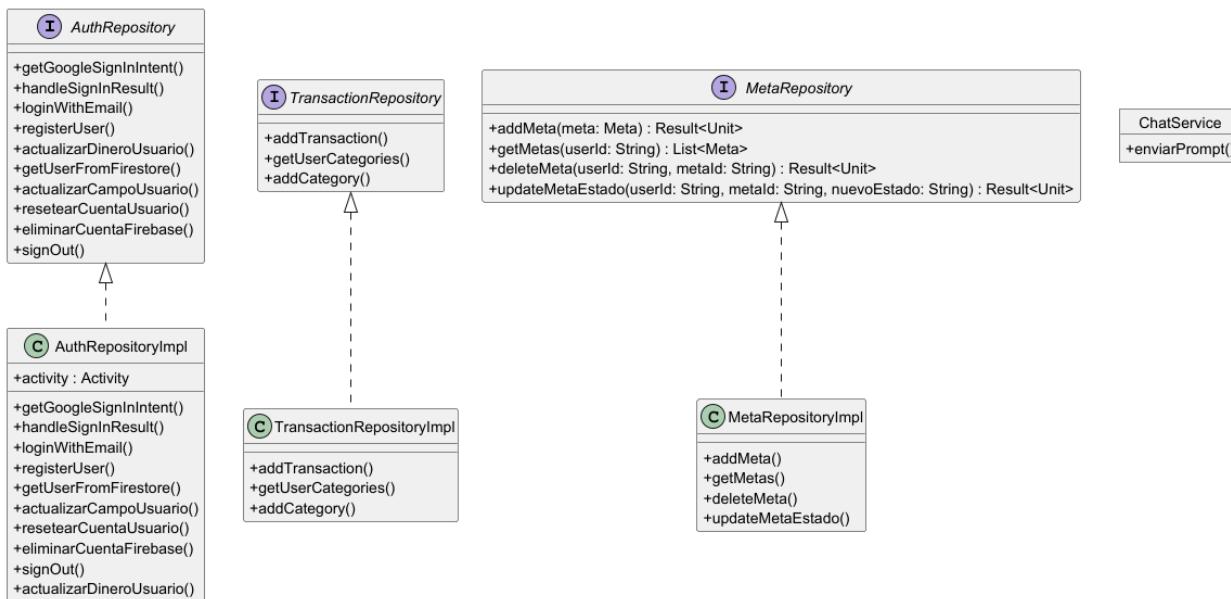
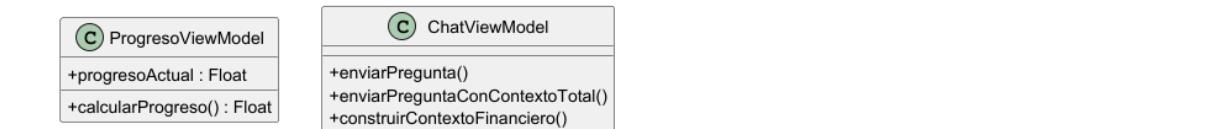
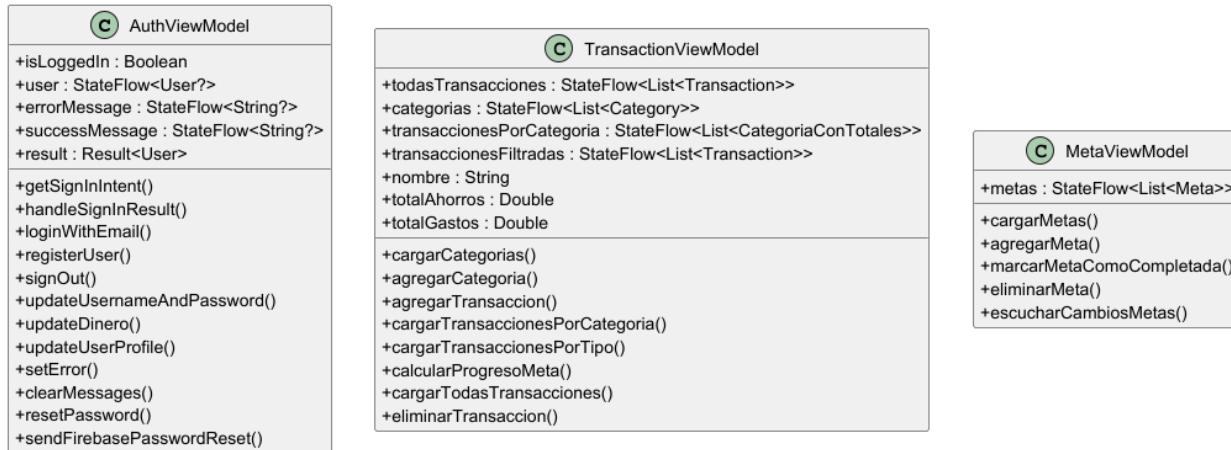
- **Diseño de la base de datos**, adaptado al uso de Firebase como solución de almacenamiento en la nube.
- **Diseño de la interfaz**, orientado a la experiencia de usuario, simplicidad y accesibilidad desde dispositivos móviles.

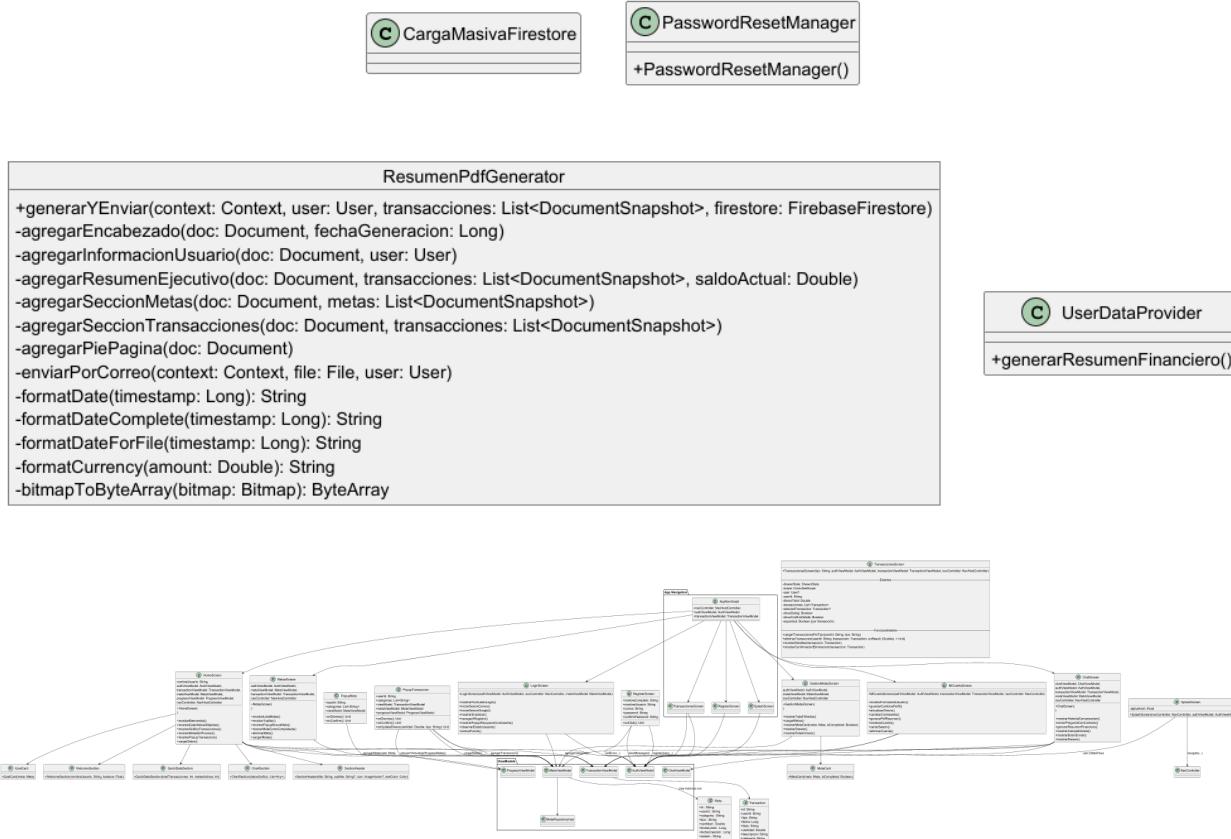
3.2 Diagrama de clases

El diseño del sistema se estructura siguiendo una arquitectura en capas, donde cada clase cumple una función específica dentro del modelo, la lógica de negocio, la presentación de datos o la interfaz gráfica.

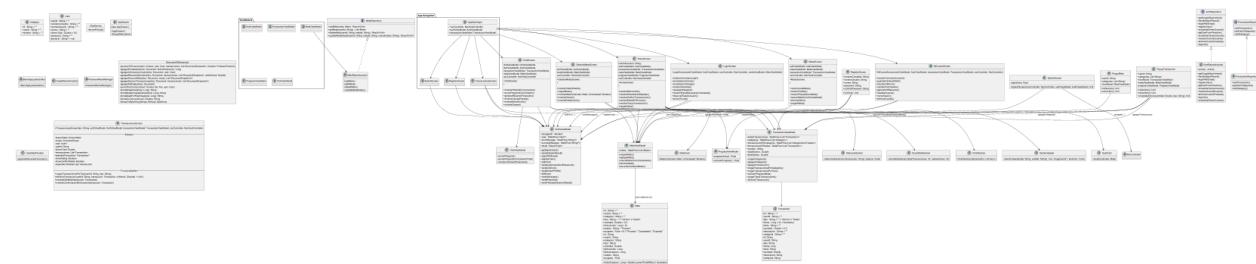
Este diseño modular facilita la escalabilidad, el mantenimiento y la reutilización del código. A continuación, se describen las principales clases detectadas en el proyecto, organizadas por paquete o responsabilidad.







Como el diagrama entre pantallas no se puede apreciar bien y mucho menos el completo:



Dejaré un enlace a las 2 imágenes en mi drive:

<https://drive.google.com/drive/folders/1xKFqooapE5PmP0cHjGLijsghgCqjshjh?usp=sharing>

También los dejaré por partes por si es necesario:

Diagrama de las screen:

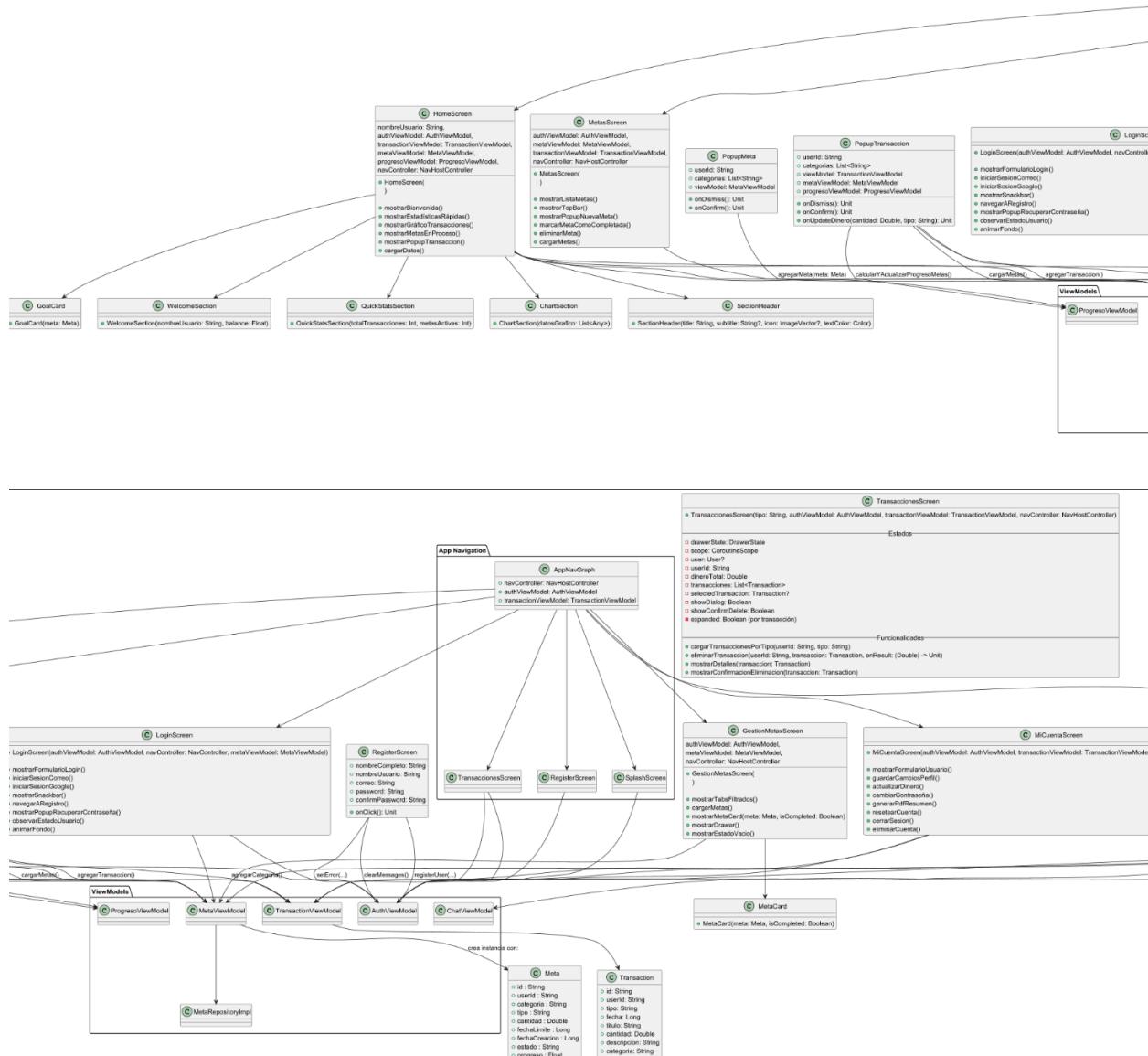
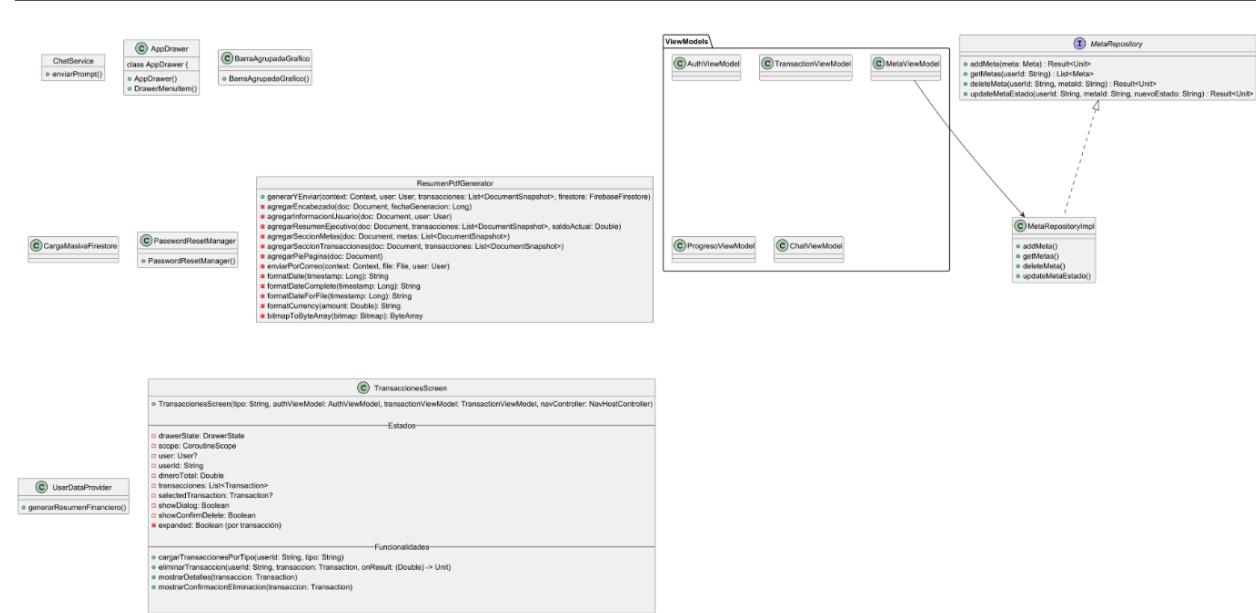
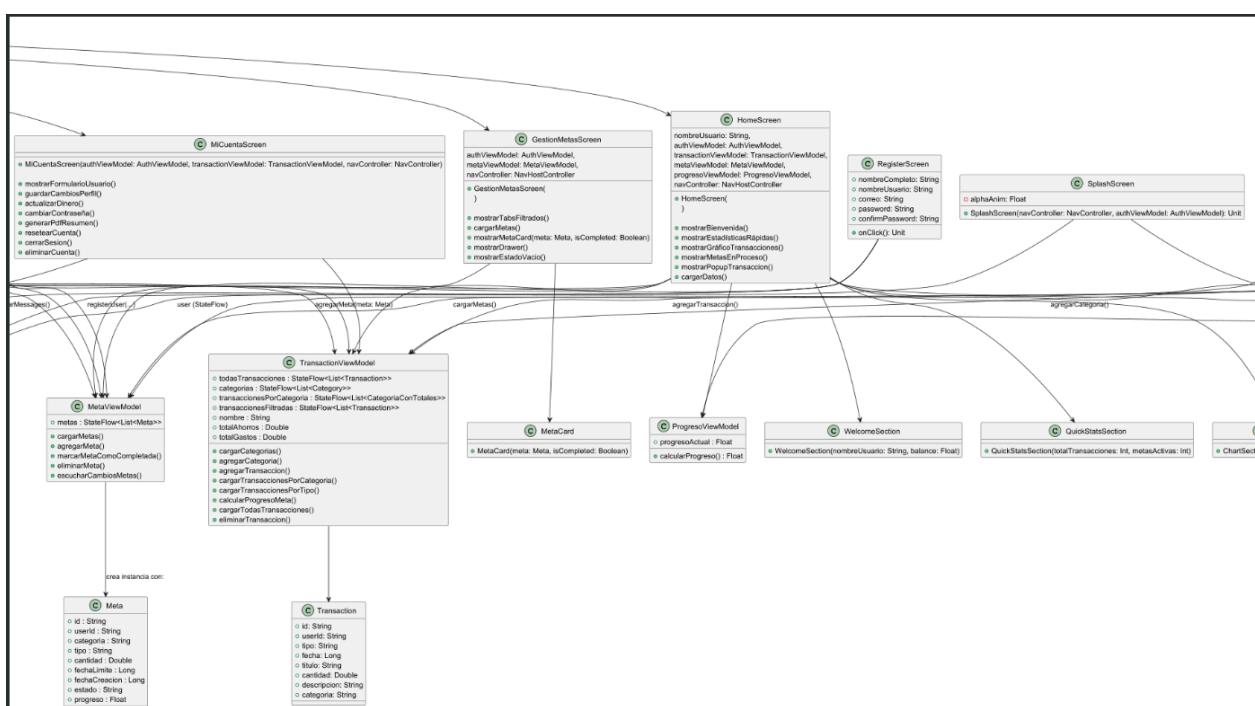
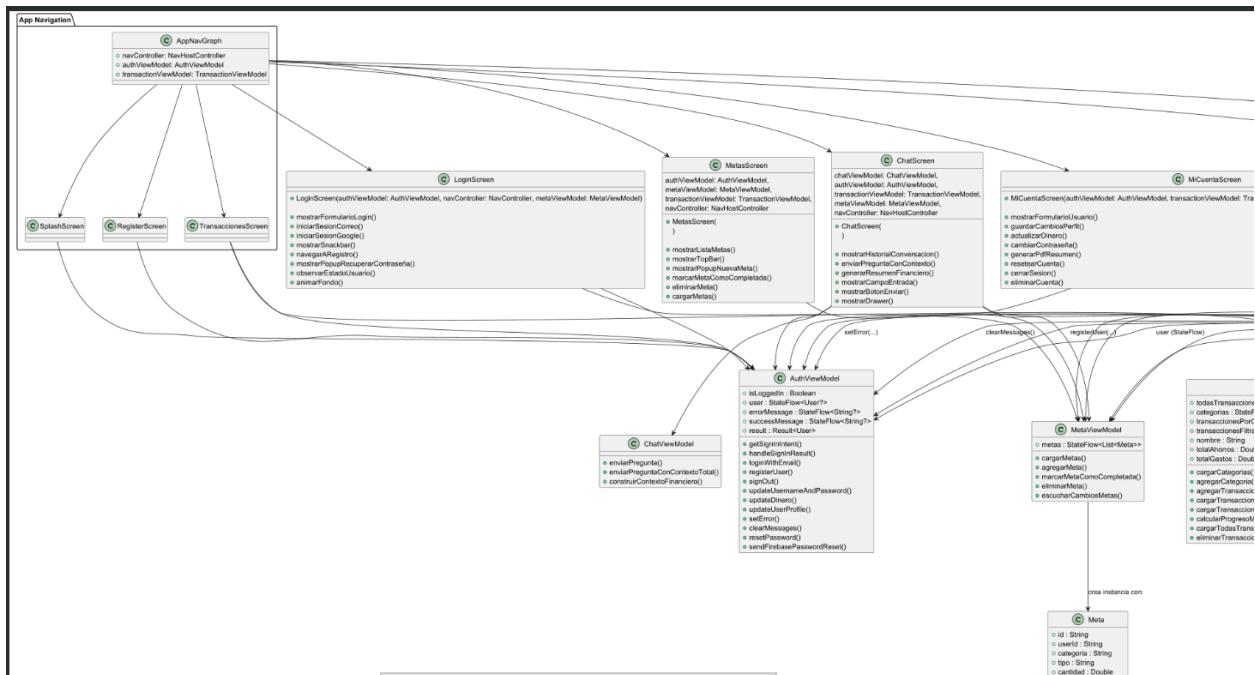
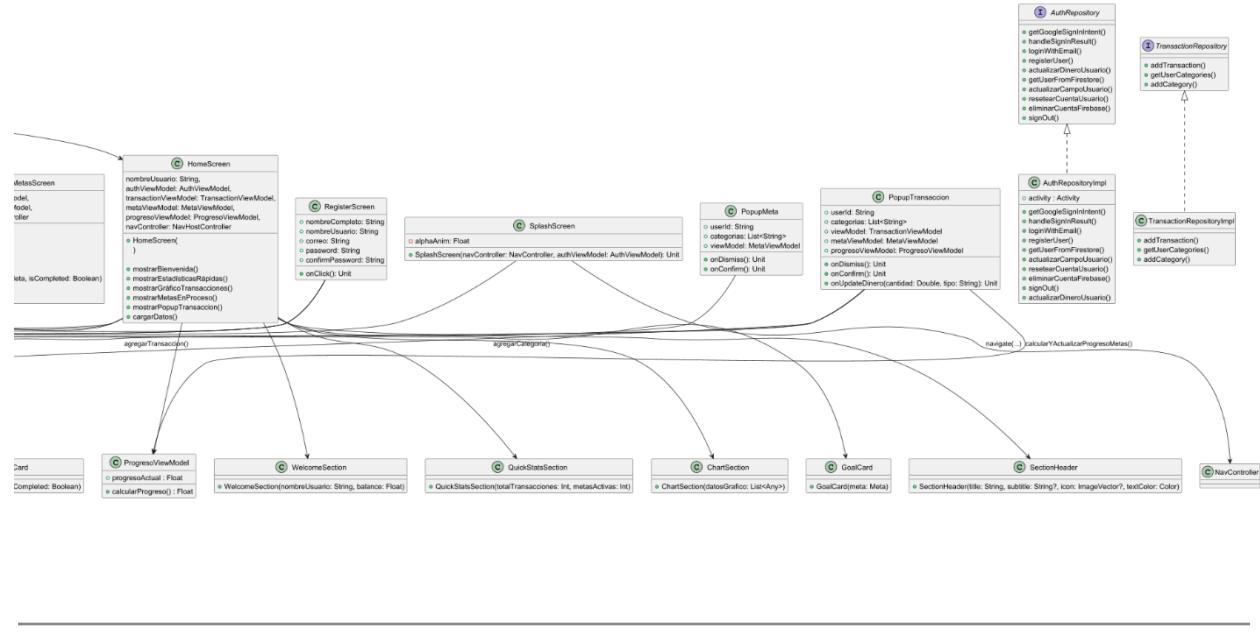




Diagrama del proyecto al completo:







Modelo de Datos (model)

Estas clases representan las entidades principales del sistema:

- **User:** Datos del perfil del usuario.
- **Transaction:** Representa movimientos financieros (ingresos o gastos).
- **Meta:** Define metas financieras.
- **Category:** Categorías personalizadas creadas por el usuario.

Todas están vinculadas por userId.

Lógica de Presentación (viewmodel)

Clases ViewModel responsables de la comunicación entre la UI y los datos:

- **AuthViewModel:** Gestión del login, logout y registro.
- **TransactionViewModel:** Operaciones con transacciones.
- **MetaViewModel:** Manejo de metas.

- **ProgresoViewModel:** Seguimiento del progreso de metas.
 - **ChatViewModel:** Manejo del chatbot.
-

Lógica de Negocio (repository)

Encargadas del acceso a datos y servicios externos:

- **AuthRepository / AuthRepositoryImpl**
 - **TransactionRepository / Impl**
 - **MetaRepository / Impl**
 - **ChatService:** Lógica para la IA financiera.
-

Interfaz de Usuario (ui.screen)

Clases que representan las pantallas o secciones de la app:

- LoginScreen, RegisterScreen, HomeScreen, TransaccionesScreen, MetasScreen, MiCuentaScreen
 - ChatScreen: interfaz del chatbot.
 - PopupTransaccion, PopupMeta: ventanas emergentes.
 - SplashScreen, NavGraph
-

Componentes (ui.components)

- AppDrawer, BarraAgrupadaGrafico: componentes gráficos.
-

Utilidades (utils)

Funciones auxiliares reutilizables:

- **CargaMasivaFirestore**: Inserción de datos predefinidos.
- **PasswordResetManager**: Gestión de recuperación de contraseña.
- **ResumenPdfGenerator**: Generación de informes PDF.
- **UserDataProvider**: Gestión del contexto de usuario.

Este conjunto de clases permite implementar un sistema organizado y funcional que cubre tanto las operaciones financieras como la experiencia de usuario, con especial atención al mantenimiento del estado, el diseño visual moderno, y el soporte de funciones avanzadas como generación de PDF o interacción por IA.

3.3 Diseño de la base de datos

La base de datos utilizada en este proyecto es **Cloud Firestore**, un sistema NoSQL proporcionado por Firebase que almacena la información en documentos y colecciones, en lugar de filas y tablas como en un modelo relacional tradicional.

Este modelo permite trabajar con datos de forma flexible, escalable y optimizada para aplicaciones móviles en tiempo real, lo cual es ideal para una app como la desarrollada en este proyecto.

3.3.1 Diseño lógico

En lugar de tablas, Firestore organiza la información en **colecciones** que contienen documentos. Cada documento puede tener atributos y/o subcolecciones. A continuación se describen las entidades principales del sistema:

- Colección usuarios
 - Documentos identificados por userId.
 - Atributos: correo, nombreCompleto, nombreUsuario, dineroTotal, photoUrl, password.
- Subcolección transacciones dentro de cada usuario
 - Atributos: tipo ("Ingreso"/"Gasto"), cantidad, fecha, descripcion, categoria, titulo..
- Subcolección categorias dentro de cada usuario

- Atributos: nombre, presupuesto.
- Subcolección metas dentro de cada usuario
 - Atributos: categoria, cantidad, fechaLímite, estado, tipo, progreso.

Relaciones lógicas:

- Cada documento de transacciones, metas y categorías pertenece a un único usuario, ya que están anidadas bajo su ID.
- No existen claves foráneas estrictas, pero el campo categoria permite vincular transacciones y metas con una categoría concreta.

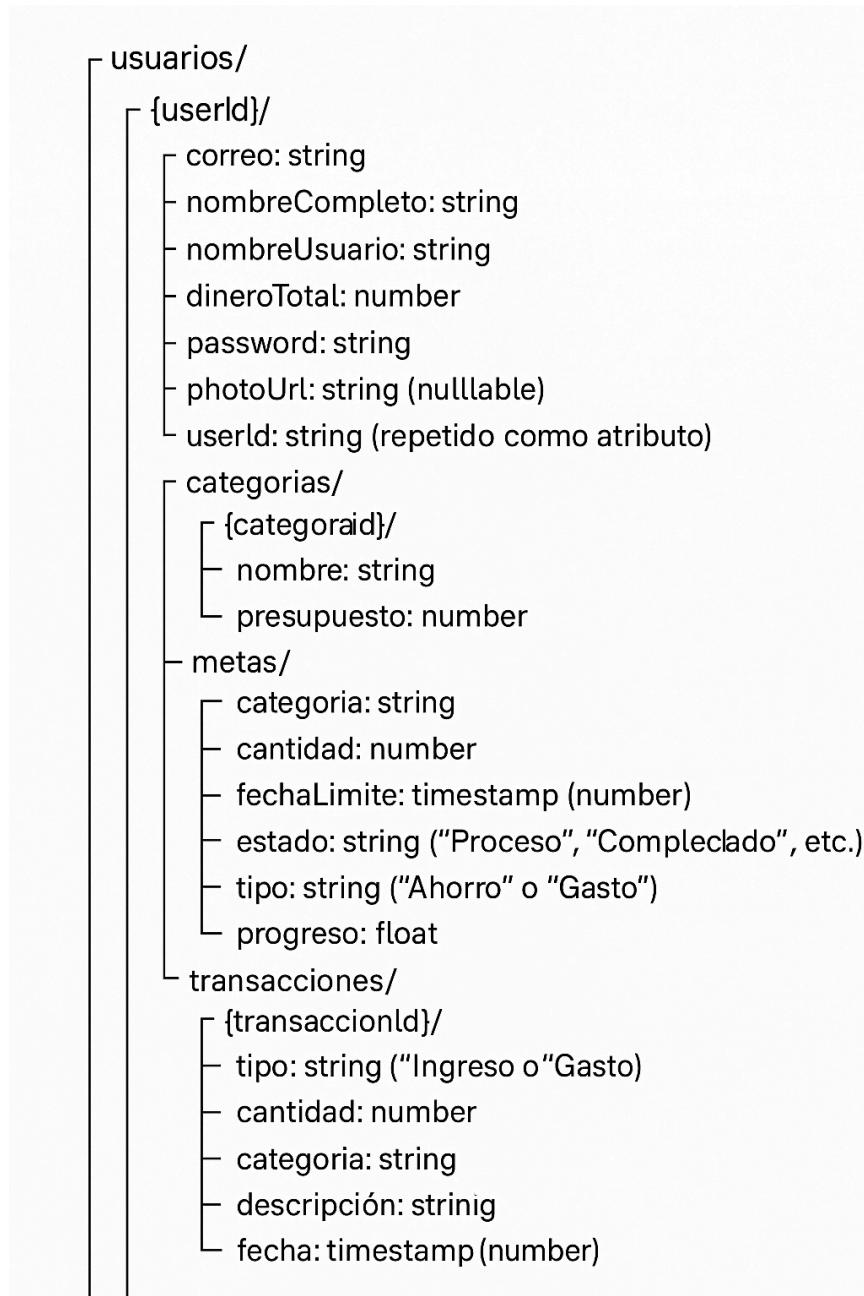
3.3.2 Diseño relacional

Dado que Firestore es una base de datos **NoSQL y no relacional, no se trabaja con tablas, claves primarias ni claves externas tradicionales**. En su lugar:

- Cada documento tiene un ID único (generado automáticamente o personalizado).
- Las relaciones se mantienen a través de la estructura de subcolecciones y la repetición de datos si es necesario (denormalización controlada).
- No hay sentencias JOIN. Las relaciones entre entidades se implementan mediante consultas manuales a diferentes colecciones.

Ventajas de este diseño:

- Mayor velocidad en la recuperación de datos específicos.
- Escalabilidad sin necesidad de rediseñar la estructura.
- Optimización para acceso frecuente desde móviles con conexión variable.



3.4 Diseño de la interfaz de usuario

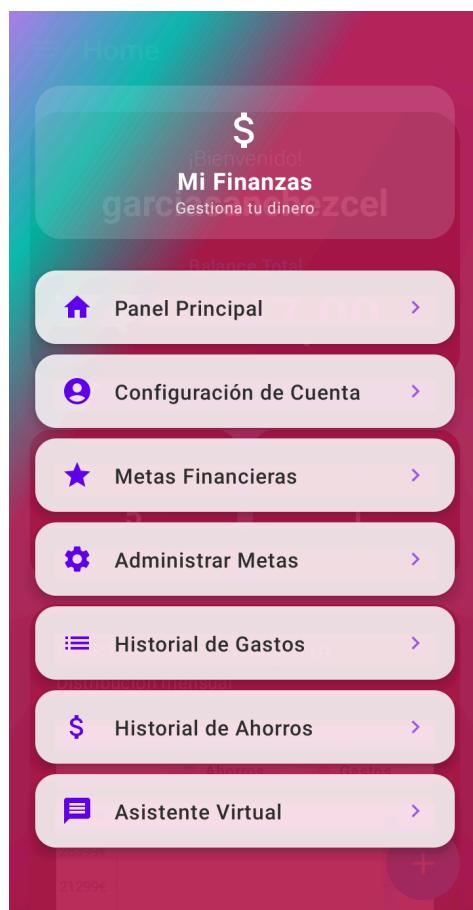
En este apartado se define cuál va a ser la **apariencia visual de la aplicación**, es decir, la interfaz que interactúa directamente con el usuario. El diseño de la interfaz se ha llevado a cabo utilizando **Jetpack Compose**, un moderno framework de UI declarativa para Android que permite construir interfaces dinámicas, modulares y visualmente atractivas con menos código.

El objetivo principal del diseño ha sido ofrecer una experiencia intuitiva, clara y agradable para el usuario, con una distribución lógica de la información y una navegación sencilla entre secciones.

Estructura general y navegación

La aplicación se organiza mediante un sistema de navegación compuesto por pantallas principales y elementos emergentes. El menú de navegación inferior y los botones flotantes permiten al usuario moverse fácilmente por las secciones de:

- Inicio
- Transacciones
- Metas
- Mi cuenta
- Asistente (chatbot)



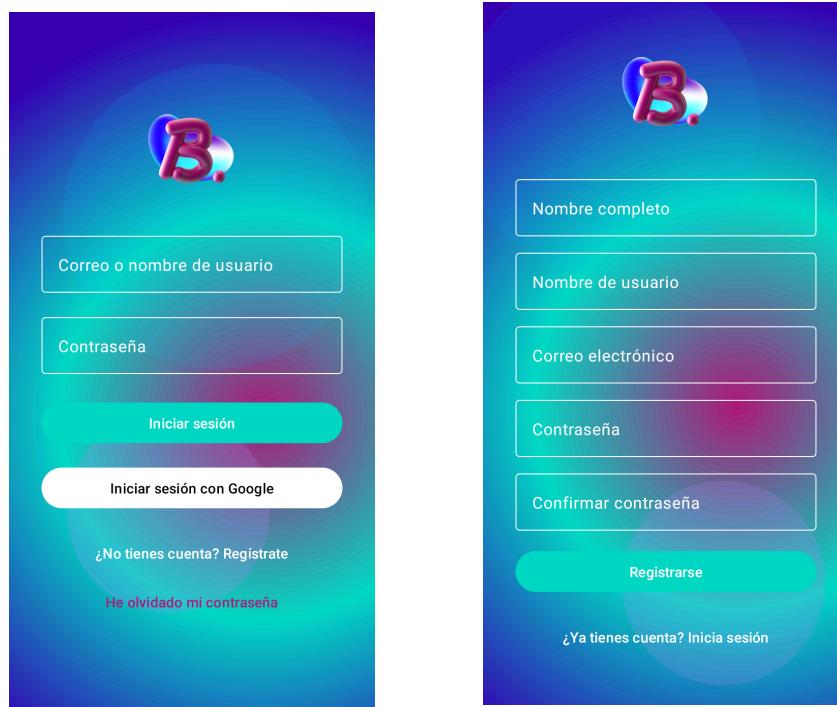
Componentes reutilizables

Se han creado varios **componentes personalizados** reutilizables como:

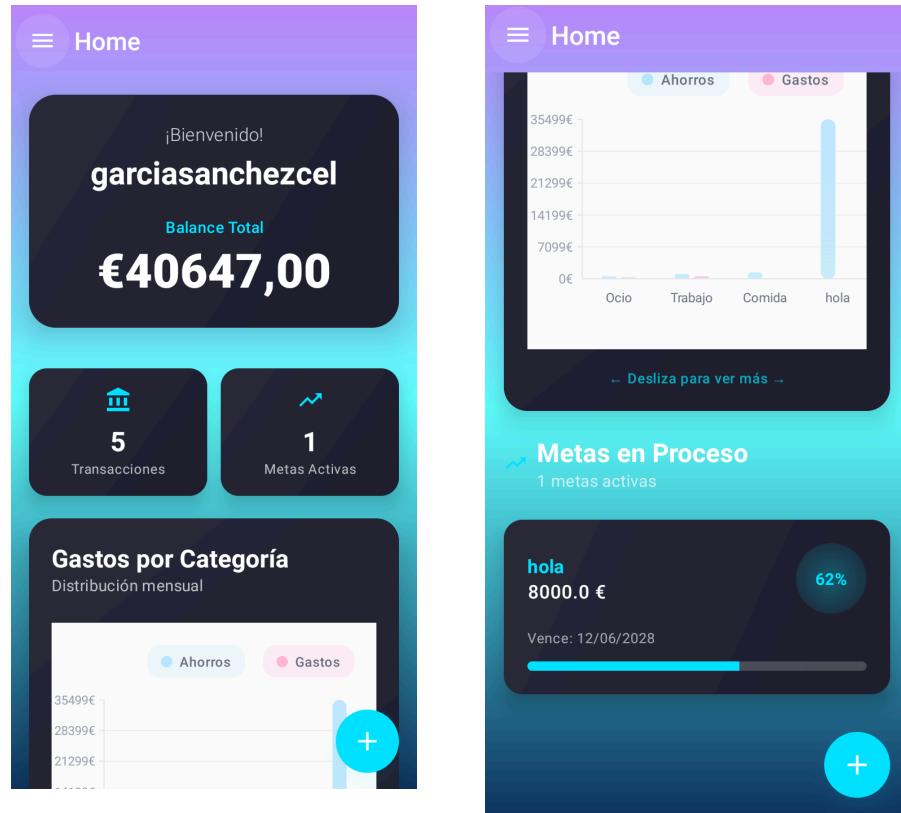
- **AppDrawer**: menú lateral de navegación.
- **BarraAgrupadaGrafico**: barra gráfica para visualizar estadísticas de forma interactiva.
- Botones y tarjetas con diseño redondeado, sombras suaves y colores adaptados a la identidad visual de la app.

Pantallas principales

- **LoginScreen & RegisterScreen**: formularios limpios con campos validados y botones accesibles.



- **HomeScreen**: acceso rápido al resumen financiero y secciones clave.



- **TransaccionesScreen:** lista detallada de movimientos con posibilidad de eliminar.

Transacciones de Ahorro (Left):

- hhuuu: Categoría: Trabajo \$ 200,00€
- hjhj: Categoría: Comida \$ 300,00€ (with 'Eliminar transacción' button)
- huj: Categoría: hola \$ 6.000,00€
- jjj: Categoría: hola \$ 200,00€
- hdjdjd: Categoría: hola \$ 5.000,00€

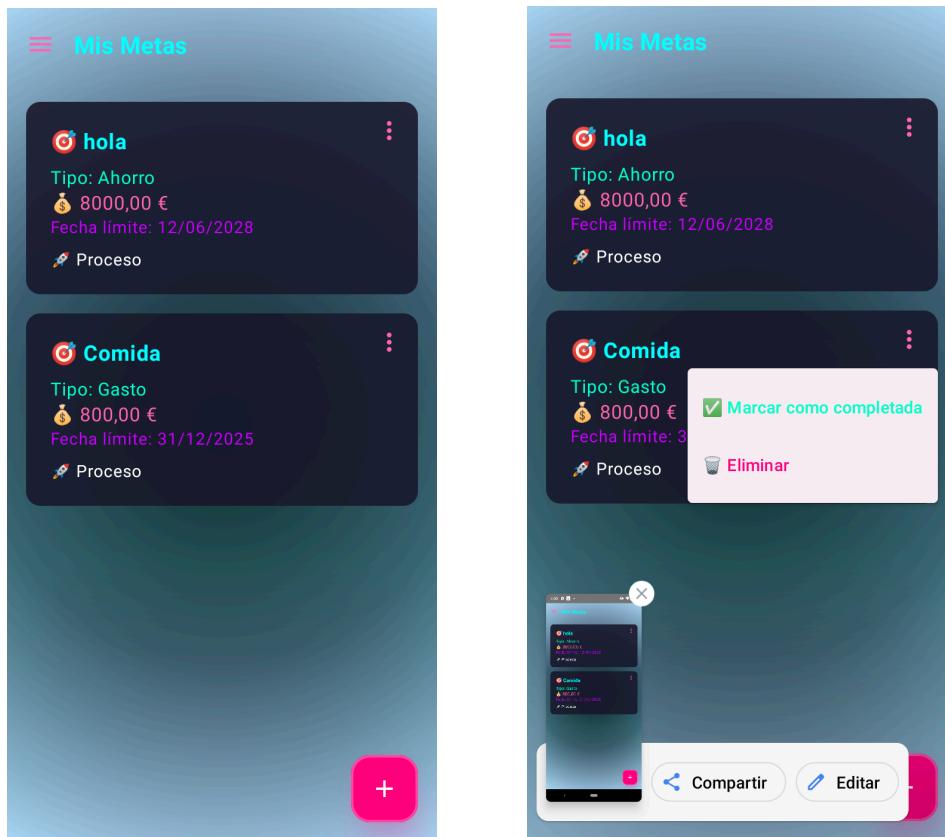
Transacciones de Ahorro (Middle):

- hhuuu: Categoría: Trabajo \$ 200,00€
- Detalles de la transacción:**
 - Título: jjj
 - Descripción: jjj
 - Categoría: hola
 - \$ 200,00€
 - 27/05/2025
- huj: Categoría: hola \$ 6.000,00€ (with 'Eliminar transacción' button)
- jjj: Categoría: hola \$ 200,00€
- hdjdjd: Categoría: hola \$ 5.000,00€

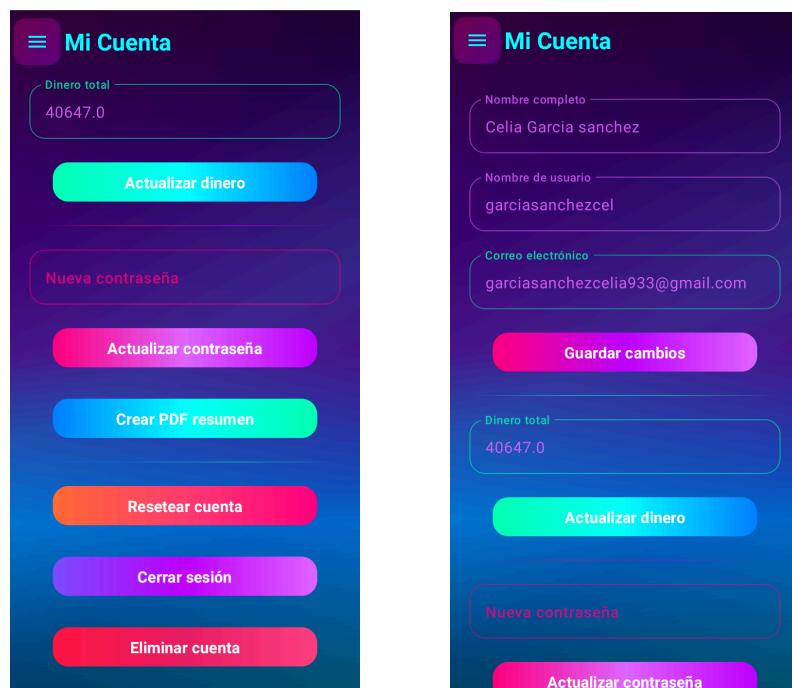
Transacciones de Gasto (Right):

- Gasto 7**: Categoría: Ocio \$ 11,00€
- Gasto 0**: Categoría: Comida \$ 94,38€
- Gasto 4**: Categoría: Ocio \$ 92,39€ (with 'Eliminar transacción' button)
- Gasto 1**: Categoría: Trabajo \$ 31,84€
- Gasto 2**: Categoría: Ocio \$ 67,33€

- **MetasScreen:** muestra objetivos activos con progreso visual.



- **MiCuentaScreen:** perfil del usuario con opciones de configuración y cierre de sesión.

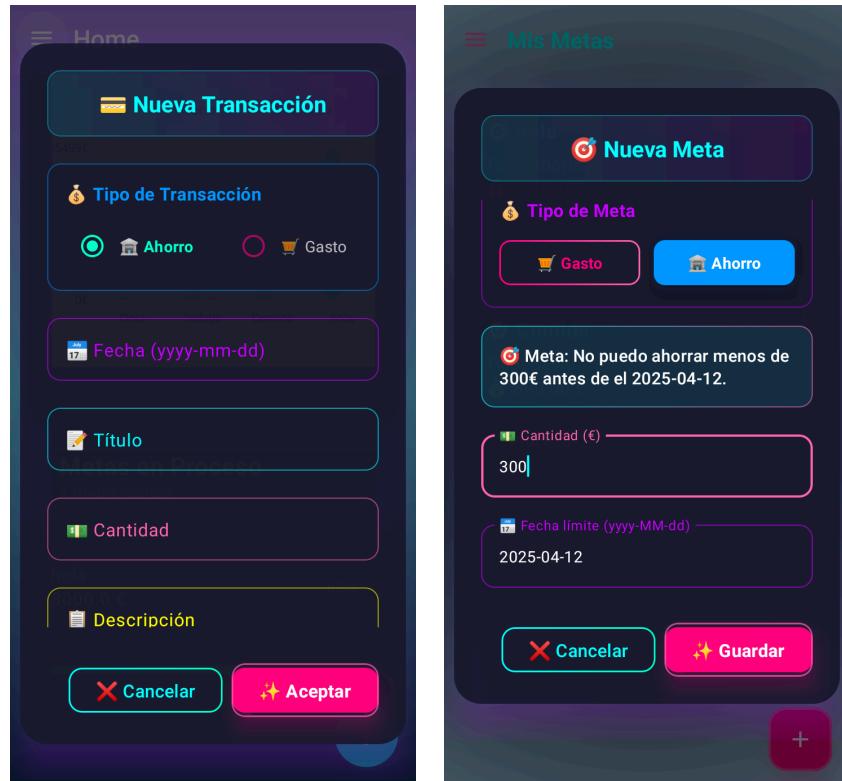


- **ChatScreen:** interfaz del chatbot con estilo de conversación limpio y centrado.



Elementos emergentes (Popups)

- **PopupTransaccion:** para añadir o editar ingresos y gastos.
- **PopupMeta:** para crear y configurar metas de ahorro o control de gasto.

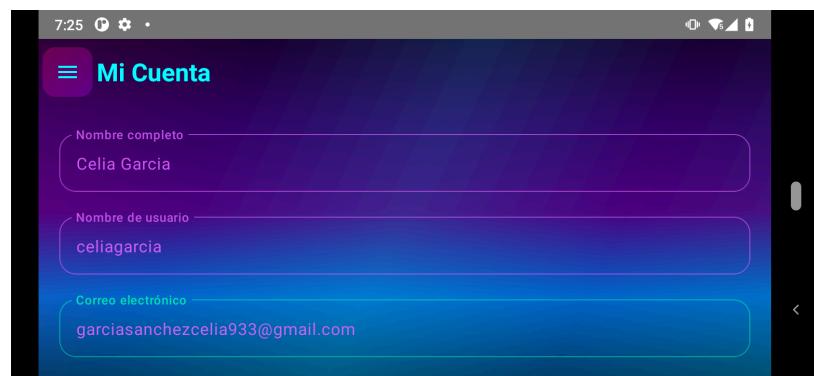
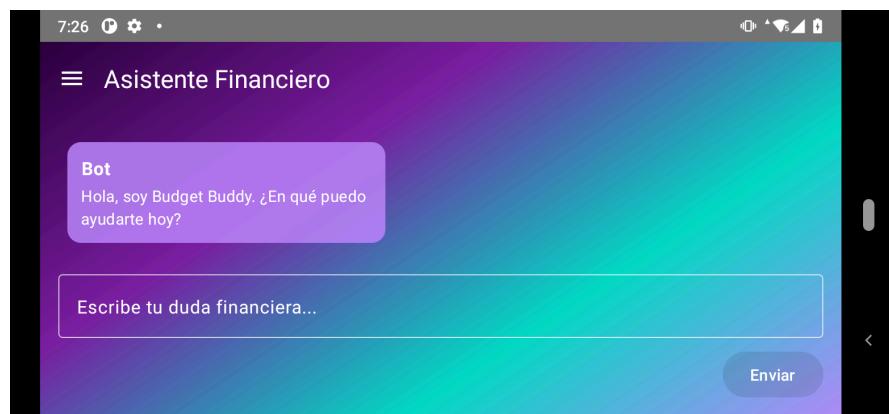
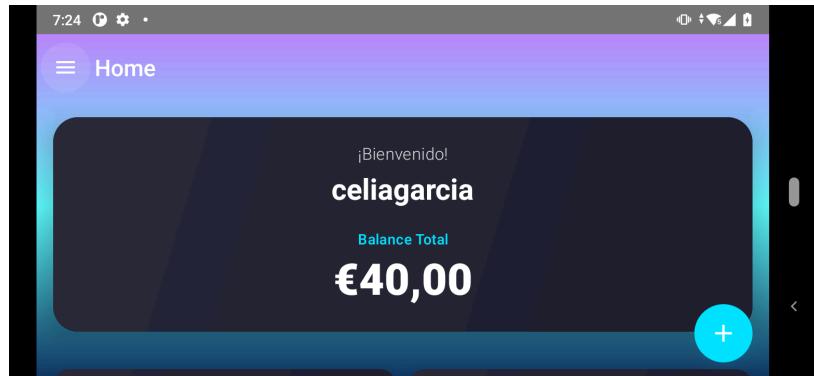


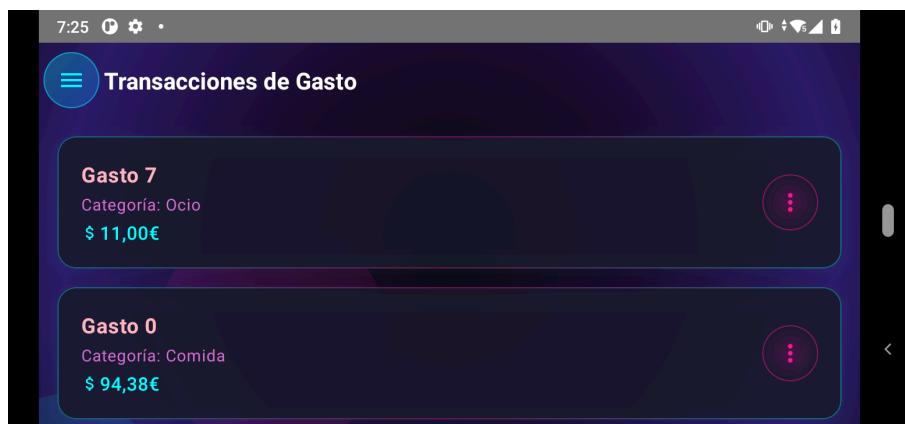
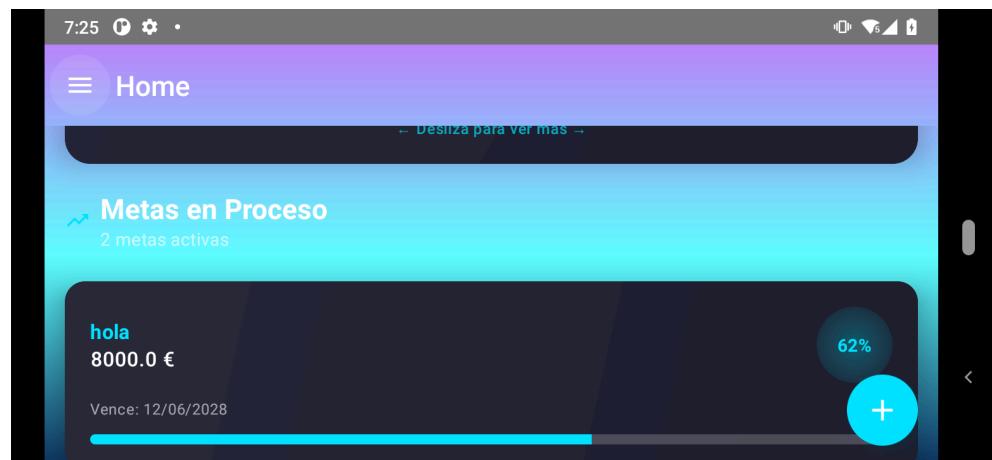
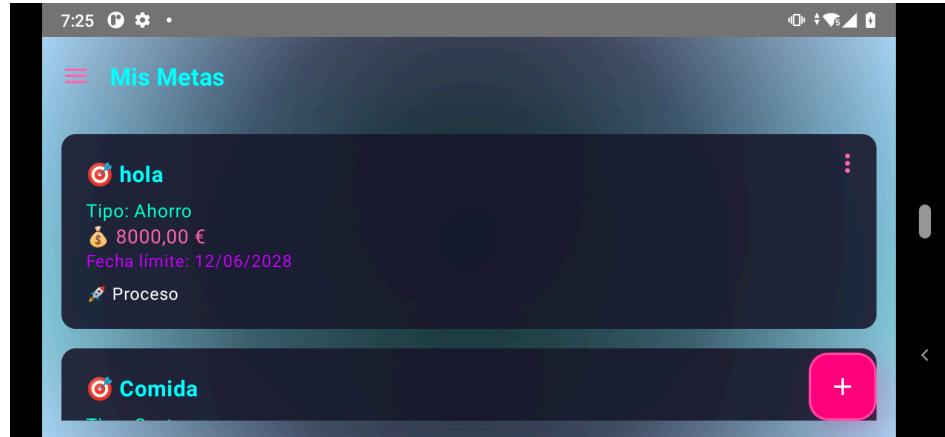
Estética general

Se ha utilizado un esquema de colores moderno y accesible, con tipografía clara y tamaño adaptable. El diseño está pensado para ofrecer una experiencia uniforme tanto en pantallas pequeñas como en otros tamaños.

3.5 Adaptación a dispositivos







4. Implementación

4.1 Introducción

En este apartado se expone la transformación del diseño lógico y estructural del sistema en código funcional. Una vez definidos los requisitos y completado el diseño, se procede a implementar la solución en una aplicación Android totalmente operativa, cumpliendo con los objetivos establecidos en las fases anteriores.

4.2 Arquitectura utilizada

Para el desarrollo de esta aplicación se ha utilizado una arquitectura basada en el patrón **MVVM (Model - View - ViewModel)**, siguiendo además algunos principios de **Clean Architecture**, lo cual permite una mejor organización del código, mayor mantenibilidad y escalabilidad futura.

Estructura general

La arquitectura se divide en varias capas bien diferenciadas:

- **Modelo (Model)**: Contiene las clases que representan los datos de la aplicación (como User, Transaction, Meta, Category). Estos modelos son utilizados en todas las capas superiores y reflejan directamente la información almacenada en la base de datos (Firebase).
- **Vista (View / UI)**: Compuesta por pantallas (Screens) desarrolladas con **Jetpack Compose**. Esta capa solo se encarga de mostrar información y recoger acciones del usuario. La UI es reactiva, y se actualiza automáticamente cuando cambia el estado.
- **ViewModel**: Contiene la lógica de presentación y actúa como intermediario entre la vista y los repositorios. Cada pantalla cuenta con su ViewModel correspondiente (AuthViewModel, TransactionViewModel, etc.), donde se gestiona el estado de la interfaz y las llamadas a la capa de datos.
- **Repositorio (Repository)**: Encapsula el acceso a los datos y las operaciones con Firebase. Esta capa se comunica con Firestore y otros servicios como el chatbot IA, ofreciendo una interfaz clara para los ViewModels.

Ventajas de esta arquitectura

- **Separación de responsabilidades**: cada clase cumple una función específica.

- **Testabilidad:** facilita la realización de pruebas unitarias al desacoplar las dependencias.
- **Escalabilidad:** se pueden añadir nuevas funcionalidades sin afectar al resto de la aplicación.
- **Mantenibilidad:** al seguir una estructura clara, el código es más fácil de entender y modificar.

Esta arquitectura ha sido clave para mantener un desarrollo organizado, especialmente en una aplicación con múltiples pantallas, funcionalidades como metas, transacciones, y comunicación con un asistente basado en IA.

4.3 Lenguajes y tecnologías empleadas

Para el desarrollo de esta aplicación móvil se han utilizado distintos lenguajes de programación y tecnologías modernas, seleccionadas por su compatibilidad con Android y su capacidad para ofrecer una experiencia de usuario fluida, escalable y segura.

Lenguajes de programación

- **Kotlin:** Es el lenguaje principal del proyecto. Kotlin es el lenguaje oficial recomendado por Google para Android, moderno, conciso y seguro. Ha sido utilizado para implementar toda la lógica de la aplicación, desde la interfaz hasta la gestión de datos y la comunicación con servicios externos.
- **XML:** Se ha utilizado de forma puntual para configuraciones del proyecto, como el `AndroidManifest.xml` y archivos de recursos (`strings.xml`, temas y colores), aunque la interfaz visual se ha desarrollado completamente con Jetpack Compose.
- **JSON:** Se utiliza implícitamente para la representación de datos en la comunicación con Firebase Firestore y para manejar la estructura de datos intercambiados con el chatbot por IA.

Tecnologías y frameworks

- **Jetpack Compose:** Framework de UI declarativa para Android. Ha permitido construir una interfaz moderna, dinámica y completamente adaptada a diferentes resoluciones de pantalla, con una gestión eficiente del estado de la interfaz.
- **Firebase:**

- **Firebase Authentication:** Para el registro e inicio de sesión de usuarios con email y con Google.
 - **Cloud Firestore:** Base de datos NoSQL en la nube para almacenar usuarios, transacciones, metas y categorías.
 - **Firebase Console:** Para gestionar reglas de seguridad, usuarios y estructura de la base de datos.
 - **OpenAI (ChatGPT API):** Utilizado para implementar un chatbot financiero integrado en la aplicación, que asiste al usuario en la gestión de sus finanzas de forma conversacional.
 - **Android Studio:** Entorno de desarrollo integrado (IDE) principal para el proyecto, con emuladores, herramientas de depuración y compatibilidad total con Compose.
-

Esta combinación de lenguajes y herramientas ha permitido desarrollar una aplicación robusta, modular, visualmente atractiva y preparada para su evolución futura.

4.4 Herramientas de desarrollo

Durante el desarrollo de la aplicación **BudgetBuddy**, se han empleado diversas herramientas y entornos que han facilitado la implementación, gestión de versiones y automatización del proyecto. A continuación, se detallan las más relevantes:

- **Android Studio:** Es el entorno de desarrollo integrado (IDE) oficial para Android. Proporciona herramientas potentes como el editor de interfaces, el sistema de depuración, la gestión de emuladores y la integración con Gradle. Fue utilizado para escribir, probar y compilar todo el código de la aplicación.
- **Git:** Sistema de control de versiones distribuido utilizado para gestionar el código fuente del proyecto. Permitió llevar un control detallado del historial de cambios, trabajar de forma colaborativa y mantener copias seguras del desarrollo.
- **GitHub:** Plataforma basada en Git que se utilizó como repositorio remoto para alojar el proyecto. También facilitó la gestión de ramas, issues y documentación colaborativa.
- **Gradle:** Herramienta de automatización utilizada para la compilación del proyecto, manejo de dependencias y construcción de versiones de la app. Su sistema de scripts permitió configurar módulos, bibliotecas externas y tareas personalizadas.

- **Firebase:** Plataforma de desarrollo backend utilizada para la autenticación de usuarios (Firebase Authentication) y el almacenamiento de datos en la nube (Firebase Firestore). También se utilizó para funciones como el registro de errores y el alojamiento de archivos.
- **Material 3 (Jetpack Compose):** Conjunto de bibliotecas de diseño que proporcionan componentes visuales modernos y adaptables, integrados de forma nativa con Jetpack Compose para una experiencia de UI fluida y coherente.
- **Emuladores de Android:** Herramientas de prueba incluidas en Android Studio que permitieron ejecutar la aplicación en diferentes configuraciones de dispositivos virtuales para asegurar la compatibilidad y el buen funcionamiento.

4.5 Dependencias y librerías utilizadas

Durante el desarrollo de la aplicación se han utilizado diversas librerías y herramientas modernas que facilitan la construcción de interfaces dinámicas, la gestión de datos en la nube y la navegación entre pantallas. A continuación, se detallan las más relevantes:

- **Jetpack Compose:** Framework declarativo de UI moderno de Android que permite construir interfaces de usuario de forma más intuitiva y eficiente. Se han empleado componentes como Material3, LazyColumn, Scaffold, OutlinedTextField, entre otros.
- **Firebase** (a través del BoM de Google):
 - **Firebase Auth:** Para la autenticación de usuarios mediante correo, contraseña y Google.
 - **Firebase Firestore:** Base de datos NoSQL en la nube utilizada para almacenar las transacciones, metas y datos de usuario.
 - **Firebase Analytics** (integrado, aunque no usado directamente en el código fuente analizado).
- **Kotlin Coroutines:** Librería para la programación asíncrona y concurrente, utilizada principalmente en ViewModels para acceder a Firestore sin bloquear el hilo principal.
- **Navigation Compose:** Para la navegación declarativa entre pantallas usando el componente NavHost.
- **Lifecycle ViewModel & LiveData / StateFlow:** Para la gestión del estado y lógica de negocio a través de ViewModel, en combinación con collectAsState().

- **Accompanist:** Librería auxiliar para animaciones y navegación con efectos en Compose (accompanist-navigation-animation).
- **Material Icons y Material3:** Para el uso de iconografía moderna y compatibilidad con el diseño Material Design actualizado.

Estas herramientas y librerías han permitido desarrollar una aplicación robusta, con un enfoque moderno, multiplataforma y con capacidades de sincronización en la nube en tiempo real.

4.6 Gestión de permisos

La aplicación está diseñada para operar con servicios en la nube, principalmente a través de Firebase (Authentication, Firestore, Storage) y no requiere el acceso a recursos sensibles del dispositivo como la cámara, contactos, almacenamiento externo o ubicación.

Permiso utilizado:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Este permiso se declara en el archivo AndroidManifest.xml para permitir que la aplicación pueda realizar peticiones a internet, necesarias para:

- Autenticación de usuarios (Firebase Authentication)
- Lectura y escritura en Firestore
- Acceso a Firebase Storage
- Llamadas HTTP mediante Retrofit
- Envío de PDF por correo electrónico desde ResumenPdfGenerator

Este es un permiso normal según la clasificación de Android, por lo que no requiere aprobación del usuario durante la instalación ni en tiempo de ejecución.

4.7 Estructura del proyecto

```
com.example.proyectodef
```

```
|
```

```
└── data → Repositorios e implementación de lógica de datos
```

```
|   └── repository      → Interfaces y clases como AuthRepositoryImpl, MetaRepositoryImpl,  
etc.  
|  
|  
└── model            → Clases de datos (Modelos)  
|   └── Meta.kt  
|   └── Transaction.kt  
|   └── User.kt  
|  
|  
└── ui  
|   └── components    → Componentes reutilizables como AppDrawer  
|   └── screen         → Todas las pantallas (LoginScreen, HomeScreen, MetasScreen, etc.)  
|  
|  
└── utils            → Utilidades como PasswordResetManager y ResumenPdfGenerator  
|  
|  
└── viewmodel        → ViewModels como AuthViewModel, MetaViewModel,  
TransactionViewModel, etc.  
|  
|  
└── MainActivity.kt  → Punto de entrada de la aplicación  
|  
|  
└── navigation       → Contiene AppNavGraph para la navegación entre pantallas
```

4.8 Codificación

En este apartado hablaré sobre cómo se han desarrollado las principales funcionalidades de mi app:

4.8.1 Navegación entre pantallas

En mi proyecto, la navegación entre pantallas la organicé utilizando **un sistema de rutas internas**, como si cada pantalla tuviera una dirección.

La navegación la controlé con un componente llamado NavHost, que se encarga de mostrar la pantalla correcta según la ruta actual. Este componente está en un archivo llamado AppNavGraph, y ahí definí todas las rutas de mi aplicación y qué pantalla corresponde a cada una.

```
NavHost(  
    navController = navController,  
    startDestination = "splash"  
) {  
    composable("splash") { SplashScreen(...) }  
    composable("login") { LoginScreen(...) }  
    composable("home") { HomeScreen(...) }  
    composable("miCuenta") { MiCuentaScreen(...) }  
    composable("transacciones/{tipo}") { ... }  
    composable("metas") { MetasScreen(...) }  
    composable("gestionMetas") { GestionMetasScreen(...) }  
    composable("chatbot") { ChatScreen(...) }  
}
```

La transición entre pantallas se realiza con navController.navigate("ruta"), como se ve en el Splash:

```
navController.navigate(if (user != null) "home" else "login") {  
    popUpTo("splash") { inclusive = true }  
    launchSingleTop = true  
}
```

El controlador de navegación (NavController) es el que lleva el control de a dónde debe ir el usuario, y yo lo utilizo para decidir la ruta según el estado del usuario.

Ejemplo real de mi app:

- En la pantalla splash, si el usuario ya está autenticado (sesión iniciada), navega automáticamente a "home".
- Si no ha iniciado sesión, lo llevo a "login".

Esto hace que la app sea inteligente y fluida, evitando que el usuario vea pantallas que no le corresponden.

4.8.2 Inicio de sesión y autenticación con Google

Además del inicio de sesión tradicional con correo y contraseña, **implementé la opción de iniciar sesión con Google**, para hacerlo más rápido y cómodo para el usuario.

Esto lo logré **conectando mi app con los servicios de autenticación de Firebase**, que permite integrar fácilmente proveedores como Google, y gestionarlo todo de forma segura.

1. **Firebase Authentication** es el sistema que usé para gestionar usuarios.
2. Al pulsar el botón “Iniciar sesión con Google”, se abre una ventana donde el usuario puede **elegir su cuenta de Google**.
3. Google valida su identidad y devuelve a mi app un “token de autenticación” (como un pase).
4. Ese token lo **intercepta Firebase** para crear o recuperar al usuario dentro de mi base de datos.
5. Finalmente, si todo es correcto, **mi app reconoce al usuario como autenticado** y lo lleva directamente a la pantalla principal (“home”).

En LoginScreen.kt, se ofrece autenticación por email y por cuenta Google. Para Google:

```
val launcher = rememberLauncherForActivityResult(  
    contract = ActivityResultContracts.StartActivityForResult()  
) { result ->  
    if (result.resultCode == Activity.RESULT_OK) {  
        val data: Intent? = result.data  
        authViewModel.handleSignInResult(data)  
    }  
}  
  
// Disparador  
  
Button(onClick = {  
    val intent = authViewModel.getSignInIntent()  
    launcher.launch(intent)  
}){
```

```
    Text("Iniciar sesión con Google")  
}
```

Esto utiliza Google Sign-In y Firebase Auth.

4.8.3 Restablecer contraseña por email

En la pantalla de login, el usuario puede iniciar el flujo de recuperación:

```
TextButton(onClick = {  
    showResetPopup = true  
}) {  
    Text("He olvidado mi contraseña")  
}
```

Esto muestra un AlertDialog personalizado (componente PasswordResetManager) que envía el email de recuperación usando Firebase:

```
Firebase.auth.sendPasswordResetEmail(email)
```

4.8.4 Añadir transacciones

Desde PopupTransaccion.kt se permite agregar transacciones con categorías:

```
val transaction = Transaction(  
    id = "",  
    userId = userId,  
    tipo = tipo,  
    fecha = System.currentTimeMillis(),  
    titulo = titulo,  
    cantidad = cantidad.toDouble(),
```

```
    descripcion = descripcion,  
    categoria = categoriaFinal  
)  
  
viewModel.agregarTransaccion(transaction)
```

También se actualiza el dinero total:

```
onUpdateDinero(transaction.cantidad, transaction.tipo)
```

Y se recalculan metas:

```
metaViewModel.cargarMetas(userId)  
progresoViewModel.calcularYActualizarProgresoMetas(userId, metas)
```

4.8.5 Visualización de gráfico agrupado (BarraAgrupadaGrafico)

El gráfico se genera con la librería MPAndroidChart y se encapsula en el componente BarraAgrupadaGrafico. Aunque su definición no estaba completa directamente, su funcionalidad se deduce de cómo se muestran los datos de transacciones por categoría en gráficos de barras agrupadas.

El ViewModel prepara los datos agrupados por categoría y tipo:

```
val datos = transacciones.groupBy { it.categoría }  
.map { (categoria, transaccionesCategoria) ->  
    categoria to transaccionesCategoria.sumOf { it.cantidad }  
}
```

Estos datos se pasan al gráfico para mostrar un resumen visual.

4.8.6 Cálculo del progreso de metas

El progreso se calcula en el ProgresoViewModel, utilizando datos de transacciones y metas. Se recorren las metas en proceso y se suman transacciones que contribuyen a ellas:

```
val progreso = metas.map { meta ->  
    val suma = transacciones.filter {
```

```
it.userId == meta.userId && it.categoría == meta.categoría && it.tipo == meta.tipo  
}.sumOf { it.cantidad }  
  
meta.copy(progreso = (suma / meta.cantidad).coerceAtMost(1.0).toFloat())  
}
```

Luego se actualizan en Firestore mediante updateMetaProgreso().

4.8.7 Generación de PDF resumen

En MiCuentaScreen, al pulsar el botón correspondiente, se obtienen las transacciones del último mes y se genera un PDF

Este generador está definido en ResumenPdfGenerator.kt, donde:

- Se construye un archivo PDF usando PdfDocument.
- Se escribe información relevante: nombre del usuario, email, resumen de transacciones.
- Se genera una estructura tipo tabla.

ResumenPdfGenerator.generarYEnviar()

```
context = navController.context,  
  
user = u,  
  
transacciones = transaccionesMes,  
  
firestore = db  
)
```

Esto utiliza Firestore para cargar datos y una clase de utilidad que genera el PDF y lo guarda o comparte.

Dentro del generador, se lanza un **Intent de tipo email** con el PDF adjunto.

4.8.8 Chatbot financiero con contexto

La funcionalidad del chatbot se implementa en ChatViewModel y ChatScreen. Cuando el usuario escribe una pregunta.

```
chatViewModel.enviarPreguntaConContextoTotal(  
    pregunta = input,  
    userId = user!!.userId,  
    onRespuesta = {}  
)
```

El ChatViewModel concatena datos del usuario, transacciones y metas, y envía esa información a la API del asistente (como contexto). El resultado se agrega al historial del chat y se muestra con estilo conversacional en LazyColumn:

```
items(history) { (q, a) ->  
    // muestra pregunta y respuesta estilizadas  
}
```

- Las respuestas son más útiles que con un chatbot sin contexto.
- El historial de chat se mantiene con chatHistory.
- Usa collectAsState() y remember para mantener datos sincronizados con el ViewModel.

Esto permite al usuario hacer preguntas como: “¿Cuánto he gastado en comida este mes?” y recibir respuestas personalizadas.

5. Pruebas de software

5.1 Introducción

En este apartado se detallan las pruebas realizadas a la aplicación desarrollada. Estas pruebas permiten comprobar el correcto funcionamiento de cada parte del sistema, detectar posibles errores y asegurar una experiencia adecuada para el usuario.

Se aplicaron principalmente pruebas funcionales (caja negra) en las pantallas más importantes de la app, como el login, la gestión de transacciones, las metas de ahorro y el chatbot.

5.2 Técnicas de prueba

En el proyecto se han considerado dos enfoques principales para la validación del software:

- **Caja blanca (o estructural)**: se enfoca en cómo está construido el sistema internamente.
- **Caja negra (o funcional)**: se centra en si el sistema cumple correctamente con los requisitos, sin conocer su interior.

5.2.1 Pruebas de caja blanca

En este proyecto, se aplicaron pruebas de caja blanca principalmente durante el desarrollo de funcionalidades como:

- El cálculo del progreso de metas, verificando que los valores no sobrepasen el 100%.
- La generación del PDF, revisando que los datos se formateen correctamente.
- El filtrado de transacciones, observando cómo se agrupan y se cargan desde Firebase.

Para facilitar las pruebas estructurales y garantizar distintos escenarios, se utilizó la clase CargaMasivaFirestore, la cual inserta datos de prueba (transacciones, metas, y categorías) en Firebase.

Estas comprobaciones se hicieron leyendo el flujo de datos y verificando manualmente los resultados en tiempo real.

5.2.2 Pruebas de caja negra

Se realizaron pruebas funcionales (caja negra) para validar que las pantallas y flujos principales funcionen como se espera. Algunas de las pruebas realizadas fueron:

- **Inicio de sesión con email y Google**: se probó iniciar sesión con distintas cuentas válidas e inválidas.
- **Registro y recuperación de contraseña**: comprobando que el email de recuperación llega correctamente.
- **Añadir, editar y eliminar transacciones**: validando que se reflejen en la base de datos y actualicen el dinero.
- **Crear y completar metas de ahorro**: confirmando que el progreso se actualiza correctamente.

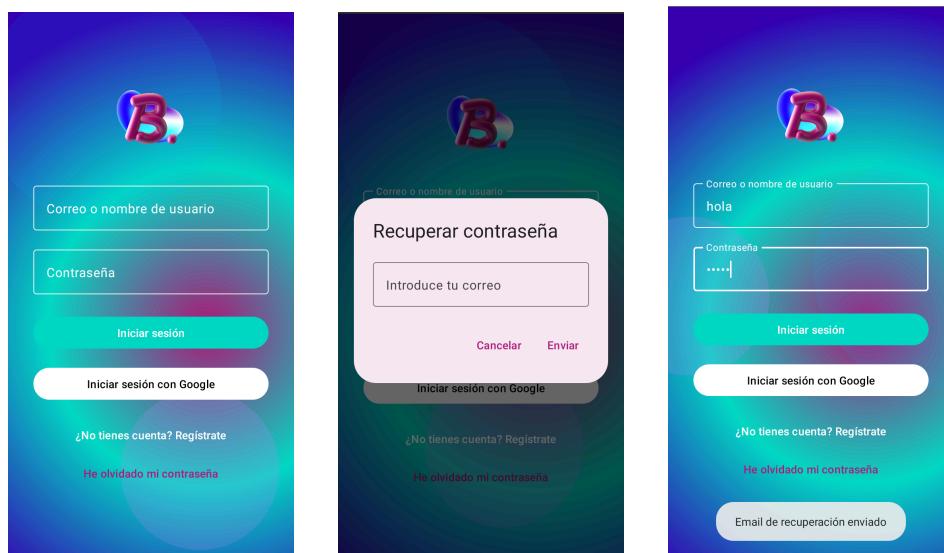
- **Uso del chatbot:** se hicieron preguntas simuladas al asistente financiero para comprobar que responde con contexto.
- **Generar PDF y enviarlo por correo:** verificando que se adjunta y contiene la información correcta.

5.3 Pruebas funcionales en dispositivos

Empezamos con el SplashScreen que se inicia y dura unos 3 segundos, si la sesión está iniciada en el dispositivo cargará directamente la pantalla de inicio, sino la de login



En el login podemos iniciar sesión normal, pedir recuperar la contraseña mediante un email de recuperación, iniciar sesión con Google o irnos a la pantalla de Registro.



Reset your password for project-269475319839 Recibidos xnoreply@proyectodef-f7224.firebaseio.com
para mí ▾19:21 (hace 24 minutos) ☆

Hello,

Follow this link to reset your project-269475319839 password for your celiara55555@gmail.com account.https://proyectodef-f7224.firebaseio.com/_auth/action?mode=resetPassword&oobCode=pXy1ZskH9ZGxhrEJmudx6wfG-IwecRyp2jx6kLONo34AAAGXHQ_RIA&apiKey=AlzaSyDamICcl7qVnupWmP24suRfx9_jL2Wzkw0&lang=en

If you didn't ask to reset your password, you can ignore this email.

Thanks,

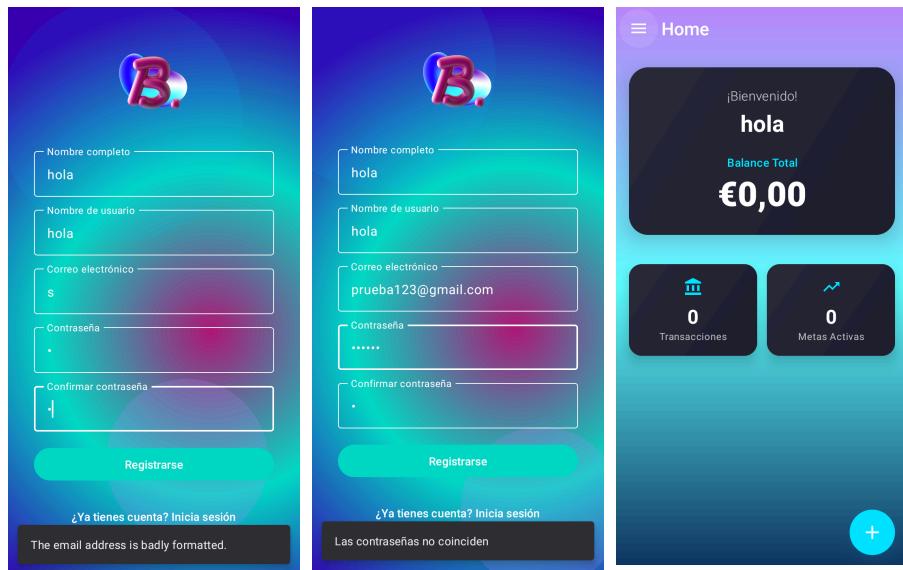
Your project-269475319839 team

En registro nos aparecerá un formulario con todas las validaciones correspondientes, también podemos ir a Inicio de sesión de nuevo.

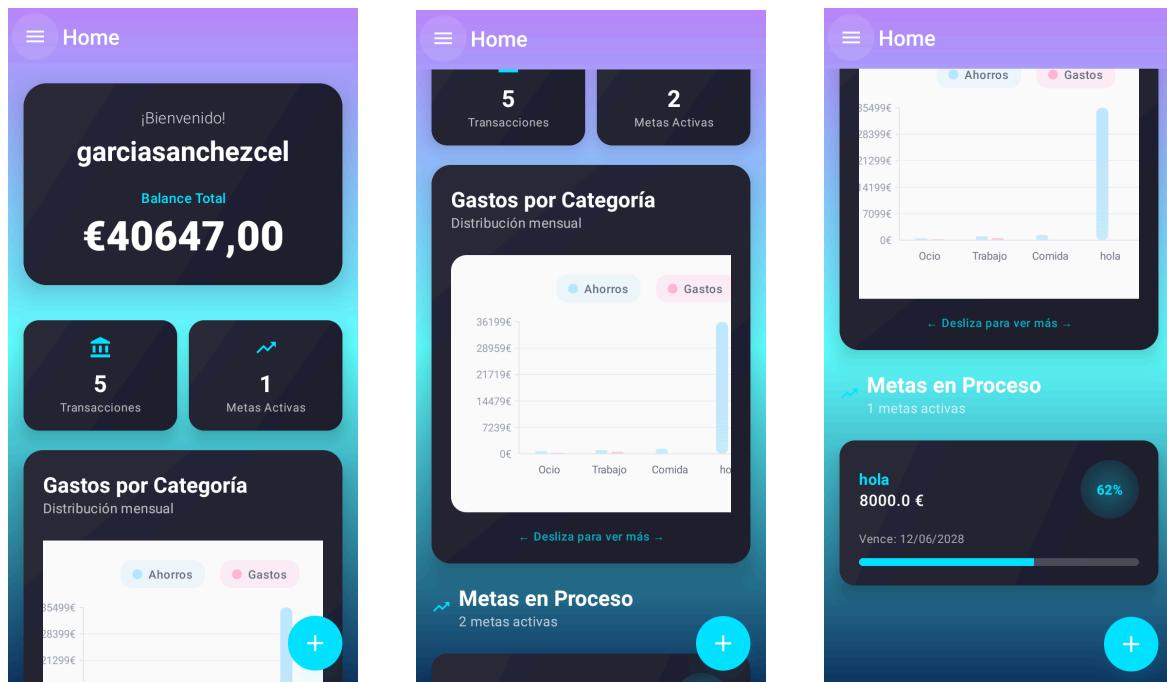
Nombre completo	Nombre de usuario	Correo electrónico	Contraseña	Confirmar contraseña
nd	x	s	•	•
hola	hola	s	•	•
hola	hola	prueba123@gmail.com	•	•

Validation messages:

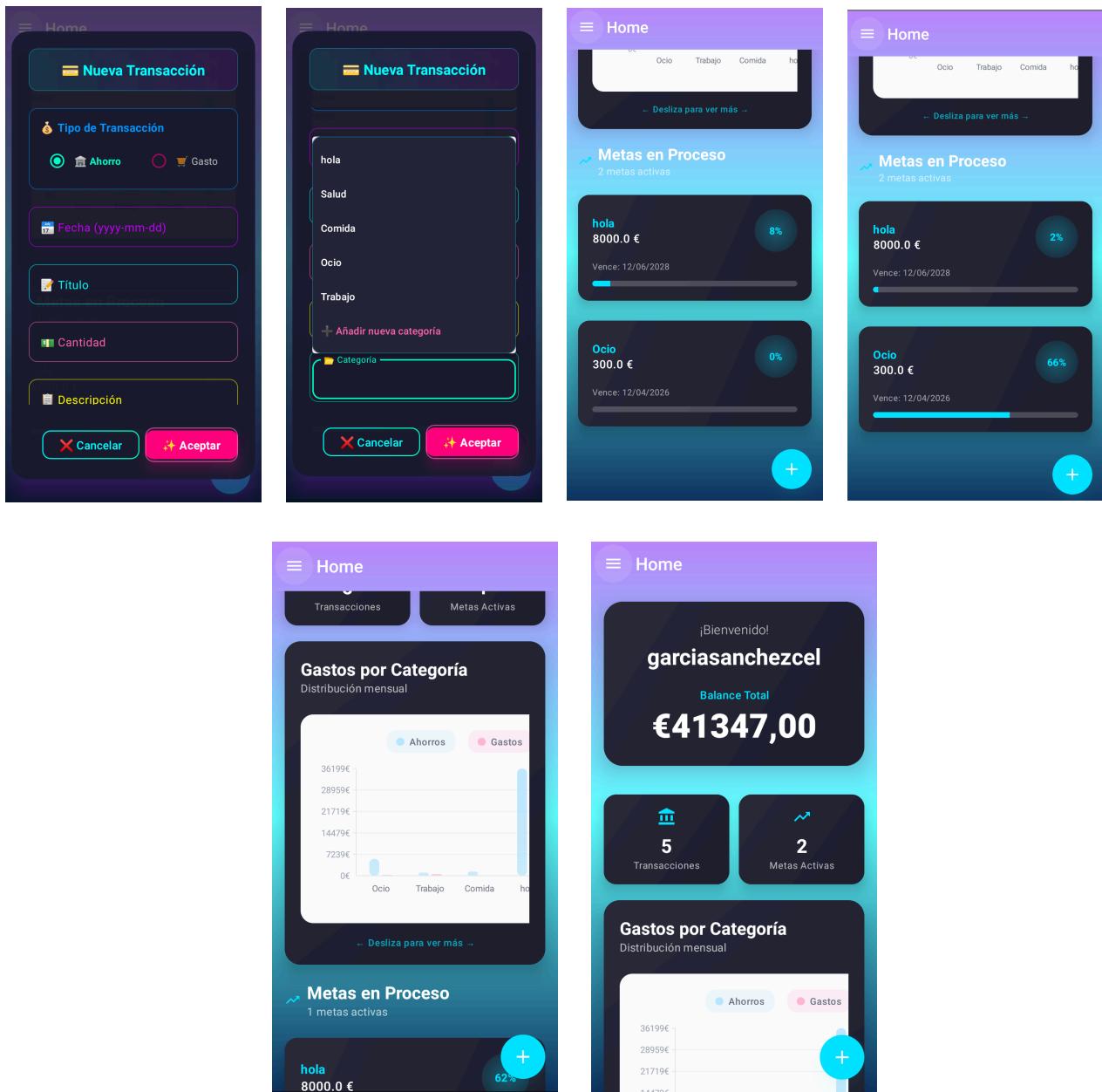
- Screenshot 2: "El nombre de usuario debe tener al menos 3 caracteres y solo letras o números"
- Screenshot 3: "Todos los campos son obligatorios"
- Screenshot 4: "La contraseña es demasiado débil"



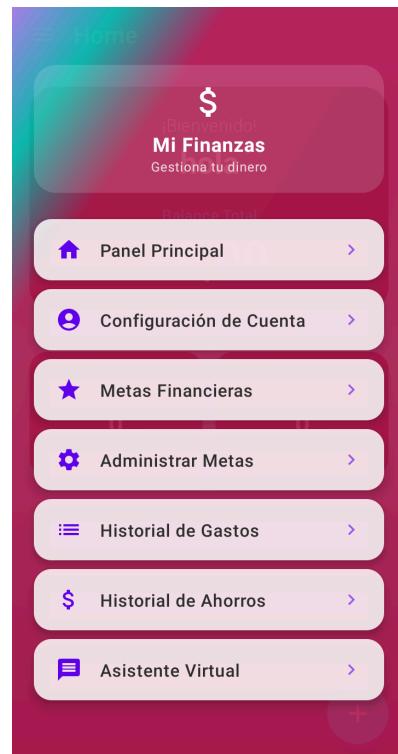
Una vez en el inicio podemos hacer varias cosas, primero vemos nuestro nombre de usuario y nuestro saldo total, también aparece las transacciones registradas y metas activas actual en esta pantalla, un gráfico deslizable con los gastos frente a los ahorros de cada categoría, un botón para añadir transacciones, el desplegable del menú y las metas activas de ahorro y sus progresos.



Podemos añadir un transacción y esta se verá reflejada en el gráfico y en la cantidad total del usuario, también afectará al progreso de las metas que tengamos relacionadas, muestro a continuación todo este funcionamiento. Por supuesto todos los campos tienen validaciones.



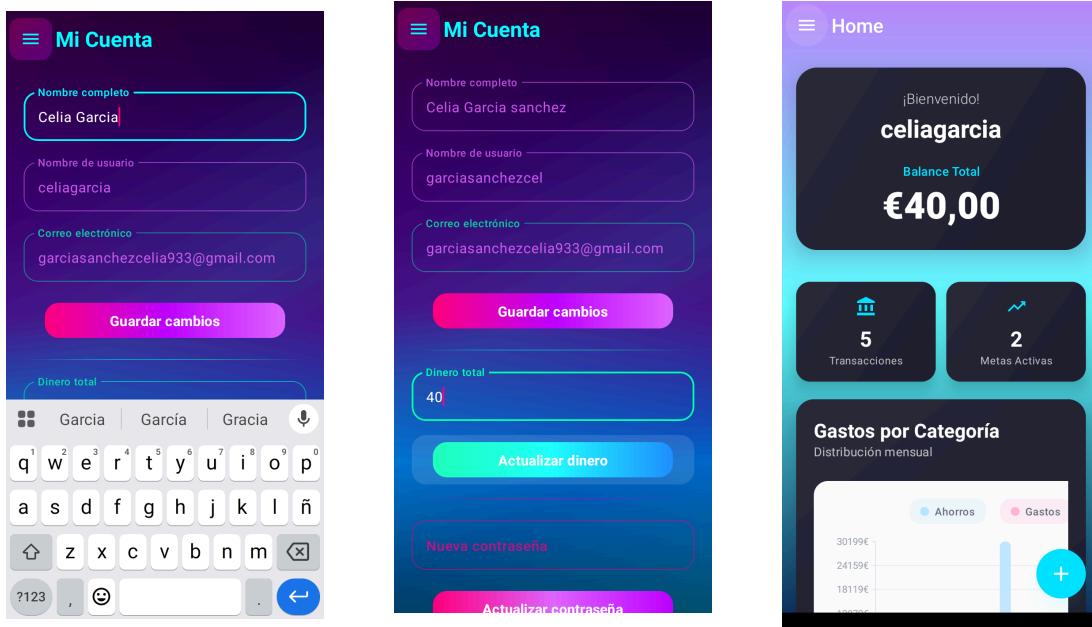
Si abrimos el menú podemos navegar por las siguientes pestañas:



Nos vamos a configuración de la cuenta donde podemos hacer varias cosas: Cambiar nombre completo y nombre de usuario (el correo no es posible cambiarlo), guardar los cambios, cambiar el saldo actual (pensado sobre todo para las personas que empiezan con la app y ya tienen una cantidad ahorrada), crear un pdf con todo el resumen financiero del último mes, resetear todos los datos de la cuenta, salir sesión y eliminar la cuenta.

The screenshots show the 'Mi Cuenta' configuration screen with the following details:

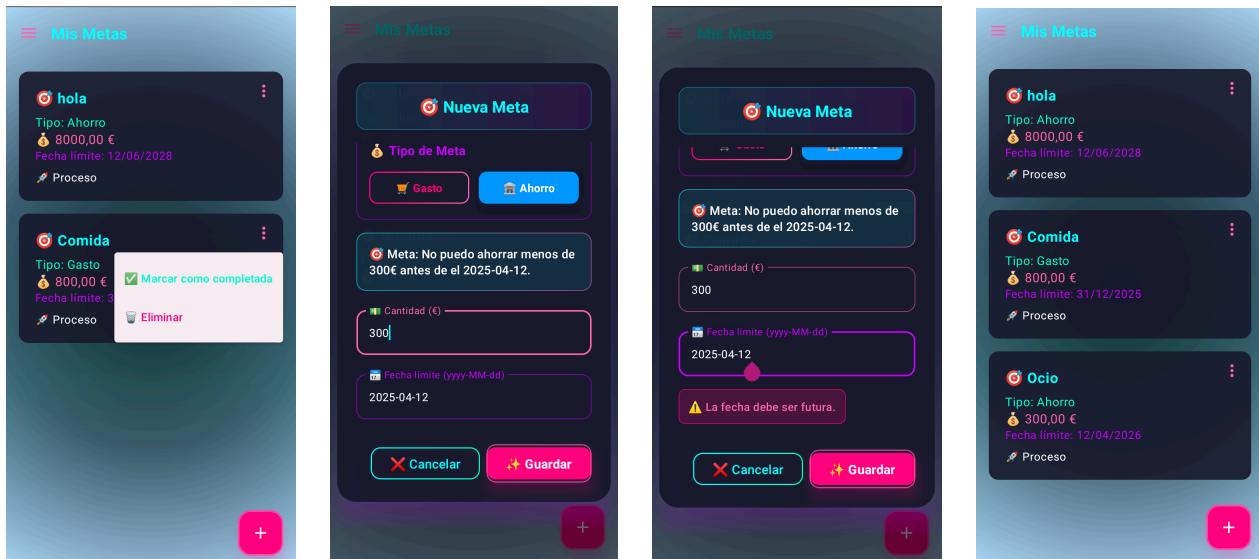
- Screenshot 1:** Shows fields for 'Nombre completo' (Celia Garcia Sanchez), 'Nombre de usuario' (garciасanchezcel), and 'Correo electrónico' (garciasanchezcelia933@gmail.com). Buttons include 'Actualizar dinero' (41347.0), 'Guardar cambios', and 'Actualizar contraseña'.
- Screenshot 2:** Shows the same fields. Buttons include 'Actualizar dinero' (41347.0), 'Actualizar contraseña', 'Crear PDF resumen', 'Resetear cuenta', 'Cerrar sesión', and 'Eliminar cuenta'.
- Screenshot 3:** Shows the same fields. Buttons include 'Actualizar dinero' (40.0), 'Guardar cambios', 'Actualizar contraseña', and 'Actualizar dinero'.
- Screenshot 4:** Shows the same fields. Buttons include 'Actualizar dinero' (40247.0), 'Actualizar contraseña', and 'Actualizar contraseña'.



The image shows three steps of a document printing process:

- Elige una impresora (Select a printer):** Set to 1 copy, carta paper size. The report title is 'Reporte Financiero Mensual'. It shows user information (Name: Celia Garcia, User: colegio, Email: garciaseanchezcelia933@gmail.com) and a summary table with total income (\$97,264.87), total expenses (\$91,154.84), and a balance of \$6,110.03. It also lists financial goals.
- Elige una impresora (Select a printer):** Set to 1 copy, carta paper size. This step shows the detailed transaction history from the previous screen.
- Elige una impresora (Select a printer):** Set to 1 copy, carta paper size. This step shows the detailed transaction history from the previous screen, followed by a note about generating reports automatically and a footer note.

En la pestaña de metas financieras podemos añadir nuevas metas y ver las que tenemos actualmente:



Nos vamos a la pestaña de gestión de metas, donde podemos ver todas nuestras metas completadas o expiradas.



Pasamos a las pestañas de transacciones, como puede ser ahorros o gastos, aquí podemos eliminarlas o ver sus detalles.

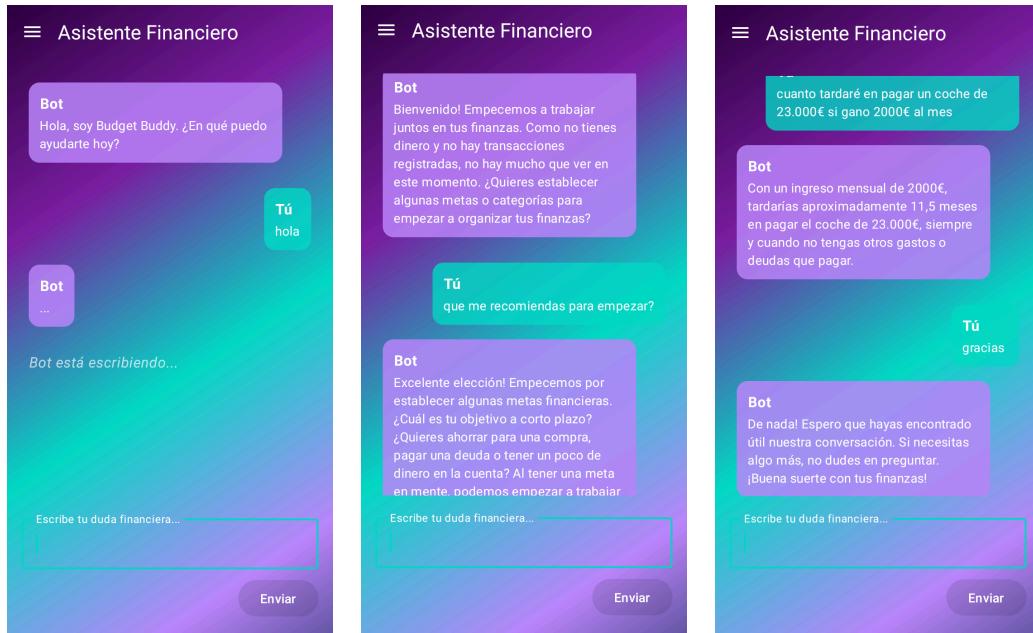
The image displays a 2x4 grid of screenshots from a mobile application for managing finances. The top row shows:

- Transacciones de Gasto:** A list of five transactions: Gasto 7 (\$11,00€), Gasto 0 (\$94,38€), Gasto 4 (\$92,39€), Gasto 1 (\$31,84€), and Gasto 2 (\$67,33€).
- Transacciones de Ahorro:** A list of five transactions: hhuuu (\$200,00€), jjj (\$200,00€), hdjdjd (\$5,000,00€), MMM (\$690,00€), and jjj (\$679,00€).
- Transacciones de Ahorro:** A list of five transactions: hhuuu (\$200,00€), hu (\$6,000,00€), jjj (\$200,00€), hdjdjd (\$5,000,00€), and MMM (\$690,00€). A modal overlay titled 'Eliminar transacción?' (Delete transaction?) is shown over the second transaction.
- Transacciones de Gasto:** A list of three transactions: Gasto 7 (\$11,00€), Gasto 0 (\$94,38€), and Gasto 2 (\$67,33€). A modal overlay titled '¿Eliminar transacción?' (Delete transaction?) is shown over the first transaction, with buttons 'Cancelar' (Cancel) and 'Eliminar' (Delete).

The bottom row shows:

- Transacciones de Ahorro:** A list of five transactions: hhuuu (\$200,00€), hhjh (\$300,00€), hu (\$6,000,00€), jjj (\$200,00€), and hdjdjd (\$5,000,00€).
- Transacciones de Gasto:** A modal overlay titled 'Detalles de la transacción' (Transaction details) for Gasto 0. It shows the title 'Gasto 0', category 'Comida', amount '\$94,38€', date '29/04/2025', and a 'Cerrar' (Close) button.

Y por último encontramos el chatbot, que te ayudará a gestionar tus finanzas y dudas que tengas sobre tus ahorros y metas.



5.4 Pruebas de rendimiento

Se evaluó el rendimiento de la aplicación desarrollada en Android con Jetpack Compose, centrándose en los tiempos de carga, fluidez de navegación y uso de recursos. A continuación se presentan los principales resultados:

- **Tiempo de carga inicial:** La pantalla principal (HomeScreen) se carga completamente en menos de **2 segundos** en dispositivos de gama media.
- **Fluidez:** Gracias al uso de LazyColumn y navegación con NavController, las transiciones entre pantallas son rápidas y sin retardos perceptibles. Las animaciones y componentes como gráficos se renderizan sin tirones visibles.
- **Consumo de recursos:** La aplicación mantiene un uso moderado de CPU y memoria durante la navegación y carga de datos, sin generar bloqueos ni cierres inesperados.
- **Rendimiento en animaciones:** Se integran animaciones mediante AnimatedVisibility y animate*AsState sin impacto negativo en el rendimiento.
- **Dispositivos de prueba:** Las pruebas se realizaron en emuladores y dispositivos físicos con Android 10 y 11, incluyendo modelos con 3 GB de RAM.

5.5 Registro y análisis de errores

Durante el proceso de desarrollo y pruebas, se utilizó **Logcat** como herramienta principal para el monitoreo de eventos, advertencias y errores en tiempo real. Se implementaron registros personalizados (Log.d, Log.e) para facilitar la detección de comportamientos anómalos en puntos críticos de la aplicación, tales como:

- Inicio de sesión y autenticación
- Acceso a Firestore y manejo de datos
- Navegación entre pantallas
- Generación de archivos PDF y carga de gráficos

Además, se emplearon bloques try-catch para capturar excepciones y prevenir cierres inesperados. Los errores detectados fueron categorizados y corregidos según su severidad. Algunos ejemplos incluyen:

- **NullPointerException** al acceder a datos no inicializados
- **FirebaseNetworkException** por fallos en la conexión
- **IllegalStateException** en composable no sincronizados con el estado de navegación

No se registraron **crashes críticos** en la versión final durante las pruebas funcionales y de rendimiento.

6. Conclusiones

6.1 Valoración del proyecto

La realización de este proyecto ha supuesto un gran reto tanto a nivel técnico como personal. A lo largo del desarrollo de esta aplicación de gestión financiera se han puesto en práctica conocimientos clave adquiridos durante el ciclo formativo, incluyendo diseño de interfaces, uso de bases de datos en la nube (Firebase), integración de servicios de autenticación (como Google), gestión de estados con ViewModel y funcionalidades avanzadas como generación de PDFs o integración de un chatbot.

Una de las mayores satisfacciones ha sido comprobar que el sistema es funcional, estable y útil, permitiendo al usuario controlar sus ingresos, gastos, metas de ahorro, generar informes y recibir asistencia inteligente. Además, el diseño visual ha sido cuidado para ofrecer una experiencia moderna y atractiva.

También se han superado dificultades, como la adaptación del sistema de navegación, la gestión de permisos o el manejo asíncrono de datos en Firebase, lo que ha reforzado la capacidad de resolución de problemas y autonomía en el desarrollo.

En resumen, el resultado ha sido muy positivo, cumpliendo con los objetivos marcados inicialmente y demostrando una evolución real en el dominio de herramientas y buenas prácticas en desarrollo de aplicaciones móviles. Esta experiencia ha sido una base sólida tanto para futuros proyectos personales como para afrontar el entorno profesional.

6.2 Dificultades encontradas

Durante el desarrollo de la aplicación se presentaron diversas dificultades que pusieron a prueba tanto mis conocimientos técnicos como mi capacidad de organización y resolución de problemas.

Una de las primeras complicaciones fue la **integración del sistema de autenticación con Google**. Aunque Firebase facilita muchas herramientas, entender el flujo completo y gestionar correctamente la identidad del usuario en distintos puntos de la app supuso un proceso de prueba y error.

Otra dificultad importante fue la **gestión asíncrona de datos en Firebase**, especialmente al trabajar con múltiples colecciones relacionadas como usuarios, transacciones, metas y categorías. Fue necesario aplicar corrutinas y manejar estados correctamente con LiveData o StateFlow para evitar bloqueos o cargas incorrectas en la interfaz.

También resultó complejo implementar el **chatbot financiero contextual**, ya que requería recopilar información del usuario en tiempo real (ingresos, gastos, metas) y construir un resumen coherente que sirviera como entrada para generar respuestas inteligentes.

Por otro lado, la generación del **PDF resumen financiero** supuso un reto técnico adicional, tanto en el diseño del documento como en su exportación y envío por correo desde el propio dispositivo.

A nivel visual, el diseño de una interfaz atractiva, funcional y adaptada a distintos tamaños de pantalla exigió invertir tiempo en ajustes finos de estilo, animaciones y composición.

Finalmente, uno de los mayores desafíos fue la **gestión del tiempo**, ya que al tratarse de un proyecto extenso desarrollado en solitario, fue necesario priorizar funcionalidades, mantener una estructura ordenada del código y avanzar con disciplina en cada fase.

6.3 Mejoras y ampliaciones futuras

A pesar de que el proyecto ha alcanzado un alto grado de funcionalidad, existen diversas áreas en las que se pueden introducir mejoras y nuevas funcionalidades para enriquecer aún más la experiencia del usuario:

- **Mejorar la inteligencia del chatbot financiero:** Aunque actualmente responde en base al contexto financiero del usuario, en el futuro podría conectarse a una API externa de asesoría financiera o incluso incorporar aprendizaje automático para personalizar aún más sus respuestas.
- **Añadir estadísticas más detalladas:** Ampliar los gráficos actuales para mostrar tendencias a lo largo del tiempo, comparativas entre meses, o promedios de gasto y ahorro por categoría, lo que permitiría un análisis más profundo.
- **Notificaciones inteligentes:** Implementar alertas automáticas cuando el usuario esté cerca de alcanzar una meta, o si supera un límite de gasto, ayudando así a mantener sus objetivos financieros.
- **Soporte multilingüe:** Internacionalizar la aplicación para que esté disponible en varios idiomas y así ampliar su público objetivo.
- **Integración bancaria:** Permitir la conexión con cuentas bancarias reales mediante APIs (como PSD2) para registrar automáticamente ingresos y gastos, lo que automatizaría gran parte de la gestión.
- **Mejora de la interfaz de usuario (UI/UX):** Aunque el diseño actual es funcional y atractivo, se podría seguir puliendo con animaciones más suaves, temas personalizados por el usuario o modos claro/oscuro.
- **Gestión de usuarios y roles:** Ampliar la lógica de usuarios para permitir cuentas familiares o compartidas, donde varias personas puedan planificar y consultar metas y finanzas comunes.
- **Funcionalidad offline:** Permitir el uso básico de la app sin conexión a internet, sincronizando los datos automáticamente cuando el dispositivo vuelve a estar online.

7. Bibliografía y referencias

7.1 Fuentes utilizadas

se han consultado diversas fuentes para la comprensión, implementación y resolución de dudas técnicas, entre las que destacan:

- **Documentación oficial de Android**
<https://developer.android.com>
Utilizada para comprender el funcionamiento de Jetpack Compose, navegación, gestión del ciclo de vida y uso de componentes modernos de UI.
- **Firebase Documentation**
<https://firebase.google.com/docs>
Fuente principal para implementar la autenticación, almacenamiento en Firestore, actualización de usuarios y manejo de contraseñas.
- **Kotlin Language Documentation**
<https://kotlinlang.org/docs/home.html>
Referencia esencial para el uso correcto del lenguaje Kotlin y sus características más relevantes.
- **Material Design 3**
<https://m3.material.io>
Guía de estilos para implementar interfaces modernas, coherentes y accesibles usando Material 3 y Compose.
- **GitHub y Stack Overflow**
<https://github.com> / <https://stackoverflow.com>
Se consultaron para resolver problemas específicos, entender patrones de arquitectura y validar buenas prácticas.
- **OpenAI Documentation**
<https://platform.openai.com/docs>
Utilizada para comprender el funcionamiento de los modelos de lenguaje aplicados en el chatbot financiero, incluyendo el uso de contexto.
- **Canva y draw.io**
Utilizados como herramientas de apoyo visual para diseñar diagramas UML, diagramas de flujo y mejorar la presentación del proyecto.

Anexos

Anexo A: Manual de instalación

Este apartado detalla los pasos necesarios para instalar y ejecutar correctamente la aplicación en un entorno de desarrollo local mediante **Android Studio**.

Requisitos previos

- **Android Studio** versión **Hedgehog | Giraffe o superior**
 - **Java Development Kit (JDK)** 17 o superior
 - **Gradle** (incluido en Android Studio)
 - **Conexión a internet** (para sincronizar dependencias)
 - Cuenta de **Firebase** (para conexión con el backend)
 - Mínimo **4 GB de RAM** recomendados
-

Pasos de instalación

Clonar o descargar el proyecto

```
git clone https://github.com/celiagarciaa05/proyectodef
```

1. Abrir el proyecto en Android Studio

- Abrir Android Studio
- Seleccionar “Open an existing project”
- Navegar a la carpeta descargada

2. Sincronizar el proyecto

- Android Studio detectará el archivo build.gradle y pedirá sincronizar.
- Clic en "Sync Now" o esperar que sincronice automáticamente.

3. Configurar Firebase

- Crear un proyecto en Firebase.

Asegurarse de que el archivo AndroidManifest.xml incluya:

```
<uses-permission android:name="android.permission.INTERNET" />
```

4. Ejecutar la aplicación

- Elegir un emulador o dispositivo físico.
- Clic en el botón de “Run” o usar Shift + F10. Agregar una aplicación Android con el mismo applicationId que en el proyecto.

Descargar el archivo google-services.json y colocarlo en:

app/google-services.json

5. Habilitar los servicios de Firebase

- Activar: Authentication (correo y Google), Firestore Database y Storage si se usa para PDF.

Dar permisos en el AndroidManifest

Anexo B: Manual de usuario

Este manual está destinado a guiar al usuario a través de las funcionalidades principales de la aplicación desarrollada. A continuación se describen las pantallas más importantes y sus funcionalidades correspondientes.

Pantalla de Inicio (Splash)

Al iniciar la app, se muestra un logo animado mientras se verifica si el usuario está autenticado.

- Si el usuario ya ha iniciado sesión, se redirige automáticamente al **Home**.
 - Si no, se redirige a la pantalla de **Login**.
-

Login

Permite al usuario acceder a la app con su cuenta.

Opciones disponibles:

- Iniciar sesión con correo y contraseña.
 - Iniciar sesión con **Google**.
 - Restablecer contraseña vía correo electrónico.
 - Ir a la pantalla de registro si no tiene cuenta.
-

Registro

Formulario para crear una nueva cuenta.

Campos requeridos:

- Nombre completo
 - Nombre de usuario
 - Correo electrónico
 - Contraseña y su confirmación
-

Home

Pantalla principal tras el login.

Opciones:

- Ver resumen del dinero total.
 - Acceder a **Metas**, **Transacciones**, **Mi cuenta**, **Chatbot**, etc.
 - Visualizar gráfico de progreso y estadísticas.
 - Añadir transacciones o metas rápidamente.
-

Transacciones

Muestra una lista filtrada de transacciones por tipo (Gasto o Ahorro).

Funciones:

- Ver detalles de cada transacción.
 - Eliminar transacciones.
 - Añadir nuevas transacciones con categoría, título, cantidad y descripción.
 - Las transacciones afectan al cálculo del dinero total y del progreso de metas automáticamente.
-

Metas

Lista de metas creadas por el usuario.

Cada meta contiene:

- Categoría
- Tipo (Gasto o Ahorro)
- Cantidad objetivo
- Fecha límite
- Estado (Proceso o Completada)
- Progreso (%)

Se pueden **añadir, marcar como completadas o eliminar** metas. El progreso se calcula automáticamente en función de las transacciones relacionadas.

Chatbot

Pantalla con un chatbot asistente financiero con contexto.

Permite:

- Hacer preguntas sobre tus finanzas.
 - Obtener consejos financieros personalizados.
 - El chatbot conoce tus metas y transacciones recientes.
-

Mi Cuenta

Permite al usuario gestionar su perfil.

Funciones disponibles:

- Editar nombre completo y nombre de usuario.
 - Ver y cambiar el correo (solo visible).
 - Cambiar contraseña.
 - Actualizar el dinero total manualmente.
 - Descargar un resumen financiero en PDF (últimos 30 días).
 - Resetear la cuenta (borrar categorías y transacciones).
 - Cerrar sesión o eliminar cuenta completamente.
-

Hasta aquí llega la documentación de este proyecto. Ha sido desarrollado con dedicación, aprendiendo y superando desafíos en cada etapa.

Agradezco el tiempo dedicado a revisar este trabajo. Espero que el resultado cumpla con las expectativas y sirva como muestra de mi compromiso y crecimiento en el desarrollo de software.

Gracias por leer, Celia.