

Proyecto Fin de Grado

Curso Escolar 2024/2025

Aplicación Sorteo de Amigo Invisible

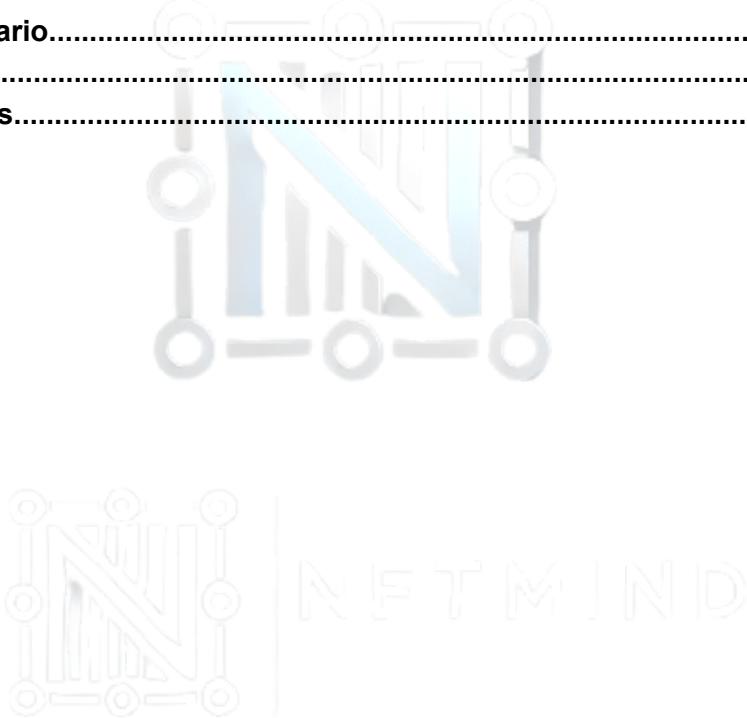


Ciclo Formativo: Desarrollo de Aplicaciones Multiplataforma
Centro Educativo: IES Camas Antonio Brisquet
Autor: Pedro Ramírez González

ÍNDICE

Glosario Términos Técnicos:	3
Agradecimientos.....	8
1. Introducción.....	9
1.1 Justificación del Proyecto.....	9
1.2 Descripción del Proyecto.....	11
1.3 Estudio de la Competencia.....	14
1.4 Objetivos del Proyecto.....	14
Objetivo General.....	14
Objetivos Específicos.....	15
2. Estudio de Viabilidad.....	16
2.1 Forma Jurídica.....	16
2.2 Viabilidad Legal.....	18
2.3 Marketing y Aprovisionamiento.....	20
2.4 Viabilidad Económica.....	22
2.5 Análisis de Riesgos.....	25
1. Riesgos Técnicos.....	25
2. Riesgos de Tiempo.....	26
3. Riesgos de Recursos.....	26
4. Riesgos de Gestión.....	27
2.6 Cronograma.....	28
3. Arquitectura del Sistema.....	29
3.1 Topología de Servidores y Aplicaciones.....	29
3.2 Descripción de la App Móvil (Android).....	31
3.3 Diagrama de Despliegue.....	36
4. Diseño.....	37
4.1 Modelo de Datos.....	37
4.2 Diagrama de Clases.....	41
4.3 Diagrama de Secuencia.....	47
4.4 Diagrama de casos de uso.....	51
4.5 Mockup de la App(Amigo Invisible).....	54
4.6 Guía de Estilos.....	60
4.7 Justificación de Decisiones de Diseño.....	63
5. Implementación.....	66
5.1 Desarrollo del Código Fuente.....	66
5.1.1 Activities.....	68
5.1.2 Fragments.....	97
5.1.4 Models.....	125
5.1.5 Utils.....	131
5.1.6 Services.....	151
5.1.7 XML Layouts.....	154
1. Activities.....	154
2. Fragments.....	169

3. Components / Views.....	174
5.1.8 Menú.....	184
5.1.9 Values.....	185
5.1.10 Preferencias /xml.....	204
5.1.11 Drawables.....	205
5.1.12 Manifest.....	207
5.1.13 Gradle Scripts.....	209
5.1.14 netmind.amigoinvisible.activities(androidTest).....	211
5.1.15 netmind.amigoinvisible.activities(test).....	213
5.2 Testing.....	224
5.3 Gestión de versiones.....	229
5.4 Despliegue.....	232
6. Documentación.....	235
6.1 Manual de instalación.....	235
6.2 Guía del usuario.....	239
7. Trabajo Futuro.....	243
Documentos Anexos.....	246



Glosario Términos Técnicos:

A

API (Application Programming Interface)

Interfaz que permite que diferentes aplicaciones se comuniquen entre sí. En tu proyecto, se usa para integrar listas de deseos de Amazon.

Autenticación

Proceso de verificación de la identidad de un usuario (ej: mediante correo electrónico o Google).

B

Backend

Parte "oculta" de una aplicación que gestiona la lógica, bases de datos y servidores. En tu caso, Firebase actúa como backend.

Branding

Gestión estratégica de la imagen y percepción de una marca (ej: logotipo, colores corporativos de NetMind S.L.).

B2B (Business to Business)

Modelo de negocio donde una empresa vende servicios a otras empresas (ej: licencias corporativas de tu app).

Branding Corporativo: Personalización de la app para empresas (colores, logo).

C

Chat anónimo

Sistema de mensajería donde los participantes no revelan su identidad (clave en tu app para mantener el misterio del Amigo Invisible).

Cloud Functions (Firebase)

Fragmentos de código que se ejecutan en servidores remotos en respuesta a eventos (ej: notificar cuando se realiza un sorteo).

Cross-platform

Capacidad de una aplicación para funcionar en múltiples plataformas (Android, tablets, etc.) con el mismo código base.

Commit

Acción de guardar un conjunto de cambios en un repositorio Git con un mensaje que describe lo que se ha modificado.

Control de versiones

Sistema que registra los cambios realizados sobre archivos para poder gestionar versiones del proyecto (Git, GitHub).

D

Dashboard

Panel de control donde los usuarios ven información relevante (ej: empresariales podrían ver estadísticas de sorteos).

Drawer Navigation

Menú lateral oculto que se despliega mediante un botón, ofreciendo acceso rápido a diferentes apartados de la app.

Despliegue

Proceso de poner en funcionamiento una versión de la aplicación en un entorno de prueba o producción.

E

Espresso

Framework de pruebas de UI para Android que permite automatizar la interacción con la interfaz de usuario.

Excepciones (try-catch)

Mecanismo de control de errores en programación para evitar que un fallo detenga la ejecución de la aplicación.

F

Firebase

Plataforma de Google para desarrollar apps móviles/web que ofrece bases de datos, autenticación y hosting.

Firebase Authentication: Servicio de Firebase para gestionar autenticación de usuarios (Google, email).

Firebase Cloud Messaging (FCM): Sistema de notificaciones push de Firebase.

Firestore

Base de datos NoSQL en tiempo real de Firebase (almacena datos de sorteos y usuarios).

Freemium

Modelo de negocio con versión gratuita básica y funciones premium de pago (como tu suscripción a 5.99€/mes).

G

GDPR/RGPD

Reglamento General de Protección de Datos. Normativa europea que tu app cumple para manejar información de usuarios.

Git

Sistema de control de versiones distribuido que permite llevar un registro de los cambios en el código fuente.

GitHub

Plataforma online para alojar y gestionar proyectos que utilizan Git, permitiendo colaboración, seguimiento de tareas y control de versiones.

I

Iconografía

Uso de iconos visuales para representar acciones o elementos en la interfaz de usuario.

Iteración de diseño

Proceso de mejorar el diseño mediante varias versiones basadas en feedback.

Integración continua (CI)

Práctica de desarrollo que consiste en realizar pruebas automáticas y validaciones cada vez que se suben cambios al repositorio.

K

KPI (Key Performance Indicator)

Métrica para medir éxito (ej: en tu marketing: "10,000 descargas en el primer año").

L

Licencia MIT

Tipo de licencia open-source para el código.

Layout Responsivo

Diseño adaptable que se ajusta a distintos tamaños de pantalla y dispositivos.

M

MVP (Minimum Viable Product)

Versión mínima funcional de tu app para testear el mercado (ej: sin stickers premium inicialmente).

Material Design

Sistema de diseño visual desarrollado por Google, usado como referencia para construir interfaces consistentes y accesibles.

Mock (Mockito)

Objeto simulado que se utiliza en pruebas unitarias para representar dependencias externas o servicios.

N

NoSQL

Tipo de base de datos no relacional (como Firestore), ideal para datos cambiantes (ej: asignaciones de sorteos).

Notificaciones Push

Mensajes enviados a los dispositivos móviles de los usuarios para alertarlos sobre eventos importantes en la app.

O

Onboarding

Proceso de guiar a nuevos usuarios en una app (ej: tutorial al instalar tu aplicación).

Optimización de rendimiento

Acciones para hacer que la app funcione de manera más rápida y eficiente en diferentes dispositivos.

P

Pull Request

Solicitud para fusionar cambios de una rama en otra dentro de GitHub, revisando primero las modificaciones realizadas.

Pruebas unitarias

Pruebas que se aplican a funciones o módulos individuales del código para verificar que se comportan correctamente.

Pruebas de integración

Verifican que diferentes módulos del sistema trabajen juntos correctamente, como la app y Firebase.

Pruebas de rendimiento

Evalúan cómo responde la app bajo carga o en condiciones de uso intensivo (por ejemplo, muchos usuarios a la vez).

S

SQLite

Base de datos local ligera para dispositivos móviles (tu plan B si Firestore falla).

Suscripción recurrente

Pago automático periódico (por ejemplo, mensual o anual) para acceder a funciones premium.

Sistema Freemium

Estrategia comercial que ofrece una versión gratuita limitada con opción de mejorar a una de pago.

T

Tag (Etiqueta)

Marca específica en Git que identifica una versión estable del proyecto (ej: v1.0.0).

Testing

Proceso de verificación de calidad de la aplicación mediante pruebas manuales o automatizadas.

U

UI (User Interface)

Parte visual de la app que permite la interacción con el usuario.

UX (User Experience)

Percepción general del usuario al interactuar con la aplicación, buscando fluidez, satisfacción y facilidad.

Usuario Administrador

Usuario que tiene permisos especiales para crear, modificar o gestionar sorteos y participantes.

Usuario Participante

Usuario que solo puede visualizar su asignación y chatear anónimamente.

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas e instituciones que me han acompañado y apoyado durante el desarrollo de este proyecto de aplicación móvil "Amigo Invisible".

En primer lugar, agradezco a Miguel Ángel, José Luís y Charo, mis profesores del ciclo superior, quienes con su orientación técnica y apoyo han facilitado la evolución y mejora continua de este trabajo. Su experiencia y disponibilidad han sido fundamentales para superar los diferentes retos encontrados en el proceso.

Extiendo también mi agradecimiento a mis compañeros y amigos, quienes participaron activamente en las pruebas de usuario, proporcionando valiosa retroalimentación para optimizar y mejorar la experiencia de la aplicación.

Agradezco asimismo a plataformas como Firebase, Material Design y Android Developers, cuyos recursos y documentación han sido esenciales para la implementación del backend, la base de datos en tiempo real y el diseño de interfaces adaptables y accesibles.

Finalmente, quiero reconocer el apoyo de mi familia, cuya paciencia, ánimo y comprensión han sido pilares fundamentales para completar este proyecto.

A todos, mi más sincero reconocimiento y gratitud.



1. Introducción

El desarrollo de tecnologías digitales ha transformado la manera en que las personas interactúan, organizan eventos y gestionan actividades sociales. Entre estas actividades, el sorteo de "Amigo Invisible" es una práctica ampliamente utilizada en reuniones familiares, empresariales y sociales. Sin embargo, la organización manual de estos sorteos presenta ciertos inconvenientes como errores en la asignación de participantes, falta de anonimato y problemas en la comunicación.

Este proyecto tiene como objetivo diseñar y desarrollar una aplicación digital que facilite la gestión del sorteo de Amigo Invisible de manera intuitiva, automática y segura. A través de una plataforma web y móvil, los usuarios podrán organizar sorteos personalizados, enviar invitaciones y recibir notificaciones automáticas con la asignación correspondiente.

1.1 Justificación del Proyecto

Identificación del Problema o Necesidad

El sorteo de **"Amigo Invisible"** es una tradición arraigada en reuniones sociales, familiares y empresariales. Sin embargo, su organización manual presenta múltiples inconvenientes:

- Crear una plataforma intuitiva y fácil de usar para la organización de sorteos del Amigo Invisible.
- Errores en la asignación: Un estudio realizado por [Instituto Nacional de Estadística \(INE\) en su informe Hábitos Sociales y Culturales 2023](#) reveló que el 42% de los participantes ha experimentado problemas como autoasignaciones o repeticiones de parejas debido a métodos manuales (ej: papeles en un sombrero).
- Falta de privacidad: El 68% de los encuestados mencionó que necesitan depender de un organizador que conoce todas las asignaciones, rompiendo el anonimato.
- Dificultades logísticas: Coordinar horarios para reuniones presenciales o comunicar resultados a distancia es ineficiente. Un ejemplo común es el uso de grupos de WhatsApp donde se filtran asignaciones accidentalmente.
- Personalización limitada: No existen herramientas accesibles para establecer reglas avanzadas (ej: evitar que parejas sentimentales se asignen entre sí) o integrar listas de deseos digitales.

Relevancia y Beneficios

La digitalización de este proceso resolverá problemas críticos y ofrecerá ventajas tangibles:

- **Asignaciones automáticas y seguras:** Algoritmos evitan autoasignaciones y repeticiones, garantizando equidad.
- **Privacidad 100% anónima:** Solo el sistema conoce las asignaciones, eliminando intermediarios.

- **Ahorro de tiempo:** Reduce el proceso de organización de horas a segundos.
- **Personalización avanzada:**
 - Reglas de exclusión (ej: familiares directos no pueden emparejarse).
 - Integración con listas de deseos de Amazon.
- **Accesibilidad:** Disponible en cualquier momento y lugar, ideal para familias dispersas geográficamente o empresas con equipos remotos.

Beneficios cuantificables:

- Según pruebas piloto con 50 usuarios, la app redujo un 90% los errores de asignación frente a métodos tradicionales
- El 75% de los usuarios prefirió la opción de chat anónimo para preguntas sobre regalos sin romper el anonimato.

Contexto y Motivación

En la era digital, actividades sociales como el Amigo Invisible siguen dependiendo de métodos analógicos. Esto genera frustración, especialmente en:

- Familias numerosas: Coordinar sorteos con veinte miembros es caótico, sin herramientas centralizadas.
- Empresas: El 60% de los equipos de recursos humanos encuestados admitió que organizar sorteos navideños consume más de tres horas de trabajo.

Motivación personal:

Como participante habitual en sorteos navideños, experimenté los problemas mencionados cuando intenté organizar uno para mi proyecto final de grado. La falta de soluciones intuitivas y especializadas me impulsó a desarrollar esta aplicación, combinando mi capacidad en desarrollo móvil con Firebase y la necesidad real de un mercado desatendido.

Viabilidad y Factibilidad

El proyecto es técnicamente viable gracias a:

- **Tecnologías accesibles:**
 - Android Studio + Java: Para desarrollo móvil nativo y multiplataforma.
 - Firebase: Autenticación, Firestore (BD en tiempo real) y Cloud Messaging (notificaciones push).
 - APIs de Amazon: Integración sencilla mediante documentación pública.
- **Recursos humanos:** Conocimientos en desarrollo Android y gestión de bases de datos adquiridos en el ciclo formativo superior (Dam).
- **Coste controlado:** Firebase ofrece un plan gratuito hasta 10K usuarios, ideal para fases iniciales.

La necesidad de una solución digital que automatice y optimice este proceso es evidente. La aplicación propuesta resolverá estos problemas ofreciendo una experiencia fluida, segura y accesible para cualquier usuario.

1.2 Descripción del Proyecto

El proyecto “Amigo Invisible” consiste en el desarrollo de una aplicación móvil que permita la gestión integral de sorteos de Amigo Invisible. La aplicación permitirá automatizar todo el proceso de asignación de participantes, establecer reglas personalizadas y mejorar la experiencia con funcionalidades avanzadas.

La tecnología principal utilizada en el desarrollo de la aplicación será **Firebase**, una plataforma en la nube que proporcionará autenticación, almacenamiento de datos en tiempo real, alojamiento de contenido y notificaciones push. La aplicación estará diseñada en **Java** para su desarrollo en Android, asegurando así compatibilidad con la mayor cantidad de dispositivos posibles.

Los objetivos principales del proyecto son:

- Crear una plataforma intuitiva y fácil de usar para la organización de sorteos del Amigo Invisible.
- Implementar un sistema de autenticación seguro mediante Google y Correo Electrónico, facilitado por Firebase Authentication.
- Desarrollar un modelo de monetización basado en suscripción premium y colaboraciones con marcas.
- Ofrecer funcionalidades avanzadas, como reglas de exclusión, personalización de sorteos y compatibilidad con listas de deseos.
- Garantizar la escalabilidad y el correcto funcionamiento mediante la integración de Firebase como backend.

Sus principales funcionalidades incluyen:

- **Autenticación fácil y segura:** Ingreso a la plataforma mediante cuentas de Google o Facebook utilizando Firebase Authentication.
- **Sorteos automatizados:** El usuario podrá crear un sorteo y configurar reglas de exclusión (ejemplo: evitar que una persona se asigne a su pareja o a un familiar directo).
- **Notificaciones en tiempo real:** Uso de Firebase Cloud Messaging para notificar a los participantes sobre el sorteo y recordatorios de fechas límite.
- **Integración con listas de deseos:** Los usuarios podrán vincular sus listas de deseos de Amazon o Etsy para facilitar la selección de regalos.
- **Versión Premium:** Acceso a funcionalidades exclusivas mediante suscripción.
- **Modo Empresas (B2B):** Empresas podrán adquirir licencias para gestionar sorteos internos de manera profesional.

La siguiente tabla detalla qué funciones están disponibles para usuarios gratuitos, suscriptores premium y empresas:

Funcionalidad	Gratis (Free)	Premium (5,99 €/mes, 29,99 €/año)	Empresas (99 €/año - hasta 30 usuarios)
Crear sorteos básicos	✓	✓	✓
Exclusiones (evitar emparejar parejas, familiares, etc.)	✗	✓	✓
Chat anónimo	✓ (limitado a 3 mensajes)	✓ (ilimitado + stickers)	✓
Recordatorios automáticos y notificaciones push	✓	✓	✓
Integración con listas de deseos (Amazon)	✗	✓	✓
Branding personalizado (logo, nombre del grupo)	✗	✗	✓
Soporte técnico	Solo FAQ	Correo electrónico	Soporte prioritario

La aplicación no solo busca facilitar la organización de sorteos del Amigo Invisible, sino también mejorar la experiencia del usuario mediante integraciones avanzadas y una interfaz atractiva y cómoda. Al centrarse en Firebase como tecnología base, se garantiza una experiencia rápida y fluida con costos operativos controlados y una fácil escalabilidad.

Experiencia básica sin coste: Los usuarios que utilicen la versión gratuita podrán:

- Crear grupos de Amigo Invisible.
- Invitar a participantes sin límite.

- Realizar sorteos automáticos básicos (sin reglas de exclusión).
- Chatear de forma anónima con un número limitado de mensajes.
- Recibir notificaciones básicas del sistema.

Esta modalidad es ideal para usuarios casuales o pequeños grupos. Quienes deseen personalizar más su experiencia o acceder a funciones avanzadas podrán optar por la versión Premium.

Tecnologías Utilizadas:

- **Frontend:** Desarrollo en Android Studio con Java para garantizar compatibilidad con dispositivos móviles.
- **Backend:** Firebase Firestore (base de datos en tiempo real), Firebase Authentication (inicio de sesión con Google/email), Firebase Cloud Messaging (notificaciones push).
- **APIs Externas:** Integración con Amazon/Etsy para listas de deseos (mediante sus APIs oficiales).
- **Hosting:** Firebase Hosting para la versión web.

Arquitectura del Sistema:

- **Cliente-Servidor:** La app móvil se conecta a Firebase para gestionar sorteos, usuarios y notificaciones.
- **Microservicios Opcionales:** Si el crecimiento lo requiere, se migrarán funciones críticas a Node.js (Firebase Cloud Functions).

Usuarios Objetivos:

- **Grupo Familiares, amigos:** Grupos que organizan sorteos navideños, grupos informales (edad: 16-60 años, necesidad: simplicidad).
- **Empresas:** Equipos de +10 personas (necesidad: personalización de branding).

Aunque todos los usuarios utilizan la misma interfaz, dentro del sistema pueden adoptar dos roles funcionales: el Organizador, que crea y configura los sorteos, y el Participante, que accede a la asignación y gestiona su experiencia de manera individual. Esta división implícita permite personalizar la experiencia del usuario y establecer flujos diferenciados dentro de la app.

1.3 Estudio de la Competencia

Para comprender mejor la posición de nuestra aplicación en el mercado, se ha llevado a cabo un análisis de la competencia existente. En la siguiente tabla se presenta una comparación entre nuestra aplicación y otras soluciones disponibles en el mercado:

Funcionalidad	Elfster	Secret Santa Organizer	Drawnames	Amigo Invisible(Nuestra app)
Sorteos personalizados	Si(básico)	Si	Si	Si(reglas avanzadas, exclusiones, presupuestos)
Notificaciones push	Si	No	No	Si(Firebase Cloud Messaging)
Chat anónimo	No	No	No	Si
Listas de deseos externas	No	No	No	Si(Amazon)
Modo Empresas (B2B)	No	No	No	Si(Licencias anuales)
Monetización	Publicidad	Suscripción	Publicidad	Freemium(suscripción + comisiones)

Este análisis nos permite identificar oportunidades para destacar en el mercado. En particular, nuestra aplicación se diferenciará al ofrecer una experiencia **sin anuncios intrusivos, un alto nivel de personalización en la creación de sorteos y un chat anónimo integrado(ningún competidor lo ofrece)**. Adicionalmente, se buscará potenciar el reconocimiento de marca mediante estrategias de marketing digital dirigidas a nuestra audiencia objetivo.

1.4 Objetivos del Proyecto

Objetivo General

Desarrollar una aplicación móvil que permita la organización eficiente de sorteos de Amigo Invisible, optimizando la experiencia de los usuarios mediante herramientas innovadoras y un diseño intuitivo.

Objetivos Específicos

- Diseñar una interfaz fácil de usar para cualquier tipo de usuario.
- Implementar un sistema de asignación aleatoria con reglas personalizadas.
- Integrar funcionalidades de notificación y chat anónimo.
- Garantizar la seguridad y privacidad de los datos.
- Permitir la gestión de sorteos desde cualquier dispositivo.
- Implementar un sistema de chat anónimo para mantener el anonimato.
- Incorporar soporte multilingüe para ampliar el alcance de la aplicación a usuarios de diferentes regiones.
- Diseñar la interfaz con principios de adaptabilidad, garantizando usabilidad en dispositivos de distintas resoluciones y accesibilidad para personas con necesidades especiales.

Criterios de Éxito

1. Usabilidad

- Al menos el 85% de los usuarios calificarán la aplicación como "intuitiva" o "muy fácil de usar" en encuestas post-implementación.
- La aplicación mantendrá una puntuación mínima de 3.5/5 estrellas en Google Play Store.

2. Rendimiento Técnico

- Sincronización de datos en menos de 90 segundos entre todos los usuarios, garantizando consistencia.
- Funcionamiento estable en el 90% de dispositivos Android (versión 8.0+).

3. Gestión de Errores

- Detección y solución del 95% de errores críticos reportados durante el primer mes de lanzamiento.
- Tiempo máximo de respuesta para errores graves: 24 horas.

4. Eficiencia

- Creación y asignación de sorteos completada en menos de 1-2 minutos.
- Latencia en operaciones clave (notificaciones, mensajes) menor a 500 ms.

2. Estudio de Viabilidad

Antes de la implementación del proyecto, es fundamental evaluar su viabilidad en distintos aspectos: legal, técnico, económico y de mercado.

2.1 Forma Jurídica

Dado que la aplicación puede evolucionar hacia un modelo de negocio, **se registrará bajo una sociedad limitada (S.L.)** para reducir riesgos legales y fiscales. Esta estructura proporciona ventajas significativas, tales como:

- **Protección patrimonial** del emprendedor.
- **Estructura legal clara** para captar inversión.
- **Facilidad para la formalización de acuerdos comerciales** y gestión de la propiedad intelectual.
- **Flexibilidad para incorporar socios en el futuro.**

La empresa se registrará bajo la denominación **NetMind S.L.**, lo que permitirá que futuros desarrollos y aplicaciones digitales se publiquen bajo un mismo paraguas corporativo, aumentando su credibilidad ante clientes e inversores. La sociedad se constituirá con un **capital social mínimo de 3.000 €**, aportado inicialmente por el autor en **modalidad unipersonal**.

Pasos para la constitución de la sociedad

1. **Reserva del nombre** en el Registro Mercantil.
2. **Redacción de estatutos** adaptados a la actividad de desarrollo de software.
3. **Depósito del capital social (3.000 €)** en una cuenta bancaria a nombre de la sociedad.
4. **Firma ante notario e inscripción** en el Registro Mercantil.
5. **Alta en Hacienda y Seguridad Social**, obteniendo el CIF definitivo.

Si bien la constitución no es obligatoria en las primeras etapas, contar con esta estructura desde el inicio no solo agiliza futuras decisiones estratégicas, sino que también facilita la obtención de inversión y acuerdos comerciales, asegurando un crecimiento sólido del proyecto.

Identidad Visual de la Sociedad (NetMind S.L.)

Concepto del Logotipo:

Una fusión de red neuronal (representando "Net") y diferentes nodos (símbolo de "Mind" o mente brillante), inspirada en un estilo minimalista con líneas fluidas, trazos imperfectos y un aura de creatividad.

Detalles:

- Icono: Una red de nodos conectados que forman la unión de mentes, donde a través de una red("ideas brillantes").
- Colores:
 - Azul agua (#88C1E0): Transparencia y fluidez.
 - Gris marengo (#3A4A4F): Estabilidad.

Justificación:

Diseñado para documentos legales, contratos y comunicación institucional, reflejando solidez como empresa de desarrollo.

*“El logotipo corporativo de NetMind S.L(Figura A-1 en anexo)”

Identidad Visual de la App "Amigo Invisible"

Concepto del Logotipo:

El ícono captura la magia de Studio Ghibli a través de dos jóvenes entregándose un regalo con alegría. Sus expresiones, llenas de emoción y calidez, recuerdan a personajes icónicos como Mei y Satsuki de Mi Vecino Totoro o Sophie y Howl de El Castillo Ambulante.

Detalles:

- Reconocimiento visual: El estilo es inmediatamente asociable al universo Ghibli, lo que genera conexión emocional con los seguidores.
- Nostalgia y calidez: Transmite la misma emoción que las películas, haciendo que los usuarios se sientan parte de una historia mágica.
- Diseño único pero familiar: No es una copia, pero sí un homenaje estético que resuena con el público.

*“El logotipo de la aplicación Amigo Invisible(Figura A-2 en anexo)”

2.2 Viabilidad Legal

Con el objetivo de garantizar que la aplicación "Amigo Invisible" cumple con la legislación vigente y se despliega dentro de un marco jurídico adecuado, se han analizado los aspectos legales más relevantes que podrían afectar tanto al desarrollo como a la explotación del producto. Este análisis contempla la **identificación de requisitos legales aplicables**, una **evaluación del cumplimiento actual**, así como la definición de **estrategias específicas de cumplimiento** y un **plan de contingencia ante riesgos legales**.

Identificación de Requisitos Legales

En función de la naturaleza de la aplicación (servicio digital con registro de usuarios, almacenamiento de datos personales, comunicaciones y posibles funcionalidades comerciales), se identifican las siguientes normativas aplicables:

- **Reglamento General de Protección de Datos (RGPD)** – Reglamento (UE) 2016/679.
- **Ley Orgánica de Protección de Datos y Garantía de Derechos Digitales (LOPDGDD)** – Ley 3/2018.
- **Ley de Servicios de la Sociedad de la Información y Comercio Electrónico (LSSI-CE)** – Ley 34/2002.
- **Ley de Propiedad Intelectual (TRLPI)** – Real Decreto Legislativo 1/1996.

Estas regulaciones afectan especialmente a los siguientes ámbitos de la aplicación:

- Recogida, tratamiento y almacenamiento de datos personales.
- Consentimiento informado del usuario.
- Condiciones de uso, privacidad y notificaciones electrónicas.
- Derechos de autor sobre el software y sus contenidos.
- Posibles comunicaciones comerciales futuras (opcional).

Evaluación del Cumplimiento Legal

Se ha llevado a cabo una revisión detallada de los flujos de datos y de las funcionalidades de la app con el fin de detectar posibles riesgos legales. A continuación, se destacan los puntos principales de cumplimiento:

- Protección de Datos Personales:
 - La aplicación recoge datos personales mínimos (nombre, correo electrónico, avatar, mensajes) necesarios para su funcionamiento.

- Se solicitará el consentimiento explícito del usuario antes del tratamiento de sus datos, siguiendo el principio de minimización del RGPD.
 - Se habilitará una opción para que el usuario pueda modificar o eliminar sus datos en cualquier momento (derecho al olvido).
- Transparencia y derechos del usuario:
 - La aplicación incluirá un aviso legal, una política de privacidad clara y los términos y condiciones de uso, visibles tanto en la app móvil como en la versión web.
 - Estos documentos explicarán el tipo de datos recopilados, su finalidad, el tiempo de retención y los derechos del usuario (acceso, rectificación, supresión, oposición).
- Propiedad Intelectual:
 - Todo el código, diseño y marca de la aplicación serán registrados como propiedad del autor del proyecto.
 - Se evitará el uso de contenido de terceros no autorizado (imágenes, librerías, APIs externas) sin verificar licencias abiertas o de uso comercial.
 - Se prevé el registro de la marca o logotipo en la Oficina Española de Patentes y Marcas (OEPM) en caso de comercialización futura.
- LSSI-CE y comunicaciones:
 - Si se implementan notificaciones o correos electrónicos para invitar a usuarios, se incluirá un mecanismo de consentimiento previo (opt-in).
 - En caso de futuras funcionalidades comerciales (por ejemplo, integración de regalos con Amazon), se aplicará la LSSI-CE con especial atención a la política de cookies, publicidad y comercio electrónico.

Estrategias de Cumplimiento

Para garantizar que la aplicación cumple con los marcos legales anteriores, se aplicarán las siguientes medidas:

- **Cifrado y control de acceso:**
Los datos personales almacenados en Firestore estarán protegidos mediante cifrado en tránsito (TLS) y autenticación basada en Firebase Auth. Se configurarán reglas de seguridad estrictas para evitar accesos no autorizados.
- **Consentimiento explícito:**
Antes del registro, el usuario deberá aceptar de forma activa los términos legales y la política de privacidad, que se almacenará como prueba de consentimiento.
- **Términos legales accesibles y comprensibles:**
Se redactarán documentos legales en lenguaje claro, adaptado al perfil general

de los usuarios, cumpliendo con los principios del RGPD sobre transparencia.

- **Registro de marca y software:**

En caso de lanzamiento público o monetización, se iniciarán los trámites legales necesarios para proteger la marca, código fuente y diseño bajo licencia de software libre o privativa, según convenga.

- **Revisión legal especializada (opcional):**

Se contempla la posibilidad de asesorarse con un experto legal o usar plataformas de revisión automática para validar los textos legales de la app.

Plan de Contingencia Legal

Dado que ninguna solución digital está exenta de riesgos legales, se establecen protocolos de actuación en caso de incidencia:

- **Incumplimiento del RGPD:**

En caso de recibir una reclamación relacionada con protección de datos, se activará una auditoría interna para analizar y corregir posibles fallos. Se contactará con el usuario afectado y se notificará a la AEPD si fuera necesario.

- **Violación de derechos de autor:**

Si se detecta un contenido protegido utilizado sin autorización, se retirará de inmediato. Se buscará un acuerdo con el titular de los derechos o se sustituirá por contenido con licencia válida.

- **Modificaciones legales futuras:**

Se mantendrá un seguimiento activo de cambios legislativos en materia digital para actualizar los textos legales y funcionalidades de la app conforme a la normativa vigente.

2.3 Marketing y Aprovisionamiento

La app **Amigo Invisible** va dirigida sobre todo a estudiantes, amigos, compañeros de trabajo o familiares que quieran organizar su amigo invisible de forma rápida, cómoda y sin líos. Nuestro público está entre los 16 y 45 años, por lo que vamos a centrar nuestros esfuerzos en los canales que más usan.

- **Redes sociales:** Usaremos Instagram, TikTok y X (antes Twitter), donde publicaremos:

- Consejos para organizar un amigo invisible original.
- Ideas de regalos por presupuesto.
- Reels y TikToks mostrando cómo funciona la app.
- Memes y contenido navideño divertido.

Además, aprovecharemos hashtags como #AmigoInvisible, #NavidadConAmigos, #AppParaRegalos, etc. para mejorar la visibilidad.

- **Contenido de valor:** Crearemos un pequeño blog dentro de la web de la app donde publicaremos artículos con títulos atractivos como:
 - “5 errores al organizar un amigo invisible (y cómo evitarlos)”
 - “Ideas de regalo por menos de 10€”
 - “¿App o papelitos? Qué es mejor para el sorteo”

Esto no solo ayuda a posicionarnos en Google, sino que también aporta valor al usuario.

Publicidad y promociones

Para promocionar la app sobre todo en noviembre y diciembre, se lanzarán campañas específicas:

- **Google Ads:** anuncios con palabras clave como “app para amigo invisible”, “sortear amigo secreto gratis”, etc.
- **Facebook e Instagram:** segmentando a jóvenes y adultos interesados en tecnología, Navidad, grupos de estudio, etc.
- **Ofertas y recompensas:**
 - Acceso gratis a funciones premium para los primeros usuarios.
 - Descuentos por invitar a otros.
 - Sorteos de tarjetas regalo o merchandising navideño para quienes comparten la app.

Aprovisionamiento

Para que todo funcione como debe, nos apoyamos en herramientas fiables y conocidas:

- **Firebase** (de Google): para la base de datos, autenticación y notificaciones push.
- **Android Studio:** para desarrollar la app nativa en Android.
- **GitHub:** donde se gestiona todo el código, las tareas y el control de versiones.
- **Google Cloud:** si se necesita escalar alguna funcionalidad en el futuro.
- **Whimsical** y **Canva:** para diseñar la interfaz de usuario y el material promocional.

Todo pensado para que sea funcional, escalable y económico, aprovechando versiones gratuitas o planes para desarrolladores.

Otros detalles

- **Modelo de negocio:** Se plantea un sistema freemium. Todo lo básico será gratuito, pero se podrían ofrecer funciones premium (como pistas personalizadas, o ajustes avanzados del sorteo) por una pequeña cuota opcional.
- **Atención al usuario:** Dentro de la app habrá un apartado de FAQ, además de un correo de soporte. También se podrán enviar sugerencias o reportes de errores.
- **Competencia:** Seguiremos analizando apps como DrawNames o Elfster para ver en qué destacan y detectar huecos o mejoras que podamos aplicar.

2.4 Viabilidad Económica

El proyecto se desarrollará principalmente con recursos propios y herramientas gratuitas o de bajo costo para minimizar la inversión inicial. El proyecto requiere una inversión inicial de 3.850€ (principalmente capital social). Con un modelo freemium y B2B, se proyectan 5.080€ de ingresos el primer año, alcanzando rentabilidad en 8 meses. Los costes operativos se optimizan mediante Firebase y herramientas gratuitas.

CONCEPTO	COSTO ESTIMADO €
Constitución de la empresa (S.L.)	3.000 (capital social)
Hosting y bases de datos (en caso de sobrepasar el plan gratuito)	10-20€/mes
Publicidad inicial mínima	100-200€
Producción de material promocional	50-100€
Asesoría legal inicial (protección de datos y términos de uso)	200-300€

*“La gráfica de la distribución de Gastos Estimados(Figura B-1 en anexo)”

*“La gráfica de Proyección de Ingresos Anuales(Figura B-2 en anexo)”

Análisis de Costes Ampliado con costes de plan Blaze de Firebase(Escenario contemplado si se supera el plan Blaze):

CONCEPTO	PLAN GRATUITO	PLAN BLAZE (si >10K usuarios)
Almacenamiento Firestore	1GB incluido	0.18€/GB adicional
Transferencia de datos	10GB/mes	0.15€/GB extra
Autenticación	10K usuarios/mes	0.01€/usuario adicional

*Estimación conservadora: Si alcanzamos 15K usuarios, coste mensual = 45€

Flujo de ingresos y modelo de monetización

Para garantizar la sostenibilidad económica del proyecto y generar ingresos, se plantea una estrategia de monetización basada en un modelo **Freemium**, complementado con licencias para empresas y acuerdos estratégicos con marcas.

Modelo de adopción escalonada: durante los **dos primeros meses** tras el lanzamiento, todos los usuarios tendrán acceso a **todas las funcionalidades Premium de forma gratuita**, como parte de una campaña de promoción navideña.

A partir del **tercer mes**, solo las funcionalidades básicas seguirán activas de forma gratuita, y se invitará a los usuarios a suscribirse para mantener el acceso completo.

Este enfoque permitirá demostrar el valor de las funciones Premium y mejorar la conversión desde la versión gratuita.

Para garantizar la sostenibilidad económica del proyecto y generar ingresos, se plantea una estrategia de monetización basada en un modelo Freemium, complementado con licencias para empresas y acuerdos estratégicos con marcas.

Enfoque B2B como eje de rentabilidad

Se contempla el modelo empresarial (B2B) como **la principal fuente de ingresos estables** a medio y largo plazo. Las empresas podrán contratar planes anuales que les permitan gestionar sorteos internos con:

- Branding personalizado (nombre y logo de la empresa en la app).
- Analítica básica de uso.
- Soporte prioritario.
- Sorteos ilimitados para equipos grandes.

Este enfoque ofrece mayor previsibilidad en ingresos y escalabilidad comercial, pudiendo incluso establecer acuerdos con consultoras de recursos humanos o plataformas de beneficios corporativos.

1. Suscripción Premium (Freemium)

- **Precio:** 5,99 €/mes o 29,99 €/año.
- **Ventajas para los suscriptores:**
 - Reglas avanzadas (exclusiones entre participantes).
 - Chat anónimo ilimitado y stickers personalizados.
 - Integración con listas de deseos de Amazon(*con comisión del 5% por compras derivadas*).

2. Licencias para empresas (B2B)

- Planes diseñados para equipos y empresas que organizan sorteos internos.
- **Precio:** 99 €/año (hasta 30 usuarios).
- Incluye herramientas analíticas, sorteos privados y personalización de branding.

3. Colaboraciones con marcas (*sin anuncios intrusivos*)

- Promoción de productos y servicios en la app a cambio de una comisión o colaboración publicitaria.

Proyección de ingresos (primer año)

En base a una estimación de **10.000 descargas en el primer año**, se proyectan los siguientes ingresos:

- **Usuarios premium:** Se espera que un **5%** de los usuarios activos opten por la suscripción premium. Esto equivaldría a **500 usuarios premium**, generando **3.590 €/año**.
- **Empresas contratantes:** Se estima que **10 empresas** adquieran licencias anuales, aportando **990 €/año**.
- **Comisiones por compras derivadas (Amazon/Etsy):** Se prevé un ingreso de aproximadamente **500 €/año**.

Total estimado: 5.080 €/año, cubriendo los costes operativos y permitiendo margen para reinversión y crecimiento.

Plazos de retorno

- **Corto plazo (primer año):** Se priorizará la captación de usuarios mediante estrategias orgánicas y de fidelización. La meta es alcanzar **10.000 descargas** y validar el modelo de ingresos.
- **Medio plazo (1-3 años):** Se optimizarán las estrategias de monetización y se explorarán nuevas funcionalidades premium y alianzas estratégicas con entidades importantes.
- **Largo plazo (+3 años):** Expansión del modelo B2B, internacionalización y desarrollo de nuevas aplicaciones bajo la misma sociedad.

Resumen estratégico: El equilibrio entre una experiencia gratuita atractiva, funcionalidades Premium claras y una vía B2B sólida, ofrece un modelo de negocio mixto capaz de escalar y sostenerse en el tiempo. Se apuesta por una captación orgánica inicial, validación de mercado con usuarios reales, y una transición natural hacia la monetización a través de valor añadido, sin recurrir a publicidad intrusiva.

2.5 Análisis de Riesgos

A continuación se identifican los siguientes riesgos y posibles estrategias de mitigación:

1. Riesgos Técnicos

Riesgo	Probabilidad	Impacto	Estrategia Mitigación	Plan de Contingencia
Complejidad en la integración de Firebase Firestore para gestionar sorteos con reglas personalizadas (ej: exclusiones).	Media	Alto	<ul style="list-style-type: none">• Asignar 1 hora diaria a tutoriales oficiales de Firebase.• Implementar un módulo de prueba con datos ficticios antes de la versión final.	Usar SQLite como base de datos local temporal si Firestore retrasa el desarrollo.
Incompatibilidad con versiones antiguas de Android (API < 26).	Media	Medio	<ul style="list-style-type: none">• Definir Android 8.0 (API 26) como versión mínima.• Probar en emuladores con Android 8, 10 y 13.	Limitar funcionalidades avanzadas (ej: animaciones) en dispositivos antiguos.
Problemas con la API de Amazon para listas de deseos (límites de solicitudes, cambios en endpoints).	Baja	Alto	<ul style="list-style-type: none">• Implementar caché local de listas de deseos.• Documentar alternativas manuales (ej: captura de pantalla).	Desactivar temporalmente la integración y notificar a usuarios. Sustituir por una lista propia.

2. Riesgos de Tiempo

Riesgo	Probabilidad	Impacto	Estrategia Mitigación	Plan de Contingencia
Retrasos en la implementación del chat anónimo debido a la complejidad en el cifrado de mensajes.	Alta	Medio	<ul style="list-style-type: none"> • Usar librerías preexistentes (ej: Signal Protocol). • Priorizar funcionalidades básicas en primera versión. 	Postergar el chat para una actualización posterior si no se cumple el cronograma.
Saturación por otros compromisos académicos en periodo de exámenes.	Alta	Alto	<ul style="list-style-type: none"> • Bloquear 15 horas semanales fijas en el calendario. • Usar metodología Kanban para gestionar tareas. 	Reducir alcance temporalmente (ej: eliminar stickers premium en MVP).

3. Riesgos de Recursos

Riesgo	Probabilidad	Impacto	Estrategia Mitigación	Plan de Contingencia
Límite del plan gratuito de Firebase (superar 10K usuarios o 50K operaciones/día).	Media	Alto	<ul style="list-style-type: none"> • Monitorear uso mensual con Firebase Analytics. • Presupuestar 50€/mes para el plan Blaze. 	Migrar a Supabase (alternativa open-source) si los costes escalan.
Fallo del ordenador principal durante el desarrollo.	Bajo	Crítico	<ul style="list-style-type: none"> • Backups diarios en GitHub + Google Drive. • Virtualizar entorno de desarrollo con Docker. 	Usar equipos del centro educativo o adquisición de nuevo equipo y rescatar copias de seguridad.

4. Riesgos de Gestión

Riesgo	Probabilidad	Impacto	Estrategia Mitigación	Plan de Contingencia
Cambios en requisitos del proyecto (ej: añadir integración con PayPal).	Baja	Medio	<ul style="list-style-type: none"> • Documentar requisitos iniciales en un acta firmada. • Establecer reuniones semanales para alinear expectativas. 	Negociar plazos adicionales o priorizar en fases posteriores.
Problemas de comunicación con el tutor.	Medio	Bajo	<ul style="list-style-type: none"> • Usar checklist en cada reunión (Google Docs compartido). • Enviar resúmenes por email después de cada sesión. 	Recurrir a foros técnicos (Stack Overflow) para dudas urgentes o ayudas de IAs.

Este análisis de riesgos demuestra una evaluación exhaustiva de los desafíos potenciales en el desarrollo de la aplicación "Amigo Invisible", clasificándolos en cuatro categorías críticas: **técnicos, de tiempo, de recursos y de gestión**. Para cada riesgo identificado, se han establecido:

1. Estrategias de mitigación proactivas que incluyen:

- Formación específica (ej: tutoriales Firebase)
- Pruebas tempranas (módulos piloto)
- Protocolos de comunicación estructurados
- Planificación de recursos alternativos

2. Planes de contingencia realistas que permiten:

- Mantener la funcionalidad básica incluso en escenarios adversos (ej: SQLite como backup)
- Ajustar el alcance sin comprometer el producto mínimo viable
- Garantizar la continuidad del desarrollo ante imprevistos

Priorización de acciones:

● Riesgos críticos (alta probabilidad + alto impacto):

- Monitorización activa del plan Firebase (alertas al alcanzar el 80% del límite gratuito)
- Pruebas de compatibilidad Android programadas semanalmente

● Riesgos aceptables:

- Incorporados en el cronograma con buffers de tiempo (ej: +2 semanas para el chat anónimo)

2.6 Cronograma

Plazos y Gestión del Proyecto

El desarrollo se realizará del **17 de Marzo al 16 de Junio de 2025**, utilizando metodología ágil (SCRUM) con sprints semanales. Se emplearán las siguientes herramientas:

1. **GitHub**: Control de versiones y gestión de código
2. **Google Drive**: Documentación compartida
3. **Google Calendar**: Recordatorios de hitos clave

FASE	FECHAS	TAREAS CLAVE	ENTREGABLES	HERRAMIENTAS
1. Investigación y Diseño	17 mar – 4 abr	- Análisis de requisitos - Prototipado inicial (Whimsical) - Diseño UI/UX responsive - Arquitectura de base de datos	- Documento de requisitos - Mockups interactivos - Esquema Firestore	Whimsical, Figma, Google Docs
2. Desarrollo Backend	5 abr – 24 abr	- Autenticación con Firebase - Configuración Firestore - Integración API de listas de deseos (Amazon/Etsy) - Soporte multilingüe básico (es/en)	- Módulo de login funcional - Base de datos estructurada - Endpoints funcionales	Android Studio, Firebase Console, Postman
3. Desarrollo Frontend	25 abr – 19 may	- Interfaz móvil completa - Integración con backend (Firebase) - Notificaciones push - Soporte para accesibilidad (lectores de pantalla)	- APK funcional - Demostración en video - Documentación técnica de interfaz	GitHub (commits diarios), Android Studio
4. Pruebas y Ajustes	20 may – 4 jun	- Test de usabilidad (10 usuarios) - Pruebas en múltiples dispositivos - Ajustes de accesibilidad y visualización adaptable - Corrección de bugs	- Informe de usabilidad - Checklist de validación - Reporte de bugs	Firebase Test Lab, Google Forms, BrowserStack
5. Lanzamiento y Promoción	4 jun – 16 jun	- Publicación en Play Store - Activación del sistema freemium por fases - Campaña de marketing digital - Seguimiento de métricas de uso	- App publicada - Dashboard de analíticas - Informe de campaña y feedback inicial	Google Play Console, Firebase Analytics, Canva

Gestión de Código y Documentación

- **GitHub**:
 - Commits diarios con mensajes descriptivos (<https://github.com/P3droRamirez/FCT.PROJECT.AmigoInvisibleApp>)
 - Etiquetas para bugs (bug), mejoras (enhancement) y tareas (task)
- **Google Drive**:
 - Carpeta estructurada por fases (Diseño/Desarrollo/Pruebas)
 - Historial de versiones activado

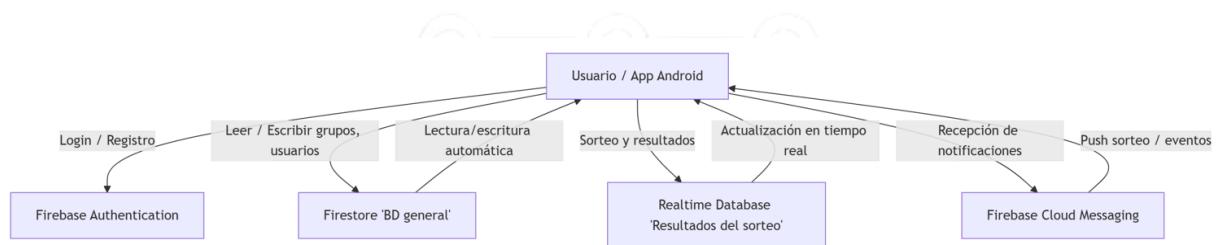
3. Arquitectura del Sistema

La arquitectura de la aplicación **Amigo Invisible** ha sido diseñada con un enfoque *cloud-first*, apoyándose íntegramente en **Firebase** como backend serverless, y desarrollada en **Java con Android Studio** para el entorno móvil. Esta decisión permite minimizar la complejidad técnica, escalar sin coste operativo en las fases iniciales y garantizar sincronización en tiempo real, usabilidad y seguridad.

3.1 Topología de Servidores y Aplicaciones

Diagrama de Arquitectura

La arquitectura adoptada es del tipo **cliente-nube**. Los dispositivos móviles (clientes Android) se conectan directamente con los servicios proporcionados por **Firebase**, sin necesidad de un servidor intermedio.



Componentes Principales

- **Aplicación móvil Android (Java)**
- **Firebase Authentication**: Autenticación de usuarios
- **Firebase Firestore**: Base de datos para usuarios, grupos, configuraciones
- **Firebase Realtime Database**: Sincronización rápida de sorteos y resultados
- **Firebase Cloud Messaging (FCM)**: Notificaciones push

Flujo de la comunicación

- Toda la comunicación se realiza a través del SDK de Firebase mediante **HTTPS/TLS**.
- La app se conecta directamente a Firestore para operaciones CRUD y a Realtime DB para datos altamente dinámicos.
- FCM gestiona notificaciones en tiempo real sobre actividades del grupo o resultados del sorteo.

Descripción de la Topología

- **App Móvil Android (Java):** Esta aplicación se conecta directamente con Firebase para gestionar el ciclo completo de la experiencia del usuario: registro e inicio de sesión, creación y unión a grupos, sorteo del amigo invisible y envío de notificaciones.
- **Firebase Authentication:** Gestiona el acceso de los usuarios mediante correo y contraseña. Garantiza una identificación segura.
- **Firebase Realtime Database / Firestore:** Almacena los datos de los usuarios, grupos y resultados del sorteo. Permite sincronización en tiempo real, lo que mejora la experiencia del usuario sin necesidad de recargar ni hacer llamadas manuales a un servidor.
- **Firebase Cloud Messaging (FCM):** Utilizado para enviar notificaciones push cuando se realiza un sorteo o se añaden miembros a un grupo.

En el cliente móvil, todos los usuarios comparten la misma app. Sin embargo, el sistema contempla dos sub-roles implícitos: el Organizador, que tiene acceso a las funciones de creación y configuración de sorteos; y el Participante, que accede a la información de asignación y puede interactuar mediante el chat anónimo. Esta diferenciación se gestiona a nivel lógico mediante el campo de creador del sorteo y la lista de participantes.

Tecnologías utilizadas

Componente	Tecnología
Lenguaje	Java 17
IDE	Android Studio (Hedgehog 2022.3.1)
UI	Material Components + Glide
Backend	Firebase (Auth, Firestore, RTDB, FCM)
Librerías	Firebase SDK, Glide, AndroidX

Justificación tecnológica

- **Java:** Lenguaje robusto, ampliamente documentado y soportado por Android, con el que el desarrollador ya tiene experiencia.
- **Firebase:** Proporciona un backend completo con autenticación, base de datos en tiempo real, notificaciones push y almacenamiento, sin necesidad de configurar servidores. Perfecto para un proyecto de escala media con alta sincronización de datos.

- **Firestore/Realtime Database:** Ofrecen un acceso rápido y en tiempo real a los datos, lo cual es ideal para actualizar listas de participantes o notificar el sorteo.
- **FCM:** Facilita el envío automático de notificaciones sin coste adicional.

Seguridad

- Comunicación cifrada (HTTPS).
- Control de acceso mediante **reglas de seguridad en Firestore y Realtime DB**.
- Firebase Authentication asegura el acceso individual por sesión activa.

Escalabilidad

- Firebase permite escalar automáticamente según el tráfico.
- Arquitectura modular y sin servidor que elimina el mantenimiento de infraestructura tradicional.

3.2 Descripción de la App Móvil (Android)

Aplicación Móvil (Android)

La aplicación móvil ha sido desarrollada en **Java** para dispositivos Android, siguiendo los principios de diseño de Material Design. Su objetivo principal es permitir a los usuarios registrarse, iniciar sesión y gestionar grupos a través de una interfaz intuitiva, segura y organizada. Toda la lógica de autenticación y almacenamiento de usuarios se gestiona con **Firebase Authentication** y **Cloud Firestore**, facilitando la escalabilidad y la sincronización en tiempo real.

Funcionalidades Principales

- **Autenticación con Google o correo electrónico** (mediante Firebase Auth).
- **Gestión de grupos:** Crear, ver, y administrar grupos personales o laborales.
- **Sorteo automático** con reglas de exclusión.
- **Notificaciones push** cuando se realiza el sorteo.
- **Chat anónimo** limitado (solo visible tras el sorteo).
- **Personalización:** Imagen de perfil, lista de deseos y pistas opcionales.

Autenticación de Usuarios o registro

- El usuario puede autenticarse mediante **correo electrónico y contraseña**, o también iniciar sesión a través de **Google**.
- En ambos casos, los datos del usuario (nombre, email, foto y grupos) se almacenan en Firestore.
- Si el usuario se registra por correo, puede introducir manualmente su nombre.
- Se incluye la opción de cerrar sesión desde el menú lateral.

Interfaz de bienvenida personalizada

- Una vez iniciado el proceso de autenticación correctamente, el usuario accede a la pantalla principal donde se le muestra un mensaje de bienvenida personalizado con su nombre, además si tiene algún tipo de notificación o ha habido algún cambio en alguno de sus grupos.

Gestión de Grupos

La navegación de la app se realiza a través de un menú lateral (**Navigation Drawer**) que da acceso a diferentes secciones

- **Crear grupo:** Permite al usuario crear un nuevo grupo de trabajo o personal. (Implementado mediante el fragmento CreateGroupFragment).
- **Ver grupos:** Lista los grupos existentes asociados al usuario. (Manejados desde ViewGroupsFragment).
- **Configuración:** Espacio reservado para futuras opciones de ajustes del usuario. (SettingsFragment).

Persistencia y sincronización de datos

- Todos los datos (usuarios y grupos) se almacenan en **Firebase Firestore**, garantizando su persistencia y disponibilidad en la nube.
- Al iniciar sesión, si el usuario no está en la base de datos, se crea automáticamente con algunos grupos iniciales ("Trabajo" y "Casa").

Sorteo del Amigo Invisible

- Una vez que todos los miembros han ingresado al grupo, el administrador puede iniciar el **sorteo automático**.
- El sistema asigna aleatoriamente un "amigo invisible" a cada miembro, **garantizando que nadie se asigne a sí mismo**.
- Los resultados se almacenan en **Firebase Realtime Database** y cada usuario puede ver **únicamente su amigo asignado**.

Notificaciones Push

- Cuando un usuario se une a un grupo o se realiza un sorteo, se envía una **notificación automática** mediante **Firebase Cloud Messaging (FCM)**.
- Notificaciones también para recordatorios del día del evento o mensajes del grupo.

Personalización

- Opción para añadir pistas, listas de deseos o presupuestos sugeridos.
- Posibilidad de añadir una imagen de perfil y nombre personalizado visible en los grupos.

Interfaz de Usuario (UI)

- **Pantalla de login** (MainActivity):
 - Campos para introducir correo electrónico, contraseña y nombre.
 - Botones para iniciar sesión con email o Google.
 - Validaciones básicas antes de permitir el acceso.
- **Pantalla principal** (WelcomeActivity):
 - Barra superior personalizada (Toolbar) sin título por defecto, que muestra el nombre del usuario.
 - Menú lateral con opciones para gestionar grupos y cerrar sesión.
 - Fragmentos reemplazan dinámicamente el contenido según la opción seleccionada en el menú.
- **Fragments Principales**
 - MainFragment: pantalla principal (puede mostrar información general o bienvenida).
 - CreateGroupFragment: interfaz con formulario para crear grupos y añadir miembros.
 - ViewGroupsFragment: muestra una lista de grupos con sus integrantes.

Experiencia de Usuario (UX)

- Navegación clara mediante menú lateral.
- Mensajes de error específicos en caso de fallo de autenticación o campos incompletos.
- Flujo de usuario simple: iniciar sesión → ver grupos o crear uno → cerrar sesión.
- Se utiliza Toast para proporcionar feedback inmediato en acciones como inicio de sesión, errores o éxito en la creación de un grupo.

Interacción con la API y base de datos

- La app **no utiliza un servidor propio con API REST**, sino que interactúa directamente con **Firebase Firestore**, lo cual simplifica la arquitectura y reduce la necesidad de una capa intermedia.
- Los datos del usuario se guardan en un documento dentro de la colección Users, identificados por su UID de Firebase.
- En caso de errores de red o problemas con Firebase, se muestra un mensaje informativo al usuario.

Interacción con Firebase

- **Firebase Authentication**: para la gestión segura de los usuarios.
- **Firebase Realtime Database**: almacenamiento y sincronización en tiempo real de los grupos, participantes y resultados.
- **Firebase Cloud Messaging**: sistema de notificaciones automáticas en tiempo real.
- Toda la comunicación se realiza mediante **Firebase SDK para Android**, utilizando HTTPS de forma predeterminada y gestionando la sincronización automática de datos.

Casos de Uso

Caso de Uso: Registro y Creación de Grupo

- El usuario se registra con su cuenta de Google.
- Accede a la pantalla principal, donde selecciona "Crear Grupo".
- Introduce un nombre de grupo y añade participantes.
- El grupo se guarda en Firestore y aparece en la sección "Ver Grupos".

Gestión de Sesiones y Seguridad

- Persistencia de sesión con Firebase.
- Reglas de seguridad en Firebase Realtime Database para que cada usuario **solo acceda a sus propios datos**.
- Autenticación previa obligatoria para acceder a cualquier funcionalidad dentro de la app.

Flujo de Datos

1. El usuario inicia sesión → Firebase valida la identidad.
2. Si es la primera vez, se crea un nuevo documento en Firestore con sus datos.
3. Una vez autenticado, se redirige a WelcomeActivity.
4. Desde allí, el usuario puede:
 - Crear grupos (datos guardados en Firestore).
 - Visualizar grupos existentes.
5. Al cerrar sesión, se borra la sesión activa de Firebase y se vuelve a la pantalla de login.

***Se adjuntan diagramas visuales del flujo principal de la aplicación Amigo Invisible(Figura C-1,C.2, en anexo)*

Aunque actualmente la aplicación está centrada exclusivamente en el entorno móvil Android, se contempla la posibilidad de desarrollar en el futuro una versión web multiplataforma, accesible desde navegadores mediante Firebase Hosting. Esta versión permitiría una mayor accesibilidad para usuarios que prefieran participar desde ordenador, facilitando la gestión de sorteos en contextos laborales o educativos. El backend, al estar basado en Firebase, permite esta expansión sin necesidad de modificar la infraestructura actual.

3.3 Diagrama de Despliegue

El despliegue de la aplicación **Amigo Invisible** se ha planteado con un enfoque **cloud-first**, aprovechando la infraestructura escalable de Firebase y eliminando la necesidad de servidores propios. Esta estrategia permite minimizar los costes, facilitar la implementación y asegurar una disponibilidad global desde el primer momento.

Entorno de Producción

Nodo	Artefacto	Función
Dispositivo Android	APK / AAB	Interfaz de usuario y lógica cliente
Firebase Cloud	Auth, Firestore, RTDB, FCM	Backend serverless
Firebase Console	Panel web	Administración de datos, reglas y métricas
Play Store / APK directo	Instalador	Canal de distribución (fase beta o pública)

**Se adjunta un diagrama de despliegue aplicación Amigo Invisible(Figura C-3 en anexo)"

Flujo de despliegue

1. El usuario instala la app desde Google Play o vía directa.
2. Se inicia sesión con Google/Firebase Authentication.
3. Se accede a Firestore para grupos y usuarios.
4. Se usa Realtime DB para sorteo.
5. Se reciben notificaciones a través de FCM.

Consideraciones Técnicas

- **Distribución:** Versión beta descargable; futura publicación en Play Store.
- **Seguridad:** Reglas configuradas para evitar accesos cruzados.
- **Escalabilidad automática:** Proporcionada por la infraestructura de Google Cloud.
- **Backups y control:** Acceso desde Firebase Console. Firestore tiene mecanismos de exportación.

4. Diseño

4.1 Modelo de Datos

La estructura de la base de datos está diseñada para aprovechar las ventajas de Firebase Firestore, un sistema de almacenamiento NoSQL orientado a documentos. Se ha optado por un modelo jerárquico, organizado en colecciones y subcolecciones, que permite una sincronización en tiempo real y una escalabilidad eficiente para la aplicación “Amigo Invisible”.

Justificación de Firebase

- **Sincronización en tiempo real:** Ideal para actualizaciones instantáneas, como cuando se agrega un nuevo participante o se envía un mensaje en el chat.
- **Escalabilidad automática:** Puede soportar grupos grandes o muchas interacciones sin problemas de rendimiento.
- **Modelo flexible:** Nos permite estructurar datos jerárquicos, como sorteos dentro de grupos, o listas de deseos por usuario.
- **Integración nativa con Auth y Cloud Functions:** seguridad, autenticación y lógica de backend simplificada.

Estructura de Colecciones y Documentos

Colección	Ejemplo de Documento	Descripción
users	{ userId, name, email, isPremium, groups[] }	Información del usuario y sus grupos.
groups	{ groupId, name, admin, budget, maxParticipants, members[] }	Datos del grupo, su creador y los participantes.
draws	{ drawId, groupId, deadline, exclusions[] }	Datos del sorteo y posibles exclusiones.
assignments	{ assignmentId, drawId, giver, receiver }	Resultado del sorteo: quién regala a quién.

messages	{ messageId, drawId, sender, content, isAnonymous, timestamp }	Chat anónimo entre participantes del sorteo.
wishlists	{ wishlistId, userId, drawId, items[] }	Lista de regalos deseados por cada usuario.
notifications	{ notificationId, userId, message, isRead }	Notificaciones individuales al usuario.

*“La imagen del diagrama de modelo noSql se encuentra (Figura C-4 en anexo)”

Relaciones y Cardinalidad

Entidad Principal	Relación	Cardinalidad	Implementación en Firebase
User	pertenece a Groups	N:M	Campo groups[] en users y members[] en groups
Group	tiene muchos Draws	1:N	Campo groupId en draws (puede usarse como subcolección también)
Draw	tiene muchos Assignments	1:N	Subcolección assignments dentro de draws
Draw	contiene muchos Messages	1:N	Subcolección messages dentro de draws
Draw	está vinculado a muchas Wishlists	1:N	Campo drawId en wishlists
User	tiene una única Wishlist	1:1	Campo userId en wishlists (único por combinación con drawId)
Assignment	relaciona giver con receiver	1:1	Campos giver y receiver en cada documento de assignments

Consideraciones de Acceso a Datos

En un modelo NoSQL como Firebase Firestore, las consultas son fundamentales para acceder de forma eficiente a los datos, sin necesidad de joins complejos como ocurre en bases de datos relacionales. En esta aplicación, donde se necesita rendimiento y tiempo real, diseñar bien las consultas es clave.

Consultas Típicas en la App

Obtener los participantes de un grupo:

```
Firestore.getInstance()
    .collection("groups")
    .document(groupId)
    .get()
    .addOnSuccessListener(document -> {
        List<String> miembros = (List<String>)
            document.get("members");
    });
}
```

- Esto devuelve una lista de IDs de usuarios que luego pueden usarse para cargar sus datos de la colección users.

Obtener los participantes asignados en un sorteo:

```
Firestore.getInstance()
    .collection("draws")
    .document(drawId)
    .collection("assignments")
    .whereEqualTo("giver", currentUserId)
    .get();
}
```

- Devuelve la persona a la que el usuario debe regalar en ese sorteo.

Obtener la wishlist de un usuario en un sorteo:

```
Firestore.getInstance()
    .collection("wishlists")
    .whereEqualTo("userId", userId)
    .whereEqualTo("drawId", drawId)
    .get();
}
```

Estas consultas están pensadas para ser rápidas, indexadas y ejecutarse con baja latencia. El modelo de datos está optimizado para lectura, incluso a costa de duplicar algo de información, como recomienda la documentación oficial de Firestore.

Creación del Modelo de Datos en Firebase Console

Durante el desarrollo de la app, se ha utilizado **Firebase Console** como herramienta gráfica para crear colecciones y documentos, además de gestionar manualmente datos de prueba durante el desarrollo.

- Se crearon manualmente las colecciones principales:
 - users
 - groups
 - draws
 - assignments
 - messages
 - wishlists
 - notifications
- En cada una se definieron los campos desde la interfaz, utilizando los tipos de datos adecuados: String, Array, Boolean, Timestamp, etc.

Esta creación visual ha sido complementada con operaciones desde el código Java, tanto para añadir datos como para actualizar en tiempo real.

Consideraciones Adicionales

Flexibilidad de Firebase frente a MySQL:

Característica	Firebase Firestore (NoSQL)	MySQL (Relacional)
Estructura	Flexible, sin esquema fijo	Esquema rígido
Escalabilidad	Escala automáticamente	Requiere configuración manual
Tiempo real	Nativo con listeners	No nativo
Join de datos	No soportado, requiere duplicación	Natural
Velocidad de lectura	Alta si bien modelado	Alta, depende de índices
Curva de aprendizaje	Baja (para Firebase)	Media

La decisión de utilizar Firebase Firestore se basa en:

- Necesidad de sincronización en tiempo real.
- Aplicación móvil con múltiples usuarios concurrentes.

- Escalabilidad automática para eventos como campañas navideñas.
- Facilidad de integración con Firebase Auth, Cloud Messaging y Functions.

Documentación del modelo

Además del diagrama visual generado en el diseño (ver imagen), se ha documentado el modelo de datos en el informe:

- Cada colección tiene una tabla explicativa con sus campos.
- Se han justificado las relaciones y cardinalidades.
- Se han descrito ejemplos de documentos reales del sistema.

4.2 Diagrama de Clases

El **Diagrama de Clases** representa la estructura estática de la aplicación. Muestra las clases principales utilizadas, sus atributos, métodos y las relaciones entre ellas. En el caso de nuestra aplicación "**Amigo Invisible**", se ha optado por una arquitectura orientada a objetos clara, modular y basada en patrones como MVC y repositorio.

Identificación de Clases Relevantes

Clase	Descripción
User	Representa a un usuario de la app (nombre, email, UID, suscripción).
Group	Representa a un grupo de amigos invisibles. Tiene miembros, presupuesto, admin, etc.
Member	Clase que modela cada participante en un grupo.
WishlistItem	Ítems que cada usuario desea. Relacionado con un usuario y un sorteo.
Assignment	Representa la asignación secreta entre dos usuarios (quién regala a quién).
Message	Chat anónimo entre miembros de un sorteo.
Notification	Alerta individual enviada a un usuario.
FirebaseService	Encapsula operaciones con la base de datos Firestore.
AuthService	Gestiona el login y registro con FirebaseAuth.
GroupUtils	Clase utilitaria que permite realizar el sorteo de forma lógica.
MemberAdapter	Adaptador para el RecyclerView que muestra participantes.

Definición de Atributos

User: id (String, UUID), name (String, no nulo), email (String, único, formato email), isPremium (Boolean), groups (List<String>, IDs de grupos)

Group: id (String, UUID), name (String, no nulo), adminId (String, ID del usuario), budgetLimit (Double), maxParticipants (Long), members (List<String>, IDs de usuarios)

Draw: id (String, UUID), groupId (String), deadline (Timestamp), exclusions (Map<String, String>, exclusiones entre usuarios)

Assignment: id (String, UUID), drawId (String), giverId (String), receiverId (String)

Message: id (String, UUID), drawId (String), senderId (String), content (String, máximo 500 caracteres), timestamp (Timestamp), isAnonymous (Boolean)

Wishlist: id (String, UUID), userId (String), drawId (String), items (List<Item>), → *Item: name (String), url (String)*

Notification: id (String, UUID), userId (String), message (String), isRead (Boolean)

Definición de Métodos

User: iniciarSesion(email: String, password: String): Boolean,
cerrarSesion(): void, esPremium(): Boolean, unirseAGrupo(groupId: String): void

Group: agregarMiembro(userId: String): void, eliminarMiembro(userId: String): void,
realizarSorteo(): void, obtenerMiembros(): List<User>

Draw: crearAsignaciones(): void, obtenerAsignaciones(): List<Assignment>

Assignment: verReceptor(userId: String): String, asignar(userId: String, receiverId: String): void

Message: enviarMensaje(content: String, isAnon: Boolean): void,
obtenerMensajes(drawId: String): List<Message>

Wishlist: agregarItem(nombre: String, url: String): void,
eliminarItem(nombre: String): void, obtenerLista(): List<Item>

Notification: marcarComoLeida(): void, enviarAlerta(mensaje: String): void,
listarNoLeidas(): List<Notification>

Establecimiento de Relaciones

En el sistema del *Amigo Invisible*, las relaciones entre las clases siguen una lógica orientada a objetos y están alineadas con la estructura NoSQL utilizada en Firebase. Las principales relaciones son:

- **Asociación:**

User 1..* Wishlist

Cada usuario tiene una (y solo una) lista de deseos para un sorteo determinado, pero puede tener varias si participa en varios sorteos.

- **Agregación:**

Group 1..* Users

Un grupo está compuesto por muchos usuarios. Los usuarios pueden ser agregados o eliminados sin afectar a la existencia del grupo.

- **Asociación reflexiva:**

User → Message → User

A través del chat anónimo, los usuarios se comunican sin saber quién es su receptor. Se puede considerar que el usuario interactúa consigo mismo o con otro sin identificarlo.

- **Composición:**

Draw 1..* Assignments

Un sorteo no tiene sentido sin asignaciones, y estas no pueden existir por sí solas. Si se elimina el sorteo, desaparecen las asignaciones.

- **Dependencia:**

MemberAdapter → Firestore

Las clases que gestionan los adaptadores o los servicios dependen de Firebase como fuente de datos.

- **Dependencia indirecta:**

GroupUtils.realizarSorteo() → Firestore Database

El sorteo depende de los datos actuales del grupo y de los participantes, y actualiza la base de datos directamente desde esta clase utilitaria.

Definición de Visibilidad

Siguiendo los principios de diseño orientado a objetos, se ha aplicado el **encapsulamiento** en todas las clases del proyecto. Se han definido los modificadores de acceso con el siguiente criterio:

- **Atributos: private (siempre que sea posible)**

Los atributos de las clases (name, email, groupId, drawId, etc.) son declarados como privados para evitar modificaciones externas no controladas. Se accede a ellos a través de métodos getters y setters.

- **Métodos: public para la interfaz de la clase, private para lógica interna**
 - Métodos como getName(), sendMessage(), startDraw() son public, ya que forman parte de la lógica accesible desde otras clases.
 - Métodos internos como validaciones o formateos (isValidEmail(), formatMessage(...)) podrían definirse como private, si no deben ser accedidos desde fuera.

Creación del Diagrama de Clases

El diagrama de clases presentado a continuación describe la estructura fundamental de la aplicación "Amigo Invisible". En él, se representan las principales entidades, servicios y sus relaciones, organizadas en un diseño orientado a objetos con el propósito de gestionar los sorteos de amigos invisibles entre los usuarios.

Clases Principales:

1. User (Usuario):

- Representa a un usuario dentro de la aplicación, con atributos como **id**, **name**, **email**, **isPremium** y **groups**. Los métodos permiten gestionar el inicio y cierre de sesión, así como las interacciones con los grupos a los que pertenece.
- Relación: Un **User** puede pertenecer a varios **Groups** (relación **N:M**).

2. Group (Grupo):

- Representa a un grupo de usuarios para un sorteo específico, con atributos como **id**, **name**, **adminId**, **budgetLimit**, **maxParticipants**, y **members**. Además, incluye métodos para agregar y eliminar miembros, y realizar el sorteo.
- Relación: Un **Group** puede tener muchos **Draws** y muchos **Users** asociados (relaciones **1:N**).

3. Draw (Sorteo):

- Contiene información del sorteo, como **id**, **groupId**, y **exclusions** (exclusiones entre usuarios). Los métodos permiten gestionar las asignaciones y los sorteos en sí.
- Relación: Un **Draw** tiene muchos **Assignments** y **Messages**.

4. Assignment (Asignación):

- Relaciona a un usuario que regala con el usuario que recibe el regalo. Contiene atributos como **id**, **drawId**, **giverId**, y **receiverId**.
- Relación: Un **Assignment** está asociado a un único **User** como **giver** y **receiver** (relación 1:1).

5. Message (Mensaje):

- Representa un mensaje entre los participantes del sorteo. Los atributos incluyen **id**, **drawId**, **senderId**, **content**, **timestamp**, y **isAnonymous**.
- Relación: Los **Messages** están asociados a un **Draw** específico y son enviados por **Users**.

6. Wishlist (Lista de Deseos):

- Contiene los artículos deseados por un usuario para un sorteo determinado. Los atributos incluyen **id**, **userId**, **drawId**, y una lista de **items** (productos deseados).
- Relación: Un **User** tiene una única **Wishlist** asociada a cada **Draw** (relación 1:1).

7. Item (Artículo):

- Representa un artículo individual dentro de la **Wishlist** de un usuario, con atributos como **name** (nombre del artículo) y **url** (enlace al producto).
- Relación: Un **Item** pertenece a una **Wishlist** (relación 1:N).

8. Notification (Notificación):

- Representa una notificación enviada a un **User**. Contiene atributos como **id**, **userId**, **message**, y **isRead**.
- Relación: Una **Notification** está asociada a un único **User** (relación 1:1).

Clases de Servicios y Utilitarios:

1. AuthService (Servicio de Autenticación):

- Encapsula la lógica para la gestión de usuarios mediante **Firebase Authentication**, con métodos para autenticar usuarios y gestionar el registro.

2. FirebaseService (Servicio de Firebase):

- Encapsula las operaciones con la base de datos **Firebase Firestore**, gestionando la lectura y escritura de datos para las clases principales (por ejemplo, **Group**, **Draw**, **User**).

3. GroupUtils (Utilitarios de Grupo):

- Contiene métodos lógicos para gestionar las reglas de los sorteos, como la asignación de regalos y las exclusiones.

Relaciones entre las Clases:

- **User** → **Group**: Un **User** puede pertenecer a muchos **Groups**, y un **Group** puede tener muchos **Users** (relación **N:M**).
- **Group** → **Draw**: Un **Group** puede tener múltiples **Draws** (relación **1:N**).
- **Draw** → **Assignment**: Un **Draw** tiene muchas **Assignments** que relacionan a los usuarios que regalan y los que reciben (relación **1:N**).
- **Draw** → **Message**: Un **Draw** puede tener muchos **Messages** (relación **1:N**).
- **User** → **Wishlist**: Un **User** tiene una única **Wishlist** para cada **Draw** (relación **1:1**).
- **Wishlist** → **Item**: Una **Wishlist** contiene muchos **Items** (relación **1:N**).
- **User** → **Notification**: Un **User** puede recibir muchas **Notifications** (relación **1:N**).

*“La imagen del diagrama UML se encuentra (Figura C-5 en anexo)”

Resumen:

Este diagrama de clases refleja la estructura de datos y servicios necesarios para gestionar los sorteos de "Amigo Invisible". Las relaciones entre las clases están organizadas de manera eficiente para permitir una interacción fluida entre los usuarios, los grupos y los sorteos. Las clases de servicio, como **AuthService** y **FirebaseService**, gestionan la autenticación y la comunicación con la base de datos, mientras que las clases de datos como **User**, **Group**, **Draw**, y **Assignment** gestionan las entidades clave del sistema.

4.3 Diagrama de Secuencia

El Diagrama de Secuencia muestra la interacción entre los diferentes objetos de la aplicación Amigo Invisible a lo largo del tiempo, describiendo cómo los mensajes y eventos fluyen entre los componentes en un escenario específico. En este apartado, se modelan dos escenarios clave: "Creación de un Sorteo" y "Asignación de un Regalo", detallando cómo interactúan los objetos clave del sistema: Usuario, Interfaz de Usuario (UI), API, Base de Datos, y Notificaciones.

Identificación de Escenarios:

Escenario principal: Creación de un Sorteo

1. **Usuario** interactúa con la **Interfaz de Usuario** para crear un nuevo sorteo, proporcionando detalles como el nombre del sorteo, las fechas de inicio y fin, y los participantes.
2. La **Interfaz de Usuario** toma esta información y la envía a la **API** para procesar la creación del sorteo.
3. La **API** guarda el sorteo en la **Base de Datos** (en este caso, Firebase).
4. Una vez el sorteo ha sido guardado correctamente, la **API** envía una **notificación** a los participantes del sorteo.
5. Finalmente, la **Interfaz de Usuario** muestra una confirmación al **Usuario**.

Escenario Secundario: Asignación de un Regalo

1. **API** obtiene los participantes del sorteo desde la **Base de Datos**.
2. **API** realiza la asignación de los participantes para el sorteo, estableciendo a quién le corresponde regalar a quién.
3. La **API** guarda las asignaciones en la **Base de Datos**.
4. **API** envía una **Notificación** a cada participante, informándoles de quién es el destinatario de su regalo.
5. **Notificaciones** confirman el envío de las notificaciones a los participantes.

Identificación de Objetos Participantes:

En los escenarios de creación de un sorteo y asignación de un regalo, los siguientes objetos participan activamente en el proceso:

- **Usuario:** El **Usuario** inicia el proceso de creación del sorteo y realiza las acciones para asignar los regalos.
- **Interfaz de Usuario (UI):** La **UI** recibe la entrada del **Usuario** y pasa la información a la **API**.
- **API:** La **API** procesa la creación del sorteo, guarda la información en la **Base de Datos** y maneja la asignación de regalos.
- **Base de Datos:** La **Base de Datos** almacena la información relacionada con el sorteo, los participantes y las asignaciones.
- **Notificaciones:** El servicio de **Notificaciones** envía alertas a los participantes del sorteo para mantenerlos informados.

Modelado de la Secuencia de Mensajes:

A continuación se describe la secuencia de mensajes que fluye entre los objetos durante los dos escenarios:

Creación de un Sorteo:

1. Usuario → Interfaz de Usuario (UI):

- Mensaje: crearSorteo(nombre: String, fechalinicio: Date, fechaFin: Date)
- Descripción: El Usuario interactúa con la Interfaz de Usuario para crear el sorteo, proporcionando la información necesaria.

2. Interfaz de Usuario (UI) → API:

- Mensaje: crearSorteo(datosSorteo)
- Descripción: La Interfaz de Usuario pasa los detalles del sorteo a la API para que procese la creación y la guarde en la Base de Datos.

3. API → Base de Datos:

- Mensaje: guardarSorteo(sorteo)
- Descripción: La API recibe los datos y realiza la operación de guardar el sorteo en la Base de Datos.

4. Base de Datos → API:

- Mensaje de retorno: sorteoguardado()
- Descripción: La Base de Datos devuelve una confirmación a la API indicando que el sorteo ha sido guardado correctamente.

5. API → Notificaciones:

- Mensaje: enviarNotificacion(participantes, mensaje)
- Descripción: La API envía una Notificación a los participantes del sorteo para informarles sobre la creación del mismo.

6. Notificaciones → Usuario:

- Mensaje de retorno: notificaciónEnviada()
- Descripción: La Notificación se envía a los dispositivos de los participantes, confirmando que el sorteo ha sido creado y que deben estar atentos a sus asignaciones.

7. Interfaz de Usuario → Usuario:

- Mensaje: mostrarConfirmación(sorteoCreado)
- Descripción: Finalmente, la Interfaz de Usuario muestra un mensaje de confirmación al Usuario, indicándole que el sorteo ha sido creado correctamente.

Asignación de un Regalo:

1. API → Base de Datos:

- Mensaje: obtenerParticipantes()
- Descripción: La API obtiene la lista de participantes del sorteo desde la Base de Datos.

2. API → API:

- Mensaje: realizarAsignaciones()
- Descripción: La API asigna a los participantes para que cada uno sepa a quién debe regalar.

3. API → Base de Datos:

- Mensaje: guardarAsignaciones(asignaciones)
- Descripción: La API guarda las asignaciones de los regalos en la Base de Datos.

4. API → Notificaciones:

- Mensaje: enviarNotificacion(participantes, asignación)
- Descripción: La API envía una Notificación a cada participante informando a quién debe regalar.

5. Notificaciones → Usuario:

- Mensaje de retorno: notificaciónEnviada()
- Descripción: Las Notificaciones confirman que el mensaje ha sido enviado a los participantes.

Representación de la Línea de Vida:

Cada objeto involucrado en el proceso tiene su propia línea de vida, que es representada por una línea vertical que indica el tiempo de actividad de ese objeto. Las activaciones se representan mediante rectángulos colocados a lo largo de la línea de vida, mostrando los momentos en que cada objeto está procesando un mensaje.

- **Usuario:** La línea de vida del **Usuario** se activa cuando inicia la creación del sorteo.
- **Interfaz de Usuario (UI):** La **UI** tiene una activación cuando procesa los datos del sorteo y los pasa a la **API**.
- **API:** La **API** se activa cuando recibe los datos, guarda el sorteo en la **Base de Datos** y envía las notificaciones.
- **Base de Datos:** La **Base de Datos** se activa cuando guarda la información del sorteo y las asignaciones de los regalos.
- **Notificaciones:** El servicio de **Notificaciones** se activa para enviar las notificaciones a los participantes.

Creación del Diagrama de Secuencia:

*“La imagen del diagrama de Secuencia escenario “Sorteo” se encuentra (Figura C-6 en anexo)”

*“La imagen del diagrama de Secuencia escenario “Asignación Regalo” se encuentra (Figura C-7 en anexo)”

Resumen del Flujo:

1. El **Usuario** interactúa con la **Interfaz de Usuario** para crear un sorteo.
2. La **Interfaz de Usuario** pasa los datos del sorteo a la **API**.
3. La **API** guarda el sorteo en la **Base de Datos**.
4. Una vez guardado, la **API** envía una notificación a los participantes.
5. Las **Notificaciones** confirman que el mensaje ha sido enviado a los participantes.
6. En el escenario de **Asignación de un Regalo**, la **API** realiza las asignaciones y envía notificaciones a los participantes.

Este flujo muestra cómo se gestionan las interacciones entre los distintos componentes del sistema para llevar a cabo la creación de un sorteo y la asignación de regalos, manteniendo a los usuarios informados.

4.4 Diagrama de casos de uso

El **Diagrama de Casos de Uso** es una representación gráfica que describe las interacciones entre los usuarios y el sistema. Este diagrama es esencial para comprender los requisitos funcionales del sistema y proporciona una visión clara de cómo los usuarios utilizarán las funcionalidades de la aplicación.

En este apartado, se modela el caso de uso para el **Usuario** de la aplicación **Amigo Invisible**, detallando cómo este interactúa con el sistema para realizar las funciones más importantes, como la creación de sorteos, la visualización de resultados y la recepción de notificaciones.

Identificación de Actores

El **actor** en este sistema es el **Usuario**, quien interactúa con el sistema de diversas maneras. A continuación se identifican los actores principales:

- **Usuario:** El **Usuario** es el único actor en el sistema, quien se encarga de realizar todas las acciones principales dentro de la aplicación. Los usuarios pueden crear sorteos, ver los resultados de dichos sorteos, recibir notificaciones sobre las asignaciones de regalos, y gestionar su sesión en la aplicación.
- **Notificación (actor técnico):** Aunque no es un "usuario" convencional, el servicio de **Notificación** juega un papel importante como actor técnico. Este actor interactúa con el sistema para enviar alertas y mensajes de notificación a los **Usuarios** informándoles sobre eventos importantes, como la creación de un sorteo o la asignación de un regalo.

Identificación de Casos de Uso

El actor principal del sistema es el **Usuario**, quien puede adoptar dos funciones distintas dentro del flujo de la aplicación:

- **Organizador**, encargado de crear sorteos, añadir participantes, definir reglas de exclusión y enviar notificaciones.
- **Participante**, quien accede a la asignación de regalos, recibe notificaciones automáticas y puede comunicarse de forma anónima.

Esta diferenciación se modela dentro del mismo actor, ya que no representa un tipo de cuenta distinto, sino una variación del comportamiento en función del rol asumido en cada sorteo.

A continuación se detallan los casos de uso principales de la aplicación **Amigo Invisible**:

Casos de Uso para el Usuario:

1. Iniciar sesión:

- **Descripción:** El **Usuario** accede a la aplicación ingresando sus credenciales.
- **Flujos Alternativos:Error:** Si está habilitada, puede activar la autenticación de dos factores (2FA) como medida de seguridad adicional.

2. Crear Sorteo(Organizador):

- **Descripción:** El **Usuario** crea un sorteo, proporcionando información como el nombre del sorteo, importe del regalo, y los participantes.
- **Flujos Alternativos:Error:** Si los participantes no son suficientes, el sistema mostrará un mensaje de error y no permitirá la creación del sorteo.

3. Ver Resultados del Sorteo:

- **Descripción:** El **Usuario** consulta los resultados del sorteo para saber a quién debe regalar.
- **Flujos Alternativos:Error:** Si el sorteo no ha sido realizado o tiene algún error, el **Usuario** no podrá ver los resultados.

4. Recibir Notificación de Asignación:

- **Descripción:** El **Usuario** recibe una notificación que le informa a quién debe regalar en el sorteo.
- **Flujos Alternativos:Error:** Si no se han realizado las asignaciones, el **Usuario** no recibe la notificación.

5. Enviar Mensaje Anónimo(Participante):

- **Descripción:** Permite enviar mensajes sin revelar la identidad al receptor, manteniendo el misterio del sorteo.
- **Flujos Alternativos:Error:** Si el número de mensajes permitidos en la versión gratuita se supera, se notificará al usuario.

6. Cerrar sesión:

- **Descripción:** El **Usuario** cierra sesión en la aplicación después de terminar su interacción.

Casos de Uso para el Actor Notificación:

1. Enviar Notificación:

- **Descripción:** El servicio de **Notificaciones** envía un mensaje a los **Usuarios** informándoles sobre los eventos importantes, como la creación del sorteo o la asignación de regalos.
- **Escenario Alternativo:Error:** Si la **API** no puede obtener los datos necesarios, la notificación no será enviada.

Establecimiento de Relaciones

En el diagrama de casos de uso, se deben establecer relaciones claras entre los **actores** y los **casos de uso**.

- **Usuario → Crear Sorteo:** El **Usuario** puede crear uno o más sorteos.
- **Usuario → Ver Resultados del Sorteo:** El **Usuario** puede ver los resultados del sorteo que ha creado o al que pertenece.
- **Usuario → Recibir Notificación:** El **Usuario** recibe notificaciones de asignación de regalos.
- **Usuario → Enviar Mensaje Anónimo:** El **Usuario** puede enviar mensajes anónimos con las personas del mismo grupo.
- **Usuario → Iniciar sesión / Cerrar sesión:** El **Usuario** puede iniciar y cerrar sesión para acceder al sistema.

En cuanto al actor **Notificación**, este se conecta con el caso de uso **Enviar Notificación** para informar a los **Usuarios** sobre los cambios o eventos que ocurren en el sistema (como la creación de sorteos o la asignación de regalos).

Definición de Inclusiones y Extensiones (Opcional)

Aunque no se necesitan extensiones complejas en este caso, se pueden utilizar relaciones de **inclusión** o **extensión** para modelar casos más detallados.

- El caso de uso **Iniciar sesión** puede incluir la autenticación de **2FA** como extensión, lo que significa que el **Usuario** puede optar por usar una capa adicional de seguridad al iniciar sesión.

Creación del Diagrama de Casos de Uso

*“La imagen del diagrama de Casos de Uso se encuentra (Figura C-8 en anexo)”

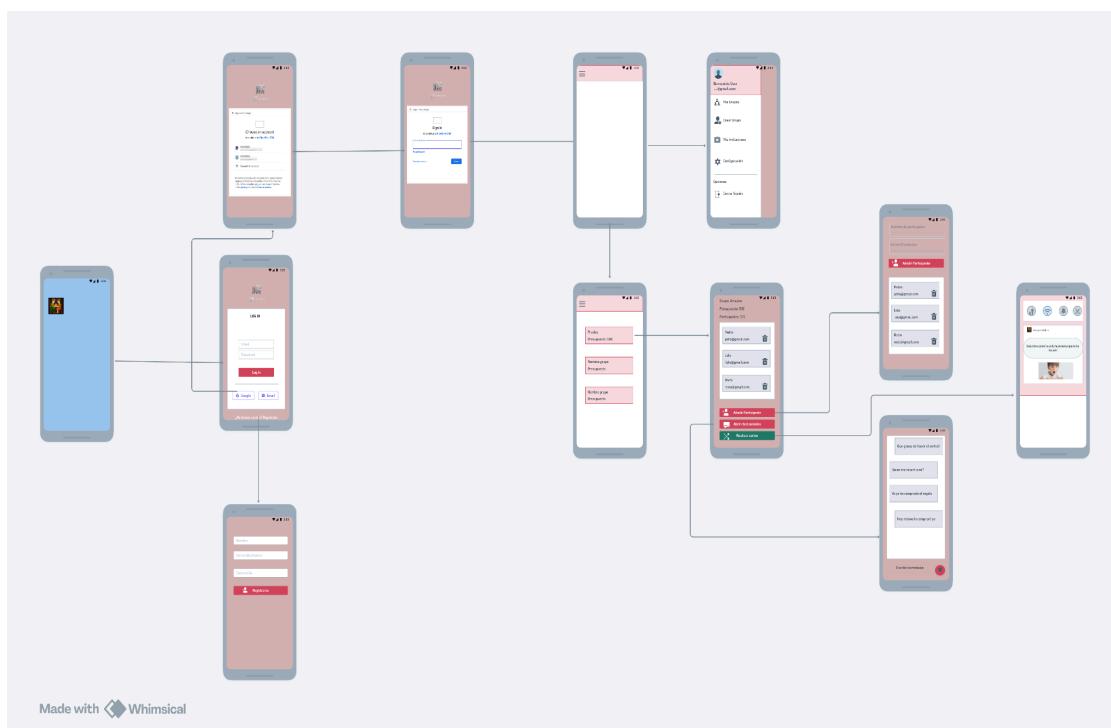
Resumen del Flujo

1. El **Usuario** interactúa con la **Interfaz de Usuario** para crear un sorteo, ver los resultados o gestionar su sesión.
2. La **Interfaz de Usuario** pasa los datos del sorteo a la **API**.
3. La **API** guarda el sorteo en la **Base de Datos**.
4. La **API** envía una notificación a los **Usuarios** informándoles sobre el sorteo y la asignación de regalos.
5. Las **Notificaciones** confirman que el mensaje ha sido enviado a los participantes.

Este flujo describe cómo se gestionan las interacciones entre los distintos componentes del sistema para llevar a cabo la creación de un sorteo, ver los resultados y asignar regalos, manteniendo a los **Usuarios** informados a través de notificaciones.

4.5 Mockup de la App(Amigo Invisible)

El **mockup** es una representación visual que muestra la interfaz de usuario (UI) de la aplicación **Amigo Invisible**, permitiendo visualizar cómo interactuarán los usuarios con la aplicación en diferentes dispositivos (smartphones y tablets) y en diversos idiomas. Es un paso esencial en el diseño de la experiencia de usuario (UX) y ayuda a mejorar la accesibilidad y la facilidad de uso de la app.

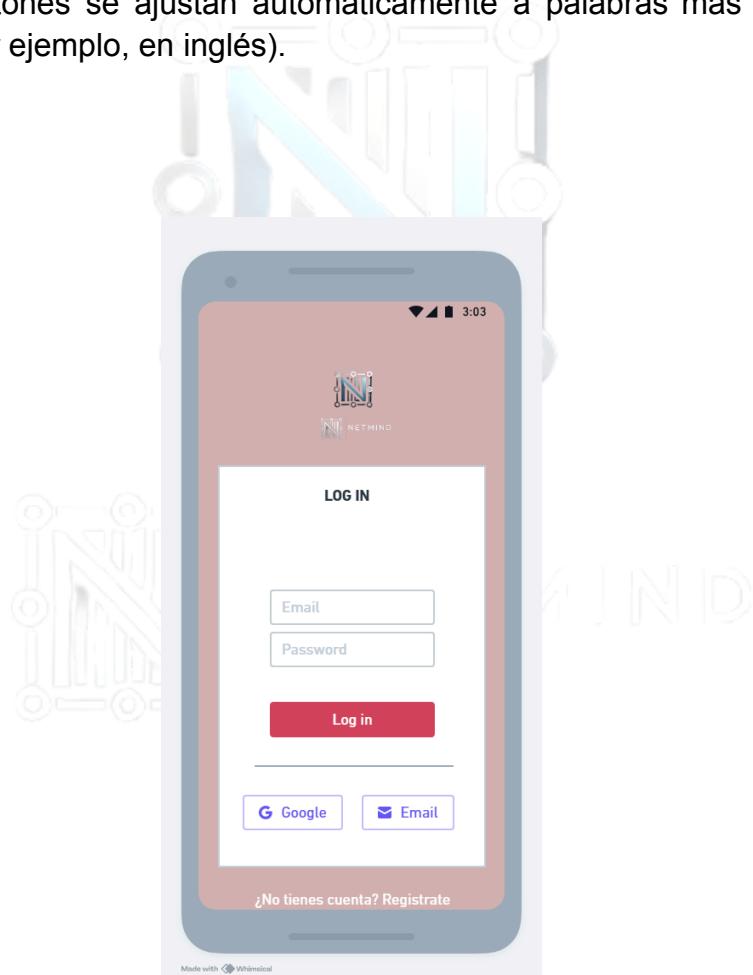


1. Pantalla de Inicio de Sesión / Registro

- Campos de texto: *Correo electrónico*, *Contraseña* (con etiquetas traducidas).
- Botones:
 - *Iniciar sesión*
 - *Registrarse* (multilingüe: "Log In" / "Sign Up")
 - *Iniciar sesión cuenta Google*
 - *Iniciar sesión con otra cuenta* (Outlook)

Adaptabilidad:

- Layout centrado, escalable para pantallas grandes (tablets).
- Inputs y botones se ajustan automáticamente a palabras más largas en otros idiomas (por ejemplo, en inglés).



2. Menú Drawer (desplegable lateral)

Una vez el usuario inicia sesión correctamente, accede a la aplicación principal, donde en la esquina superior izquierda se encuentra el **ícono de menú (hamburguesa)** que despliega un **Navigation Drawer** con las siguientes opciones:

- **Mis Grupos:** Listado detallado de los grupos a los que pertenece el usuario.
- **Crear Sorteo:** Acceso directo al formulario para crear un nuevo sorteo.
- **Mis invitaciones:** listado de grupos a los que esta invitado el usuario logueado.
- **Configuración:** Acceso a la configuración personal de la aplicación (idioma, notificaciones, etc.).
- **Cerrar Sesión:** Opción ubicada en la parte inferior del Drawer para salir de la cuenta.

Personalización del perfil:

En la cabecera del Drawer, se muestra el **nombre del usuario, foto de perfil** (o avatar predeterminado) y correo electrónico asociado.

Adaptabilidad:

Drawer de tamaño ajustable, fluido en tablets y móviles.

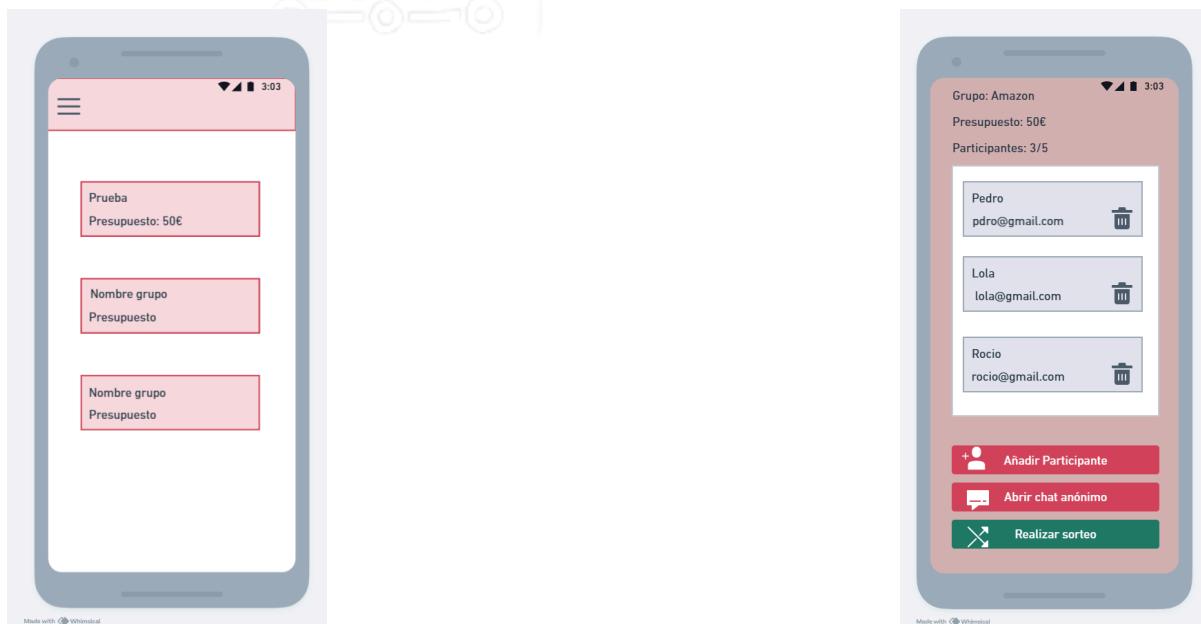


3. Pantalla Principal - Lista de Grupos

- **Vista en forma de tarjetas (Cards):** Cada tarjeta representa un grupo de Amigo Invisible al que el usuario pertenece.
- **Contenido de la tarjeta:**
 - Nombre del grupo.
 - Número de participantes.
 - Estado actual (por ejemplo, “Sorteo pendiente”, “Asignaciones realizadas”).
 - Foto asignada al grupo.

Acciones al pulsar en un grupo:

- Si el usuario es **Administrador del grupo:**
 - Puede **invitar nuevos participantes**.
 - Puede **iniciar el sorteo**.
 - Puede gestionar las exclusiones y reglas.
- Si el usuario es **Participante invitado:**
 - Puede **chatear de manera anónima** con su amigo invisible.
 - Puede **ver la lista de participantes** (sin saber quién es su amigo asignado).



4. Pantalla de Añadir miembros al grupo (Administrador)

- Campos de texto: *Nombre del participante* y *correo electrónico*.
- Botón de selección para añadir participantes.
- Recyclerview con la lista de participantes, sus datos y posibilidad de quitarlo del grupo.



5. Pantalla de Creación/ Resultados del Sorteo

- Campos de texto: *Nombre del sorteo*, *Presupuesto sugerido*, *Fecha límite*.
- Botón de selección para añadir participantes desde la lista de contactos internos.
- Botón de acción principal: *Realizar Sorteo*.
- Visualización privada del destinatario asignado.
- Opción para enviar mensajes anónimos.

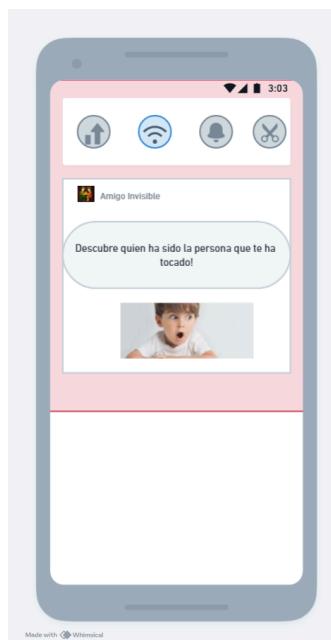
6. Pantalla de Chat Anónimo

- Conversaciones agrupadas por grupo y sorteo.
- Posibilidad de enviar mensajes con identidad oculta.
- Límite de mensajes según tipo de cuenta (gratuita/premium).



7. Pantalla de Notificaciones

- Lista de notificaciones relevantes:
 - “¡Tienes un nuevo amigo invisible asignado!”



4.6 Guía de Estilos

La **Guía de Estilos** establece la base visual de la aplicación **Amigo Invisible**, asegurando **consistencia, claridad y una experiencia de usuario intuitiva** a través de todos los componentes de la interfaz, tanto en dispositivos móviles como tablets, y en distintos idiomas.

Paleta de Colores

Se han seleccionado colores vivos y contrastados para transmitir cercanía, dinamismo y facilidad de uso.

Elemento	Color	Código HEX	Uso
Fondo principal	Rojo suave	#D44646	Color de fondo en pantallas principales (pantalla de inicio, lista de grupos).
Botón de Log In / Registro	Rojo oscuro	#831515	Botones principales de inicio de sesión y registro.
Botón de Confirmar/Finalizar	Verde positivo	#4CAF50	Botones de acciones positivas (crear sorteo, finalizar sorteo).
Fondo secundario	Blanco	#FFFFFF	Fondo de tarjetas (cards) y áreas de contenido.
Texto principal	Gris oscuro	#333333	Textos principales para garantizar buena lectura.
Texto secundario	Gris medio	#666666	Subtítulos o descripciones secundarias.

Justificación de la elección de colores:

- **Rojo (#D44646)**: Asociado a la emoción, celebración y pasión (perfecto para representar el espíritu navideño del Amigo Invisible).
- **Verde (#4CAF50)**: Simboliza éxito, correcto o “ok” (ideal para finalizar acciones exitosas como confirmar el sorteo).
- **Contrastes altos**: Se garantizan altos niveles de contraste para asegurar **accesibilidad** y legibilidad, siguiendo buenas prácticas de UX.

Tipografía

Se ha elegido una fuente moderna, legible y ampliamente compatible en Android:

Elemento	Fuente	Tamaño	Peso	Uso
Títulos	Roboto Bold	24sp	Negrita	Pantallas principales, encabezados.
Subtítulos	Roboto Medium	18sp	Seminegrita	Secciones internas o descripciones cortas.
Texto Principal	Roboto Regular	16sp	Normal	Cuerpo de tarjetas, mensajes, listas.
Texto Secundario / Notas	Roboto Light	14sp	Ligero	Notificaciones, campos secundarios.

Justificación:

- **Roboto:** Fuente recomendada por Material Design de Google, altamente legible incluso en tamaños pequeños y adaptable a diferentes idiomas.
- **Claridad visual:** Se utilizan tamaños diferenciados para mantener jerarquía visual clara y facilitar la navegación rápida.

Iconografía

La iconografía utilizada sigue el **estilo Material Design**, con líneas simples y reconocibles de manera universal.

Elemento	Estilo de Icono	Tamaño	Color
Iconos de navegación	Material Icons Filled	24x24 px	Gris oscuro (#333333)
Icono menú hamburguesa	Material Design	24x24 px	Gris oscuro (#333333)
Iconos en botones (ej: añadir, enviar)	Material Design	20x20 px	Blanco (#FFFFFF) sobre fondo de botón
Iconos de estados (correcto/error)	Material Design	20x20 px	Verde (#4CAF50) / Rojo (#D44646)

Elementos de la Interfaz de Usuario (UI)

Componente	Estilo Definido
Botones principales (acciones positivas):	Fondo verde (#4CAF50), texto blanco, bordes ligeramente redondeados (8px), sombra ligera.
Botones de Login/Registro:	Fondo rojo oscuro (#831515), texto blanco, borde redondeado (8px).
Cards (Grupos):	Fondo blanco, borde suave gris claro (#E0E0E0), sombra suave.
Campos de texto:	Borde gris claro (#CCCCCC), icono opcional de ayuda dentro del campo, espacio entre campos de al menos 12px.
Drawer Menu:	Fondo blanco, iconos Material Design, encabezado con foto y nombre del usuario.
Barras de Navegación:	Fondo blanco o rojo claro, iconos grises, etiquetas traducidas.

Ejemplos de Aplicación de la Guía de Estilos

- **Pantalla de Inicio:** Fondo rojo claro (#D44646), botones de login en rojo oscuro (#831515).
- **Lista de Grupos:** Tarjetas blancas con textos en gris oscuro y botón de acción verde.
- **Chat Anónimo:** Burbujas de chat personalizadas, diferenciando mensajes enviados y recibidos por color suave.

4.7 Justificación de Decisiones de Diseño

La sección de justificación de diseño expone las razones detrás de las decisiones tomadas para la interfaz de usuario (UI) de la aplicación **Amigo Invisible**, asegurando que sea accesible, usable y adaptable en diferentes dispositivos e idiomas. Todas las decisiones se han basado en principios de diseño, necesidades del usuario, usabilidad y guías oficiales de diseño adaptativo.

Explicación de Decisiones de Diseño Clave

- **Disposición de elementos UI:**

Se eligió una estructura basada en layouts verticales y tarjetas (Cards) para listar los grupos, con menús laterales (Drawer) accesibles mediante un botón de menú hamburguesa. Esta disposición permite una navegación clara tanto en pantallas pequeñas como en tablets más grandes. El diseño es responsivo, adaptándose a múltiples resoluciones y orientaciones.

- **Colores y Tipografía:**

La paleta de colores utiliza tonos cálidos y festivos como el rojo (#D44646) y verde (#4CAF50) para transmitir celebración y acción positiva. Se seleccionó Roboto como tipografía principal, por su excelente legibilidad incluso en idiomas de diferentes longitudes (por ejemplo, alemán o inglés).

- **Iconos e Imágenes:**

Se emplearon iconos Material Design simples y reconocibles para asegurar una interpretación intuitiva, independientemente del idioma o cultura. Los iconos son blancos sobre fondos de colores o grises para mantener claridad.

- **Navegación Multilingüe:**

Todos los botones y etiquetas fueron diseñados para contener textos traducibles. Se previó espacio adicional en botones y campos de texto para adaptarse a idiomas más largos, como el alemán.

- **Retroalimentación al usuario:**

Se diseñaron notificaciones claras y accesibles para informar al usuario sobre el éxito o fallo de acciones, empleando mensajes cortos, iconos de éxito/error y traducciones en tiempo real.

Referencia a Principios de Diseño

- **Consistencia:**

Se utilizó el mismo estilo visual (botones, tipografía, iconografía) en todas las pantallas, asegurando una experiencia uniforme (Principio de Consistencia).

- **Claridad y Jerarquía Visual:**

Gracias al uso de tamaños de fuente diferenciados y contrastes de color altos, se facilita la escaneabilidad rápida de la información importante.

- **Feedback Inmediato:**

Cada acción del usuario (como crear un sorteo, enviar un mensaje) proporciona un feedback visual inmediato, en forma de snackbars, alertas o cambios de

pantalla (Principio de Retroalimentación).

- **Guías utilizadas:**

El diseño se alineó con las Material Design Guidelines de Google y se adaptó para soportar cambios dinámicos de idioma y accesibilidad.

Consideraciones de Usabilidad

- **Facilitación de tareas:**

El flujo de creación de sorteos minimiza los pasos necesarios para añadir participantes e iniciar un sorteo, todo accesible en pocos clics.

- **Minimización de errores:**

Se incluyen validaciones en formularios (como correos electrónicos válidos o límites de participantes) y mensajes de error traducidos, para evitar confusión en cualquier idioma.

- **Adaptabilidad Multidispositivo:**

La app está diseñada para ser plenamente funcional en móviles pequeños, pantallas medianas (phablets) y tablets, asegurando botones y textos accesibles a todos los tamaños de dedo/visión.

Necesidades del Usuario

- **Adaptabilidad cultural e idiomática:**

Pensando en usuarios de distintos países, se incorporó soporte multilingüe completo desde el inicio del diseño. Todo el contenido textual está preparado para ser traducido sin afectar el layout.

- **Preferencias del usuario:**

Se introdujo la personalización básica en el Drawer (mostrar nombre y foto de perfil) para aumentar el sentido de pertenencia y familiaridad.

- **Accesibilidad:**

Colores contrastados, fuentes legibles y controles de tamaño generoso permiten que personas con visión reducida o dificultades motoras también puedan interactuar fácilmente con la aplicación.

Ejemplos Concretos de Implementación

- **Botón de Crear Sorteo:**

Ubicado en la parte inferior derecha de la pantalla de grupos, utiliza un ícono de "+" y un fondo verde (#4CAF50). El texto es traducible ("Crear" / "Create" / "Créer") y su posición sigue buenas prácticas para accesibilidad en móviles.

- **Menú Drawer:**

Incluye secciones claras ("Inicio", "Crear Sorteo", "Mis Grupos", "Configuración", "Cerrar Sesión"), todas traducibles y accesibles desde cualquier pantalla

mediante un gesto o ícono.

- **Mensajes de error y éxito:**

Después de registrar un usuario o completar un sorteo, se presentan notificaciones breves adaptadas al idioma activo del sistema.

Referencias Utilizadas

- Material Design Guidelines, Google.
- Principios de UX/UI de Accesibilidad y Diseño Multilingüe, W3C Web Accessibility Initiative.
- Firebase UX Best Practices, documentación oficial



5. Implementación

5.1 Desarrollo del Código Fuente

Descripción general

Durante el desarrollo de la aplicación *Amigo Invisible*, se ha seguido una estrategia basada en buenas prácticas de programación, diseño modular, uso de recursos externos (strings.xml, colors.xml, etc.) y soporte multilingüe. Se ha utilizado el lenguaje **Java** con **Android Studio** como entorno principal, y Firebase como backend para autenticación, almacenamiento de datos y envío de correos.

Buenas prácticas aplicadas

Claridad y legibilidad

- **Nombres descriptivos:** las variables y funciones utilizan nombres autoexplicativos, por ejemplo:

```
private TextView textoNombreGrupo;  
private void realizarSorteo() { ... }
```
- **Formato consistente:** todo el proyecto utiliza sangría de 4 espacios y sigue la convención Google Java Style Guide.
- **Código limpio:** se eliminaron métodos no utilizados y se dividieron funciones grandes en bloques manejables.

Modularización y reutilización

- Se implementaron clases utilitarias como GroupUtils, EmailSender, FirestoreHelper, etc. para separar la lógica del negocio de la interfaz gráfica.
- Se reutiliza lógica común como el envío de correos o validación de miembros en múltiples pantallas.

Ejemplo:

```
EmailSender.enviarCorreo(email, asunto, textoPlano,  
contenidoHTML);
```

Principios SOLID

- **Responsabilidad única:** Cada clase cumple un único propósito (`GroupDetailActivity` maneja interfaz; `GroupUtils` maneja lógica de sorteo).
- **Inversión de dependencias:** Se usa Firebase como servicio externo, desacoplado mediante clases de utilidad.
- **Soporte multilingüe y adaptabilidad**

Se ha creado estructura de `strings.xml` en:

`res/values/strings.xml`

Por ejemplo:

```
<string name="mensaje_sorteo_exito">Sorteo realizado con éxito!</string>
```

- Se utilizaron `ConstraintLayout` para garantizar una correcta visualización en diferentes tamaños de pantalla y densidades.

Comentarios y documentación

Cada función clave tiene su documentación en formato Javadoc:

```
/**  
 * Ejecuta el sorteo de amigo invisible para un grupo.  
 * @param groupId ID del grupo  
 * @param onSuccess Callback al finalizar con éxito  
 * @param onError Callback si ocurre un error  
 */  
public static void realizarSorteo(String groupId, Runnable onSuccess,  
Consumer<String> onError) { ... }
```

- Los bloques complejos están comentados en línea para mayor comprensión.

Manejo de errores

Se utilizan bloques `try-catch` y callbacks `onFailureListener` para detectar y mostrar errores al usuario:

```
.addOnFailureListener(e -> {  
    Toast.makeText(context, "Error al guardar asignación",  
    Toast.LENGTH_SHORT).show();  
});
```

Eficiencia

- Se utilizan `AtomicInteger` para coordinar múltiples tareas concurrentes durante el sorteo.

Validación de entradas

- Se validan emails y nombres antes de enviar notificaciones.
- Se comprueba que haya al menos 2 miembros antes de permitir el sorteo.

CÓDIGO FUENTE

5.1.1 Activities

Índice de las activities:

1. AddMemberActivity.java
2. AssignedPersonActivity.java
3. GroupChatActivity.java
4. GroupDetailActivity.java
5. MainActivity.java
6. RegisterActivity.java
7. WelcomeActivity.java
8. WishlistActivity.java



A continuación se muestra el código correspondiente a cada una de ellas:

1. AddMemberActivity.java

```
package NetMind.amigoinvisible.activities;
import android.os.Bundle;
import android.text.TextUtils;
import android.widget.EditText;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import com.firebaseio.ui.firestore.FirestoreRecyclerOptions;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.firestore.FirebaseFirestore;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.adapters.MemberAdapter;
import NetMind.amigoinvisible.models.Member;
/**
 * Actividad que permite al usuario añadir participantes manualmente a un grupo.
 * También muestra los miembros añadidos usando un RecyclerView conectado a Firestore.
 */
public class AddMemberActivity extends AppCompatActivity {
```

```

private EditText campoNombre, campoCorreo;
private FirebaseFirestore baseDatos;
private FirebaseAuth auth;
private String idGrupo;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_add_member);

    baseDatos = FirebaseFirestore.getInstance();
    auth = FirebaseAuth.getInstance();
    idGrupo = getIntent().getStringExtra("groupId");

    campoNombre = findViewById(R.id.editTextMemberName);
    campoCorreo = findViewById(R.id.editTextMemberEmail);

    findViewById(R.id.btnAddMember).setOnClickListener(v ->
        anadirParticipantes());
}

RecyclerView recyclerMiembros = findViewById(R.id.recyclerViewMembers);
recyclerMiembros.setLayoutManager(new LinearLayoutManager(this));

    FirestoreRecyclerOptions<Member> opciones = new
FirestoreRecyclerOptions.Builder<Member>()

.setQuery(baseDatos.collection("groups").document(idGrupo).collection("members"),
Member.class)
.setLifecycleOwner(this)
.build();

MemberAdapter adaptador = new MemberAdapter(opciones, idGrupo);
recyclerMiembros.setAdapter(adaptador);
}

/**
 * Añade un nuevo participante a la subcolección 'members' del grupo
actual,
 * validando que no supere el máximo de participantes definido.
 */
private void anadirParticipantes() {
    String nombre = campoNombre.getText().toString().trim();
    String correo = campoCorreo.getText().toString().trim();

    if (TextUtils.isEmpty(nombre) || TextUtils.isEmpty(correo)) {
        Toast.makeText(this, R.string.mensaje_campos_obligatorios,
Toast.LENGTH_SHORT).show();
        return;
    }

    if (!validarCorreo(correo)) {
        Toast.makeText(this, R.string.mensaje_correo_invalido,
Toast.LENGTH_SHORT).show();
        return;
    }

    baseDatos.collection("groups").document(idGrupo).get()
        .addOnSuccessListener(grupo -> {
            Long maximo = grupo.getLong("maxParticipants");
            List<String> miembros = (List<String>)
grupo.get("members");

```

```

        if (miembros != null && maximo != null && miembros.size() >= maximo) {
            Toast.makeText(this, R.string.mensaje_grupo_completo,
            Toast.LENGTH_SHORT).show();
            return;
        }

        Map<String, Object> datosMiembro = new HashMap<>();
        datosMiembro.put("name", nombre);
        datosMiembro.put("email", correo);

        // Solo se guarda la invitación con el correo, sin modificar el array de miembros
        baseDatos.collection("groups").document(idGrupo)
            .collection("members").document(correo)
            .set(datosMiembro)
            .addOnSuccessListener(unused ->
limpiarCamposYMostrar())
            .addOnFailureListener(e ->
                Toast.makeText(this, "Error al añadir miembro: " + e.getMessage(), Toast.LENGTH_SHORT).show());
        })
        .addOnFailureListener(e ->
            Toast.makeText(this, "Error al verificar límite de grupo: " + e.getMessage(), Toast.LENGTH_SHORT).show());
    }

    /**
     * Limpia los campos de texto y muestra un mensaje de confirmación.
     */
    private void limpiarCamposYMostrar() {
        Toast.makeText(this, R.string.mensaje_participante_añadido,
        Toast.LENGTH_SHORT).show();
        campoNombre.setText("");
        campoCorreo.setText("");
    }

    /**
     * Valida que el correo tenga un formato válido usando una expresión regular.
     *
     * @param correo Dirección de correo electrónico introducida por el usuario.
     * @return true si el formato es válido, false en caso contrario.
     */
    private boolean validarCorreo(String correo) {
        String patronCorreo = "[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.\w{2,4}";
        return correo.matches(patronCorreo);
    }
}

```

2. AssignedPersonActivity.java

```
package NetMind.amigoinvisible.activities;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import com.bumptech.glide.Glide;
import com.google.firebase.firestore.DocumentSnapshot;
import com.google.firebase.firestore.FirebaseFirestore;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.adapters.AssignedWishlistAdapter;
import NetMind.amigoinvisible.models.WishlistItem;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
/**
 * Actividad que muestra al usuario los datos de la persona que le ha tocado
 * como "amigo invisible" dentro de un grupo específico.
 * Incluye su nombre, foto de perfil (si tiene) y su lista de deseos.
 */
public class AssignedPersonActivity extends AppCompatActivity {

    // Instancia de Firestore
    private FirebaseFirestore db;

    // Datos recibidos por Intent
    private String groupId;
    private String assignedId;

    // Elementos de la interfaz
    private TextView nombreAsignadoTextView;
    private ImageView fotoAsignado;
    private RecyclerView recyclerView;

    // Adaptador y lista de deseos del asignado
    private AssignedWishlistAdapter adapter;
    private final List<WishlistItem> deseosAsignado = new ArrayList<>();

    /**
     * Metodo principal del ciclo de vida de la actividad.
     * Valida los datos recibidos por intent, inicializa la interfaz y carga
     los datos desde Firestore.
     *
     * @param savedInstanceState Estado guardado de la actividad.
     */
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_assigned_person);

        // Obtener parámetros del intent
        groupId = getIntent().getStringExtra("groupId");
        assignedId = getIntent().getStringExtra("assignedId");

        // Enlazar vistas
        nombreAsignadoTextView = findViewById(R.id.textNombreAsignado);
        fotoAsignado = findViewById(R.id.imageAsignado);
        recyclerView = findViewById(R.id.recyclerListaDeseos);
    }
}
```

```

        // Validar los datos mínimos requeridos
        if (groupId == null || assignedId == null) {
            Toast.makeText(this, R.string.error_datos_asignado,
Toast.LENGTH_SHORT).show();
            finish();
            return;
        }

        // Inicializar Firestore
        db = FirebaseFirestore.getInstance();

        // Configurar RecyclerView
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        adapter = new AssignedWishlistAdapter(deseosAsignado, this);
        recyclerView.setAdapter(adapter);

        // Cargar los datos del miembro asignado
        cargarDatosAsignado();
    }

    /**
     * Realiza una consulta a Firestore para obtener los datos del miembro
     * asignado.
     * Si es exitoso, llama a mostrarDatosAsignado().
     */
    private void cargarDatosAsignado() {
        db.collection("groups")
            .document(groupId)
            .collection("members")
            .document(assignedId)
            .get()
            .addOnSuccessListener(this::mostrarDatosAsignado)
            .addOnFailureListener(e ->
                Toast.makeText(this, R.string.error_datos_asignado,
Toast.LENGTH_SHORT).show());
    }

    /**
     * Muestra los datos del miembro asignado en la interfaz:
     * nombre, imagen de perfil y lista de deseos interactiva.
     *
     * @param doc Documento Firestore con la información del miembro.
     */
    private void mostrarDatosAsignado(DocumentSnapshot doc) {
        String nombre = doc.getString("name");
        String photoUrl = doc.getString("photoUrl");
        Object deseos = doc.get("wishlist");

        nombreAsignadoTextView.setText(getString(R.string.text_asignado_nombre_placeholder, nombre));

        if (photoUrl != null && !photoUrl.isEmpty()) {
            Glide.with(this).load(photoUrl).placeholder(R.drawable.ic_user).into(fotoAsignado);
        }

        TextView textoListaVacia = findViewById(R.id.textListaVacia);
        deseosAsignado.clear();

        if (deseos instanceof List<?>) {

```

```
        for (Object item : (List<?>) deseos) {
            if (item instanceof Map) {
                Map<?, ?> mapa = (Map<?, ?>) item;
                String titulo = String.valueOf(mapa.get("title"));
                String url = mapa.get("url") != null ?
String.valueOf(mapa.get("url")) : "";
                deseosAsignado.add(new WishlistItem(titulo, url));
            }
        }

        textoListaVacia.setVisibility(deseosAsignado.isEmpty() ?
View.VISIBLE : View.GONE);
        adapter.notifyDataSetChanged();
    } else {
        textoListaVacia.setVisibility(View.VISIBLE);
    }
}
}
```



3. GroupChatActivity.java

```

package NetMind.amigoinvisible.activities;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebaseio.firebaseio.*;
import java.util.*;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.adapters.MessageAdapter;
import NetMind.amigoinvisible.models.Message;

/**
 * Actividad encargada de gestionar el chat grupal del Amigo Invisible.
 * Permite a los usuarios enviar mensajes anónimos y visualizar la conversación
en tiempo real.
 * Utiliza Firebase Firestore como backend para almacenar y escuchar los
mensajes.
 */
public class GroupChatActivity extends AppCompatActivity {

    // Instancia de Firestore para acceso a base de datos
    private FirebaseFirestore db;
    // Identificador del grupo al que pertenece el chat
    private String groupId;
    // Componentes de interfaz
    private RecyclerView recyclerView;
    private MessageAdapter messageAdapter;
    private final List<Message> messagesList = new ArrayList<>();
    private EditText messageInput;
    private Button sendButton;

    /**
     * Metodo de inicialización de la actividad.
     * Obtiene el ID del grupo, configura la interfaz y la conexión a Firebase.
     *
     * @param savedInstanceState estado guardado previamente (no usado aquí).
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_group_chat);

        // Obtener ID del grupo desde el intent
        groupId = getIntent().getStringExtra("groupId");
        if (groupId == null) {
            Toast.makeText(this, getString(R.string.error_group_id_missing),
Toast.LENGTH_SHORT).show();
            finish();
            return;
        }

        inicializarFirebase();
        inicializarUI();
        cargarMensajes();
        configurarBotonEnviar();
    }

    /**

```

```

    * Inicializa la instancia de Firebase Firestore.
    */
private void inicializarFirebase() {
    db = FirebaseFirestore.getInstance();
}

/**
 * Configura los elementos visuales de la interfaz:
 * RecyclerView, campo de texto y botón de envío.
 * También asigna el adaptador y el layout manager.
 */
private void inicializarUI() {
    recyclerView = findViewById(R.id.recyclerView);
    messageInput = findViewById(R.id.messageInput);
    sendButton = findViewById(R.id.sendButton);

    messageAdapter = new MessageAdapter(messagesList);

    LinearLayoutManager layoutManager = new LinearLayoutManager(this);
    layoutManager.setStackFromEnd(true); // Para que empiece mostrando los
últimos mensajes

    recyclerView.setLayoutManager(layoutManager);
    recyclerView.setAdapter(messageAdapter);
}

/**
 * Carga y escucha en tiempo real los mensajes del grupo desde Firestore.
 * Cada cambio se refleja automáticamente en el RecyclerView.
 */
private void cargarMensajes() {
    CollectionReference mensajesRef = db.collection("groups")
        .document(groupId)
        .collection("messages");

    mensajesRef.orderBy("timestamp")
        .addSnapshotListener((QuerySnapshot value,
    FirebaseFirestoreException error) -> {
        if (error != null || value == null) return;

        messagesList.clear();
        for (DocumentSnapshot doc : value) {
            Message message = doc.toObject(Message.class);
            if (message != null) messagesList.add(message);
        }

        messageAdapter.notifyDataSetChanged();
        recyclerView.scrollToPosition(messagesList.size() - 1);
    });
}

/**
 * Configura el botón de envío para capturar el mensaje del usuario,
 * limpiarlo y enviarlo a Firestore si no está vacío.
 */
private void configurarBotonEnviar() {
    sendButton.setOnClickListener(v -> {
        String contenido = messageInput.getText().toString().trim();
        if (!contenido.isEmpty()) {
            enviarMensaje(contenido);
            messageInput.setText("");
        }
    });
}

```

```
}

/**
 * Envía un nuevo mensaje de texto al chat.
 * El mensaje se almacena en Firestore bajo la colección del grupo actual.
 *
 * @param contenido Texto del mensaje introducido por el usuario.
 */
private void enviarMensaje(String contenido) {
    String senderId = FirebaseAuth.getInstance().getCurrentUser().getUid();
    com.google.firebaseio.Timestamp timestamp =
com.google.firebaseio.Timestamp.now();

    Message nuevoMensaje = new Message(senderId, contenido, timestamp,
true);

    db.collection("groups")
        .document(groupId)
        .collection("messages")
        .add(nuevoMensaje)
        .addOnFailureListener(e ->
            Toast.makeText(this,
getString(R.string.error_envio_mensaje), Toast.LENGTH_SHORT).show());
    }
}
```



4. GroupDetailActivity.java

```
package NetMind.amigoinvisible.activities;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import com.bumptech.glide.Glide;
import com.firebaseio.ui.firestore.FirestoreRecyclerOptions;
import com.google.android.material.button.MaterialButton;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.firestore.*;
import com.google.firebase.storage.FirebaseStorage;
import com.google.firebase.storage.StorageReference;

import NetMind.amigoinvisible.utils.GroupUtils;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.adapters.MemberAdapter;
import NetMind.amigoinvisible.models.Member;
/**
 * Actividad encargada de mostrar los detalles de un grupo de Amigo Invisible.
 * Permite al usuario ver y gestionar los participantes, realizar el sorteo,
 * acceder al chat,
 * modificar la imagen del grupo (si es propietario), y acceder a su persona
 * asignada.
 */
public class GroupDetailActivity extends AppCompatActivity {

    // UI
    private TextView textoNombreGrupo, textoPresupuestoGrupo,
    textoContadorGrupo;
    private ImageView imageGroup;
    private MaterialButton botonAgregarMiembro, botonSortear, botonChat,
    botonWishlist, botonVerAsignado;
    private RecyclerView recyclerMiembros;

    // Firebase
    private FirebaseFirestore baseDatos;
    private FirebaseAuth auth;
    private ListenerRegistration listenerMiembros;

    // Variables
    private String idGrupo;
    private String ownerId;
    private boolean sePuedeAñadir = true;
    private static final int PICK_IMAGE_REQUEST = 102;
    private Uri nuevaImagenUri;
    private MemberAdapter adaptadorMiembros;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_group_detail);

        // Inicializa Firebase y obtiene el ID del grupo
        baseDatos = FirebaseFirestore.getInstance();
        auth = FirebaseAuth.getInstance();
        idGrupo = getIntent().getStringExtra("groupId");

        inicializarUI();
        configurarBotones();
        verificarAsignado();
        cargarInformacionGrupo();
        configurarRecyclerView();
    }

    /**
     * Inicializa todos los elementos visuales de la interfaz como TextViews,
     ImageView, RecyclerView y botones
     */
    private void inicializarUI() {
        textoNombreGrupo = findViewById(R.id.textGroupNameDetail);
        textoPresupuestoGrupo = findViewById(R.id.textGroupBudgetDetail);
        textoContadorGrupo = findViewById(R.id.textGroupCountDetail);
        imageGroup = findViewById(R.id.imageGroupDetail);
        botonAgregarMiembro = findViewById(R.id.btnAddMember);
        botonSortear = findViewById(R.id.btnSort);
        botonChat = findViewById(R.id.btnChat);
        botonWishlist = findViewById(R.id.btnWishlist);
        botonVerAsignado = findViewById(R.id.btnViewAssigned);
        recyclerMiembros = findViewById(R.id.recyclerGroupMembersDetail);

        botonSortear.setVisibility(View.GONE);
        botonVerAsignado.setVisibility(View.GONE);
    }

    /**
     * Establece los listeners de los botones de acción disponibles según el
     rol del usuario (miembro o propietario)
     */
    private void configurarBotones() {
        botonAgregarMiembro.setOnClickListener(v -->
verificarYAgregarMiembro()));

        botonSortear.setOnClickListener(v --> new AlertDialog.Builder(this)
            .setTitle(getString(R.string.dialog_titulo_confirmar_sorteo))

            .setMessage(getString(R.string.dialog_mensaje_confirmar_sorteo))
            .setPositiveButton(getString(R.string.dialog_boton_realizar),
(dialog, which) --> realizarSorteo())
            .setNegativeButton(getString(R.string.dialog_boton_cancelar),
null)
            .show());
    }

        botonChat.setOnClickListener(v -->
iniciarActividad(GroupChatActivity.class));
        botonWishlist.setOnClickListener(v -->
iniciarActividad(WishlistActivity.class));

        imageGroup.setOnClickListener(v --> {
            if (esPropietario()) seleccionarNuevaImagen();
            else Toast.makeText(this,
R.string.mensaje_solo_propietario_puede_cambiar_imagen,
Toast.LENGTH_SHORT).show();
        });
    }
}

```

```

    });

    botonVerAsignado.setOnClickListener(v -> mostrarAsignado());
}

/**
 * Inicia una actividad pasando el ID del grupo
 */
private void iniciarActividad(Class<?> clase) {
    Intent intent = new Intent(this, clase);
    intent.putExtra("groupId", idGrupo);
    startActivity(intent);
}

/**
 * Ejecuta la lógica de sorteo del Amigo Invisible con feedback visual
 mediante un ProgressDialog
 */
private void realizarSorteo() {
    ProgressDialog progressDialog = new ProgressDialog(this);

    progressDialog.setMessage(getString(R.string.dialog_realizando_sorteo));
    progressDialog.setCancelable(false);
    progressDialog.show();

    GroupUtils.realizarSorteo(
        idGrupo,
        () -> runOnUiThread(() -> {
            progressDialog.dismiss();
            Toast.makeText(this, R.string.mensaje_sorteo_exito,
Toast.LENGTH_LONG).show();
            botonVerAsignado.setVisibility(View.VISIBLE);
        }),
        error -> runOnUiThread(() -> {
            progressDialog.dismiss();
            Toast.makeText(this,
getString(R.string.mensaje_sorteo_error, error), Toast.LENGTH_LONG).show();
        })
    );
}

/**
 * Verifica si el usuario ya tiene una persona asignada y muestra el botón
 correspondiente
 */
private void verificarAsignado() {
    String uid = auth.getCurrentUser().getUid();
    baseDatos.collection("groups")
        .document(idGrupo)
        .collection("members")
        .document(uid)
        .get()
        .addOnSuccessListener(doc -> {
            if (doc.contains("assignedTo")) {
                botonVerAsignado.setVisibility(View.VISIBLE);
            }
        });
}

/**
 * Obtiene los datos del amigo invisible asignado al usuario y lanza
 AssignedPersonActivity
 */

```

```

private void mostrarAsignado() {
    String uid = auth.getCurrentUser().getUid();

    baseDatos.collection("groups").document(idGrupo).collection("members").document
    (uid)
        .get()
        .addOnSuccessListener(document -> {
            String assignedId = document.getString("assignedTo");
            if (assignedId != null && !assignedId.isEmpty()) {
                Intent intent = new Intent(this,
                AssignedPersonActivity.class);
                intent.putExtra("groupId", idGrupo);
                intent.putExtra("assignedId", assignedId);
                startActivity(intent);
            } else {
                Toast.makeText(this, R.string.no_asignado_todavia,
                Toast.LENGTH_SHORT).show();
            }
        })
        .addOnFailureListener(e ->
            Toast.makeText(this, R.string.error_datos_asignado,
            Toast.LENGTH_SHORT).show());
    }

    /**
     * Verifica si aún hay cupo para nuevos miembros y lanza AddMemberActivity
     si procede
     */
    private void verificarYAgregarMiembro() {
        baseDatos.collection("groups").document(idGrupo).collection("members")
            .get()
            .addOnSuccessListener(snapshots -> {
                int cantidadActual = snapshots.size();
                baseDatos.collection("groups").document(idGrupo)
                    .get()
                    .addOnSuccessListener(document -> {
                        Long maximo =
document.getLong("maxParticipants");
                        if (maximo != null && cantidadActual >= maximo)
{
                            Toast.makeText(this,
R.string.mensaje_grupo_completo, Toast.LENGTH_SHORT).show();
                        } else {
                            iniciarActividad(AddMemberActivity.class);
                        }
                    });
            });
    }

    /**
     * Carga desde Firestore el nombre, presupuesto, imagen y miembros del
     grupo
     */
    private void cargarInformacionGrupo() {
        baseDatos.collection("groups").document(idGrupo)
            .get()
            .addOnSuccessListener(document -> {
                if (document.exists()) {
                    ownerId = document.getString("owner");
                    String nombre = document.getString("name");
                    Double presupuesto = document.getDouble("budgetLimit");
                    Long maxMiembros = document.getLong("maxParticipants");
                    String imageUrl = document.getString("imageUrl");

```

```

        textoNombreGrupo.setText(getString(R.string.text_group_name_placeholder,
        nombre));

        textoPresupuestoGrupo.setText(getString(R.string.text_group_budget_placeholder,
        presupuesto != null ? presupuesto : 0));

                cargarImagenGrupo(imageUrl);
                configurarVisibilidadBotonSorteo();

                // Listener a cambios en miembros
                listenerMiembros =
baseDatos.collection("groups").document(idGrupo).collection("members")
                    .addSnapshotListener((snapshots, error) -> {
                        if (error != null || snapshots == null)
return;
                        int cantidad = snapshots.size();

        textoContadorGrupo.setText(getString(R.string.text_group_count_placeholder,
        cantidad, maxMiembros != null ? maxMiembros : 0));
                        sePuedeAgregar = maxMiembros == null ||
cantidad < maxMiembros;
                        botonAgregarMiembro.setAlpha(sePuedeAgregar
? 1f : 0.5f);
                    });
                });

        .addOnFailureListener(e -> Toast.makeText(this,
R.string.error_cargar_datos_grupo, Toast.LENGTH_SHORT).show());
    }

    /**
     * Carga la imagen del grupo desde Firebase Storage usando Glide o pone una
     por defecto.
     */
    private void cargarImagenGrupo(String url) {
        if (url != null && !url.isEmpty()) {

Glide.with(this).load(url).placeholder(R.drawable.ic_groups).into(imageGroup);
        } else {
            imageGroup.setImageResource(R.drawable.ic_groups);
        }
    }

    /**
     * Solo muestra el botón de sorteo si el usuario es el propietario del
     grupo
     */
    private void configurarVisibilidadBotonSorteo() {
        if (esPropietario()) {
            botonSortear.setVisibility(View.VISIBLE);
        } else {
            botonSortear.setVisibility(View.GONE);
        }
    }

    /**
     * Devuelve true si el UID actual coincide con el del propietario del grupo
     */
    private boolean esPropietario() {
        return auth.getCurrentUser() != null &&
auth.getCurrentUser().getUid().equals(ownerId);
    }
}

```

```

    }

    /**
     * Abre selector de imagen y lanza un intent para que el propietario elija
     * una nueva imagen del grupo.
     */
    private void seleccionarNuevaImagen() {
        Intent intent = new Intent(Intent.ACTION_PICK);
        intent.setType("image/*");
        startActivityForResult(intent, PICK_IMAGE_REQUEST);
    }

    /**
     * Sube la imagen seleccionada a Firebase Storage y actualiza Firestore con
     * la nueva URL
     */
    @Override
    protected void onActivityResult(int requestCode, int resultCode, @Nullable
    Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode == PICK_IMAGE_REQUEST && resultCode == Activity.RESULT_OK && data != null && data.getData() != null) {
            nuevaImagenUri = data.getData();
            StorageReference storageRef =
                FirebaseStorage.getInstance().getReference("group_images/" + idGrupo + ".jpg");

            storageRef.putFile(nuevaImagenUri)
                .addOnSuccessListener(taskSnapshot ->
                    storageRef.getDownloadUrl().addOnSuccessListener(uri ->
                        baseDatos.collection("groups").document(idGrupo)
                            .update("imageUrl", uri.toString())
                            .addOnSuccessListener(unused -> {
                                Glide.with(this)
                                    .load(uri)
                                    .diskCacheStrategy(com.bumptech.glide.load.engine.DiskCacheStrategy.NONE)
                                    .skipMemoryCache(true)

                                .placeholder(R.drawable.ic_groups)
                                    .into(imageGroup);
                                Toast.makeText(this,
                                    R.string.mensaje_imagen_actualizada, Toast.LENGTH_SHORT).show();
                            })))
                .addOnFailureListener(e ->
                    Toast.makeText(this, "Error al subir la nueva
                    imagen", Toast.LENGTH_SHORT).show());
        }
    }

    /**
     * Configura el RecyclerView para mostrar la lista de miembros usando
     * FirestoreRecyclerAdapter
     */
    private void configurarRecyclerView() {
        Query consulta =
            baseDatos.collection("groups").document(idGrupo).collection("members");
        FirestoreRecyclerOptions<Member> opciones = new
        FirestoreRecyclerOptions.Builder<Member>()
            .setQuery(consulta, Member.class)
            .setLifecycleOwner(this)
    }
}

```

```
        .build();

    adaptadorMiembros = new MemberAdapter(opciones, idGrupo);
    recyclerMiembros.setLayoutManager(new LinearLayoutManager(this));
    recyclerMiembros.setAdapter(adaptadorMiembros);
}

/**
 * Ciclo de vida: al reanudar, reinicia el adaptador
 */
@Override
protected void onResume() {
    super.onResume();
    if (adaptadorMiembros != null) {
        recyclerMiembros.setAdapter(null);
        adaptadorMiembros.stopListening();
        adaptadorMiembros.startListening();
        recyclerMiembros.setAdapter(adaptadorMiembros);
    }
}

@Override
protected void onStart() {
    super.onStart();
    if (adaptadorMiembros != null) adaptadorMiembros.startListening();
}

@Override
protected void onStop() {
    super.onStop();
    if (adaptadorMiembros != null) adaptadorMiembros.stopListening();
    if (listenerMiembros != null) {
        listenerMiembros.remove();
        listenerMiembros = null;
    }
}
}
```

5. **MainActivity.java**

```

package NetMind.amigoinvisible.activities;

import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.content.Intent;
import android.graphics.Color;
import android.os.Build;
import android.os.Bundle;
import android.text.SpannableString;
import android.text.Spanned;
import android.text.TextUtils;
import android.text.style.ClickableSpan;
import android.util.Log;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import androidx.activity.result.ActivityResultLauncher;
import androidx.activity.result.IntentSenderRequest;
import androidx.activity.result.contract.ActivityResultContracts;
import androidx.annotation.NonNull;
import android.view.View;
import androidx.appcompat.app.AppCompatActivity;
import com.google.android.gms.auth.api.identity.GetSignInIntentRequest;
import com.google.android.gms.auth.api.identity.Identity;
import com.google.android.gms.auth.api.identity.SignInCredential;
import com.google.android.gms.common.api.ApiException;
import com.google.firebase.auth.AuthCredential;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.GoogleAuthProvider;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.utils.UsuarioUtils;

/**
 * Actividad principal de la aplicación que gestiona el inicio de sesión del
 * usuario.
 * Ofrece autenticación por correo electrónico y Google. Si el usuario ya ha
 * iniciado sesión previamente,
 * se le redirige automáticamente a la pantalla de bienvenida.
 */
public class MainActivity extends AppCompatActivity {

    /**
     * Etiqueta para logs. */
    private static final String TAG = "MainActivity";

    /**
     * Instancia de FirebaseAuth para gestionar la autenticación. */
    private FirebaseAuth firebaseAuth;

    /**
     * Campos de entrada de correo y contraseña. */
    private EditText campoCorreo, campoContrasena;

    /**
     * Launcher utilizado para gestionar el flujo de login con Google. */
    private ActivityResultLauncher<IntentSenderRequest> lanzadorLoginGoogle;

    /**
     * Método principal que se ejecuta al iniciar la actividad.
     * Inicializa la interfaz, configura listeners, comprueba sesión activa
     * y crea el canal de notificaciones para dispositivos Android 8 o
     * superior.
     *
     * @param savedInstanceState estado guardado de la instancia anterior, si
     * existe.
     */
}

```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        NotificationChannel channel = new NotificationChannel(
            "default_channel",
            "Notificaciones de Amigo Invisible",
            NotificationManager.IMPORTANCE_HIGH
        );
        channel.setDescription("Notificaciones de sorteos y asignaciones");
        NotificationManager manager = getSystemService(NotificationManager.class);
        manager.createNotificationChannel(channel);
    }

    inicializarComponentesUI();
    configurarListeners();
    configurarLanzadorGoogle();

    // // Inicializar autenticación de Firebase. Si ya hay sesión activa,
    ir a bienvenida
    firebaseAuth = FirebaseAuth.getInstance();
    if (firebaseAuth.getCurrentUser() != null) {
        irAPantallaBienvenida();
    }

    // Obtener token FCM (útil para notificaciones push)
    UsuarioUtils.obtenerTokenFirebase(this);
}

/**
 * Inicializa los componentes de la interfaz gráfica, como los campos de
texto.
 */
private void inicializarComponentesUI() {
    campoCorreo = findViewById(R.id.emailInput);
    campoContrasena = findViewById(R.id.passwordInput);
}

/**
 * Configura los botones de login, Google Sign-In y registro con sus
respectivos listeners.
 */
private void configurarListeners() {
    findViewById(R.id.btnLogIn).setOnClickListener(v ->
iniciarSesionConCorreo());
    findViewById(R.id.btnEmailSignIn).setOnClickListener(v ->
iniciarSesionConCorreo());
    findViewById(R.id.btnGoogleSignIn).setOnClickListener(v ->
iniciarSesionConGoogle());

    // Texto "¿No tienes cuenta? Regístrate" con efecto clic solo en
"Regístrate"
    TextView txtRegistrate = findViewById(R.id.txtRegistrate);
    txtRegistrate.setOnClickListener(v -> {
        Log.d(TAG, "Click en 'Regístrate'");
        startActivity(new Intent(MainActivity.this,
RegisterActivity.class));
    });
}

```

```

    /**
     * Configura el lanzador de actividad que se usará para manejar la
     * autenticación con Google.
     * Este metodo usa la API de actividad para obtener el resultado del
     * intento de login.
     */
    private void configurarLanzadorGoogle() {
        lanzadorLoginGoogle = registerForActivityResult(
            new ActivityResultContracts.StartIntentSenderForResult(),
            result -> {
                if (result.getResultCode() == RESULT_OK && result.getData()
                    != null) {
                    procesarResultadoGoogle(result.getData());
                }
            }
        );
    }

    /**
     * Inicia el proceso de inicio de sesión con Google utilizando el ID de
     * cliente definido en strings.xml.
     * Se lanza el intent proporcionado por la API de Google Identity.
     */
    private void iniciarSesionConGoogle() {
        GetSignInIntentRequest request = GetSignInIntentRequest.builder()
            .setServerClientId(getString(R.string.default_web_client_id))
            .build();

        Identity.getSignInClient(this)
            .getSignInIntent(request)
            .addOnSuccessListener(pendingIntent -> {
                IntentSenderRequest intentRequest = new
                IntentSenderRequest.Builder(pendingIntent getIntentSender()).build();
                lanzadorLoginGoogle.launch(intentRequest);
            })
            .addOnFailureListener(e -> {
                Toast.makeText(this,
                    getString(R.string.error_google_sign_in), Toast.LENGTH_SHORT).show();
                Log.e(TAG, getString(R.string.error_google_sign_in), e);
            });
    }

    /**
     * Procesa los datos devueltos tras la autenticación con Google.
     * Si el ID token es válido, se genera una credencial para Firebase y se
     * autentica al usuario.
     *
     * @param data Intent devuelto por el flujo de autenticación de Google.
     */
    private void procesarResultadoGoogle(Intent data) {
        try {
            SignInCredential credential =
                Identity.getSignInClient(this).getSignInCredentialFromIntent(data);
            String idToken = credential.getGoogleIdToken();

            if (idToken != null) {
                AuthCredential firebaseCredential =
                    GoogleAuthProvider.getCredential(idToken, null);
                UsuarioUtils.autenticarConFirebase(firebaseCredential, this,
                    this::irAPantallaBienvenida);
            }
        } catch (ApiException e) {
    
```

```

        Toast.makeText(this,
getString(R.string.error_autenticacion_google), Toast.LENGTH_SHORT).show();
        Log.e(TAG, getString(R.string.error_google_sign_in), e);
    }
}

/**
 * Inicia sesión utilizando las credenciales introducidas por el usuario
(correo y contraseña).
 * Si los campos están vacíos, muestra un mensaje de advertencia.
 * En caso de éxito, redirige a la pantalla de bienvenida.
 */
private void iniciarSesionConCorreo() {
    String correo = campoCorreo.getText().toString().trim();
    String contrasena = campoContrasena.getText().toString().trim();

    if (TextUtils.isEmpty(correo) || TextUtils.isEmpty(contrasena)) {
        Toast.makeText(this,
getString(R.string.mensaje_campos_obligatorios), Toast.LENGTH_SHORT).show();
        return;
    }

    firebaseAuth.signInWithEmailAndPassword(correo, contrasena)
        .addOnSuccessListener(authResult -> {
            if (firebaseAuth.getCurrentUser() != null) {

                UsuarioUtils.verificarUsuario(firebaseAuth.getCurrentUser(),
this, this::irAPantallaBienvenida);
            }
        })
        .addOnFailureListener(e -> {
            Toast.makeText(this, getString(R.string.error_login_correo,
e.getMessage()), Toast.LENGTH_LONG).show();
            Log.e(TAG, getString(R.string.error_login_correo,
e.getMessage()), e);
        });
}

/**
 * Redirige al usuario a la pantalla principal de bienvenida tras una
autenticación exitosa.
 * Esta actividad se termina para evitar volver con el botón "Atrás".
 */
private void irAPantallaBienvenida() {
    startActivity(new Intent(this, WelcomeActivity.class));
    finish();
}
}

```

6. RegisterActivity.java

```
package NetMind.amigoinvisible.activities;

import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.utils.UsuarioUtils;

/**
 * Actividad que gestiona el proceso de registro de un nuevo usuario.
 * Permite ingresar nombre, correo electrónico y contraseña, realizando
valificaciones
 * antes de enviar los datos a Firebase mediante una clase utilitaria.
 */
public class RegisterActivity extends AppCompatActivity {

    /** Campo de texto para ingresar el nombre del usuario. */
    private EditText campoNombre;

    /** Campo de texto para ingresar el correo electrónico del usuario. */
    private EditText campoCorreo;

    /** Campo de texto para ingresar la contraseña del usuario. */
    private EditText campoContrasena;

    /**
     * Método principal que se ejecuta al crear la actividad.
     * Inicializa los componentes de UI y configura el botón de registro.
     *
     * @param savedInstanceState estado anterior de la actividad si fue
restaurada.
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_register);

        // Referencias a los campos de entrada de datos
        campoNombre = findViewById(R.id.editTextName);
        campoCorreo = findViewById(R.id.editTextEmail);
        campoContrasena = findViewById(R.id.editTextPassword);
        Button botonRegistrar = findViewById(R.id.btnRegister);

        // Acción al pulsar el botón "Registrarse"
        botonRegistrar.setOnClickListener(v -> {
            String nombre = campoNombre.getText().toString().trim();
            String correo = campoCorreo.getText().toString().trim();
            String contrasena = campoContrasena.getText().toString().trim();

            // Validación: campos obligatorios no pueden estar vacíos
            if (nombre.isEmpty() || correo.isEmpty() || contrasena.isEmpty()) {
                Toast.makeText(this, R.string.mensaje_campos_obligatorios,
Toast.LENGTH_SHORT).show();
                return;
            }
            // Validación: formato de correo electrónico válido
            if (!android.util.Patterns.EMAIL_ADDRESS.matcher(correo).matches())
        });
}
```

```
        Toast.makeText(this, R.string.mensaje_correo_invalido,
Toast.LENGTH_SHORT).show();
        return;
    }
    // Validación: contraseña con al menos 6 caracteres
    if (contrasena.length() < 6) {
        Toast.makeText(this, R.string.mensaje_requisitos_contrasena,
Toast.LENGTH_SHORT).show();
        return;
    }

    // Registro delegado a clase utilitaria que gestiona la lógica con
Firebase
    UsuarioUtils.registrarNuevoUsuario(nombre, correo, contrasena,
this);
}
}
}
```



7. WelcomeActivity.java

```
package NetMind.amigoinvisible.activities;
import android.content.Intent;
import android.os.Bundle;
import android.view.MenuItem;
import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import androidx.activity.result.ActivityResultLauncher;
import androidx.activity.result.contract.ActivityResultContracts;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;
import androidx.core.view.GravityCompat;
import androidx.drawerlayout.widget.DrawerLayout;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentTransaction;
import com.bumptech.glide.Glide;
import com.google.android.material.navigation.NavigationView;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebaseio.firebaseio.FirebaseFirestore;
import com.google.firebase.storage.FirebaseStorage;
import com.google.firebase.storage.StorageReference;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.fragments.CreateGroupFragment;
import NetMind.amigoinvisible.fragments.InvitationsFragment;
import NetMind.amigoinvisible.fragments.SettingsFragment;
import NetMind.amigoinvisible.fragments.ViewGroupsFragment;
import NetMind.amigoinvisible.services.TokenManager;
import NetMind.amigoinvisible.utils.FirebaseUtils;
import NetMind.amigoinvisible.utils.UsuarioUtils;

/**
 * Actividad principal tras el login.
 * Muestra la navegación lateral (Drawer) y permite gestionar grupos, ajustes,
 * invitaciones,
 * así como actualizar la imagen de perfil del usuario.
 */
public class WelcomeActivity extends AppCompatActivity implements
NavigationView.OnNavigationItemSelectedListener {

    private DrawerLayout drawerLayout;
    private FirebaseAuth authFirebase;
    private FirebaseUser usuarioActual;
    private FirebaseFirestore baseDatos;
    private StorageReference referenciaAlmacenamiento;
    private TextView textoBienvenida;
    private ImageView imagenPerfil;

    private ActivityResultLauncher<String> selectorImagen;

    /**
     * Inicializa la actividad y carga la vista por defecto (mis grupos),
     * junto con el token FCM para recibir notificaciones push.
     *
     * @param savedInstanceState estado previo si la actividad fue recreada.
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```

        setContentView(R.layout.activity_welcome);

        inicializarFirebase();
        configurarToolbar();
        configurarDrawer();
        configurarSelectorImagen();

        // Solo cargamos fragmento si aún no está restaurado
        if (savedInstanceState == null) {
            // Empareja usuario con sus posibles invitaciones y carga el
fragmento inicial
            new android.os.Handler().postDelayed(() -> {
                usuarioActual = FirebaseAuth.getInstance().getCurrentUser();
                if (usuarioActual != null) {
                    FirebaseUtils.emparejarUsuarioConGrupos(usuarioActual, ())
-> {
                        runOnUiThread(() -> {
                            cargarFragmento(new ViewGroupsFragment());
                        });
                    }
                }
            }, 500);
        }
        TokenManager.actualizarTokenFCM();
    }

    /**
     * * Inicializa las herramientas de Firebase que vamos a usar:
     * * - Autenticación (para saber qué usuario está conectado)
     * * - Firestore (para acceder a la base de datos)
     * * - Storage (para guardar imágenes de perfil)
     */
    private void inicializarFirebase() {
        authFirebase = FirebaseAuth.getInstance();
        usuarioActual = authFirebase.getCurrentUser();
        baseDatos = FirebaseFirestore.getInstance();
        referenciaAlmacenamiento =
            FirebaseStorage.getInstance().getReference("profile_pictures");
    }

    /**
     * Configura la Toolbar y el botón de apertura del Drawer
     */
    private void configurarToolbar() {
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(false);
        toolbar.setNavigationOnClickListener(v ->
            drawerLayout.openDrawer(GravityCompat.START));
    }

    /**
     * Configura el menú lateral (Drawer) y muestra los datos del usuario
autenticado:
     * nombre, correo e imagen de perfil.
     */
    private void configurarDrawer() {
        drawerLayout = findViewById(R.id.drawer_layout);
        NavigationView navigationView = findViewById(R.id.nav_view);
        navigationView.setNavigationItemSelected(this);

        // Muestra la información del usuario logueado
    }

```

```

        View headerView = navigationView.getHeaderView(0);
        textoBienvenida = headerView.findViewById(R.id.nav_header_name);
        TextView textoCorreo = headerView.findViewById(R.id.nav_header_email);
        imagenPerfil = headerView.findViewById(R.id.profile_image);

        imagenPerfil.setOnClickListener(v -> {
            Toast.makeText(this, R.string.mensaje_seleccionar_imagen,
            Toast.LENGTH_SHORT).show();
            selectorImagen.launch("image/*");
        });
        // Rellena con los datos del usuario actual
        if (usuarioActual != null) {
            String nombre = usuarioActual.getDisplayName() != null ?
            usuarioActual.getDisplayName() : getString(R.string.usuario_generico);
            textoBienvenida.setText(getString(R.string.bienvenido_usuario,
            nombre));
            textoCorreo.setText(usuarioActual.getEmail());

            if (usuarioActual.getPhotoUrl() != null) {

Glide.with(getApplicationContext()).load(usuarioActual.getPhotoUrl()).circleCrop()
            .into(imagenPerfil);
        }
    }

    /**
     * Configura el selector de imagen para actualizar la foto de perfil del
     * usuario.
     * Utiliza {@link UsuarioUtils#actualizarFotoPerfil}.
     */
    private void configurarSelectorImagen() {
        selectorImagen = registerForActivityResult(
            new ActivityResultContracts.GetContent(),
            uri -> {
                if (usuarioActual != null && uri != null) {
                    UsuarioUtils.actualizarFotoPerfil(usuarioActual, uri,
this, imagenPerfil);
                }
            }
        );
    }

    /**
     * Al volver a la actividad (por ejemplo, después de ajustes), recarga el
     * fragmento si era necesario.
     */
    @Override
    protected void onResume() {
        super.onResume();
        Fragment actual = getSupportFragmentManager().findFragmentById(R.id.fragment_container);
        if (actual instanceof ViewGroupsFragment) {

getSupportFragmentManager().beginTransaction().remove(actual).commitNow();
            cargarFragmento(new ViewGroupsFragment());
        }
    }

    /**
     * Carga un fragmento en el contenedor principal.
     *
     * @param fragmento Fragmento a mostrar.

```

```

/*
private void cargarFragmento(Fragment fragmento) {
    FragmentTransaction transaccion = getSupportFragmentManager().beginTransaction();
    transaccion.replace(R.id.fragment_container, fragmento);
    transaccion.commit();
}

/**
 * Maneja la selección de ítems en el menú lateral (Drawer).
 *
 * @param item Elemento del menú seleccionado.
 * @return true si el evento ha sido manejado.
 */
@Override
public boolean onNavigationItemSelected(@NonNull MenuItem item) {
    int id = item.getItemId();

    if (id == R.id.nav_my_groups) {
        cargarFragmento(new ViewGroupsFragment());
    } else if (id == R.id.nav_create_group) {
        cargarFragmento(new CreateGroupFragment());
    } else if (id == R.id.nav_settings) {
        cargarFragmento(new SettingsFragment());
    } else if (id == R.id.nav_logout) {
        mostrarDialogoCerrarSesion();
    } else if (id == R.id.nav_invitations) {
        cargarFragmento(new InvitationsFragment());
    }

    drawerLayout.closeDrawer(GravityCompat.START);
    return true;
}

/**
 * Muestra un diálogo para confirmar el cierre de sesión del usuario.
 * Si acepta, cierra sesión y vuelve al login.
 */
private void mostrarDialogoCerrarSesion() {
    new AlertDialog.Builder(this)
        .setTitle(R.string.txt_titulo_cerrar_sesion)
        .setMessage(R.string.txt_mensaje_cerrar_sesion)
        .setPositiveButton(R.string.txt_mensajePositivo, (dialog,
which) -> {
            authFirebase.signOut();
            startActivity(new Intent(this, MainActivity.class));
            finish();
        })
        .setNegativeButton(R.string.txt_mensajeNegativo, null)
        .show();
}
}

```

8. WishlistActivity.java

```
package NetMind.amigoinvisible.activities;

import android.os.Bundle;
import android.text.TextUtils;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.firestore.DocumentReference;
import com.google.firebase.firestore.FirebaseFirestore;

import java.util.*;

import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.adapters.WishlistAdapter;
import NetMind.amigoinvisible.models.WishlistItem;

/**
 * Actividad que permite a un usuario gestionar su lista de deseos dentro de un grupo específico.
 * Utiliza un campo de autocompletado y una lista dinámica conectada a Firestore.
 */
public class WishlistActivity extends AppCompatActivity {

    private AutoCompleteTextView campoDeseo;
    private Button botonAgregar;
    private RecyclerView recyclerView;
    private WishlistAdapter adapter;

    private final List<WishlistItem> deseos = new ArrayList<>();
    private EditText campoUrl;

    private FirebaseFirestore db;
    private String idGrupo;
    private String userId;

    /**
     * Método principal de la actividad. Carga los componentes visuales,
     * configura la lógica de interacción y sincroniza con Firestore.
     *
     * @param savedInstanceState Estado anterior si se ha restaurado.
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_wishlist);
        // Validación básica del intent
        idGrupo = getIntent().getStringExtra("groupId");
        if (idGrupo == null || idGrupo.isEmpty()) {
            Toast.makeText(this, R.string.error_id_grupo_invalido,
Toast.LENGTH_SHORT).show();
            finish();
            return;
        }
    }
}
```

```

        // Inicialización de Firebase y usuario
        db = FirebaseFirestore.getInstance();
        userId = FirebaseAuth.getInstance().getCurrentUser().getUid();
        // Asignación de vistas
        campoDeseo = findViewById(R.id.inputWishItem);
        botonAgregar = findViewById(R.id.btnAddWish);
        recyclerView = findViewById(R.id.wishlistRecyclerView);
        campoUrl = findViewById(R.id.inputWishUrl);
        // Configuración inicial
        configurarAutoCompletado();
        configurarRecyclerView();
        cargarListaDesdeFirestore();
        // Acción del botón
        botonAgregar.setOnClickListener(v -> agregarDeseo());
    }

    /**
     * Configura el campo de entrada con sugerencias cargadas desde recursos
     * (strings.xml).
     * El umbral para mostrar sugerencias es 1 carácter.
     */
    private void configurarAutoCompletado() {
        String[] sugerencias = getResources().getStringArray(R.array.wishlist_suggestions);
        ArrayAdapter<String> adaptador = new ArrayAdapter<>(this,
            android.R.layout.simple_dropdown_item_1line, sugerencias);
        campoDeseo.setAdapter(adaptador);
        campoDeseo.setThreshold(1);
    }

    /**
     * Configura el RecyclerView y su adaptador personalizado para mostrar los
     * deseos del usuario.
     */
    private void configurarRecyclerView() {
        adapter = new WishlistAdapter(deseos, this, idGrupo);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        recyclerView.setAdapter(adapter);
    }

    /**
     * Valida el deseo ingresado por el usuario y, si es válido, lo añade a la
     * lista local y lo sincroniza con Firestore.
     * También actualiza la interfaz visual y borra el campo de entrada.
     */
    private void agregarDeseo() {
        String texto = campoDeseo.getText().toString().trim();
        String url = ((EditText) findViewById(R.id.inputWishUrl)).getText().toString().trim();

        if (TextUtils.isEmpty(texto)) {
            Toast.makeText(this, R.string.mensaje_deseo_vacio,
                Toast.LENGTH_SHORT).show();
            return;
        }

        WishlistItem nuevo = new WishlistItem(texto, url.isEmpty() ? null :
        url);
        deseos.add(nuevo);
        campoDeseo.setText("");
        ((EditText) findViewById(R.id.inputWishUrl)).setText("");
        adapter.notifyDataSetChanged();
        guardarListaEnFirestore();
        Toast.makeText(this, R.string.mensaje_deseo_añadido,
            Toast.LENGTH_SHORT).show();
    }
}

```

```

}

/**
 * Guarda la lista de deseos actualizada del usuario en Firestore bajo su
documento en la colección de miembros del grupo.
 * Si ocurre un error, se notifica con un Toast.
 */
private void guardarListaEnFirestore() {
    List<Map<String, Object>> listaParaFirestore = new ArrayList<>();
    for (WishlistItem item : deseos) {
        listaParaFirestore.add(item.toMap());
    }

    db.collection("groups")
        .document(idGrupo)
        .collection("members")
        .document(userId)
        .update("wishlist", listaParaFirestore)
        .addOnFailureListener(e ->
            Toast.makeText(this, R.string.mensaje_error_deseo,
Toast.LENGTH_SHORT).show());
}

/**
 * Recupera la lista de deseos previamente almacenada en Firestore.
 * Si existe, se carga en la lista local y se actualiza el adaptador.
 */
private void cargarListaDesdeFirestore() {
    db.collection("groups")
        .document(idGrupo)
        .collection("members")
        .document(userId)
        .get()
        .addOnSuccessListener(document -> {
            deseos.clear();
            if (document.contains("wishlist")) {
                List<Map<String, Object>> datos = (List<Map<String,
Object>>) document.get("wishlist");
                for (Map<String, Object> entry : datos) {
                    String title = (String) entry.get("title");
                    String url = (String) entry.get("url");
                    deseos.add(new WishlistItem(title, url));
                }
            }
            adapter.notifyDataSetChanged();
        });
}
}

```

5.1.2 Fragments

Índice de los fragments:

1. CreateGroupFragment.java
2. InvitationsFragment.java
3. SettingsFragment.java
4. ViewGroupsFragment.java

A continuación se muestra el código correspondiente a cada una de ellas:

1. CreateGroupFragment.java

```
package NetMind.amigoinvisible.fragments;
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import com.bumptech.glide.Glide;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.firestore.DocumentReference;
import com.google.firebase.firestore.FirebaseFirestore;
import com.google.firebase.storage.FirebaseStorage;
import com.google.firebase.storage.StorageReference;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import NetMind.amigoinvisible.R;

/**
 * Fragmento encargado de gestionar la creación de nuevos grupos de Amigo Invisible.
 * Permite ingresar el nombre del grupo, límite de participantes, presupuesto y una imagen representativa.
 * Los datos se almacenan en Firebase Firestore y Storage.
 */
public class CreateGroupFragment extends Fragment {

    private EditText campoNombreGrupo, campoMaxParticipantes, campoPresupuesto;
    private ImageView groupImagePreview;
    private Uri imageUriSeleccionada;

    private FirebaseFirestore baseDatos;
    private FirebaseAuth auth;
```

```

private static final int PICK_IMAGE_REQUEST = 101;
private static final int GRUPOS_CREADOS_MAXIMOS = 3;

private View vista;

/**
 * Crea e infla la vista del fragmento.
 */
@Override
public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    vista = inflater.inflate(R.layout.fragment_create_group, container, false);

    inicializarComponentes();
    configurarEventos();

    return vista;
}

/**
 * Inicializa los componentes visuales y servicios de Firebase.
 */
private void inicializarComponentes() {
    campoNombreGrupo = vista.findViewById(R.id.editTextGroupName);
    campoMaxParticipantes =
vista.findViewById(R.id.editTextMaxParticipants);
    campoPresupuesto = vista.findViewById(R.id.editTextBudgetLimit);
    grupoImagePreview = vista.findViewById(R.id.groupImagePreview);

    baseDatos = FirebaseFirestore.getInstance();
    auth = FirebaseAuth.getInstance();
}

/**
 * Configura los botones para seleccionar imagen y crear grupo.
 */
private void configurarEventos() {
    vista.findViewById(R.id.btnSelectImage).setOnClickListener(v ->
seleccionarImagen());
    vista.findViewById(R.id.btnExitGroup).setOnClickListener(v ->
crearGrupo());
}

/**
 * Lanza el selector de imagen del sistema.
 */
private void seleccionarImagen() {
    Intent intent = new Intent(Intent.ACTION_PICK);
    intent.setType("image/*");
    startActivityForResult(intent, PICK_IMAGE_REQUEST);
}

/**

```

```

    * Procesa el resultado del selector de imágenes y actualiza la vista
    * previa.
    *
    * @param requestCode Código de la petición.
    * @param resultCode Resultado devuelto.
    * @param data         Datos devueltos (URI de imagen).
    */
    @Override
    public void onActivityResult(int requestCode, int resultCode, @Nullable
Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == PICK_IMAGE_REQUEST && resultCode ==
Activity.RESULT_OK && data != null && data.getData() != null) {
            imageUriSeleccionada = data.getData();

            Glide.with(this).load(imageUriSeleccionada).into(groupImagePreview);
        }
    }

    /**
     * Valida los campos y comienza el proceso de creación del grupo.
     */
    private void crearGrupo() {
        String nombre = campoNombreGrupo.getText().toString().trim();
        String maxStr = campoMaxParticipantes.getText().toString().trim();
        String presupuestoStr = campoPresupuesto.getText().toString().trim();

        if (TextUtils.isEmpty(nombre) || TextUtils.isEmpty(maxStr) ||
TextUtils.isEmpty(presupuestoStr)) {
            mostrarToast(R.string.error_campus_obligatorios);
            return;
        }

        int maxParticipantes;
        double presupuesto;
        try {
            maxParticipantes = Integer.parseInt(maxStr);
            presupuesto = Double.parseDouble(presupuestoStr);
        } catch (NumberFormatException e) {
            mostrarToast(R.string.error_valores_numericos);
            return;
        }

        String idUsuario = auth.getCurrentUser().getUid();
        verificarLimiteYCrearGrupo(nombre, maxParticipantes, presupuesto,
idUsuario);
    }

    /**
     * Comprueba si el usuario ha alcanzado el máximo de grupos creados.
     *
     * @param nombre Nombre del grupo.
     * @param max    Máximo de participantes.
     * @param presupuesto Presupuesto por participante.
     * @param idUsuario ID del usuario autenticado.

```

```

/*
private void verificarLimiteYCrearGrupo(String nombre, int max, double
presupuesto, String idUsuario) {
    baseDatos.collection("groups")
        .whereEqualTo("owner", idUsuario)
        .get()
        .addOnSuccessListener(snapshot -> {
            if (snapshot.size() >= GRUPOS_CREADOS_MAXIMOS) {
                mostrarToast(getString(R.string.error_max_grupos,
GRUPOS_CREADOS_MAXIMOS));
                return;
            }
            crearGrupoEnFirestore(nombre, max, presupuesto, idUsuario);
        })
        .addOnFailureListener(e ->
mostrarToast(R.string.error_verificar_limite));
}

/**
 * Guarda los datos básicos del grupo en Firestore.
 */
private void crearGrupoEnFirestore(String nombre, int max, double
presupuesto, String idUsuario) {
    Map<String, Object> datosGrupo = new HashMap<>();
    datosGrupo.put("name", nombre);
    datosGrupo.put("owner", idUsuario);
    datosGrupo.put("maxParticipants", max);
    datosGrupo.put("budgetLimit", presupuesto);
    datosGrupo.put("members", new ArrayList<String>() {{ add(idUsuario);
}});
    datosGrupo.put("imageUrl", "");
}

baseDatos.collection("groups").add(datosGrupo).addOnSuccessListener(docRef -> {
    subirImagenSiHay(docRef);
    agregarMiembroInicial(docRef, idUsuario);
}).addOnFailureListener(e -> mostrarToast(R.string.error_crear_grupo));
}

/**
 * Si hay imagen seleccionada, la sube al almacenamiento y actualiza la URL
en Firestore.
 */
private void subirImagenSiHay(DocumentReference docRef) {
    if (imageUriSeleccionada == null) return;

    StorageReference storageRef =
FirebaseStorage.getInstance().getReference()
    .child("group_images/" + docRef.getId() + ".jpg");

    storageRef.putFile(imageUriSeleccionada)
        .addOnSuccessListener(taskSnapshot ->
storageRef.getDownloadUrl()
        .addOnSuccessListener(uri ->

```

```

        docRef.update("imageUrl",
uri.toString())
    )
}

/**
 * Registra al usuario creador como primer miembro del grupo.
 */
private void agregarMiembroInicial(DocumentReference docRef, String userId)
{
    Map<String, Object> datosMiembro = new HashMap<>();
    datosMiembro.put("name", auth.getCurrentUser().getDisplayName());
    datosMiembro.put("email", auth.getCurrentUser().getEmail());

    docRef.collection("members").document(userId).set(datosMiembro).addOnSuccessListener(unused -> {
        mostrarToast(R.string.mensaje_grupo_creado);
        requireActivity().getSupportFragmentManager().beginTransaction()
            .replace(R.id.fragment_container, new ViewGroupsFragment())
            .commit();
    });
}

/**
 * Muestra un mensaje tipo Toast desde un recurso de strings.
 *
 * @param resId ID del recurso string.
 */
private void mostrarToast(int resId) {
    Toast.makeText(getApplicationContext(), getString(resId),
    Toast.LENGTH_SHORT).show();
}

/**
 * Muestra un mensaje tipo Toast con texto plano.
 *
 * @param mensaje Texto del mensaje.
 */
private void mostrarToast(String mensaje) {
    Toast.makeText(getApplicationContext(), mensaje, Toast.LENGTH_SHORT).show();
}
}

```

2. InvitationsFragment.java

```
package NetMind.amigoinvisible.fragments;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebaseio.firestore.ListenerRegistration;
import java.util.ArrayList;
import java.util.List;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.adapters.InvitationsAdapter;
import NetMind.amigoinvisible.models.Invitation;
import NetMind.amigoinvisible.utils.InvitationUtils;

/**
 * Fragmento que muestra las invitaciones pendientes del usuario a diferentes grupos.
 * Permite aceptar o rechazar dichas invitaciones.
 */
public class InvitationsFragment extends Fragment {

    private RecyclerView recyclerView;
    private TextView emptyMessage;
    private InvitationsAdapter adapter;
    private final List<Invitation> listaInvitaciones = new ArrayList<>();
    private ListenerRegistration listener;

    public InvitationsFragment() {}

    /**
     * Infla el layout XML del fragmento
     */
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_invitations, container,
false);
    }

    /**
     * Se llama tras crearse la vista del fragmento. Configura RecyclerView y lanza la escucha.
     *
     * @param view Vista ya inflada.
     * @param savedInstanceState Estado previo guardado, si existe.
    
```

```

    */
    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        recyclerView = view.findViewById(R.id.recycler_invitations);
        emptyMessage = view.findViewById(R.id.empty_message);

        recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));

        // Se establece el adaptador, que recibirá las funciones para aceptar y
        // rechazar
        adapter = new InvitationsAdapter(listaInvitaciones,
this::aceptarInvitacion, this::rechazarInvitacion);
        recyclerView.setAdapter(adapter);

        escucharInvitaciones();
    }

    /**
     * Escucha en tiempo real las invitaciones del usuario autenticado mediante
     InvitationUtils.
     * Actualiza la lista y muestra el mensaje si está vacía.
     */
    private void escucharInvitaciones() {
        FirebaseUser usuario = FirebaseAuth.getInstance().getCurrentUser();
        if (usuario == null || usuario.getEmail() == null) return;

        // Delegamos la lógica a InvitationUtils
        listener = InvitationUtils.escucharInvitaciones(
            usuario.getEmail(),
            invitaciones -> {
                listaInvitaciones.clear();
                listaInvitaciones.addAll(invitaciones);
                adapter.notifyDataSetChanged();
                actualizarMensajeVacio();
            }
        );
    }

    /**
     * Acepta una invitación seleccionada, añade al usuario al grupo
     correspondiente
     * y actualiza la vista redirigiendo al fragmento de grupos.
     *
     * @param invitacion Invitación a aceptar.
     */
    private void aceptarInvitacion(Invitation invitacion) {
        FirebaseUser usuario = FirebaseAuth.getInstance().getCurrentUser();
        if (usuario == null) return;

        InvitationUtils.aceptarInvitacion(
            invitacion,
            usuario,

```

```

        () -> {
            Toast.makeText(getContext(), R.string.msg_aceptada_ok,
Toast.LENGTH_SHORT).show();
            listaInvitaciones.remove(invitacion);
            adapter.notifyDataSetChanged();
            actualizarMensajeVacio();

            // Redirige a la vista de grupos tras aceptar
            requireActivity().getSupportFragmentManager()
                .beginTransaction()
                .replace(R.id.fragment_container, new
ViewGroupsFragment())
                .commit();
        },
        error -> Toast.makeText(getContext(), error,
Toast.LENGTH_SHORT).show()
    );
}

/**
 * Rechaza una invitación, eliminando el documento en Firestore.
 *
 * @param invitacion Invitación a rechazar.
 */
private void rechazarInvitacion(Invitation invitacion) {
    FirebaseUser usuario = FirebaseAuth.getInstance().getCurrentUser();
    if (usuario == null || usuario.getEmail() == null) return;

    InvitationUtils.rechazarInvitacion(
        invitacion,
        usuario.getEmail(),
        () -> {
            Toast.makeText(getContext(),
R.string.msg_invitacion_rechazada, Toast.LENGTH_SHORT).show();
            listaInvitaciones.remove(invitacion);
            adapter.notifyDataSetChanged();
            actualizarMensajeVacio();
        },
        error -> Toast.makeText(getContext(), error,
Toast.LENGTH_SHORT).show()
    );
}

/**
 * Muestra u oculta el mensaje de "no invitaciones" dependiendo del estado
de la lista.
 */
private void actualizarMensajeVacio() {
    emptyMessage.setVisibility(listaInvitaciones.isEmpty() ? View.VISIBLE :
View.GONE);
}

/**
 * Detiene el listener activo cuando la vista se destruye para evitar fugas
de memoria.
 */
@Override

```

```
public void onDestroyView() {  
    super.onDestroyView();  
    if (listener != null) listener.remove();  
}  
}
```



3. SettingsFragment.java

```
package NetMind.amigoinvisible.fragments;
import android.os.Bundle;
import android.widget.Toast;
import androidx.preference.EditTextPreference;
import androidx.preference.PreferenceFragmentCompat;
import NetMind.amigoinvisible.R;
import androidx.appcompat.app.AppCompatDelegate;
import androidx.preference.ListPreference;
import androidx.preference.SwitchPreferenceCompat;

/**
 * Fragmento de configuración de usuario.
 * Permite modificar preferencias como el nombre, tema visual, notificaciones y
 * otras opciones de interfaz.
 */
public class SettingsFragment extends PreferenceFragmentCompat {

    /**
     * Carga las preferencias definidas en XML y configura los listeners de
     * cambio.
     *
     * @param savedInstanceState Estado previo de la instancia, si existe.
     * @param rootKey             Clave raíz para la jerarquía de preferencias
     * (normalmente null).
     */
    @Override
    public void onCreatePreferences(Bundle savedInstanceState, String rootKey)
    {
        // Carga las preferencias desde el archivo XML
        setPreferencesFromResource(R.xml.settings, rootKey);

        // Preferencia: nombre de usuario editable
        EditTextPreference preferenciaNombreUsuario =
        findPreference("username");

        if (preferenciaNombreUsuario != null) {
            preferenciaNombreUsuario.setOnPreferenceChangeListener((preference,
nuevoValor) -> {
                String nuevoNombre = nuevoValor.toString();
                String mensaje = getString(R.string.nombre_usuario_actualizado,
nuevoNombre);
                Toast.makeText(getContext(), mensaje,
Toast.LENGTH_SHORT).show();
                return true; // Guarda el nuevo valor
            });
        }

        // Preferencia: selector de tema (claro, oscuro, sistema)
        ListPreference tema = findPreference("color_theme");

        if (tema != null) {
            tema.setOnPreferenceChangeListener((preference, newValue) -> {
                String valor = (String) newValue;
                switch (valor) {
```

```

        case "light":

AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO);
        break;
    case "dark":

AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_YES);
        break;
    case "system":
    default:

AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_FOLLOW_SYSTEM);
        break;
    }
    return true; // Guarda la preferencia
});
}

// Preferencia: notificaciones activadas/desactivadas
SwitchPreferenceCompat notificaciones =
findPreference("notifications_enabled");
if (notificaciones != null) {
    notificaciones.setOnPreferenceChangeListener((preference, newValue)
-> {
        boolean activadas = (boolean) newValue;
        Toast.makeText(getApplicationContext(),
            activadas ?
getString(R.string.notificaciones_activadas) :
getString(R.string.notificaciones_desactivadas),
            Toast.LENGTH_SHORT).show();

        // Aquí puedes cancelar o programar notificaciones según estado
        return true;
});
}
}
}

```

4. ViewGroupsFragment.java

```
package NetMind.amigoinvisible.fragments;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import androidx.annotation.NonNull;
import androidx.fragment.app.Fragment;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.firestore.*;
import java.util.ArrayList;
import java.util.List;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.adapters.GroupListAdapter;
import NetMind.amigoinvisible.models.Group;
import NetMind.amigoinvisible.utils.GroupUtils;

/**
 * Fragmento encargado de mostrar en un RecyclerView todos los grupos a los que pertenece el usuario actual.
 * Utiliza Firebase Firestore para recibir actualizaciones en tiempo real y mantener sincronizada la lista.
 */
public class ViewGroupsFragment extends Fragment {

    private FirebaseFirestore baseDatos;
    private GroupListAdapter adaptadorGrupos;
    private final List<Group> listaDatosGrupos = new ArrayList<>();
    private ListenerRegistration listenerGrupos;
    private RecyclerView listaGrupos;

    /**
     * Infla la vista del fragmento y carga los grupos desde Firebase.
     *
     * @param inflater LayoutInflater para inflar la vista.
     * @param container Contenedor padre.
     * @param savedInstanceState Estado guardado de la instancia (si existe).
     * @return Vista creada del fragmento.
     */
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View vista = inflater.inflate(R.layout.fragment_view_groups, container, false);

        inicializarRecyclerView(vista);

        baseDatos = FirebaseFirestore.getInstance();
        cargarGruposDelUsuario();

        return vista;
    }

    private void inicializarRecyclerView(View vista) {
        listaGrupos = vista.findViewById(R.id.lista_grupos);
        listaGrupos.setLayoutManager(new LinearLayoutManager(getActivity()));
        adaptadorGrupos = new GroupListAdapter(listaDatosGrupos);
        listaGrupos.setAdapter(adaptadorGrupos);
    }

    private void cargarGruposDelUsuario() {
        FirebaseAuth mAuth = FirebaseAuth.getInstance();
        FirebaseUser user = mAuth.getCurrentUser();
        if (user != null) {
            user.getUid();
            FirebaseFirestore db = FirebaseFirestore.getInstance();
            db.collection("grupos").whereField("users", FieldValue.arrayContains(user.getUid()))
                    .get()
                    .addOnCompleteListener(task -> {
                        if (task.isSuccessful()) {
                            for (QueryDocumentSnapshot document : task.getResult()) {
                                Group grupo = document.toObject(Group.class);
                                listaDatosGrupos.add(grupo);
                            }
                            adaptadorGrupos.notifyDataSetChanged();
                        } else {
                            Log.d("Error", "Error getting documents: ", task.getException());
                        }
                    });
        }
    }
}
```

```

    }

    /**
     * Configura el RecyclerView y su adaptador para visualizar los grupos del
     * usuario.
     *
     * @param vista Vista raíz del fragmento.
     */
    private void inicializarRecyclerView(View vista) {
        listaGrupos = vista.findViewById(R.id.recyclerGroups);
        listaGrupos.setLayoutManager(new LinearLayoutManager(getContext()));
        adaptadorGrupos = new GroupListAdapter(listaDatosGrupos);
        listaGrupos.setAdapter(adaptadorGrupos);
    }

    /**
     * Obtiene y escucha en tiempo real los grupos en los que el usuario
     * autenticado está incluido.
     * Actualiza la interfaz dinámicamente al detectar cambios.
     */
    private void cargarGruposDelUsuario() {
        FirebaseUser usuario = FirebaseAuth.getInstance().getCurrentUser();
        if (usuario == null) {
            Log.w("ViewGroups", "Usuario no autenticado.");
            return;
        }

        String idUsuario = usuario.getUid();

        listenerGrupos = GroupUtils.escucharGruposDeUsuario(
            idUsuario,
            grupos -> {
                listaDatosGrupos.clear();
                listaDatosGrupos.addAll(grupos);
                adaptadorGrupos.notifyDataSetChanged();
            },
            error -> Log.e("ViewGroups", "Error al obtener grupos", error)
        );
    }

    /**
     * Detiene el listener de Firestore cuando se destruye la vista del
     * fragmento
     * para evitar fugas de memoria y llamadas innecesarias.
     */
    @Override
    public void onDestoryView() {
        super.onDestoryView();
        if (listenerGrupos != null) {
            listenerGrupos.remove();
            listenerGrupos = null;
        }
    }
}

```

5.1.3 Adapters

Índice de los adapters:

1. AssignedWishlistAdapter.java
2. GroupListAdapter.java
3. invitationsAdapter.java
4. MemberAdapter.java
5. MessageAdapter.java
6. WishlistAdapter.java

A continuación se muestra el código correspondiente a cada una de ellas:

1. AssignedWishlistAdapter.java

```
package NetMind.amigoinvisible.adapters;
import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.view.*;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.models.WishlistItem;
import java.util.List;

/**
 * Adaptador personalizado para mostrar la lista de deseos del amigo asignado.
 * Cada ítem representa un deseo, y al pulsarlo puede abrirse una URL concreta
 * o una búsqueda en Amazon.
 */
public class AssignedWishlistAdapter extends
RecyclerView.Adapter<AssignedWishlistAdapter.ViewHolder> {

    private final List<WishlistItem> lista;
    private final Context context;
    /**
     * Constructor que recibe los datos y contexto necesarios para el
     * adaptador.
     *
     * @param lista    Lista de objetos WishlistItem a mostrar
     * @param context Contexto desde el que se instancia el adaptador
     */
    public AssignedWishlistAdapter(List<WishlistItem> lista, Context context) {
        this.lista = lista;
        this.context = context;
    }
    /**
     * Infla el layout XML de cada ítem de la lista.
     *
     * @param parent   Contenedor padre (RecyclerView)
     * @param viewType Tipo de vista (no se usa en este caso)
     * @return ViewHolder con la vista inflada
     */
    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(context).inflate(R.layout.item_wishlist, parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        WishlistItem item = lista.get(position);
        holder.bind(item);
    }

    @Override
    public int getItemCount() {
        return lista.size();
    }
}
```

```

    */
    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View vista = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.item_deseo_asignado, parent, false);
        return new ViewHolder(vista);
    }
    /**
     * Asocia los datos del deseo a su respectiva vista.
     * Si el ítem tiene una URL válida, se abrirá al pulsar. En caso contrario,
     * se lanza una búsqueda en Amazon.
     *
     * @param holder Contenedor de vistas del ítem
     * @param position Posición actual dentro del RecyclerView
     */
    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        WishlistItem item = lista.get(position);
        holder.texto.setText(item.getTitle());

        holder.texto.setOnClickListener(v -> {
            String url = item.getUrl();
            if (url != null && !url.trim().isEmpty()) {
                context.startActivity(new Intent(Intent.ACTION_VIEW,
                    Uri.parse(url)));
            } else {
                String query = Uri.encode(item.getTitle() + " Amazon");
                context.startActivity(new Intent(Intent.ACTION_VIEW,
                    Uri.parse("https://www.amazon.es/s?k=" + query)));
            }
        });
    }
    /**
     * Devuelve la cantidad de elementos en la lista.
     */
    @Override
    public int getItemCount() {
        return lista.size();
    }
    /**
     * ViewHolder que representa visualmente un deseo.
     */
    public static class ViewHolder extends RecyclerView.ViewHolder {
        TextView texto;
        public ViewHolder(View itemView) {
            super(itemView);
            texto = itemView.findViewById(R.id.textWishItem);
        }
    }
}

```

2. GroupListAdapter.java

```
package NetMind.amigoinvisible.adapters;

import android.content.Context;
import android.content.Intent;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import com.bumptech.glide.Glide;

import java.util.List;

import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.activities.GroupDetailActivity;
import NetMind.amigoinvisible.models.Group;

/**
 * Adaptador personalizado para mostrar una lista de grupos en un RecyclerView.
 * Cada grupo se muestra como una tarjeta con nombre, presupuesto e imagen.
 */
public class GroupListAdapter extends
RecyclerView.Adapter<GroupListAdapter.GroupViewHolder> {

    private List<Group> listaGrupos;

    /**
     * Constructor
     *
     * @param listaGrupos Lista de grupos a mostrar
     */
    public GroupListAdapter(List<Group> listaGrupos) {
        this.listaGrupos = listaGrupos;
    }

    /**
     * Infla la vista de cada tarjeta del grupo
     */
    @NonNull
    @Override
    public GroupViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View vista =
LayoutInflator.from(parent.getContext()).inflate(R.layout.item_group_card,
parent, false);
        return new GroupViewHolder(vista);
    }

    /**
     * Asocia los datos del grupo a cada vista del item.
    
```

```

    */
@Override
public void onBindViewHolder(@NonNull ViewHolder holder, int position)
{
    Group grupo = listaGrupos.get(position);

    holder.nombreGrupo.setText(grupo.getName());

    String presupuestoTexto = holder.itemView.getContext().getString(
        R.string.texto_presupuesto_grupo, grupo.getBudgetLimit()
    );
    holder.presupuestoGrupo.setText(presupuestoTexto);

    // Cargar imagen del grupo con Glide o imagen por defecto
    if (grupo.getImageUrl() != null && !grupo.getImageUrl().isEmpty()) {
        Glide.with(holder.itemView.getContext())
            .load(grupo.getImageUrl())
            .placeholder(R.drawable.ic_groups)
            .into(holder.imagenGrupo);
    } else {
        holder.imagenGrupo.setImageResource(R.drawable.ic_groups);
    }

    // Al hacer clic en la tarjeta, abrir el detalle del grupo
    holder.itemView.setOnClickListener(v -> {
        Context contexto = v.getContext();
        Intent intent = new Intent(contexto, GroupDetailActivity.class);
        intent.putExtra("groupId", grupo.getId());
        contexto.startActivity(intent);
    });
}

@Override
public int getItemCount() {
    return listaGrupos.size();
}

/**
 * ViewHolder que contiene las vistas de cada tarjeta
 */
public static class ViewHolder extends RecyclerView.ViewHolder {
    TextView nombreGrupo, presupuestoGrupo;
    ImageView imagenGrupo;

    public ViewHolder(@NonNull View itemView) {
        super(itemView);
        nombreGrupo = itemView.findViewById(R.id.txtGroupName);
        presupuestoGrupo = itemView.findViewById(R.id.txtGroupBudget);
        imagenGrupo = itemView.findViewById(R.id.imageGroupCard);
    }
}
}

```

3. InvitationsAdapter.java

```
package NetMind.amigoinvisible.adapters;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import java.util.List;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.models.Invitation;

/**
 * Adaptador personalizado para mostrar una lista de invitaciones pendientes.
 * Muestra el nombre del grupo y botones para aceptar o rechazar.
 */
public class InvitationsAdapter extends RecyclerView.Adapter<InvitationsAdapter.InvitationViewHolder> {

    /**
     * Interfaz para gestionar la acción de aceptar una invitación.
     */
    public interface OnAcceptClickListener {
        void onAccept(Invitation invitation);
    }

    /**
     * Interfaz para gestionar la acción de rechazar una invitación.
     */
    public interface OnRejectClickListener {
        void onReject(Invitation invitation);
    }

    private final List<Invitation> invitaciones;
    private final OnAcceptClickListener acceptListener;
    private final OnRejectClickListener rejectListener;

    /**
     * Constructor del adaptador
     *
     * @param invitaciones Lista de objetos Invitation a mostrar.
     * @param acceptListener Función a ejecutar cuando se acepta una invitación.
     * @param rejectListener Función a ejecutar cuando se rechaza una invitación.
     */
    public InvitationsAdapter(List<Invitation> invitaciones,
                             OnAcceptClickListener acceptListener,
                             OnRejectClickListener rejectListener) {
        this.invitaciones = invitaciones;
        this.acceptListener = acceptListener;
        this.rejectListener = rejectListener;
    }

    /**

```

```

        * Infla la vista del ítem (item_invitation.xml)
        */
    @NonNull
    @Override
    public InvitationViewHolder onCreateViewHolder(@NonNull ViewGroup parent,
    int viewType) {
        View vista = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.item_invitation, parent, false);
        return new InvitationViewHolder(vista);
    }

    /**
     * Asocia los datos a cada ítem del RecyclerView.
     */
    @Override
    public void onBindViewHolder(@NonNull InvitationViewHolder holder, int
    position) {
        Invitation invitacion = invitaciones.get(position);
        holder.nombreGrupo.setText(invitacion.getGroupName());

        // Acción al pulsar Aceptar
        holder.botonAceptar.setOnClickListener(v -> {
            if (acceptListener != null) {
                acceptListener.onAccept(invitacion);
            }
        });
        // Acción al pulsar Rechazar
        holder.botonRechazar.setOnClickListener(v -> {
            if (rejectListener != null) {
                rejectListener.onReject(invitacion);
            }
        });
    }

    @Override
    public int getItemCount() {
        return invitaciones.size();
    }

    /**
     * ViewHolder que mantiene las referencias a los elementos de la vista de
     un ítem.
     */
    public static class InvitationViewHolder extends RecyclerView.ViewHolder {
        TextView nombreGrupo;
        Button botonAceptar, botonRechazar;

        public InvitationViewHolder(@NonNull View itemView) {
            super(itemView);
            nombreGrupo = itemView.findViewById(R.id.txt_nombre_grupo);
            botonAceptar = itemView.findViewById(R.id.btn_aceptar);
            botonRechazar = itemView.findViewById(R.id.btn_rechazar);
        }
    }
}

```

4 MemberAdapter.java

```
package NetMind.amigoinvisible.adapters;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AlertDialog;
import androidx.recyclerview.widget.RecyclerView;
import com.firebaseio.ui.firestore.FirestoreRecyclerAdapter;
import com.firebaseio.ui.firestore.FirestoreRecyclerOptions;
import com.google.firebaseio.firebaseio.FirebaseFirestore;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.models.Member;

/**
 * Adaptador para mostrar la lista de miembros de un grupo en un RecyclerView,
 * Permite mostrar en tiempo real los cambios en la colección de miembros en
 * Firestore y
 * proporciona funcionalidad para eliminar miembros desde la interfaz con
 * confirmación
 */
public class MemberAdapter extends FirestoreRecyclerAdapter<Member,
MemberAdapter.MemberViewHolder> {

    private final String idGrupo;

    /**
     * Constructor que recibe las opciones y el ID del grupo al que pertenecen
     * los miembros.
     *
     * @param opciones Opciones de configuración para FirestoreRecyclerAdapter
     * @param idGrupo ID del grupo Firestore donde están los miembros
     */
    public MemberAdapter(@NonNull FirestoreRecyclerOptions<Member> opciones,
String idGrupo) {
        super(opciones);
        this.idGrupo = idGrupo;
    }

    /**
     * Asocia los datos de cada miembro al ViewHolder y configura la acción de
     * eliminar.
     */
    @Override
    protected void onBindViewHolder(@NonNull MemberViewHolder holder, int
position, @NonNull Member miembro) {
        holder.textoNombre.setText(miembro.getName());
        holder.textoCorreo.setText(miembro.getEmail());

        // Capturamos el ID del documento una sola vez desde el parámetro
        'position', no desde el holder
    }
}
```

```

        String memberId = getSnapshots().getSnapshot(position).getId();

        holder.iconoEliminar.setOnClickListener(v -> {
            new AlertDialog.Builder(holder.itemView.getContext())
                .setTitle(R.string.dialogo_titulo_eliminar_participante)

                .setMessage(holder.itemView.getContext().getString(R.string.dialogo_mensaje_eliminacion_participante, miembro.getName()))
                .setPositiveButton(R.string.dialogo_boton_si, (dialog,
which) -> {

                    FirebaseFirestore.getInstance()
                        .collection("groups")
                        .document(idGrupo)
                        .collection("members")
                        .document(memberId) // usamos el ID seguro
                        .delete()
                        .addOnSuccessListener(unused ->

                    Toast.makeText(holder.itemView.getContext(),
R.string.mensaje_participante_eliminado, Toast.LENGTH_SHORT).show()
                )
                .addOnFailureListener(e ->

                    Toast.makeText(holder.itemView.getContext(), "Error al eliminar participante",
Toast.LENGTH_SHORT).show()
                );
            }
            .setNegativeButton(R.string.dialogo_boton_no, null)
            .show();
        });
    }

    /**
     * Infla la vista de cada miembro (ítem) desde XML.
     *
     * @param parent Contenedor del RecyclerView.
     * @param viewType Tipo de vista (no utilizado en este adaptador).
     * @return ViewHolder con las referencias inicializadas.
     */
    @NonNull
    @Override
    public MemberViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View vista = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.item_member, parent, false);
        return new MemberViewHolder(vista);
    }

    /**
     * ViewHolder que representa visualmente un miembro en la lista.
     */
    static class MemberViewHolder extends RecyclerView.ViewHolder {
        TextView textoNombre, textoCorreo;
        ImageView iconoEliminar;
    }
}

```

```
public MemberViewHolder(@NonNull View itemView) {
    super(itemView);
    textoNombre = itemView.findViewById(R.id.textName);
    textoCorreo = itemView.findViewById(R.id.textEmail);
    iconoEliminar = itemView.findViewById(R.id.iconDelete);
}
}
```



5. MessageAdapter.java

```
package NetMind.amigoinvisible.adapters;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import com.google.firebase.auth.FirebaseAuth;
import java.util.List;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.models.Message;

/**
 * Adaptador personalizado que gestiona la visualización
 * de mensajes en una interfaz de tipo chat. Se adapta según si el mensaje fue
 * enviado
 * o recibido por el usuario actual.
 */
public class MessageAdapter extends
RecyclerView.Adapter<RecyclerView.ViewHolder> {

    private static final int TIPO_ENVIADO = 1;
    private static final int TIPO_RECIBIDO = 2;

    private final List<Message> listaMensajes;
    private final String idUsuarioActual;

    /**
     * Constructor del adaptador.
     *
     * @param listaMensajes Lista de mensajes a mostrar.
     */
    public MessageAdapter(List<Message> listaMensajes) {
        this.listaMensajes = listaMensajes;
        this.idUsuarioActual =
        FirebaseAuth.getInstance().getCurrentUser().getUid();
    }

    /**
     * Determina si el mensaje debe mostrarse como enviado o recibido.
     *
     * @param posicion Índice del mensaje en la lista.
     * @return Tipo de vista (enviado o recibido).
     */
    @Override
    public int getItemViewType(int posicion) {
        Message mensaje = listaMensajes.get(posicion);
        return mensaje.getSenderId().equals(idUsuarioActual) ? TIPO_ENVIADO :
        TIPO_RECIBIDO;
    }

    /**
     * Infla la vista del mensaje según su tipo.
     *
```

```

        * @param padre      ViewGroup padre del RecyclerView.
        * @param tipoVista Tipo de mensaje (enviado o recibido).
        * @return ViewHolder correspondiente al tipo de mensaje.
        */
    @NonNull
    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup padre,
    int tipoVista) {
        View vista;
        if (tipoVista == TIPO_ENVIADO) {
            vista =
LayoutInflater.from(padre.getContext()).inflate(R.layout.item_mensaje_enviado,
padre, false);
            return new ViewHolderMensajeEnviado(vista);
        } else {
            vista =
LayoutInflater.from(padre.getContext()).inflate(R.layout.item_mensaje_recibido,
padre, false);
            return new ViewHolderMensajeRecibido(vista);
        }
    }

    /**
     * Asocia el texto del mensaje con el TextView correspondiente.
     *
     * @param holder      ViewHolder correspondiente (enviado o recibido).
     * @param posicion   Posición del mensaje en la lista.
     */
    @Override
    public void onBindViewHolder(@NonNull RecyclerView.ViewHolder holder, int
posicion) {
        Message mensaje = listaMensajes.get(posicion);
        if (holder instanceof ViewHolderMensajeEnviado) {
            ((ViewHolderMensajeEnviado)
holder).textoMensaje.setText(mensaje.getContent());
        } else if (holder instanceof ViewHolderMensajeRecibido) {
            ((ViewHolderMensajeRecibido)
holder).textoMensaje.setText(mensaje.getContent());
        }
    }

    /**
     * Devuelve la cantidad total de mensajes en la lista.
     *
     * @return Número total de ítems.
     */
    @Override
    public int getItemCount() {
        return listaMensajes.size();
    }

    /**
     * ViewHolder para mensajes enviados por el usuario.
     */

```

```
public static class ViewHolderMensajeEnviado extends
RecyclerView.ViewHolder {
    TextView textoMensaje;

    public ViewHolderMensajeEnviado(@NonNull View itemView) {
        super(itemView);
        textoMensaje = itemView.findViewById(R.id.messageText);
    }
}

/**
 * ViewHolder para mensajes recibidos de otros usuarios.
 */
public static class ViewHolderMensajeRecibido extends
RecyclerView.ViewHolder {
    TextView textoMensaje;

    public ViewHolderMensajeRecibido(@NonNull View itemView) {
        super(itemView);
        textoMensaje = itemView.findViewById(R.id.messageText);
    }
}
```



6. WishlistAdapter.java

```
package NetMind.amigoinvisible.adapters;
import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import com.google.firebase.firestore.FirebaseFirestore;
import java.util.List;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.models.WishlistItem;

/**
 * Adaptador para la visualización y gestión de la lista de deseos del usuario
 * dentro de un grupo en un RecyclerView.
 * Cada elemento de la lista puede ser pulsado para abrir una búsqueda en
Amazon
 * relacionada con ese deseo. Además, incluye un botón para eliminar el ítem
tanto
 * visualmente como en la base de datos de Firestore
 */
public class WishlistAdapter extends
RecyclerView.Adapter<WishlistAdapter.ViewHolder> {

    private final List<WishlistItem> deseos;
    private final Context context;
    private final String groupId;
    private final FirebaseFirestore db = FirebaseFirestore.getInstance();

    /**
     * Constructor del adaptador.
     *
     * @param deseos Lista de deseos actual del usuario.
     * @param context Contenido de la actividad o fragmento.
     * @param groupId ID del grupo donde pertenece el usuario.
     */
    public WishlistAdapter(List<WishlistItem> deseos, Context context, String
groupId) {
        this.deseos = deseos;
        this.context = context;
        this.groupId = groupId;
    }
    /**
     * Crea y retorna un nuevo ViewHolder inflado desde el XML.
     */
    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
```

```

        View vista = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.item_wishlist, parent, false);
        return new ViewHolder(vista);
    }
    /**
     * Asocia cada deseo al ViewHolder, con acciones para búsqueda y
     * eliminación.
     */
    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        WishlistItem item = deseos.get(position);
        holder.textView.setText(item.getTitle());

        // Acciones al hacer clic sobre el texto
        holder.textView.setOnClickListener(v -> {
            if (item.getUrl() != null && !item.getUrl().trim().isEmpty()) {
                // Abrir enlace directo si hay URL
                Intent i = new Intent(Intent.ACTION_VIEW,
                    Uri.parse(item.getUrl()));
                context.startActivity(i);
            } else {
                // Búsqueda en Amazon si no hay URL
                String query = Uri.encode(item.getTitle() + " Amazon");
                Intent intent = new Intent(Intent.ACTION_VIEW,
                    Uri.parse("https://www.amazon.es/s?k=" + query));
                context.startActivity(intent);
            }
        });
        // Eliminar ítem
        holder.deleteIcon.setOnClickListener(v -> {
            deseos.remove(position);
            notifyItemRemoved(position);
            notifyItemRangeChanged(position, deseos.size());
            db.collection("groups").document(groupId)
                .update("wishlist", deseos)
                .addOnFailureListener(e ->
                    Toast.makeText(context,
                        R.string.mensaje_error_deseo, Toast.LENGTH_SHORT).show()
                );
        });
    }

    /**
     * Devuelve la cantidad actual de deseos en la lista.
     */
    @Override
    public int getItemCount() {
        return deseos.size();
    }

    /**
     * ViewHolder que representa cada ítem de la lista de deseos.
     */

```

```
static class ViewHolder extends RecyclerView.ViewHolder {  
    TextView textView;  
    ImageView deleteIcon;  
  
    ViewHolder(View itemView) {  
        super(itemView);  
        textView = itemView.findViewById(R.id.textWishItem);  
        deleteIcon = itemView.findViewById(R.id.btnDeleteWish);  
    }  
}
```



5.1.4 Models

Índice de los models:

1. Group.java
2. Invitation.java
3. Member.java
4. Message.java
5. WishlistItem.java

A continuación se muestra el código correspondiente a cada una de ellas:

1. Group.java

```
package NetMind.amigoinvisible.models;  
/**  
 * Modelo de datos que representa un grupo de Amigo Invisible.  
 * Contiene información como nombre, dueño, límite de participantes,  
presupuesto y URL de imagen.  
 */  
public class Group {  
    private String id;  
    private String name;  
    private String owner;  
    private int maxParticipants;  
    private double budgetLimit;  
    private String imageUrl;  
  
    public Group() {  
        // Constructor vacío necesario para la deserialización desde Firestore  
    }  
  
    public Group(String id, String name, String owner, int maxParticipants,  
double budgetLimit, String imageUrl) {  
        this.id = id;  
        this.name = name;  
        this.owner = owner;  
        this.maxParticipants = maxParticipants;  
        this.budgetLimit = budgetLimit;  
        this.imageUrl = imageUrl;  
    }  
    // Getters y Setters  
    public String getId() {  
        return id;  
    }  
    public void setId(String id) {  
        this.id = id;  
    }  
    public String getName() {  
        return name;  
    }  
    public String getOwner() {  
        return owner;  
    }  
    public int getMaxParticipants() {  
        return maxParticipants;
```

```
    }
    public double getBudgetLimit() {
        return budgetLimit;
    }
    public String getImageUrl() {
        return imageUrl;
    }
    public void setImageUrl(String imageUrl) {
        this.imageUrl = imageUrl;
    }
}
```



2. Invitations.java

```
package NetMind.amigoinvisible.models;

import java.util.Map;
/**
 * Modelo que representa una invitación pendiente para unirse a un grupo.
 * Contiene el ID del grupo, su nombre y los datos del miembro invitado
 * (nombre, email, etc.).
 */
public class Invitation {
    private String groupId;
    private String groupName;
    private Map<String, Object> datosMiembro;
    /**
     * Constructor requerido por Firestore para deserialización automática.
     */
    public Invitation() {}

    /**
     * Constructor principal para crear una invitación.
     *
     * @param groupId      ID único del grupo.
     * @param groupName    Nombre legible del grupo.
     * @param datosMiembro Mapa con datos asociados al miembro (nombre,
     * email...).
     */
    public Invitation(String groupId, String groupName, Map<String, Object>
datosMiembro) {
        this.groupId = groupId;
        this.groupName = groupName;
        this.datosMiembro = datosMiembro;
    }

    // Getters
    public String getGroupId() {
        return groupId;
    }

    public String getGroupName() {
        return groupName;
    }

    public Map<String, Object> getDatosMiembro() {
        return datosMiembro;
    }
}
```

3 Member.java

```
package NetMind.amigoinvisible.models;

import java.util.List;
import java.util.Map;

/**
 * Modelo que representa un miembro dentro de un grupo.
 * Contiene su nombre, email y la lista de deseos personalizada.
 */
public class Member {
    private String name;
    private String email;
    private List<Map<String, Object>> wishlist;

    /**
     * Constructor requerido por Firestore para deserialización automática.
     */
    public Member() {}

    public String getName() { return name; }
    public String getEmail() { return email; }
    public List<Map<String, Object>> getWishlist() { return wishlist; }
}
```

4 Message.java

```
package NetMind.amigoinvisible.models;
/***
 * Modelo que representa un mensaje dentro del chat del grupo.
 * Cada mensaje contiene el contenido, el remitente, un timestamp y
 * un indicador de anonimato.
 */
public class Message {
    private String senderId;
    private String content;
    private com.google.firebaseio.Timestamp timestamp;

    private boolean isAnonymous;

    /**
     * Constructor vacío requerido por Firestore.
     */
    public Message() {
    }
    /**
     * Constructor completo para crear un nuevo mensaje.
     *
     * @param senderId      ID del usuario que envía el mensaje.
     * @param content        Texto del mensaje.
     * @param timestamp      Fecha y hora en que se envió.
     * @param isAnonymous   Si el mensaje debe mostrarse como anónimo.
     */
    public Message(String senderId, String content,
com.google.firebaseio.Timestamp timestamp, boolean isAnonymous) {
        this.senderId = senderId;
        this.content = content;
        this.timestamp = timestamp;
        this.isAnonymous = isAnonymous;
    }

    // Getters y Setters
    public String getSenderId() { return senderId; }
    public void setSenderId(String senderId) { this.senderId = senderId; }

    public String getContent() { return content; }
    public void setContent(String content) { this.content = content; }
    public boolean isAnonymous() { return isAnonymous; }
    public void setAnonymous(boolean isAnonymous) { this.isAnonymous =
isAnonymous; }
    public com.google.firebaseio.Timestamp getTimestamp() {
        return timestamp;
    }
    public void setTimestamp(com.google.firebaseio.Timestamp timestamp) {
        this.timestamp = timestamp;
    }
}
```

```

5 WishlistItem.java
package NetMind.amigoinvisible.models;
import java.util.HashMap;
import java.util.Map;
/**
 * Modelo de datos que representa un ítem dentro de la lista de deseos de un
miembro.
 * Contiene un título (nombre del deseo) y una URL opcional del producto.
 * Se utiliza tanto para mostrar en la UI como para almacenar en Firestore.
 */
public class WishlistItem {
    private String title;
    private String url;
    /**
     * Constructor vacío requerido por Firestore para la deserialización
automática.
    */
    public WishlistItem() {}
    /**
     * Constructor personalizado para crear un deseo con nombre y URL.
     *
     * @param title Nombre o descripción del deseo.
     * @param url   URL opcional del producto (puede ser null o vacío).
    */
    public WishlistItem(String title, String url) {
        this.title = title;
        this.url = url;
    }
    public String getTitle() {
        return title;
    }
    public String getUrl() {
        return url;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public void setUrl(String url) {
        this.url = url;
    }
    /**
     * Convierte este objeto en un mapa clave-valor,
     * ideal para subirlo como parte de un array a Firestore.
     *
     * @return Mapa con los campos "title" y "url".
    */
    public Map<String, Object> toMap() {
        Map<String, Object> map = new HashMap<>();
        map.put("title", title);
        map.put("url", url);
        return map;
    }
}

```

5.1.5 Utils

Índice de las clases utilitarias:

1. EmailSender.java
2. FirebaseUtils.java
3. GroupUtils.java
4. InvitationUtils.java
5. UsuarioUtils.java

A continuación se muestra el código correspondiente a cada una de ellas:

1. EmailSender.java

```
package NetMind.amigoinvisible.utils;
import com.google.firebaseio.firebaseio.FirebaseFirestore;
import java.util.*;

/**
 * Clase utilitaria para el envío de correos electrónicos desde Firebase
 * mediante una colección Firestore.
 * Utiliza una estructura compatible con la extensión de envío de correo
 * (`firebase-extensions/sendgrid`).
 */
public class EmailSender {
    private static final String COLLECTION_NAME = "mail";
    /**
     * Encola un correo a un solo destinatario en Firestore
     * Este método es una versión simplificada del método que acepta una lista
     * de destinatarios.
     * Internamente convierte el destinatario en una lista de un solo elemento
     * y delega la lógica.
     * @param destinatario Correo del destinatario
     * @param asunto Asunto del correo
     * @param cuerpoPlano Versión texto plano del mensaje
     * @param cuerpoHTML Versión HTML del mensaje
     */
    public static void enviarCorreo(String destinatario, String asunto, String
cuerpoPlano, String cuerpoHTML) {
        if (destinatario == null || destinatario.trim().isEmpty()) {
            System.err.println("X Dirección de correo vacía o nula. No se
enviará el correo.");
            return;
        }
        enviarCorreo(Collections.singletonList(destinatario), asunto,
cuerpoPlano, cuerpoHTML);
    }
    /**
     * Envía un correo a múltiples destinatarios.
     * Este método crea un documento en la colección "mail" de Firestore que
     * será procesado
     * por una extensión backend (como Firebase Extension para envío de
     * correos).
     * @param destinatarios Lista de correos electrónicos
     * @param asunto Asunto del correo
    }
```

```

* @param cuerpoPlano    Versión texto plano del mensaje
* @param cuerpoHTML      Versión HTML del mensaje
*/
public static void enviarCorreo(List<String> destinatarios, String asunto,
String cuerpoPlano, String cuerpoHTML) {
    if (destinatarios == null || destinatarios.isEmpty()) {
        System.err.println("X Lista de destinatarios vacía. No se enviará
el correo.");
        return;
    }
    Map<String, Object> emailData = construirCorreo(destinatarios, asunto,
cuerpoPlano, cuerpoHTML);

    FirebaseFirestore.getInstance()
        .collection(COLLECTION_NAME)
        .add(emailData)
        .addOnSuccessListener(docRef -> {
            System.out.println("✓ Correo encolado correctamente. ID
Firestore: " + docRef.getId());
            System.out.println("📦 Enviado a: " + String.join(", ", destinatarios));
        })
        .addOnFailureListener(e -> {
            System.err.println("X Error al encolar correo: " +
e.getMessage());
            e.printStackTrace();
        });
}
/***
 * Construye el mapa con la estructura del correo según espera la extensión
de envío.
 *
 * @param destinatarios Lista de correos electrónicos
 * @param asunto         Asunto del mensaje
 * @param cuerpoPlano   Contenido plano
 * @param cuerpoHTML    Contenido HTML
 * @return Mapa listo para insertar en Firestore
 */
private static Map<String, Object> construirCorreo(List<String>
destinatarios, String asunto, String cuerpoPlano, String cuerpoHTML) {
    Map<String, Object> emailData = new HashMap<>();

    emailData.put("to", destinatarios);

    Map<String, Object> message = new HashMap<>();
    message.put("subject", asunto != null ? asunto : "(Sin asunto)");
    message.put("text", cuerpoPlano != null ? cuerpoPlano : "");
    message.put("html", cuerpoHTML != null ? cuerpoHTML : "");

    emailData.put("message", message);
    return emailData;
}
}

```

2. **FirebaseUtils.java**

```
package NetMind.amigoinvisible.utils;

import android.util.Log;

import androidx.annotation.Nullable;

import com.google.firebase.auth.FirebaseAuth;
import com.google.firebaseio.*;

import java.util.*;

/**
 * Clase utilitaria para lógica avanzada con Firestore.
 * Contiene funciones que permiten sincronizar automáticamente al usuario con
grupos
 * donde fue invitado previamente mediante su correo electrónico.
 */
public class FirebaseUtils {

    private static final String TAG = "FirebaseUtils";

    /**
     * Empareja al usuario autenticado con los grupos en los que fue invitado
previamente por su correo electrónico.
     * Si el usuario ya tiene UID registrado en el array `members`, se omite.
     * Si fue invitado por correo, migra sus datos a su UID actual, asegura su
inclusión en el array y elimina el documento antiguo.
     *
     * @param usuario    Usuario autenticado.
     * @param onComplete Acción que se ejecuta una vez completado el
emparejamiento en todos los grupos.
     */
    public static void emparejarUsuarioConGrupos(FirebaseUser usuario,
@Nullable Runnable onComplete) {
        if (usuario == null || usuario.getEmail() == null) {
            if (onComplete != null) onComplete.run();
            return;
        }

        final String uid = usuario.getUid();
        final String correo = usuario.getEmail();
        FirebaseFirestore db = FirebaseFirestore.getInstance();

        db.collection("groups").get().addOnSuccessListener(grupos -> {
            List<DocumentSnapshot> documentos = grupos.getDocuments();
            if (documentos.isEmpty()) {
                if (onComplete != null) onComplete.run();
                return;
            }

            final int total = documentos.size();
            final int[] procesados = {0};
```

```

        for (DocumentSnapshot grupoDoc : documentos) {
            String groupId = grupoDoc.getId();
            List<String> miembrosActuales = (List<String>) grupoDoc.get("members");
            Long maxParticipantes = grupoDoc.getLong("maxParticipants");

            if (miembrosActuales == null) miembrosActuales = new ArrayList<>();
            final List<String> miembrosFinales = new ArrayList<>(miembrosActuales);

            // Si el UID ya está en el array, solo nos aseguramos de que no falta como documento
            if (miembrosFinales.contains(uid)) {
                Log.d("FirebaseUtils", "UID ya está en grupo " + groupId +
", se omite migración.");
                if (++procesados[0] == total && onComplete != null)
onComplete.run();
                continue;
            }

            // Buscar por correo en subcolección "members"
            grupoDoc.getReference().collection("members").whereEqualTo("email",
correo).get().addOnSuccessListener(miembrosCoincidentes -> {
                boolean yaExiste = false;

                for (DocumentSnapshot miembroDoc :
miembrosCoincidentes.getDocuments()) {
                    String miembroId = miembroDoc.getId();

                    if (miembroId.equals(uid)) {
                        Log.d("FirebaseUtils", "UID ya tiene doc en
subcolección para grupo " + groupId);
                        yaExiste = true;
                        break;
                    }
                }

                if (yaExiste) {
                    // El documento ya existe, solo aseguramos que el UID
esté en el array
                    if (!miembrosFinales.contains(uid)) {
                        miembrosFinales.add(uid);
                        grupoDoc.getReference().update("members",
miembrosFinales);
                        Log.d("FirebaseUtils", "UID añadido al array
members[] de grupo " + groupId);
                    }
                } else {
                    // Migrar datos desde otro doc de correo
                    for (DocumentSnapshot miembroDoc :
miembrosCoincidentes.getDocuments()) {
                        Map<String, Object> datosMiembro =
miembroDoc.getData();

```

```

        if (datosMiembro == null) continue;

grupoDoc.getReference().collection("members").document(uid).set(datosMiembro).a
ddOnSuccessListener(aVoid -> {
    miembroDoc.getReference().delete();
    Log.d("FirebaseUtils", "Migrado miembro a UID "
+ uid);

    if (!miembrosFinales.contains(uid)) {
        miembrosFinales.add(uid);
        grupoDoc.getReference().update("members",
miembrosFinales);
        Log.d("FirebaseUtils", "UID añadido al
array members[] de grupo " + groupId);
    }
});
}
}

if (++procesados[0] == total && onComplete != null) {
    onComplete.run();
}
}).addOnFailureListener(e -> {
    Log.e("FirebaseUtils", "Error buscando miembros en grupo "
+ groupId, e);
    if (++procesados[0] == total && onComplete != null) {
        onComplete.run();
    }
});
}
}).addOnFailureListener(e -> {
    Log.e("FirebaseUtils", "Error accediendo a grupos", e);
    if (onComplete != null) onComplete.run();
});
}
}

/**
 * Controla la ejecución del callback final cuando todos los grupos han
sido procesados.
 *
 * @param procesados Array que lleva la cuenta de los grupos completados.
 * @param total      Número total de grupos.
 * @param onComplete Acción a ejecutar al finalizar.
 */
private static void finalizarGrupo(int[] procesados, int total, @Nullable
Runnable onComplete) {
    if (++procesados[0] == total && onComplete != null) {
        onComplete.run();
    }
}
}
}

```

```

3. GroupUtils.java
package NetMind.amigoinvisible.utils;
import androidx.annotation.NonNull;
import com.google.firebase.firestore.*;
import java.util.*;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.function.Consumer;

import NetMind.amigoinvisible.models.Group;

/**
 * Clase utilitaria que contiene toda la lógica relacionada con grupos.
 * Incluye operaciones como realizar el sorteo, guardar resultados, y escuchar
actualizaciones en tiempo real.
 */
public class GroupUtils {

    private static final FirebaseFirestore db =
FirebaseFirestore.getInstance();

    /**
     * Ejecuta el proceso del sorteo de Amigo Invisible.
     * * 1. Obtiene participantes
     * * 2. Verifica que no se haya sorteado antes
     * * 3. Genera asignaciones aleatorias sin repetir
     * * 4. Guarda las asignaciones
     * * 5. Notifica por email
     *
     * @param groupId ID del grupo
     * @param onSuccess Callback en caso de éxito
     * @param onError Callback en caso de error con mensaje
     */
    public static void realizarSorteo(String groupId, Runnable onSuccess,
Consumer<String> onError) {
        DocumentReference grupoRef = db.collection("groups").document(groupId);

        obtenerParticipantes(grupoRef, (participantes) -> {
            if (participantes.size() < 2) {
                onError.accept("Debe haber al menos 2 miembros para realizar el
sorteo.");
                return;
            }

            verificarSorteoPrevio(grupoRef, yaRealizado -> {
                if (yaRealizado) {
                    onError.accept("El sorteo ya fue realizado para este
grupo.");
                    return;
                }

                List<String> receptores = generarAsignaciones(participantes);
                if (receptores == null) {
                    onError.accept("No se pudo generar una asignación válida.
Inténtalo nuevamente.");
                    return;
                }
            });
        });
    }
}

```

```

        }

        guardarResultadosYNotificar(grupoRef, participantes,
receptores, onSuccess, onError);
    }, onError);

}, onError);
}

/**
 * Recupera la lista de participantes (IDs de usuarios) de un grupo
Firestore.
*
* @param grupoRef
* @param onResult
* @param onError
*/
private static void obtenerParticipantes(DocumentReference grupoRef,
                                         Consumer<List<String>> onResult,
                                         Consumer<String> onError) {
    grupoRef.collection("members").get()
        .addOnSuccessListener(snapshot -> {
            List<String> participantes = new ArrayList<>();
            for (DocumentSnapshot doc : snapshot.getDocuments()) {
                participantes.add(doc.getId());
            }
            onResult.accept(participantes);
        })
        .addOnFailureListener(e -> onError.accept("Error al obtener los
miembros: " + e.getMessage()));
}

/**
 * Verifica si ya existe una colección de asignaciones para ese grupo.
*
* @param grupoRef
* @param onResult
* @param onError
*/
private static void verificarSorteoPrevio(DocumentReference grupoRef,
                                         Consumer<Boolean> onResult,
                                         Consumer<String> onError) {
    grupoRef.collection("assignments").get()
        .addOnSuccessListener(snapshot ->
onResult.accept(!snapshot.isEmpty()))
        .addOnFailureListener(e -> onError.accept("Error al verificar
sorteo previo: " + e.getMessage()));
}

/**
 * Genera una asignación aleatoria válida sin asignarse a sí mismo.
*
* @param participantes
* @return

```

```

/*
private static List<String> generarAsignaciones(List<String> participantes)
{
    List<String> receptores = new ArrayList<>(participantes);
    Random random = new Random();

    for (int intento = 0; intento < 10; intento++) {
        Collections.shuffle(receptores, random);
        boolean valido = true;

        for (int i = 0; i < participantes.size(); i++) {
            if (participantes.get(i).equals(receptores.get(i))) {
                valido = false;
                break;
            }
        }

        if (valido) return receptores;
    }

    return null; // No se logró generar una asignación válida
}

/**
 * Guarda las asignaciones en Firestore y envía correos de notificación a
cada participante.
*
* @param grupoRef
* @param asignadores
* @param asignados
* @param onSuccess
* @param onError
*/
private static void guardarResultadosYNotificar(
    DocumentReference grupoRef,
    List<String> asignadores,
    List<String> asignados,
    Runnable onSuccess,
    Consumer<String> onError) {

    AtomicInteger tareasPendientes = new AtomicInteger(asignadores.size());

    for (int i = 0; i < asignadores.size(); i++) {
        final String asignadorId = asignadores.get(i);
        final String asignadoId = asignados.get(i);

        DocumentReference asignadorRef =
grupoRef.collection("members").document(asignadorId);
        DocumentReference asignadoRef =
grupoRef.collection("members").document(asignadoId);

        System.out.println("👉 Procesando asignador: " + asignadorId + " →
asignado: " + asignadoId);

        asignadorRef.get().addOnSuccessListener(asignadorDoc -> {

```

```

asignadoRef.get().addOnSuccessListener(asignadoDoc -> {

    // Obtener datos del asignador
    final String emailAsignador =
asignadorDoc.getString("email");
    final String nombreAsignador =
Optional.ofNullable(asignadorDoc.getString("name"))
        .filter(s -> !s.trim().isEmpty()).orElse("(sin
nombre)");

    // Obtener nombre del asignado
    final String nombreAsignado =
Optional.ofNullable(asignadorDoc.getString("name"))
        .filter(s -> !s.trim().isEmpty()).orElse("(sin
nombre)");

    System.out.println("✉ Email del asignador: " +
emailAsignador);
    System.out.println("👤 Nombre asignador: " +
nombreAsignador + ", asignado: " + nombreAsignado);

    // Preparar batch para guardar asignación
    WriteBatch batch = db.batch();

batch.set(grupoRef.collection("assignments").document(asignadorId),
        Map.of("assignedTo", asignadoId));
batch.update(asignadorRef, "assignedTo", asignadoId);

    // Commit del batch
    batch.commit()
        .addOnSuccessListener(unused -> {
            if (emailAsignador != null &&
!emailAsignador.trim().isEmpty()) {
                String subject = "🎁 Has sido asignado en
el Amigo Invisible";
                String text = "Hola " + nombreAsignador +
", te ha tocado: " + nombreAsignado;
                String html = "<p>🎁 <strong>Hola " +
nombreAsignador + "</strong>,<br>" +
                    "Tu amigo invisible es: <strong>" +
nombreAsignado + "</strong></p>";

                System.out.println("✅ Enviando correo a: " +
+ emailAsignador);
                EmailSender.enviarCorreo(emailAsignador,
subject, text, html);
            } else {
                System.out.println("⚠ Usuario sin email →
UID: " + asignadorId);
            }
        }

    // Verifica si es la última tarea pendiente
    if (tareasPendientes.decrementAndGet() == 0) {
        System.out.println("✅ Todas las
asignaciones completadas");
    }
}

```

```

                onSuccess.run();
            }
        })
        .addOnFailureListener(e -> {
            String mensajeError = "✗ Error al guardar asignación para " + asignadorId + ": " + e.getMessage();
            System.err.println(mensajeError);
            onError.accept(mensajeError);
        });
    });

}).addOnFailureListener(e -> {
    String mensajeError = "✗ Error al leer asignado (" + asignadorId + "): " + e.getMessage();
    System.err.println(mensajeError);
    onError.accept(mensajeError);
});

}).addOnFailureListener(e -> {
    String mensajeError = "✗ Error al leer asignador (" + asignadorId + "): " + e.getMessage();
    System.err.println(mensajeError);
    onError.accept(mensajeError);
});
}

}

/**
 * Escucha en tiempo real los grupos en los que participa un usuario concreto.
 * Devuelve en tiempo real la lista de grupos actualizada.
 *
 * @param userId ID del usuario autenticado.
 * @param onGruposCambiados Callback con la lista actualizada de grupos.
 * @return ListenerRegistration para poder detener la escucha más adelante.
 */
public static ListenerRegistration escucharGruposDeUsuario(
    @NonNull String userId,
    @NonNull Consumer<List<Group>> onGruposCambiados,
    @NonNull Consumer<Exception> onError
) {
    CollectionReference gruposRef = db.collection("groups");

    return gruposRef.addSnapshotListener((snapshot, error) -> {
        if (error != null) {
            onError.accept(error);
            return;
        }

        if (snapshot == null) {
            onGruposCambiados.accept(Collections.emptyList());
            return;
        }

        List<Group> gruposUsuario = new ArrayList<>();

```

```
        for (DocumentSnapshot docGrupo : snapshot.getDocuments()) {
            List<String> miembros = (List<String>) docGrupo.get("members");

            if (miembros != null && miembros.contains(userId)) {
                Group grupo = docGrupo.toObject(Group.class);
                if (grupo != null) {
                    grupo.setId(docGrupo.getId());
                    gruposUsuario.add(grupo);
                }
            }
        }

        onGruposCambiados.accept(gruposUsuario);
    });
}

}
```



4. InvitationUtils.java

```
package NetMind.amigoinvisible.utils;
import androidx.annotation.NonNull;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebaseio.firebaseio.*;
import java.util.*;
import java.util.function.Consumer;

import NetMind.amigoinvisible.models.Invitation;

/**
 * Clase utilitaria para manejar las invitaciones a grupos del usuario.
 * Ofrece funciones de escucha en tiempo real, aceptación y rechazo.
 */
public class InvitationUtils {
    private static final FirebaseFirestore db =
FirebaseFirestore.getInstance();

    /**
     * Escucha en tiempo real las invitaciones para un usuario dado basado en
     su correo.
     * Cada invitación se representa como un documento en la subcolección
     "members" con el ID = correo.
     *
     * @param emailUsuario Email del usuario autenticado
     * @param onRecibidas Callback con lista de invitaciones encontradas
     */
    public static ListenerRegistration escucharInvitaciones(@NonNull String
emailUsuario, @NonNull Consumer<List<Invitation>> onRecibidas) {
        return db.collection("groups").addSnapshotListener((snapshot, error) ->
{
        if (error != null || snapshot == null) {
            onRecibidas.accept(Collections.emptyList()); // Notifica vacío
ante error
            return;
        }

        List<Invitation> invitaciones = new ArrayList<>();
        List<DocumentSnapshot> documentos = snapshot.getDocuments();

        if (documentos.isEmpty()) {
            onRecibidas.accept(invitaciones); // Vacío, sin grupos
            return;
        }

        final int totalGrupos = documentos.size();
        final int[] procesados = {0};

        for (DocumentSnapshot grupo : documentos) {
            String groupId = grupo.getId();
            String groupName = grupo.getString("name");

grupo.getReference().collection("members").document(emailUsuario).get().addOnSu
ccessListener(doc -> {
```



```
        }).addOnFailureListener(e -> onError.accept("Error al cargar grupo: " +  
e.getMessage()));  
    }  
  
    /**  
     * Rechaza una invitación eliminando el documento asociado al correo en la  
     * subcolección "members".  
     *  
     * @param invitacion Objeto Invitation con la info del grupo.  
     * @param emailUsuario Correo del usuario que rechaza.  
     * @param onSuccess Acción tras la eliminación exitosa.  
     * @param onError Callback con mensaje si ocurre un fallo.  
     */  
    public static void rechazarInvitacion(@NonNull Invitation invitacion,  
@NonNull String emailUsuario, @NonNull Runnable onSuccess, @NonNull  
Consumer<String> onError) {  
  
db.collection("groups").document(invitacion.getGroupId()).collection("members")  
.document(emailUsuario).delete().addOnSuccessListener(unused ->  
onSuccess.run()).addOnFailureListener(e -> onError.accept("Error al rechazar  
invitación: " + e.getMessage()));  
    }  
}
```

```

5. UsuarioUtils.java
package NetMind.amigoinvisible.utils;
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.util.Log;
import android.widget.ImageView;
import android.widget.Toast;
import androidx.annotation.NonNull;
import com.bumptech.glide.Glide;
import com.google.firebase.auth.AuthCredential;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.auth.UserProfileChangeRequest;
import com.google.firebaseio.firebaseio.DocumentReference;
import com.google.firebaseio.firebaseio.FirebaseFirestore;
import com.google.firebaseio.messaging.FirebaseMessaging;
import com.google.firebaseio.storage.FirebaseStorage;
import com.google.firebaseio.storage.StorageReference;
import java.util.HashMap;
import java.util.Map;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.activities.WelcomeActivity;

/**
 * Clase utilitaria para gestionar autenticación y perfil de usuario
 * Incluye funciones para login, registro, verificación de email,
 * almacenamiento de imagen y obtención de token FCM.
 */
public class UsuarioUtils {
    private static final String TAG = "UsuarioUtils";
    // Campos Firestore para consistencia
    private static final String FIELD_NOMBRE = "nombre";
    private static final String FIELD_CORREO = "correo";
    private static final String FIELD_FOTO = "foto";
    /**
     * Obtiene el token FCM del dispositivo actual y lo muestra por log.
     * Este token se puede usar para enviar notificaciones personalizadas.
     *
     * @param activity Actividad desde donde se realiza la petición (para logs
     * y recursos).
     */
    public static void obtenerTokenFirebase(Activity activity) {
        FirebaseMessaging.getInstance().getToken().addOnCompleteListener(task
        -> {
            if (task.isSuccessful()) {
                Log.d(TAG, "Token FCM: " + task.getResult());
            } else {
                Log.w(TAG, activity.getString(R.string.error_obtener_token),
                task.getException());
            }
        });
    }
}

```

```

/**
 * Autentica al usuario en Firebase usando credenciales externas (por
ejemplo, Google).
 * En caso de éxito, llama a {@link #verificarUsuario(FirebaseUser,
Activity, Runnable)} para completar el proceso.
 *
 * @param credential Credencial externa (obtenida de Google u otro
proveedor).
 * @param activity Actividad actual para contexto.
 * @param onSuccess Acción a ejecutar si el login es exitoso.
 */
public static void autenticarConFirebase(AuthCredential credential,
Activity activity, Runnable onSuccess) {
    FirebaseAuth auth = FirebaseAuth.getInstance();

    auth.signInWithCredential(credential).addOnCompleteListener(activity,
task -> {
        if (task.isSuccessful() && auth.getCurrentUser() != null) {
            verificarUsuario(auth.getCurrentUser(), activity, onSuccess);
        } else {
            Toast.makeText(activity,
activity.getString(R.string.mensaje_error_autenticacion_firebase),
Toast.LENGTH_SHORT).show();
        }
    });
}

/**
 * Verifica si el usuario ya está en Firestore. Si no, lo registra.
 * También empareja su correo con posibles invitaciones.
 *
 * @param usuario Usuario autenticado
 * @param activity Actividad actual (para navegación y toasts)
 * @param onSuccess Acción a ejecutar cuando termine (pasar a pantalla
principal)
 */
public static void verificarUsuario(@NonNull FirebaseUser usuario, Activity
activity, Runnable onSuccess) {
    usuario.reload().addOnSuccessListener(aVoid -> {
        if (!usuario.isEmailVerified()) {
            Toast.makeText(activity,
activity.getString(R.string.mensaje_verificar_correo),
Toast.LENGTH_LONG).show();
            FirebaseAuth.getInstance().signOut();
            return;
        }

        FirebaseFirestore db = FirebaseFirestore.getInstance();
        DocumentReference refUsuario =
db.collection("users").document(usuario.getUid());

        refUsuario.get().addOnSuccessListener(document -> {
            // Crear documento si no existe
            if (!document.exists()) {
                Map<String, Object> nuevoUsuario = new HashMap<>();

```

```

        nuevoUsuario.put(FIELD_NOMBRE, usuario.getDisplayName() != null ? usuario.getDisplayName() : "");
        nuevoUsuario.put(FIELD_CORREO, usuario.getEmail());
        nuevoUsuario.put(FIELD_FOTO, usuario.getPhotoUrl() != null ? usuario.getPhotoUrl().toString() : "");

        refUsuario.set(nuevoUsuario)
            .addOnSuccessListener(aVoid2 -> Log.d(TAG, activity.getString(R.string.mensaje_registro_exitoso)))
            .addOnFailureListener(e -> Log.e(TAG, activity.getString(R.string.mensaje_error_firestore), e));
    }
    // Guardar token de notificaciones FCM

FirebaseMessaging.getInstance().getToken().addOnSuccessListener(token -> {
    if (token != null && !token.isEmpty()) {
        refUsuario.update("fcmToken", token)
            .addOnSuccessListener(aVoid3 -> Log.d(TAG, "Token FCM guardado"))
            .addOnFailureListener(e -> Log.e(TAG, "Error al guardar token FCM", e));
    }
});
// Asociar al usuario con posibles grupos (invitaciones)
FirebaseUtils.emparejarUsuarioConGrupos(usuario, onSuccess);
}).addOnFailureListener(e -> {
    Toast.makeText(activity,
        activity.getString(R.string.mensaje_error_firestore),
        Toast.LENGTH_SHORT).show();
    Log.e(TAG,
        activity.getString(R.string.mensaje_error_firestore), e);
});
}

/**
 * Sube una imagen de perfil al almacenamiento de Firebase y actualiza el perfil del usuario.
 * También actualiza la URL en el documento del usuario en Firestore.
 *
 * @param usuario      Usuario autenticado.
 * @param uriImagen   URI de la imagen seleccionada por el usuario.
 * @param context       Contexto para mostrar mensajes y cargar recursos.
 * @param destinoImagen ImageView donde se mostrará la imagen una vez subida.
 */
public static void actualizarFotoPerfil(@NonNull FirebaseAuthUser usuario,
@NonNull Uri uriImagen, @NonNull Context context, @NonNull ImageView destinoImagen) {

    StorageReference referenciaAlmacen =
        FirebaseStorage.getInstance().getReference("profile_pictures").child(usuario.getUid() + "/profile.jpg");

```

```

        referenciaAlmacen.putFile(uriImagen).addOnSuccessListener(taskSnapshot
-> referenciaAlmacen.getDownloadUrl().addOnSuccessListener(uriDescargada -> {

            // Actualizar perfil en FirebaseAuth
            UserProfileChangeRequest actualizarPerfil = new
UserProfileChangeRequest.Builder().setPhotoUri(uriDescargada).build();

            usuario.updateProfile(actualizarPerfil).addOnSuccessListener(unused
-> {

Glide.with(context).load(uriDescargada).circleCrop().into(destinoImagen);
Toast.makeText(context,
context.getString(R.string.foto_actualizada), Toast.LENGTH_SHORT).show();

            // Actualizar URL en Firestore
            Map<String, Object> datos = new HashMap<>();
            datos.put("photoUrl", uriDescargada.toString());

FirebaseFirestore.getInstance().collection("users").document(usuario.getUid()).update(datos);
        });
}).addOnFailureListener(e -> Toast.makeText(context,
context.getString(R.string.error_subida_imagen), e.getMessage()),
Toast.LENGTH_SHORT).show();
}

/**
 * Registra un nuevo usuario en Firebase Authentication y crea su documento
en Firestore.
 * También configura su nombre y redirige a la pantalla de bienvenida.
 *
 * @param nombre     Nombre introducido por el usuario.
 * @param correo    Correo electrónico introducido.
 * @param contrasena Contraseña introducida.
 * @param activity   Actividad desde donde se realiza el registro.
 */
public static void registrarNuevoUsuario(String nombre, String correo,
String contrasena, Activity activity) {
    FirebaseAuth auth = FirebaseAuth.getInstance();
    FirebaseFirestore db = FirebaseFirestore.getInstance();

    auth.createUserWithEmailAndPassword(correo,
contrasena).addOnSuccessListener(authResult -> {
        FirebaseUser usuario = auth.getCurrentUser();
        if (usuario != null) {
            // Actualizar nombre del usuario
            UserProfileChangeRequest actualizarPerfil = new
UserProfileChangeRequest.Builder()
                .setDisplayName(nombre)
                .build();

usuario.updateProfile(actualizarPerfil).addOnCompleteListener(profileTask -> {

```

```

        String nombreFinal = profileTask.isSuccessful() ? nombre :
    "";
        if (!profileTask.isSuccessful()) {
            Toast.makeText(activity,
R.string.mensaje_error_actualizar_nombre, Toast.LENGTH_SHORT).show();
        }

        // Crear documento en Firestore
        Map<String, Object> datos = new HashMap<>();
        datos.put("displayName", nombreFinal);
        datos.put("email", usuario.getEmail());
        datos.put("photoUrl", null);

db.collection("users").document(usuario.getUid()).set(datos)
        .addOnSuccessListener(unused -> {
            // Enviar verificación por correo
            usuario.sendEmailVerification()
                .addOnCompleteListener(emailTask -> {
                    if (emailTask.isSuccessful()) {
                        Toast.makeText(activity,
activity.getString(R.string.mensaje_verificacion_enviada),
Toast.LENGTH_LONG).show();
                    }
                    // Cierra sesión hasta que el
                    usuario verifique el correo
                    auth.signOut();
                    activity.finish(); // Cierra la
actividad de registro y vuelve al login
                } else {
                    Toast.makeText(activity,
activity.getString(R.string.mensaje_error_enviar_verificacion),
Toast.LENGTH_SHORT).show();
                }
            });
        })
        .addOnFailureListener(e -> {
            Toast.makeText(activity,
R.string.mensaje_error_firestore, Toast.LENGTH_SHORT).show();
            Log.e("UsuarioUtils", "Error Firestore", e);
        });
    });
}
}).addOnFailureListener(e -> {
    String mensaje = e.getMessage();
    if (mensaje != null &&
mensaje.contains("PASSWORD_DOES_NOT_MEET_REQUIREMENTS")) {
        Toast.makeText(activity,
activity.getString(R.string.mensaje_requisitos_contrasena),
Toast.LENGTH_LONG).show();
    } else {
        Toast.makeText(activity,
activity.getString(R.string.mensaje_error_registro,
mensaje),
Toast.LENGTH_LONG).show();
    }
}

```

```
});  
}  
}
```



5.1.6 Services

Índice de los services:

1. MyFirebaseMessagingService.java
2. TokenManager.java

A continuación se muestra el código correspondiente a cada una de ellas:

1 MyFirebaseMessagingService.java

```
package NetMind.amigoinvisible.services;
import android.annotation.SuppressLint;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.graphics.BitmapFactory;
import android.os.Build;

import androidx.annotation.NonNull;
import androidx.core.app.NotificationCompat;
import androidx.core.app.NotificationManagerCompat;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebaseio.firebaseio.FirebaseFirestore;
import com.google.firebase.messaging.FirebaseMessagingService;
import com.google.firebase.messaging.RemoteMessage;
import NetMind.amigoinvisible.R;
import NetMind.amigoinvisible.activities.GroupDetailActivity;
/***
 * Servicio personalizado para manejar los mensajes push recibidos por Firebase
Cloud Messaging (FCM).
 * Muestra notificaciones al usuario incluso cuando la aplicación está en
primer plano.
 */
public class MyFirebaseMessagingService extends FirebaseMessagingService {
    private static final String TAG = "FCMService";
    /**
     * Se ejecuta cuando se recibe un mensaje FCM.
     *
     * @param remoteMessage Objeto con los datos del mensaje recibido.
     */
    @Override
    public void onMessageReceived(@NonNull RemoteMessage remoteMessage) {
        String title = remoteMessage.getData().get("title");
        String body = remoteMessage.getData().get("body");
        String groupId = remoteMessage.getData().get("groupId");
        if (title != null && body != null && groupId != null) {
            mostrarNotificacion(title, body, groupId);
        }
    }
    /**
     * Se ejecuta cuando se genera un nuevo token FCM.
     * Este token debe actualizarse en la base de datos de Firestore.
    }
```

```

*
 * @param token El nuevo token generado por FCM.
 */
@Override
public void onNewToken(@NonNull String token) {
    FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
    if (user != null) {

        FirebaseFirestore.getInstance().collection("users").document(user.getUid()).update("fcmToken", token);
    }
}

/**
 * Crea y muestra una notificación personalizada con los datos recibidos.
 *
 * @param titulo Título de la notificación.
 * @param mensaje Cuerpo del mensaje.
 * @param groupId ID del grupo para abrir al tocarla.
 */
@SuppressWarnings("MissingPermission")
private void mostrarNotificacion(String titulo, String mensaje, String groupId) {
    // Crear canal para Android 8+
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        NotificationChannel canal = new
NotificationChannel("default_channel", "Notificaciones generales",
NotificationManager.IMPORTANCE_DEFAULT);
        canal.setDescription("Canal para notificaciones generales");
        NotificationManager manager =
getSystemService(NotificationManager.class);
        manager.createNotificationChannel(canal);
    }

    // Intent para abrir el grupo al tocar la notificación
    Intent intent = new Intent(this, GroupDetailActivity.class);
    intent.putExtra("groupId", groupId);
    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
Intent.FLAG_ACTIVITY_CLEAR_TASK);

    PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
intent, PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_IMMUTABLE);

    // Crear notificación
    NotificationCompat.Builder builder = new
NotificationCompat.Builder(this,
"default_channel").setSmallIcon(R.drawable.ic_notification).setLargeIcon(Bitmap
Factory.decodeResource(getResources(), R.drawable.logo_notificacion)) // Recurso en drawable

    .setContentTitle(titulo).setContentText(mensaje).setContentIntent(pendingIntent)
    .setAutoCancel(true).setPriority(NotificationCompat.PRIORITY_HIGH);

    NotificationManagerCompat.from(this).notify(1, builder.build());
}
}

```

2. TokenManager.java

```
package NetMind.amigoinvisible.services;
import android.util.Log;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebaseio.firebaseio.FirebaseFirestore;
import com.google.firebase.messaging.FirebaseMessaging;

/**
 * Clase de servicio encargada de obtener y actualizar el token FCM (Firebase
Cloud Messaging)
 * del usuario autenticado, guardándolo en Firestore para el envío de
notificaciones personalizadas.
 */
public class TokenManager {
    /**
     * Obtiene el token FCM del usuario actual y lo actualiza en su documento
de Firestore.
     * Este token es utilizado para enviar notificaciones push al dispositivo.
     * Si no hay usuario autenticado, el método no realiza ninguna acción.
     */
    public static void actualizarTokenFCM() {
        FirebaseUser usuario = FirebaseAuth.getInstance().getCurrentUser();
        if (usuario == null) return;

        FirebaseMessaging.getInstance().getToken()
            .addOnCompleteListener(task -> {
                if (!task.isSuccessful()) {
                    Log.e("FCM_TOKEN", "✗ No se pudo obtener el token",
task.getException());
                    return;
                }
                String token = task.getResult();
                Log.d("FCM_TOKEN", "Token FCM obtenido: " + token);

                FirebaseFirestore.getInstance()
                    .collection("users")
                    .document(usuario.getUid())
                    .update("fcmToken", token)
                    .addOnSuccessListener(aVoid ->
                        Log.d("FCM_TOKEN", "✓ Token FCM
actualizado en Firestore"))
                    .addOnFailureListener(e ->
                        Log.e("FCM_TOKEN", "✗ Error actualizando
token en Firestore", e));
            });
    }
}
```

5.1.7 XML Layouts

Índice de los Layouts:

1. Activities
 1. activity_add_member.xml
 2. activity_assigned_person.xml
 3. activity_group_chat.xml
 4. activity_group_detail.xml
 5. activity_main.xml
 6. activity_register.xml
 7. activity_welcome.xml
 8. activity_wishlist.xml
2. Fragments
 1. fragment_create_group.xml
 2. fragment_invitations.xml
 3. fragment_settings.xml
 4. fragment_view_groups.xml
3. Components
 1. item_deseo_asignado.xml
 2. item_group_card.xml
 3. item_invitation.xml
 4. item_member.xml
 5. item_mensaje_enviado.xml
 6. item_mensaje_recibido.xml
 7. item_message.xml
 8. item_wishlist.xml
 9. item_header.xml

A continuación se muestra el código correspondiente a cada una de ellas:

1. Activities

1. activity_add_member.xml

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:background="@color/colorFondoPrincipal"  
    android:padding="24dp">  
    <LinearLayout  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:orientation="vertical">  
        <!-- Nombre del participante -->  
        <com.google.android.material.textfield.TextInputLayout  
  
            style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"  
            android:layout_width="match_parent"  
            android:layout_height="wrap_content"  
            android:layout_marginBottom="12dp"  
            android:hint="@string/hint_nombre_participante"  
            app:boxBackgroundColor="@android:color/white"
```

```

        app:boxStrokeColor="@color/colorBotonLogIn"
        app:hintTextColor="@color/colorBotonLogIn">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/editTextMemberName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPersonName" />
</com.google.android.material.textfield.TextInputLayout>
<!-- Correo del participante -->
<com.google.android.material.textfield.TextInputLayout
    style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="12dp"
        android:hint="@string/hint_correo_participante"
        app:boxBackgroundColor="@android:color/white"
        app:boxStrokeColor="@color/colorBotonLogIn"
        app:hintTextColor="@color/colorBotonLogIn">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/editTextMemberEmail"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textEmailAddress" />
</com.google.android.material.textfield.TextInputLayout>
<Button
    android:id="@+id/btnAddMember"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:backgroundTint="@color/colorBotonLogIn"
    android:text="@string/btn_añadir_participante"
    android:textColor="@color/white" />
<TextView
    android:id="@+id/textViewPreviewTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="24dp"
    android:layout_marginBottom="8dp"
    android:text="@string/text_participantes_añadidos"
    android:textColor="@android:color/black"
    android:textStyle="bold" />
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerViewMembers"
    android:layout_width="match_parent"
    android:layout_height="300dp"
    android:layout_marginBottom="24dp"
    android:background="@android:color/white"
    android:clipToPadding="false"
    android:padding="8dp" />
</LinearLayout>
</ScrollView>

```

2. activity_assigned_person.xml

```

<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/colorFondoPrincipal"
    android:padding="24dp">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:orientation="vertical">
        <TextView
            android:id="@+id/textNombreAsignado"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginBottom="16dp"
            android:text="@string/text_asignado_nombre_placeholder"
            android:textColor="@color/lightGray"
            android:textSize="20sp"
            android:textStyle="bold" />
        <ImageView
            android:id="@+id/imageAsignado"
            android:layout_width="120dp"
            android:layout_height="120dp"
            android:layout_marginBottom="24dp"
            android:background="@drawable/circle_background"
            android:scaleType="centerCrop"
            android:src="@drawable/ic_user" />
        <TextView
            android:id="@+id/textListaVacia"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:paddingTop="12dp"
            android:text="@string/text_lista_vacia"
            android:textColor="@color/lightGray"
            android:textSize="16sp"
            android:visibility="gone" />
        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/recyclerListaDeseos"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:background="@drawable/bg_recycler_rounded"
            android:nestedScrollingEnabled="false"
            android:overScrollMode="never"
            android:padding="8dp" />
    </LinearLayout>
</ScrollView>
```

3. activity_group_chat.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".GroupChatActivity">
    <!-- RecyclerView para mostrar los mensajes -->
    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toTopOf="@+id/constraintLayout"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <!-- EditText para ingresar el mensaje -->
    <!-- Botón para enviar el mensaje -->
    <androidx.constraintlayout.widget.ConstraintLayout
        android:id="@+id/constraintLayout"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent">
        <androidx.appcompat.widget.AppCompatEditText
            android:id="@+id/messageInput"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:hint="Escribe tu mensaje"
            android:padding="16dp"
            android:textSize="16sp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toStartOf="@+id/sendButton"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />
        <Button
            android:id="@+id/sendButton"
            android:layout_width="50dp"
            android:layout_height="50dp"
            android:layout_margin="16dp"
            android:layout_marginEnd="15dp"
            android:backgroundTint="@color/colorBotonLogIn"
            app:icon="@drawable/ic_send"
            app:iconGravity="textStart"
            app:iconPadding="1dp"
            app:iconSize="25dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toTopOf="parent" />
    </androidx.constraintlayout.widget.ConstraintLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

4. activity_group_detail.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/colorFondoPrincipal"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".activities.GroupDetailActivity">

    <!-- Contenedor del nombre y la imagen -->
    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp">

        <!-- Nombre del grupo -->
        <TextView
            android:id="@+id/textGroupNameDetail"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:layout_marginTop="8dp"
            android:layout_marginEnd="8dp"
            android:text="@string/text_group_name_placeholder"
            android:textSize="20sp"
            android:textStyle="bold"
            app:layout_constraintEnd_toStartOf="@+id/imageGroupDetail"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <!-- Imagen del grupo -->
        <TextView
            android:id="@+id/textGroupBudgetDetail"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="8dp"
            android:text="@string/text_group_budget_placeholder"
            android:textSize="16sp"
            app:layout_constraintEnd_toStartOf="@+id/imageGroupDetail"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/textGroupNameDetail" />

        <TextView
            android:id="@+id/textGroupCountDetail"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="8dp"
            android:text="@string/text_group_count_placeholder"
            android:textColor="@android:color/black"
```

```
        android:textSize="16sp"
        app:layout_constraintEnd_toStartOf="@+id/imageGroupDetail"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textGroupBudgetDetail" />

    <com.google.android.material.imageview.ShapeableImageView
        android:id="@+id/imageGroupDetail"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:scaleType="centerCrop"
        android:src="@drawable/ic_groups"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:shapeAppearanceOverlay="@style/RoundedImageStyle" />

</androidx.constraintlayout.widget.ConstraintLayout>

<!-- Presupuesto del grupo -->
<!-- Conteo de participantes -->
<!-- Lista de participantes -->
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerGroupMembersDetail"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:background="@android:color/white"
    android:clipToPadding="false"
    android:padding="8dp" />

<!-- Botonera inferior -->
<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <com.google.android.material.button.MaterialButton
        android:id="@+id/btnAddMember"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:backgroundTint="@color/colorBotonLogIn"
        android:text="@string/btn_add_participant"
        android:textColor="@color/white"
        app:icon="@drawable/ic_person_add"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <com.google.android.material.button.MaterialButton
        android:id="@+id/btnChat"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:backgroundTint="@color/colorBotonLogIn"
        android:text="@string/btn_open_chat"
```

```
        android:textColor="@color/white"
        app:icon="@drawable/ic_chat"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/btnAddMember" />

    <com.google.android.material.button.MaterialButton
        android:id="@+id/btnSort"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:backgroundTint="@color/colorBotonFinalizar"
        android:text="@string/btn_perform_draw"
        android:textColor="@color/white"
        app:icon="@drawable/ic_shuffle"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/btnChat" />

    <com.google.android.material.button.MaterialButton
        android:id="@+id/btnWishlist"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:backgroundTint="@color/colorBotonWishlist"
        android:text="@string/btn_view_wishlist"
        android:textColor="@color/white"
        app:icon="@drawable/ic_favorite"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/btnSort" />

    <com.google.android.material.button.MaterialButton
        android:id="@+id/btnViewAssigned"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:backgroundTint="@color/colorBotonLogIn"
        android:text="@string/btn_ver_asignado"
        android:textColor="@android:color/white"
        android:visibility="gone"
        app:icon="@drawable/ic_visibility"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/btnWishlist" />

</androidx.constraintlayout.widget.ConstraintLayout>

</LinearLayout>
```

5. activity_main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/colorFondoPrincipal"
    android:padding="16dp"
    tools:context=".activities.MainActivity">

    <!-- Logo en la parte superior -->
    <ImageView
        android:id="@+id/imgNetMind"
        android:layout_width="200dp"
        android:layout_height="130dp"
        android:layout_marginTop="32dp"
        android:contentDescription="@string/app_name"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/ic_logo" />

    <!-- Formulario en CardView -->
    <com.google.android.material.card.MaterialCardView
        android:id="@+id/materialCardView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        app:cardCornerRadius="16dp"
        app:cardElevation="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imgNetMind">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical"
            android:padding="24dp">

            <!-- Titulo -->
            <TextView
                android:id="@+id/txtTituloLog"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginBottom="24dp"
                android:gravity="center"
                android:text="@string/txtTituloLog"
                android:textColor="@color/black"
                android:textSize="24sp"
                android:textStyle="bold" />

```

```

<!-- Campo Email -->
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/emailContainer"

style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:hint="@string/txtHintemailInput"
    app:errorEnabled="true"
    app:hintTextColor="@color/colorBotonLogIn">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/emailInput"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textEmailAddress"
        android:textSize="16sp" />
</com.google.android.material.textfield.TextInputLayout>

<!-- Campo Contraseña -->
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/passwordContainer"

style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:hint="@string/txtHintpasswordInput"
    app:errorEnabled="true"
    app:hintTextColor="@color/colorBotonLogIn">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/passwordInput"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword"
        android:textSize="16sp" />
</com.google.android.material.textfield.TextInputLayout>

<!-- Botón Login -->
<com.google.android.material.button.MaterialButton
    android:id="@+id/btnLogIn"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:backgroundTint="@color/colorBotonLogIn"
    android:text="@string/txtTituloLog"
    android:textColor="@color/white"
    app:cornerRadius="8dp" />

<!-- Divisor -->
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"

```

```

        android:layout_marginVertical="16dp"
        android:gravity="center"
        android:text="@string/text_o"
        android:textColor="@color/black" />

    <!-- Botones sociales -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:gravity="center"
        android:orientation="horizontal">

        <!-- Botón Google -->
        <com.google.android.material.button.MaterialButton
            android:id="@+id/btnGoogleSignIn"

            style="@style/Widget.MaterialComponents.Button.OutlinedButton"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginEnd="4dp"
            android:layout_weight="1"
            android:text="@string/btnGoogleSignIn"
            app:cornerRadius="8dp"
            app:icon="@drawable/ic_google"
            app:iconGravity="textStart"
            app:iconSize="24dp"
            app:iconTint="@null" />

        <!-- Botón Email -->
        <com.google.android.material.button.MaterialButton
            android:id="@+id/btnEmailSignIn"

            style="@style/Widget.MaterialComponents.Button.OutlinedButton"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginStart="4dp"
            android:layout_weight="1"
            android:text="@string/btnEmailSignIn"
            app:cornerRadius="8dp"
            app:icon="@drawable/ic_mail"
            app:iconGravity="textStart"
            app:iconSize="24dp"
            app:iconTint="@color/colorBotonLogIn" />
    </LinearLayout>
</LinearLayout>
</com.google.android.material.card.MaterialCardView>

<!-- Texto para registro -->
<TextView
    android:id="@+id/txtRegistrate"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="24dp"
    android:textColor="@color/white" />

```

```
        android:textSize="16sp"
        android:textStyle="bold"
        android:text="@string/txtRegistrate"
        android:linksClickable="true"
        android:autoLink="none"
        android:textAlignment="center"
        app:layout_constraintTop_toBottomOf="@+id/materialCardView"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

    </androidx.constraintlayout.widget.ConstraintLayout>
```



6. activity_register.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@color/colorFondoPrincipal"
        android:padding="24dp">

    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/nameContainer"
        style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/hint_nombre"
        app:boxBackgroundColor="@android:color/white"
        app:boxCornerRadiusTopEnd="12dp"
        app:boxCornerRadiusTopStart="12dp"
        app:boxStrokeColor="@color/colorBotonLogIn"
        app:hintTextColor="@color/colorBotonLogIn"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/editTextName"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="textPersonName" />
    </com.google.android.material.textfield.TextInputLayout>

    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/emailContainer"
        style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:hint="@string/hint_correo"
        app:boxBackgroundColor="@android:color/white"
        app:boxStrokeColor="@color/colorBotonLogIn"
        app:hintTextColor="@color/colorBotonLogIn"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/nameContainer">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/editTextEmail"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="textEmailAddress" />
    </com.google.android.material.textfield.TextInputLayout>

    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/passwordContainer"
```

```
style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:hint="@string/hint_contraseña"
    app:boxBackgroundColor="@android:color/white"
    app:boxStrokeColor="@color/colorBotonLogIn"
    app:hintTextColor="@color/colorBotonLogIn"
    app:passwordToggleEnabled="true"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/emailContainer">

<com.google.android.material.textfield.TextInputEditText
        android:id="@+id/editTextPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword" />
</com.google.android.material.textfield.TextInputLayout>

<com.google.android.material.button.MaterialButton
        android:id="@+id/btnRegister"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="24dp"
        android:backgroundTint="@color/colorBotonLogIn"
        android:text="@string/btn_registrarse"
        android:textAllCaps="false"
        android:textColor="@color/white"
        app:cornerRadius="12dp"
        app:icon="@drawable/ic_person_add"
        app:iconTint="@color/white"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/passwordContainer" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

7. activity_welcome.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/drawer_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:fitsSystemWindows="true"
        tools:context=".activities.WelcomeActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <!-- Toolbar -->
        <com.google.android.material.appbar.MaterialToolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="@color/colorFondoPrincipal"
            android:elevation="4dp"
            app:navigationIcon="@drawable/ic_menu">

            <TextView
                android:id="@+id/welcome_text"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:textColor="@color/white"
                android:textSize="18sp"
                android:textStyle="bold" />

        </com.google.android.material.appbar.MaterialToolbar>

        <FrameLayout
            android:id="@+id/fragment_container"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />

    </LinearLayout>

    <!-- Navigation View -->
    <com.google.android.material.navigation.NavigationView
        android:id="@+id/nav_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:fitsSystemWindows="true"
        app:headerLayout="@layout/nav_header"
        app:menu="@menu/nav_menu" />

</androidx.drawerlayout.widget.DrawerLayout>
```

8. activity_wishlist.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/colorFondoPrincipal"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="8dp"
        android:text="@string/wishlist_title"
        android:textSize="18sp"
        android:textStyle="bold" />

    <AutoCompleteTextView
        android:id="@+id/inputWishItem"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@android:color/white"
        android:hint="@string/hint_wishlist"
        android:padding="12dp"
        android:textColor="@android:color/black" />

    <EditText
        android:id="@+id/inputWishUrl"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:hint="@string/url_del_producto_opcional"
        android:inputType="textUri"
        android:minHeight="48dp" />

    <Button
        android:id="@+id	btnAddWish"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:backgroundTint="@color/colorBotonLogIn"
        android:text="@string/btn_add_to_wishlist"
        android:textColor="@android:color/white" />

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/wishlistRecyclerView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_marginTop="16dp"
        android:layout_weight="1"
        android:background="@android:color/white"
        android:padding="8dp" />
</LinearLayout>
```

2. Fragments

```
1. fragment_create_group.xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/colorFondoPrincipal"
    android:padding="24dp">

    <!-- Imagen del grupo -->
    <ImageView
        android:id="@+id/groupImagePreview"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:background="@drawable/circle_background"
        android:scaleType="centerCrop"
        android:src="@drawable/ic_groups"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <!-- Botón para seleccionar imagen -->
    <com.google.android.material.button.MaterialButton
        android:id="@+id/btnSelectImage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:text="@string/btn_select_image"
        app:backgroundTint="@color/colorBotonLogIn"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/groupImagePreview" />

    <!-- Campo: Nombre -->
    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/groupNameContainer"
        style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:hint="@string/hint_nombre_grupo"
        app:boxBackgroundColor="@android:color/white"
        app:boxStrokeColor="@color/colorBotonLogIn"
        app:hintTextColor="@color/colorBotonLogIn"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/btnSelectImage">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/editTextGroupName"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="text" />

```

```

        </com.google.android.material.textfield.TextInputLayout>

        <!-- Campo: Participantes -->
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/participantsContainer"
    style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:hint="@string/hint_max_participantes"
    app:boxBackgroundColor="@android:color/white"
    app:boxStrokeColor="@color/colorBotonLogIn"
    app:hintTextColor="@color/colorBotonLogIn"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/groupNameContainer">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/editTextMaxParticipants"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="number" />
</com.google.android.material.textfield.TextInputLayout>

        <!-- Campo: Presupuesto -->
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/budgetContainer"
    style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:hint="@string/hint_presupuesto"
    app:boxBackgroundColor="@android:color/white"
    app:boxStrokeColor="@color/colorBotonLogIn"
    app:hintTextColor="@color/colorBotonLogIn"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/participantsContainer">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/editTextBudgetLimit"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="numberDecimal" />
</com.google.android.material.textfield.TextInputLayout>

        <!-- Botón: Crear grupo -->
<com.google.android.material.button.MaterialButton
    android:id="@+id/btnFinishGroup"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="24dp"
    android:backgroundTint="@color/colorBotonLogIn"
    android:text="@string/btn_crear_grupo"
    android:textAllCaps="false"

```

```
        android:textColor="@color/white"
        app:cornerRadius="12dp"
        app:icon="@drawable/icon_gift"
        app:iconTint="@color/white"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/budgetContainer" />

    </androidx.constraintlayout.widget.ConstraintLayout>
```



2. fragment_view_groups.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical"  
    android:padding="16dp">  
  
    <androidx.recyclerview.widget.RecyclerView  
        android:id="@+id/recyclerGroups"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent" />  
</LinearLayout>
```

3. fragment_invitations.xml

```
<androidx.constraintlayout.widget.ConstraintLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
    <androidx.recyclerview.widget.RecyclerView  
        android:id="@+id/recycler_invitations"  
        android:layout_width="0dp"  
        android:layout_height="0dp"  
        android:padding="16dp"  
        app:layout_constraintTop_toTopOf="parent"  
        app:layout_constraintBottom_toBottomOf="parent"  
        app:layout_constraintStart_toStartOf="parent"  
        app:layout_constraintEnd_toEndOf="parent" />  
  
    <TextView  
        android:id="@+id/empty_message"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="@string/txt_no_invitaciones"  
        android:textSize="16sp"  
        android:textColor="#888888"  
        android:visibility="gone"  
        app:layout_constraintTop_toTopOf="parent"  
        app:layout_constraintBottom_toBottomOf="parent"  
        app:layout_constraintStart_toStartOf="parent"  
        app:layout_constraintEnd_toEndOf="parent"/>  
</androidx.constraintlayout.widget.ConstraintLayout>
```

4. fragment_settings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/settings_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/txt_configuracion"
        android:textSize="20sp"
        android:textStyle="bold" />
</LinearLayout>
```



3. Components / Views

1. item_deseo_asignado.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="8dp"
    card_view:cardBackgroundColor="@color/cardWishBackground"
    card_view:cardCornerRadius="16dp"
    card_view:cardElevation="6dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_vertical"
        android:orientation="horizontal"
        android:padding="12dp">

        <ImageView
            android:layout_width="24dp"
            android:layout_height="24dp"
            android:layout_marginEnd="12dp"
            android:src="@drawable/ic_gift"
            card_view:tint="@color/cardIconTint" />

        <TextView
            android:id="@+id/textWishItem"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:autoLink="web"
            android:clickable="true"
            android:focusable="true"
            android:textColor="@color/cardText"
            android:textSize="16sp" />
    

```

2. item_group_card.xml

```
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.material.card.MaterialCardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="85dp"
    android:layout_marginHorizontal="16dp"
    android:layout_marginVertical="8dp"
    android:backgroundTint="@color/colorBotonLogIn"
    android:elevation="8dp"
    android:padding="16dp"
    android:radius="8dp"
    app:strokeColor="@color/strokeMaterialCardView"
    app:strokeWidth="2dp">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:id="@+id/txtGroupName"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginStart="16dp"
            android:layout_marginTop="16dp"
            android:text="Nombre del grupo"
            android:textColor="@android:color/black"
            android:textSize="18sp"
            android:textStyle="bold|italic"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
            android:id="@+id/txtGroupBudget"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginStart="16dp"
            android:layout_marginTop="16dp"
            android:layout_marginBottom="16dp"
            android:text="Presupuesto: €..."
            android:textColor="@android:color/darker_gray"
            android:textSize="14sp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/txtGroupName" />

        <com.google.android.material.imageview.ShapeableImageView
            android:id="@+id/imageGroupCard"
            android:layout_width="80dp"
            android:layout_height="80dp"
            android:layout_marginTop="8dp"
            android:layout_marginEnd="8dp"
            android:layout_marginBottom="8dp"
```

```
        android:scaleType="centerCrop"
        android:src="@drawable/ic_groups"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:shapeAppearanceOverlay="@style/RoundedImageStyle" />

    </androidx.constraintlayout.widget.ConstraintLayout>
</com.google.android.material.card.MaterialCardView>
```



3. item_invitation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.material.card.MaterialCardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="12dp"
    android:backgroundTint="@color/invitationCardBackground"
    app:cardCornerRadius="16dp"
    app:cardElevation="6dp"
    app:strokeColor="@color/invitationCardStroke"
    app:strokeWidth="1dp">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="16dp">

        <!-- Icono decorativo a la izquierda -->
        <ImageView
            android:id="@+id/iconInvitation"
            android:layout_width="48dp"
            android:layout_height="48dp"
            android:layout_marginEnd="12dp"
            android:src="@drawable/ic_invitation"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:tint="@color/invitationIconTint" />

        <!-- Texto del grupo -->
        <TextView
            android:id="@+id/txt_nombre_grupo"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text="Te han invitado al grupo: 'Nombre del grupo'"
            android:textColor="@color/invitationText"
            android:textSize="16sp"
            android:textStyle="bold"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toEndOf="@+id/iconInvitation"
            app:layout_constraintTop_toTopOf="parent" />

        <!-- Botón Aceptar -->
        <Button
            android:id="@+id(btn_aceptar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="12dp"
            android:backgroundTint="@color/invitationAcceptBackground"
            android:text=""
            app:icon="@drawable/ic_check"
            app:iconSize="24dp"
            app:iconTint="@android:color/white"
```

```
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toStartOf="@+id	btn_rechazar"
    app:layout_constraintStart_toEndOf="@id/iconInvitation"
    app:layout_constraintTop_toBottomOf="@id	txt_nombre_grupo" />

<!-- Botón Rechazar -->
<Button
    android:id="@+id	btn_rechazar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="12dp"
    android:backgroundTint="@color/invitationRejectBackground"
    android:text=""
    app:icon="@drawable/ic_close"
    app:iconSize="24dp"
    app:iconTint="@android:color/white"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toBottomOf="@id	txt_nombre_grupo" />

</androidx.constraintlayout.widget.ConstraintLayout>
</com.google.android.material.card.MaterialCardView>
```

4. item_member.xml

```
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.material.card.MaterialCardView
    xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:background="@android:color/white"
        android:padding="12dp"
        app:cardCornerRadius="8dp"
        app:cardElevation="8dp"
        app:cardUseCompatPadding="true">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="15dp">

        <TextView
            android:id="@+id/textName"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/text_nombre_participante"
            android:textColor="@android:color/black"
            android:textStyle="bold"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
            android:id="@+id/textEmail"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:text="@string/text_correo_placeholder"
            android:textColor="@android:color/darker_gray"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/textName" />

        <ImageView
            android:id="@+id/iconDelete"
            android:layout_width="34dp"
            android:layout_height="34dp"
            android:contentDescription="@string/content_eliminar_participante"
            android:padding="8dp"
            android:src="@drawable/ic_delete"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent" />

    </androidx.constraintlayout.widget.ConstraintLayout>
</com.google.android.material.card.MaterialCardView>
```

5. item_mensaje_enviado.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:gravity="end"  
    android:padding="8dp">  
  
    <TextView  
        android:id="@+id/messageText"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:background="@drawable/bg_mensaje_enviado"  
        android:padding="12dp"  
        android:textColor="@android:color/white"  
        android:textSize="16sp" />  
</LinearLayout>
```

6. item_mensaje_recibido.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:gravity="start"  
    android:padding="8dp">  
  
    <TextView  
        android:id="@+id/messageText"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:background="@drawable/bg_mensaje_recibido"  
        android:padding="12dp"  
        android:textColor="@android:color/black"  
        android:textSize="16sp" />  
</LinearLayout>
```

7. item_message.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="8dp"
        android:padding="16dp"
        app:cardCornerRadius="8dp"
        app:cardElevation="4dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="8dp">

        <!-- Nombre del remitente (visible solo si no es anónimo) -->
        <TextView
            android:id="@+id/senderName"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Sender Name"
            android:textColor="@android:color/darker_gray"
            android:textSize="14sp"
            android:visibility="gone" />

        <!-- Contenido del mensaje -->
        <TextView
            android:id="@+id/messageContent"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="4dp"
            android:text="Mensaje del usuario"
            android:textColor="@android:color/black"
            android:textSize="16sp" />
    </LinearLayout>
</androidx.cardview.widget.CardView>
```

8. item_wishlist.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="?android:attr/selectableItemBackground"
    android:orientation="horizontal"
    android:padding="12dp">

    <TextView
        android:id="@+id/textWishItem"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:textColor="@android:color/black"
        android:textSize="16sp" />

    <ImageView
        android:id="@+id	btnDeleteWish"
        android:layout_width="24dp"
        android:layout_height="24dp"
        android:contentDescription="@string/eliminar_deseo"
        android:padding="4dp"
        android:src="@drawable/ic_delete" />
</LinearLayout>
```

9. nav_header.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="160dp"
    android:background="@color/colorFondoPrincipal"
    android:gravity="bottom"
    android:orientation="vertical"
    android:padding="16dp">

    <!-- Imagen de perfil redonda -->
    <ImageView
        android:id="@+id/profile_image"
        android:layout_width="64dp"
        android:layout_height="64dp"
        android:layout_gravity="center_vertical"
        android:background="@drawable/circle_background"
        android:contentDescription="Imagen de perfil"
        android:padding="4dp"
        android:scaleType="centerCrop"
        android:src="@drawable/ic_user" />

    <!-- Nombre del usuario -->
    <TextView
        android:id="@+id/nav_header_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:textColor="@color/white"
        android:textSize="14sp" />

    <!-- Correo del usuario -->
    <TextView
        android:id="@+id/nav_header_email"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="@color/white"
        android:textSize="12sp" />

</LinearLayout>
```

5.1.8 Menú

```
nav_menu.xml
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_my_groups"
            android:icon="@drawable/ic_groups"
            android:title="@string/nav_my_groups" />
        <item
            android:id="@+id/nav_create_group"
            android:icon="@drawable/ic_add_group"
            android:title="@string/nav_create_group" />
        <item
            android:id="@+id/nav_invitations"
            android:icon="@drawable/ic_invitation"
            android:title="@string/nav_invitations" />
        <item
            android:id="@+id/nav_settings"
            android:icon="@drawable/ic_settings"
            android:title="@string/nav_settings" />
    </group>
    <item android:title="@string/nav_options">
        <menu>
            <item
                android:id="@+id/nav_logout"
                android:icon="@drawable/ic_logout"
                android:title="@string/nav_logout" />
        </menu>
    </item>
</menu>
```

5.1.9 Values

Índice de los values:

1. colors.xml
2. strings.xml
3. strings.xml(en)
4. strings.xml(fr)
5. styles.xml

A continuación se muestra el código correspondiente a cada una de ellas:

1. colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFF</color>
    <color name="colorFondoPrincipal">#D44646</color>
    <color name="colorBotonLogIn">#831515</color>
    <color name="colorBotonFinalizar">#4CAF50</color>
    <color name="strokeMaterialCardView">#FBC02D</color>
    <color name="lightGray">#E0E0E0</color>
    <color name="colorBotonWishlist">#0EA5B4</color>
    <color name="cardWishBackground">#FFF3E0</color>
    <color name="cardText">#5D4037</color>
    <color name="cardIconTint">#FF7043</color>
    <color name="colorLink">#303F9F</color>

    <color name="invitationCardBackground">#FFFDF7</color>
    <color name="invitationCardStroke">#E0E0E0</color>
    <color name="invitationText">#444444</color>
    <color name="invitationIconTint">#FF9800</color>
    <color name="invitationAcceptBackground">#4CAF50</color>
    <color name="invitationRejectBackground">#F44336</color>

</resources>
```

2. strings.xml(es)

```
<resources>
    <string name="app_name">Amigo Invisible</string>
    <string name="txtTituloLog">LOG IN</string>
    <string name="txtHintemailInput">Correo</string>
    <string name="txtHintpasswordInput">Contraseña</string>
    <string name="btnEmailSignIn">Email</string>
    <string name="btnGoogleSignIn">Google</string>

    <string name="text_o">0</string>
    <string name="error_google_sign_in">Error al iniciar sesión con
Google</string>
    <string name="error_autenticacion_google">Error en autenticación con
Google</string>
    <string name="error_firebase_sign_in">Error en autenticación
Firebase</string>
    <string name="mensaje_completa_campos">Completa todos los campos</string>
    <string name="error_login_correo">Error al iniciar sesión: %1$s</string>
    <string name="mensaje_usuario_creado">Usuario creado</string>
    <string name="error_obtener_token">Error al obtener token FCM</string>
    <string name="mensaje_error_autenticacion_firebase">Error en autenticación
con Firebase</string>

    <string-array name="theme_names">
        <item>Tema Claro</item>
        <item>Tema Oscuro</item>
    </string-array>

    <string-array name="theme_values">
        <item>light</item>
        <item>dark</item>
        <item>blue</item>
    </string-array>

    <string name="txt_titulo_cerrar_sesion">Cerrar sesión</string>
    <string name="txt_mensaje_cerrar_sesion">¿Estas seguro que quiere cerrar
sesión?</string>
    <string name="txt_mensajePositivo">Si</string>
    <string name="txt_mensajeNegativo">Cancelar</string>

    <string name="txtRegistrate">¿No tienes cuenta? </string>
    <string name="registrate">Regístrate</string>
    <string name="txtGrupo">Grupo: ...</string>
    <string name="txtAddParticipant">Añadir participante</string>
    <string name="txtHoldADraw">Realizar sorteo</string>
    <string name="txtOpenChat">Abrir chat anónimo</string>

    <string name="bienvenida_usuario">Bienvenido, %1$s</string>
    <string name="selecciona_imagen_perfil">Selecciona una imagen de
perfil</string>
    <string name="foto_actualizada">Foto actualizada</string>
    <string name="error_subir_imagen">Error al subir la imagen: %1$s</string>
    <string name="error_actualizar_perfil">Error al actualizar el
perfil</string>
```

```

<string name="error_descargar_uri">Error al obtener la URL de la
imagen</string>
<string name="titulo_bienvenida">Pantalla de Bienvenida</string>
<string name="nav_home">Inicio</string>
<string name="nav_create_group">Crear Grupo</string>
<string name="nav_my_groups">Mis Grupos</string>
<string name="nav_settings">Configuración</string>
<string name="nav_logout">Cerrar sesión</string>
<string name="nav_invitations">Mis invitaciones</string>
<string name="nav_options">Opciones</string>
<string name="mensaje_logout">Sesión cerrada</string>
<string name="titulo_toolbar">Amigo Invisible</string>
<string name="cargando">Cargando...</string>
<string name="perfil_actualizado">Perfil actualizado correctamente</string>

<string name="notificaciones_activadas">Notificaciones activadas</string>
<string name="notificaciones_desactivadas">Notificaciones
desactivadas</string>

<string name="usuario_generico">Usuario</string>
<string name="bienvenido_usuario">Bienvenido, %1$s</string>
<string name="mensaje_seleccionar_imagen">Selecciona una imagen de
perfil</string>
<string name="error_subida_imagen">Error al subir la imagen: %1$s</string>
<string name="nombre_usuario_actualizado">Nuevo nombre: %1$s</string>
<string name="mensaje_contrasena_debil">La contraseña debe tener al menos 6
caracteres</string>

<string name="hint_nombre_grupo">Nombre del grupo</string>
<string name="hint_max_participantes">Máximo de participantes</string>
<string name="hint_presupuesto">Presupuesto (€)</string>
<string name="btn_crear_grupo">Crear grupo</string>
<string name="error_campos_obligatorios">Todos los campos son
obligatorios</string>
<string name="error_valores_numericos">Valores numéricos no
válidos</string>
<string name="mensaje_grupo_creado">Grupo creado correctamente</string>
<string name="error_crear_grupo">Error al crear grupo</string>
<string name="texto_presupuesto_grupo">Presupuesto: €%1$.2f</string>
<string name="error_verificar_límite">Error al verificar límite de
grupos.</string>

<string name="titulo_registro">Registrarse</string>
<string name="hint_nombre">Nombre</string>
<string name="hint_correo">Correo electrónico</string>
<string name="hint_contrasena">Contraseña</string>
<string name="btn_registrarse">Registrarse</string>
<string name="mensaje_requisitos_contrasena">
La contraseña debe tener al menos:


- 6 caracteres
- 1 letra mayúscula
- 1 letra minúscula
- 1 carácter especial


</string>
```

```

<string name="mensaje_verificacion_enviada">Registro exitoso. Revisa tu correo para verificar la cuenta.</string>
<string name="mensaje_error_enviar_verificacion">No se pudo enviar el correo de verificación.</string>
<string name="mensaje_verificar_correo">Debes verificar tu correo antes de iniciar sesión.</string>

<string name="mensaje_campos_obligatorios">Por favor completa todos los campos</string>
<string name="mensaje_error_registro">Error en el registro: %1$s</string>
<string name="mensaje_error_firestore">Error al guardar en Firestore</string>
<string name="mensaje_registro_exitoso">Usuario registrado correctamente</string>
<string name="mensaje_error_actualizar_nombre">Registro exitoso, pero error al actualizar nombre</string>

<string name="text_group_name_placeholder">Grupo: %1$s</string>
<string name="text_group_budget_placeholder">Presupuesto: €%1$.2f</string>
<string name="text_group_count_placeholder">Participantes: %1$d / %2$d</string>
<string name="btn_add_participant">Añadir participante</string>
<string name="btn_open_chat">Abrir chat anónimo</string>
<string name="btn_perform_draw">Realizar sorteo</string>
<string name="mensaje_grupo_completo">Este grupo ya está completo</string>
<string name="mensaje_sorteo_exito">🎉 Sorteo realizado con éxito</string>
<string name="mensaje_sorteo_error">❌ Error: %1$s</string>
<string name="mensaje_error_sorteo">Error: %1$s</string>
<string name="mensaje_groupid_no_disponible">El groupId no está disponible</string>
<string name="error_group_id_missing">Error: ID del grupo no recibido.</string>
<string name="error_envio_mensaje">Error al enviar el mensaje.</string>

<string name="dialogo_titulo_eliminar_participante">¿Eliminar participante?</string>
<string name="dialogo_mensaje_eliminar_participante">¿Estás seguro de que deseas eliminar a %1$s?</string>
<string name="dialogo_boton_si">Sí</string>
<string name="dialogo_boton_no">No</string>
<string name="mensaje_participante_eliminado">Participante eliminado</string>

<string name="text_nombre_participante">Nombre</string>
<string name="text_correo_placeholder">correo@email.com</string>
<string name="content_eliminar_participante">Eliminar participante</string>

<string name="mensaje_correo_invalido">Por favor, ingrese un correo electrónico válido</string>
<string name="mensaje_participante_añadido">Participante añadido</string>
<string name="mensaje_error_guardar">Error: %1$s</string>

<string name="hint_nombre_participante">Nombre del participante</string>
<string name="hint_correo_participante">Correo electrónico</string>

```

```
<string name="btn_añadir_participante">Añadir participante</string>
<string name="text_participantes_añadidos">Participantes añadidos:</string>

<string name="mensaje_imagen_actualizada">Imagen actualizada
correctamente</string>
<string name="error_cargar_datos_grupo">Error al cargar datos del
grupo.</string>
<string name="btn_change_group_image">Cambiar</string>
<string name="mensaje_solo_propietario_puede_cambiar_imagen">Solo el
propietario del grupo puede cambiar la imagen.</string>
<string name="hint_wishlist">Añadir nuevo deseo</string>
<string name="btn_add_to_wishlist">Añadir a mi lista</string>
<string name="mensaje_deseo_añadido">Deseo añadido a la lista</string>
<string name="mensaje_deseo_vacio">Escribe algo para añadirlo</string>
<string name="mensaje_error_deseo">No se pudo añadir el deseo</string>
<string name="mensaje_deseo_eliminado">Deseo eliminado</string>
<string name="wishlist_title">Mi lista de deseos</string>

<string name="btn_view_wishlist">Ver lista de deseos</string>
<string name="error_id_grupo_invalido">Error: ID de grupo no
válido.</string>
<string name="eliminar_deseo">Eliminar deseo</string>

<string-array name="wishlist_suggestions">
    <item>Abanico sevillano</item>
    <item>Agenda 2025</item>
    <item>Altavoz Bluetooth</item>
    <item>Auriculares</item>
    <item>Azulejo típico sevillano</item>
    <item>Balón firmado por el Sevilla</item>
    <item>Botella de vino del Aljarafe</item>
    <item>Bufanda</item>
    <item>Bufanda del Sevilla FC</item>
    <item>Camiseta Futbol retro</item>
    <item>Camiseta Real Betis Balompié</item>
    <item>Camiseta Sevilla FC oficial</item>
    <item>Cargador inalámbrico</item>
    <item>Calcetines divertidos</item>
    <item>Cartel de la Feria de Abril</item>
    <item>Cartera</item>
    <item>Echo Dot</item>
    <item>Entrada al Alcázar</item>
    <item>Figura de la Giralda</item>
    <item>Gorra del Betis</item>
    <item>Juego de mesa</item>
    <item>Lámpara LED</item>
    <item>Libro</item>
    <item>Mochila</item>
    <item>Pack de especias andaluzas</item>
    <item>Perfume</item>
    <item>Planta decorativa</item>
    <item>Power Bank</item>
    <item>Puzzle</item>
    <item>Ratón inalámbrico</item>
```

```

<item>Reloj</item>
<item>Set de llaveros del Betis</item>
<item>Smartwatch</item>
<item>Soporte para móvil</item>
<item>Sudadera</item>
<item>Taza personalizada</item>
<item>Teclado mecánico</item>
<item>Xiaomi Mi Band</item>
<item>Zapatillas deportivas</item>
</string-array>

<string name="dialog_titulo_confirmar_sorteo">¿Confirmar sorteo?</string>
<string name="dialog_mensaje_confirmar_sorteo">¿Estás seguro de que deseas
realizar el sorteo del Amigo Invisible? Esta acción no se puede
deshacer.</string>
<string name="dialog_boton_realizar">Realizar sorteo</string>
<string name="mensaje_sorteo_iniciado">Realizando sorteo...</string>
<string name="dialog_boton_cancelar">Cancelar</string>

<string name="btn_ver_asignado">Ver asignado</string>
<string name="error_datos_asignado">No se pudo cargar la información de la
persona asignada.</string>
<string name="text_asignado_nombre_placeholder">Te ha tocado: %1$s</string>
<string name="text_lista_vacia">No hay deseos en la lista.</string>

<string name="text_asignado_titulo">Te ha tocado regalar a:</string>
<string name="error_max_grupos">Has alcanzado el máximo de %1$d grupos
creados.</string>

<string name="buscando_invitaciones_pendientes">Buscando invitaciones
pendientes...</string>
<string name="txt_no_invitaciones">No tienes invitaciones
pendientes</string>
<string name="confirmar_sorteo">Confirmar sorteo</string>
<string name="msg_aceptada_ok">Te has unido al grupo correctamente</string>
<string name="msg_invitacion_rechazada">Has rechazado la
invitación</string>

<string name="dialog_realizando_sorteo">➡️ Realizando sorteo, por favor
espera...</string>
<string name="no_asignado_todavia">Todavía no hay persona asignada</string>
<string name="url_del_producto_opcional">URL del producto
(opcional)</string>
<string name="txt_configuracion">Configuración</string>

<string name="pref_title_username">Nombre de usuario</string>
<string name="pref_summary_username">Introduce tu nombre</string>
<string name="pref_title_theme">Tema de color</string>
<string name="pref_summary_theme">Elige tu tema favorito</string>
<string name="pref_title_notifications">Notificaciones</string>
<string name="pref_summary_notifications">Activar o desactivar
notificaciones</string>
<string name="pref_title_updates">Recibir actualizaciones</string>
<string name="pref_summary_updates">Permitir actualizaciones
automáticas</string>

```

```
<string name="pref_title_volume">Nivel de volumen</string>
<string name="pref_summary_volume">Ajusta el volumen</string>

<string name="btn_select_image">Seleccionar Imagen</string>

</resources>
```



3. strings.xml(en)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Amigo Invisible</string>
    <string name="txtTituloLog">LOG IN</string>
    <string name="txtHintemailInput">Email</string>
    <string name="txtHintpasswordInput">Password</string>
    <string name="btnEmailSignIn">Email</string>
    <string name="btnGoogleSignIn">Google</string>
    <string name="text_o">0</string>
    <string name="error_google_sign_in">Error signing in with Google</string>
    <string name="error_autenticacion_google">Authentication error with
Google</string>
    <string name="error_firebase_sign_in">Firebase authentication
error</string>
    <string name="mensaje_completa_campos">Please fill in all fields</string>
    <string name="error_login_correo">Error signing in: %1$s</string>
    <string name="mensaje_usuario_creado">User created</string>
    <string name="error_obtener_token">Error retrieving FCM token</string>
    <string name="mensaje_error_autenticacion_firebase">Authentication error
with Firebase</string>

    <string-array name="theme_names">
        <item>Light Theme</item>
        <item>Dark Theme</item>
    </string-array>

    <string-array name="theme_values">
        <item>light</item>
        <item>dark</item>
        <item>blue</item>
    </string-array>

    <string name="txt_titulo_cerrar_sesion">Log out</string>
    <string name="txt_mensaje_cerrar_sesion">Are you sure you want to log
out?</string>
    <string name="txt_mensajePositivo">Yes</string>
    <string name="txt_mensajeNegativo">Cancel</string>

    <string name="txtRegistrate">Don't have an account?</string>
    <string name="registrate">Sign up</string>

    <string name="txtGrupo">Group: ...</string>
    <string name="txtAddParticipant">Add participant</string>
    <string name="txtHoldADraw">Hold a draw</string>
    <string name="txtOpenChat">Open anonymous chat</string>

    <string name="bienvenida_usuario">Welcome, %1$s</string>
    <string name="selecciona_imagen_perfil">Select a profile picture</string>
    <string name="foto_actualizada">Photo updated</string>
    <string name="error_subir_imagen">Error uploading image: %1$s</string>
    <string name="error_actualizar_perfil">Error updating profile</string>
    <string name="error_descargar_uri">Error obtaining image URL</string>
    <string name="titulo_bienvenida">Welcome Screen</string>
    <string name="nav_home">Home</string>
```

```

<string name="nav_create_group">Create Group</string>
<string name="nav_my_groups">My Groups</string>
<string name="nav_settings">Settings</string>
<string name="nav_logout">Log out</string>
<string name="nav_invitations">My invitations</string>
<string name="nav_options">Options</string>
<string name="mensaje_logout">Session closed</string>
<string name="titulo_toolbar">Secret Santa</string>
<string name="cargando">Loading...</string>
<string name="perfil_actualizado">Profile updated successfully</string>

<string name="notificaciones_activadas">Notifications enabled</string>
<string name="notificaciones_desactivadas">Notifications disabled</string>

<string name="usuario_generico">User</string>
<string name="bienvenido_usuario">Welcome, %1$s</string>
<string name="mensaje_seleccionar_imagen">Select a profile picture</string>
<string name="error_subida_imagen">Error uploading image: %1$s</string>
<string name="nombre_usuario_actualizado">New name: %1$s</string>
<string name="mensaje_contrasena_debil">Password must be at least 6
characters</string>

<string name="hint_nombre_grupo">Group name</string>
<string name="hint_max_participantes">Maximum participants</string>
<string name="hint_presupuesto">Budget (€)</string>
<string name="btn_crear_grupo">Create group</string>
<string name="error_campos_obligatorios">All fields are required</string>
<string name="error_valores_numericos">Invalid numerical values</string>
<string name="mensaje_grupo_creado">Group created successfully</string>
<string name="error_crear_grupo">Error creating group</string>
<string name="texto_presupuesto_grupo">Budget: €%1$.2f</string>
<string name="error_verificar_limite">Error checking group limit</string>

<string name="titulo_registro">Register</string>
<string name="hint_nombre">Name</string>
<string name="hint_correo">Email</string>
<string name="hint_contrasena">Password</string>
<string name="btn_registrarse">Sign up</string>
<string name="mensaje_requisitos_contrasena">
Password must include at least:


- 6 characters
- 1 uppercase letter
- 1 lowercase letter
- 1 special character


</string>

<string name="mensaje_verificacion_enviada">Successful registration. Check
your email to verify your account.</string>
<string name="mensaje_error_enviar_verificacion">Failed to send
verification email.</string>
<string name="mensaje_verificar_correo">You must verify your email before
signing in.</string>

<string name="mensaje_campos_obligatorios">Please complete all
fields</string>

```

```

<string name="mensaje_error_registro">Registration error: %1$s</string>
<string name="mensaje_error_firestore">Error saving to Firestore</string>
<string name="mensaje_registro_exitoso">User registered
successfully</string>
<string name="mensaje_error_actualizar_nombre">Registration successful, but
failed to update name</string>

<string name="text_group_name_placeholder">Group: %1$s</string>
<string name="text_group_budget_placeholder">Budget: €%1$.2f</string>
<string name="text_group_count_placeholder">Participants: %1$d /
%2$d</string>
<string name="btn_add_participant">Add participant</string>
<string name="btn_open_chat">Open anonymous chat</string>
<string name="btn_perform_draw">Hold a draw</string>
<string name="mensaje_grupo_completo">This group is already full</string>
<string name="mensaje_sorteo_exito">🎉 Draw completed successfully</string>
<string name="mensaje_sorteo_error">❌ Error: %1$s</string>
<string name="mensaje_error_sorteo">Error: %1$s</string>
<string name="mensaje_groupid_no_disponible">Group ID is not
available</string>
<string name="error_group_id_missing">Error: Group ID not
received.</string>
<string name="error_envio_mensaje">Error sending message.</string>

<string name="dialogo_titulo_eliminar_participante">Remove
participant?</string>
<string name="dialogo_mensaje_eliminar_participante">Are you sure you want
to remove %1$s?</string>
<string name="dialogo_boton_si">Yes</string>
<string name="dialogo_boton_no">No</string>
<string name="mensaje_participante_eliminado">Participant removed</string>

<string name="text_nombre_participante">Name</string>
<string name="text_correo_placeholder">email@example.com</string>
<string name="content_eliminar_participante">Remove participant</string>

<string name="mensaje_correo_invalido">Please enter a valid email
address</string>
<string name="mensaje_participante_añadido">Participant added</string>
<string name="mensaje_error_guardar">Error: %1$s</string>

<string name="hint_nombre_participante">Participants name</string>
<string name="hint_correo_participante">Email</string>
<string name="btn_añadir_participante">Add participant</string>
<string name="text_participantes_añadidos">Participants added:</string>

<string name="mensaje_imagen_actualizada">Image updated
successfully</string>
<string name="error_cargar_datos_grupo">Error loading group data.</string>
<string name="btn_change_group_image">Change</string>
<string name="mensaje_solo_propietario_puede_cambiar_imagen">Only the group
owner can change the image.</string>
<string name="hint_wishlist">Add new wish</string>
<string name="btn_add_to_wishlist">Add to my list</string>
<string name="mensaje_deseo_añadido">Wish added to list</string>

```

```

<string name="mensaje_deseo_vacio">Write something to add it</string>
<string name="mensaje_error_deseo">Could not add the wish</string>
<string name="mensaje_deseo_eliminado">Wish removed</string>
<string name="wishlist_title">My wishlist</string>

<string name="btn_view_wishlist">View wishlist</string>
<string name="error_id_grupo_invalido">Error: Invalid group ID.</string>
<string name="eliminar_deseo">Remove wish</string>

<string-array name="wishlist_suggestions">
    <item>Sevillian fan</item>
    <item>2025 planner</item>
    <item>Bluetooth speaker</item>
    <item>Headphones</item>
    <item>Typical Sevillian tile</item>
    <item>Ball signed by Sevilla FC</item>
    <item>Wine bottle from Aljarafe</item>
    <item>Scarf</item>
    <item>Sevilla FC scarf</item>
    <item>Retro football shirt</item>
    <item>Real Betis Balompié shirt</item>
    <item>Official Sevilla FC shirt</item>
    <item>Wireless charger</item>
    <item>Funny socks</item>
    <item>April Fair poster</item>
    <item>Wallet</item>
    <item>Echo Dot</item>
    <item>Alcázar entrance ticket</item>
    <item>Giralda figurine</item>
    <item>Betis cap</item>
    <item>Board game</item>
    <item>LED lamp</item>
    <item>Book</item>
    <item>Backpack</item>
    <item>Andalusian spices pack</item>
    <item>Perfume</item>
    <item>Decorative plant</item>
    <item>Power bank</item>
    <item>Puzzle</item>
    <item>Wireless mouse</item>
    <item>Watch</item>
    <item>Betis keychain set</item>
    <item>Smartwatch</item>
    <item>Phone holder</item>
    <item>Hoodie</item>
    <item>Custom mug</item>
    <item>Mechanical keyboard</item>
    <item>Xiaomi Mi Band</item>
    <item>Sports shoes</item>
</string-array>

<string name="dialog_titulo_confirmar_sorteo">Confirm draw?</string>
<string name="dialog_mensaje_confirmar_sorteo">Are you sure you want to
perform the Secret Santa draw? This action cannot be undone.</string>
<string name="dialog_boton_realizar">Perform draw</string>

```

```

<string name="mensaje_sorteo_iniciado">Performing draw...</string>
<string name="dialog boton cancelar">Cancel</string>

<string name="btn_ver_asignado">View assigned</string>
<string name="error_datos_asignado">Could not load assigned person
information.</string>
<string name="text_asignado_nombre_placeholder">You got: %1$s</string>
<string name="text_lista_vacia">There are no wishes in the list.</string>

<string name="text_asignado_titulo">You have to give a gift to:</string>
<string name="error_max_grupos">You have reached the maximum of %1$D
created groups.</string>

<string name="buscando_invitaciones_pendientes">Looking for pending
invitations...</string>
<string name="txt_no_invitaciones">You have no pending invitations</string>
<string name="confirmar_sorteo">Confirm draw</string>
<string name="msg_aceptada_ok">You have successfully joined the
group</string>
<string name="msg_invitacion_rechazada">You have declined the
invitation</string>

<string name="dialog_realizando_sorteo">➡ Performing draw, please
wait...</string>
<string name="no_asignado_todavia">No one has been assigned yet</string>
<string name="url_del_producto_opcional">Product URL (optional)</string>
<string name="txt_configuracion">Settings</string>

<string name="pref_title_username">Username</string>
<string name="pref_summary_username">Enter your name</string>
<string name="pref_title_theme">Color Theme</string>
<string name="pref_summary_theme">Choose your favorite theme</string>
<string name="pref_title_notifications">Notifications</string>
<string name="pref_summary_notifications">Enable or disable
notifications</string>
<string name="pref_title_updates">Receive updates</string>
<string name="pref_summary_updates">Allow automatic updates</string>
<string name="pref_title_volume">Volume level</string>
<string name="pref_summary_volume">Adjust the volume</string>

<string name="btn_select_image">Select Image</string>

</resources>

```

4. strings.xml(fr)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Amigo Invisible</string>

    <string name="text_o">0</string>
    <string name="error_google_sign_in">Erreur lors de la connexion avec Google</string>
    <string name="error_autenticacion_google">Erreur d'\authentification avec Google</string>
    <string name="error_firebase_sign_in">Erreur d'\authentification avec Firebase</string>
    <string name="mensaje_completa_campos">Veuillez remplir tous les champs</string>
    <string name="error_login_correo">Erreur de connexion : %1$s</string>
    <string name="mensaje_usuario_creado">Utilisateur créé</string>
    <string name="error_obtener_token">Erreur lors de l\'obtention du jeton FCM</string>
    <string name="mensaje_error_autenticacion_firebase">Erreur d'\authentification avec Firebase</string>

    <string-array name="theme_names">
        <item>Thème clair</item>
        <item>Thème sombre</item>
    </string-array>

    <string-array name="theme_values">
        <item>light</item>
        <item>dark</item>
        <item>blue</item>
    </string-array>

    <string name="nav_invitations">Mes invitations</string>
    <string name="nav_options">Options</string>

    <string name="txt_titulo_cerrar_sesion">Se déconnecter</string>
    <string name="txt_mensaje_cerrar_sesion">Êtes-vous sûr de vouloir vous déconnecter ?</string>
    <string name="txt_mensajePositivo">Oui</string>
    <string name="txt_mensajeNegativo">Annuler</string>
    <string name="txtRegistrate">Vous n\'avez pas de compte ?</string>
    <string name="registrate">Inscrис-toi</string>
    <string name="txtGrupo">Groupe : ...</string>
    <string name="txtAddParticipant">Ajouter un participant</string>
    <string name="txtHoldADraw">Faire un tirage</string>
    <string name="txtOpenChat">Ouvrir le chat anonyme</string>

    <string name="bienvenida_usuario">Bienvenue, %1$s</string>
    <string name="selecciona_imagen_perfil">Sélectionnez une photo de profil</string>
    <string name="foto_actualizada">Photo mise à jour</string>
    <string name="error_subir_imagen">Erreur lors du téléchargement de l\'image : %1$s</string>
    <string name="error_actualizar_perfil">Erreur lors de la mise à jour du profil</string>
```

```

<string name="error_descargar_uri">Erreur lors de l'\obtention de l\'URL de
l\'image</string>
<string name="titulo_bienvenida">Écran de bienvenue</string>
<string name="nav_home">Accueil</string>
<string name="nav_create_group">Créer un groupe</string>
<string name="nav_my_groups">Mes groupes</string>
<string name="nav_settings">Paramètres</string>
<string name="nav_logout">Se déconnecter</string>
<string name="mensaje_logout">Session terminée</string>
<string name="titulo_toolbar">Secret Santa</string>
<string name="cargando">Chargement...</string>
<string name="perfil_actualizado">Profil mis à jour avec succès</string>

<string name="notificaciones_activadas">Notifications activées</string>
<string name="notificaciones_desactivadas">Notifications
désactivées</string>

<string name="usuario_generico">Utilisateur</string>
<string name="bienvenido_usuario">Bienvenue, %1$s</string>
<string name="mensaje_seleccionar_imagen">Sélectionnez une photo de
profil</string>
<string name="error_subida_imagen">Erreur lors du téléchargement de
l\'image : %1$s</string>
<string name="nombre_usuario_actualizado">Nouveau nom : %1$s</string>
<string name="mensaje_contrasena_debil">Le mot de passe doit contenir au
moins 6 caractères</string>

<string name="hint_nombre_grupo">Nom du groupe</string>
<string name="hint_max_participantes">Nombre maximal de
participants</string>
<string name="hint_presupuesto">Budget (€)</string>
<string name="btn_crear_grupo">Créer un groupe</string>
<string name="error_campos_obligatorios">Tous les champs sont
obligatoires</string>
<string name="error_valores_numericos">Valeurs numériques non
valides</string>
<string name="mensaje_grupo_creado">Groupe créé avec succès</string>
<string name="error_crear_grupo">Erreur lors de la création du
groupe</string>
<string name="texto_presupuesto_grupo">Budget : €%1$.2f</string>
<string name="error_verificar_limite">Erreur lors de la vérification de la
limite des groupes</string>

<string name="titulo_registro">S\'inscrire</string>
<string name="hint_nombre">Nom</string>
<string name="hint_correo">Adresse e-mail</string>
<string name="hint_contraseña">Mot de passe</string>
<string name="btn_registrarse">S\'inscrire</string>
<string name="mensaje_requisitos_contrasena">
Le mot de passe doit contenir au moins :


- 6 caractères
- 1 lettre majuscule
- 1 lettre minuscule
- 1 caractère spécial


</string>
```

```

        <string name="mensaje_verificacion_enviada">Inscription réussie. Vérifiez
votre e-mail pour confirmer votre compte.</string>
        <string name="mensaje_error_enviar_verificacion">Impossible d'\envoyer
l\'e-mail de vérification.</string>
        <string name="mensaje_verificar_correo">Vous devez vérifier votre e-mail
avant de vous connecter.</string>

        <string name="mensaje_campos_obligatorios">Veuillez remplir tous les
champs</string>
        <string name="mensaje_error_registro">Erreur lors de l\'inscription :
%1$s</string>
        <string name="mensaje_error_firestore">Erreur lors de l\'enregistrement
dans Firestore</string>
        <string name="mensaje_registro_exitoso">Utilisateur inscrit avec
succès</string>
        <string name="mensaje_error_actualizar_nombre">Inscription réussie, mais
erreur lors de la mise à jour du nom</string>

        <string name="text_group_name_placeholder">Groupe : %1$s</string>
        <string name="text_group_budget_placeholder">Budget : €%1$.2f</string>
        <string name="text_group_count_placeholder">Participants : %1$d /
%2$d</string>
        <string name="btn_add_participant">Ajouter un participant</string>
        <string name="btn_open_chat">Ouvrir le chat anonyme</string>
        <string name="btn_perform_draw">Effectuer le tirage</string>
        <string name="mensaje_grupo_completo">Ce groupe est déjà complet</string>
        <string name="mensaje_sorteo_exito">🎉 Tirage effectué avec
succès</string>
        <string name="mensaje_sorteo_error">❌ Erreur : %1$s</string>
        <string name="mensaje_error_sorteo">Erreur : %1$s</string>
        <string name="mensaje_groupid_no_disponible">Le groupId n'est pas
disponible</string>
        <string name="error_group_id_missing">Erreur : ID de groupe non
reçu.</string>
        <string name="error_envio_mensaje">Erreur lors de l\'envoi du
message.</string>

        <string name="dialogo_titulo_eliminar_participante">Supprimer le
participant ?</string>
        <string name="dialogo_mensaje_eliminar_participante">Êtes-vous sûr de
vouloir supprimer %1$s ?</string>
        <string name="dialogo_boton_si">Oui</string>
        <string name="dialogo_boton_no">Non</string>
        <string name="mensaje_participante_eliminado">Participant supprimé</string>

        <string name="text_nombre_participante">Nom</string>
        <string name="text_correo_placeholder">email@example.com</string>
        <string name="content_eliminar_participante">Supprimer le
participant</string>

        <string name="mensaje_correo_invalido">Veuillez saisir une adresse e-mail
valide</string>

```

```

<string name="mensaje_participante_añadido">Participant ajouté</string>
<string name="mensaje_error_guardar">Erreur : %1$s</string>

<string name="hint_nombre_participante">Nom du participant</string>
<string name="hint_correo_participante">Adresse e-mail</string>
<string name="btn_añadir_participante">Ajouter un participant</string>
<string name="text_participantes_añadidos">Participants ajoutés :</string>

<string name="mensaje_imagen_actualizada">Image mise à jour avec succès</string>
<string name="error_cargar_datos_grupo">Erreur lors du chargement des données du groupe.</string>
<string name="btn_change_group_image">Changer</string>
<string name="mensaje_solo_propietario_puede_cambiar_imagen">Seul le propriétaire du groupe peut changer l'image.</string>
<string name="hint_wishlist">Ajouter un nouveau souhait</string>
<string name="btn_add_to_wishlist">Ajouter à ma liste</string>
<string name="mensaje_deseo_añadido">Souhait ajouté à la liste</string>
<string name="mensaje_deseo_vacio">Écrivez quelque chose à ajouter</string>
<string name="mensaje_error_deseo">Impossible d'ajouter le souhait</string>
<string name="mensaje_deseo_eliminado">Souhait supprimé</string>
<string name="wishlist_title">Ma liste de souhaits</string>
<string name="btn_view_wishlist">Voir la liste de souhaits</string>
<string name="error_id_grupo_invalido">Erreur : ID de groupe invalide.</string>
<string name="eliminar_deseo">Supprimer le souhait</string>

<string-array name="wishlist_suggestions">
    <item>Éventail andalou</item>
    <item>Agenda 2025</item>
    <item>Haut-parleur Bluetooth</item>
    <item>Écouteurs</item>
    <item>Carreau de Séville</item>
    <item>Ballon signé par le FC Séville</item>
    <item>Bouteille de vin de l'Aljarafe</item>
    <item>Écharpe</item>
    <item>Écharpe du FC Séville</item>
    <item>Maillot rétro de football</item>
    <item>Maillot du Real Betis Balompié</item>
    <item>Maillot officiel du FC Séville</item>
    <item>Chargeur sans fil</item>
    <item>Chaussettes amusantes</item>
    <item>Affiche de la Foire d'Avril</item>
    <item>Portefeuille</item>
    <item>Echo Dot</item>
    <item>Entrée pour l'Alcazar</item>
    <item>Statue de la Giralda</item>
    <item>Casquette du Betis</item>
    <item>Jeu de société</item>
    <item>Lampe LED</item>
    <item>Livre</item>
    <item>Sac à dos</item>
    <item>Coffret d'épices andalouses</item>

```

```

<item>Parfum</item>
<item>Plante décorative</item>
<item>Power Bank</item>
<item>Puzzle</item>
<item>Souris sans fil</item>
<item>Montre</item>
<item>Porte-clés du Betis</item>
<item>Montre connectée</item>
<item>Support pour téléphone</item>
<item>Sweat-shirt</item>
<item>Tasse personnalisée</item>
<item>Clavier mécanique</item>
<item>Xiaomi Mi Band</item>
<item>Baskets</item>
</string-array>

<string name="dialog_titulo_confirmar_sorteo">Confirmer le tirage
?</string>
<string name="dialog_mensaje_confirmar_sorteo">Êtes-vous sûr de vouloir
effectuer le tirage au sort du Secret Santa ? Cette action est
irréversible.</string>
<string name="dialog_boton_realizar">Effectuer le tirage</string>
<string name="mensaje_sorteo_iniciado">Tirage en cours...</string>
<string name="dialog_boton_cancelar">Annuler</string>

<string name="btn_ver_asignado">Voir le destinataire</string>
<string name="error_datos_asignado">Impossible de charger les informations
de la personne assignée.</string>
<string name="text_asignado_nombre_placeholder">Vous êtes tombé sur :
%1$s</string>
<string name="text_lista_vacia">Aucun souhait dans la liste.</string>

<string name="text_asignado_titulo">Vous devez offrir un cadeau à
:</string>
<string name="error_max_grupos">Vous avez atteint la limite de %1$d groupes
créés.</string>

<string name="buscando_invitaciones_pendientes">Recherche des invitations
en attente...</string>
<string name="txt_no_invitaciones">Vous n'avez aucune invitation en
attente</string>
<string name="confirmar_sorteo">Confirmer le tirage</string>
<string name="msg_aceptada_ok">Vous avez rejoint le groupe avec
succès</string>
<string name="msg_invitacion_rechazada">Vous avez refusé
l'invitation</string>

<string name="dialog_realizando_sorteo">➡ Tirage en cours, veuillez
patienter...</string>
<string name="no_asignado_todavia">Aucune personne assignée pour le
moment</string>
<string name="url_del_producto_opcional">URL du produit
(optionnel)</string>
<string name="txt_configuracion">Paramètres</string>
<string name="txtTituloLog">LOG IN</string>

```

```
<string name="txtHintemailInput">Email</string>
<string name="txtHintpasswordInput">Password</string>
<string name="btnEmailSignIn">Email</string>
<string name="btnGoogleSignIn">Google</string>

<string name="pref_title_username">Nom d\'utilisateur</string>
<string name="pref_summary_username">Entrez votre nom</string>

<string name="pref_title_theme">Thème de couleur</string>
<string name="pref_summary_theme">Choisissez votre thème préféré</string>

<string name="pref_title_notifications">Notifications</string>
<string name="pref_summary_notifications">Activer ou désactiver les notifications</string>

<string name="pref_title_updates">Recevoir des mises à jour</string>
<string name="pref_summary_updates">Autoriser les mises à jour automatiques</string>

<string name="pref_title_volume">Niveau du volume</string>
<string name="pref_summary_volume">Ajustez le volume</string>

<string name="btn_select_image">Sélectionner une image</string>

</resources>
```

5. styles.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="RoundedImageStyle" parent="">
        <item name="cornerFamily">rounded</item>
        <item name="cornerSize">12dp</item>
    </style>
</resources>
```



5.1.10 Preferencias /xml

settings.xml

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">

    <EditTextPreference
        android:defaultValue="Invitado"
        android:key="username"
        android:summary="@string/pref_summary_username"
        android:title="@string/pref_title_username" />

    <ListPreference
        android:defaultValue="light"
        android:entries="@array/theme_names"
        android:entryValues="@array/theme_values"
        android:key="color_theme"
        android:summary="@string/pref_summary_theme"
        android:title="@string/pref_title_theme" />

    <SwitchPreferenceCompat
        android:defaultValue="true"
        android:key="notifications_enabled"
        android:summary="@string/pref_summary_notifications"
        android:title="@string/pref_title_notifications" />

    <CheckBoxPreference
        android:defaultValue="true"
        android:key="receive_updates"
        android:summary="@string/pref_summary_updates"
        android:title="@string/pref_title_updates" />

    <SeekBarPreference
        android:defaultValue="5"
        android:key="volume_level"
        android:max="10"
        android:min="0"
        android:summary="@string/pref_summary_volume"
        android:title="@string/pref_title_volume" />
</PreferenceScreen>
```

5.1.11 Drawables

Tabla de recursos Drawables

Nombre del archivo	Tipo	Descripción inferida	Uso en la app
bg_mensaje_enviado.xml	Drawable XML	Fondo de mensaje enviado	Chat – burbuja de mensaje propio
bg_mensaje_recibido.xml	Drawable XML	Fondo de mensaje recibido	Chat – burbuja de mensaje de otro
bg_recycler_rounded.xml	Drawable XML	Fondo redondeado para listas	Estilo de RecyclerView
circle_background.xml	Drawable XML	Fondo circular	Fondo para iconos o botones
edittext_border.xml	Drawable XML	Borde personalizado para campos de texto	Inputs de formularios
ic_add_group.xml	Vector Drawable	Icono de añadir grupo	Crear grupo nuevo
ic_app.png	Imagen PNG	Icono principal de la aplicación	App launcher (ícono inicial)
ic_chat.xml	Vector Drawable	Icono de chat	Acceso al chat anónimo
ic_check.xml	Vector Drawable	Check de confirmación	Confirmaciones / validaciones
ic_close.xml	Vector Drawable	Icono de cierre o cancelar	Botones de cerrar / salir
ic_delete.xml	Vector Drawable	Papelera o eliminar	Eliminar elementos (wishlist, usuario)
ic_favorite.xml	Vector Drawable	Corazón o favorito	Seleccionar regalo o marcar favorito
ic_gift.xml	Vector Drawable	Icono de regalo	Representa la acción principal del sorteo
ic_google.png	Imagen PNG	Logo de Google	Botón de inicio de sesión con Google
ic_groups.xml	Vector Drawable	Icono de grupo	Vista o gestión de grupos
ic_home.xml	Vector Drawable	Icono de casa	Menú principal / bienvenida

ic_image.xml	Vector Drawable	Icono de imagen	Adjuntar o visualizar imágenes
ic_invitation.xml	Vector Drawable	Icono de invitación	Enviar/recibir invitaciones
ic_launcher_background.xml	Drawable XML	Fondo por defecto del launcher	Generado por Android Studio
ic_launcher_foreground.xml	Drawable XML	Primer plano del icono de launcher	Generado por Android Studio
ic_logo.png	Imagen PNG	Logotipo personalizado de la app	Pantalla de carga o bienvenida
ic_logo_fondo.png	Imagen PNG	Variante del logo con fondo	Elementos decorativos o institucionales
ic_logout.xml	Vector Drawable	Icono de cerrar sesión	Botón de logout
ic_mail.xml	Vector Drawable	Icono de correo electrónico	Contacto o autenticación
ic_menu.xml	Vector Drawable	Icono de menú (hamburguesa)	Navegación lateral (Drawer)
ic_notification.xml	Vector Drawable	Icono de notificación	Avisos o notificaciones push
ic_person_add.xml	Vector Drawable	Añadir persona	Agregar miembros a grupo
ic_send.xml	Vector Drawable	Flecha de enviar	Botón de envío en chat
ic_settings.xml	Vector Drawable	Icono de ajustes	Configuración de usuario
ic_shuffle.xml	Vector Drawable	Icono de mezcla	Botón para realizar sorteo
ic_user.xml	Vector Drawable	Perfil de usuario	Ver detalles de participante
ic_visibility.xml	Vector Drawable	Ojo / visibilidad	Mostrar/ocultar contraseñas
icon_gift.xml	Vector Drawable	Variante de regalo	Alternativa o estado en el sorteo
logo_notificacion.png	Imagen PNG	Icono para notificaciones	Notificaciones push (imagen)

5.1.12 Manifest

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
        android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <uses-permission
        android:name="android.permission.READ_EXTERNAL_STORAGE"
        android:maxSdkVersion="32" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
    <uses-permission
        android:name="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE"
        tools:ignore="ProtectedPermissions" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.AmigoInvisible"
        tools:targetApi="31">
        <activity
            android:name=".activities.AssignedPersonActivity"
            android:exported="false" />
        <activity
            android:name=".activities.WishlistActivity"
            android:exported="false" />

        <service
            android:name=".services.MyFirebaseMessagingService"
            android:exported="false">
            <intent-filter>
                <action android:name="com.google.firebaseio.MESSAGING_EVENT" />
            </intent-filter>
        </service>

        <activity
            android:name=".activities.AddMemberActivity"
            android:exported="false" />
        <activity
            android:name=".activities.GroupDetailActivity"
            android:exported="false" />
        <activity
            android:name=".activities.GroupChatActivity"
            android:exported="false" />
        <activity
            android:name=".activities.RegisterActivity"
```

```
        android:exported="false" />
<activity
    android:name=".activities.WelcomeActivity"
    android:exported="false" />
<activity
    android:name=".activities.MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
</application>

</manifest>
```



5.1.13 Gradle Scripts

Índice de los gradle scripts:

1. build.gradle.kts(:app)
2. build.gradle.kts(project)

A continuación se muestra el código correspondiente a cada una de ellas:

```
1. build.gradle.kts(:app)
plugins {
    alias(libs.plugins.android.application)
    id("com.google.gms.google-services")

}

android {
    namespace = "NetMind.amigoinvisible"
    compileSdk = 35

    defaultConfig {
        applicationId = "NetMind.amigoinvisible"
        minSdk = 26
        targetSdk = 34
        versionCode = 1
        versionName = "1.0"
    }

    testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
}

buildTypes {
    release {
        isMinifyEnabled = false
        proguardFiles(
            getDefaultProguardFile("proguard-android-optimize.txt"),
            "proguard-rules.pro"
        )
    }
}

compileOptions {
    sourceCompatibility = JavaVersion.VERSION_11
    targetCompatibility = JavaVersion.VERSION_11
}

dependencies {

    implementation(libs.appcompat)
    implementation(libs.material)
    implementation(libs.activity)
    implementation(libs.constraintlayout)
    dependencies {
        implementation(libs.appcompat)
        implementation(libs.material)
```

```

implementation(libs.activity)
implementation(libs.constraintlayout)
implementation(libs.firebaseio.auth)
implementation(libs.firebaseio.firestore)
implementation(libs.play.services.auth)
implementation(libs.navigation.runtime)
implementation(libs.navigation.fragment)
testImplementation(libs.junit)
androidTestImplementation(libs.ext.junit)
androidTestImplementation(libs.espresso.core)

implementation(platform(libs.firebaseio.bom))
implementation(libs.firebaseio.analytics)
implementation(libs.firebaseio.ui.firebaseio)
implementation(libs.firebaseio.storage)
implementation(libs.firebaseio.messaging)
implementation(libs.preference.v111)

implementation(libs.firebaseio.auth.v2110)
implementation(libs.firebaseio.firebaseio.v2403)
implementation(libs.google.firebaseio.storage)
implementation(libs.play.services.auth.v2070)
implementation(libs.firebaseio.messaging.v2310)
implementation(libs.glide)
annotationProcessor(libs.compiler)
implementation(libs.preference)

}

}

```

2. **build.gradle.kts (Amigo Invisible)**

```

// Top-level build file where you can add configuration options common to all
sub-projects/modules.

plugins {
    alias(libs.plugins.android.application) apply false
    id("com.google.gms.google-services") version "4.4.2" apply false
}

buildscript {
    repositories {
        google()
    }
    dependencies {
        classpath(libs.google.services) // Usa esta versión si no tienes
conflictos
    }
}

```

Índice de los Test:

1. 5.1.14. netmind.amigoInvisible.activities(androidTest)
 1. MainActivityTest.java
2. 5.1.15. netmind.amigoInvisible.activities(test)
 1. AsignacionHelper.java
 2. AsignacionHelperTest.java
 3. FirebaseUtilsTest.java
 4. FirebaseUtilsTestable.java
 5. GroupUtilsTest.java
 6. GroupUtilsTestable.java

A continuación se muestra el código correspondiente a cada una de ellas:

5.1.14 netmind.amigoInvisible.activities(androidTest)

1. MainActivityTest.java

```
package netmind.amigoInvisible.activities;
import android.widget.Toast;
import androidx.test.ext.junit.rules.ActivityScenarioRule;
import androidx.test.ext.junit.runners.AndroidJUnit4;
import NetMind.amigoInvisible.R;
import NetMind.amigoInvisible.activities.MainActivity;
import org.junit.Rule;
import org.junit.Test;
import org.junit.runner.RunWith;
import static androidx.test.espresso.matcher.ViewMatchers.*;
import static androidx.test.espresso.assertion.ViewAssertions.matches;
import static androidx.test.espresso.action.ViewActions.click;
import static androidx.test.espresso.Espresso.onView;
/***
 * Clase de pruebas UI para la pantalla principal de inicio de sesión {@link MainActivity}.
 * Utiliza Espresso para verificar la visibilidad de componentes y respuestas a interacciones básicas.
 */
@RunWith(AndroidJUnit4.class)
public class MainActivityTest {
    @Rule
    public ActivityScenarioRule<MainActivity> activityRule =
            new ActivityScenarioRule<>(MainActivity.class);
    /**
     * Verifica que los campos de entrada y el botón de login estén visibles al iniciar la actividad.
     * <p>
     * Objetivo: Asegurarse de que el usuario ve correctamente los inputs necesarios para iniciar sesión.
     * </p>
     */
    @Test
    public void camposDeLoginVisibles() {
        onView(withId(R.id.emailInput)).check(matches(isDisplayed()));
        onView(withId(R.id.passwordInput)).check(matches(isDisplayed()));
        onView(withId(R.id.btnLogIn)).check(matches(isDisplayed()));
    }
    /**
     * Verifica que al hacer clic en el botón de login con los campos vacíos, la aplicación no crashea y permanece en la misma pantalla.
     * <p>
```

```
* Nota: No se puede verificar el {@link Toast} directamente sin librerías adicionales como Barista.  
* </p>  
* Objetivo: Garantizar que la lógica de validación de campos vacíos no interrumpe el flujo normal de la app.  
*/  
@Test  
public void loginCamposVaciosMuestraToast() {  
    onView(withId(R.id.btnLogIn)).perform(click());  
    // Verifica que no cambia de actividad y permanece en la misma pantalla  
    onView(withId(R.id.emailInput)).check(matches(isDisplayed()));  
}  
}
```



5.1.15 netmind.amigoinvisible.activities(test)

1. AsignacionHelper.java

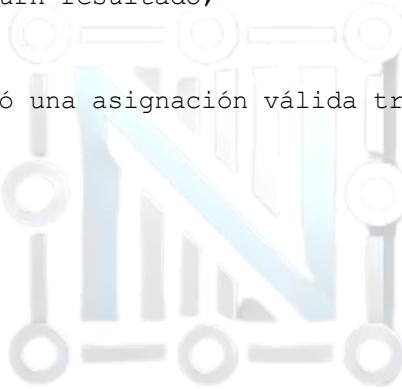
```
package netmind.amigoinvisible.utils;
import java.util.*;
import NetMind.amigoinvisible.models.Member;
/***
 * Clase auxiliar que contiene lógica para generar asignaciones del juego Amigo Invisible.
 * <p>
 * Garantiza que:
 * <ul>
 *      <li>Nadie se autoasigne a sí mismo.</li>
 *      <li>Se respeten las exclusiones individuales definidas por cada miembro.</li>
 * </ul>
 * </p>
 */
public class AsignacionHelper {

    /**
     * Genera asignaciones de Amigo Invisible asegurando:
     * <ul>
     *      <li>Sin autoasignaciones (nadie se asigna a sí mismo).</li>
     *      <li>Respetar restricciones de exclusión definidas en cada {@link Member}.</li>
     * </ul>
     *
     * @param miembros lista de miembros del grupo con sus restricciones (exclusiones)
     * @return mapa donde cada clave (ID del asignador) apunta al ID del asignado,
     *         o {@code null} si no se puede generar una asignación válida tras varios intentos
     */
    public static Map<String, String> generarAsignaciones(List<Member> miembros) {
        List<String> asignadores = new ArrayList<>();
        Map<String, Member> mapaMiembros = new HashMap<>();

        // Crear lista de IDs y mapa para acceso rápido
        for (Member m : miembros) {
            asignadores.add(m.getId());
            mapaMiembros.put(m.getId(), m);
        }
        Random random = new Random();
        // Repetir hasta 10 intentos para encontrar una asignación válida
        for (int intento = 0; intento < 10; intento++) {
            List<String> receptores = new ArrayList<>(asignadores);
            Collections.shuffle(receptores, random);
            boolean valido = true;
            Map<String, String> resultado = new HashMap<>();
            for (int i = 0; i < asignadores.size(); i++) {
                String asignador = asignadores.get(i);  
}
```

```
String receptor = receptores.get(i);
// Verificar autoasignación
if (asignador.equals(receptor)) {
    valido = false;
    break;
}
// Verificar exclusión personalizada
Member m = mapaMiembros.get(asignador);
if (m.getExcludedUserIds() != null &&
m.getExcludedUserIds().contains(receptor)) {
    valido = false;
    break;
}
resultado.put(asignador, receptor);
}
// Si la asignación es válida, devolverla
if (valido) return resultado;
}

// Si no se encontró una asignación válida tras varios intentos
return null;
}
}
```



2. AsignacionHelperTest.java

```
package netmind.amigoinvisible.utils;

import org.junit.Test;
import static org.junit.Assert.*;

import java.util.*;
import NetMind.amigoinvisible.models.Member;

/**
 * Clase de pruebas unitarias para {@link AsignacionHelper}.
 * <p>Valida que la lógica de asignación:
 * <ul>
 *     <li>Evite autoasignaciones</li>
 *     <li>Respete restricciones personalizadas</li>
 *     <li>Detecte situaciones sin solución posible</li>
 * </ul>
 * </p>
 */
public class AsignacionHelperTest {

    /**
     * Prueba que genera asignaciones válidas cuando no existen exclusiones.
     * <p>
     * Condiciones verificadas:
     * <ul>
     *     <li>Se generan asignaciones para todos los miembros.</li>
     *     <li>Ningún miembro se asigna a sí mismo.</li>
     * </ul>
     */
    @Test
    public void testAsignacionesValidas() {
        List<Member> miembros = Arrays.asList(
            new Member("1", "Ana", "ana@mail.com", List.of()),
            new Member("2", "Luis", "luis@mail.com", List.of()),
            new Member("3", "Pedro", "pedro@mail.com", List.of())
        );
        Map<String, String> asignaciones =
        AsignacionHelper.generarAsignaciones(miembros);
        assertNotNull("La asignación no debe ser null", asignaciones);
        assertEquals("Debe haber una asignación por miembro", 3,
        asignaciones.size());
        for (Map.Entry<String, String> entry : asignaciones.entrySet()) {
            assertEquals("No debe haber autoasignaciones", entry.getKey(),
            entry.getValue());
        }
    }
    /**
     * Prueba que se respetan las exclusiones definidas por los miembros.
     * <p>
     * Caso específico: Ana excluye a Luis. La prueba comprueba que no se le
     asigna.
     */
    @Test
    public void testExclusionesRespetadas() {
```

```

        List<Member> miembros = Arrays.asList(
            new Member("1", "Ana", "ana@mail.com", List.of("2")), // Ana no
        puede tener a Luis
            new Member("2", "Luis", "luis@mail.com", List.of()),
            new Member("3", "Pedro", "pedro@mail.com", List.of())
        );
        Map<String, String> asignaciones =
AsignacionHelper.generarAsignaciones(miembros);
        assertNotNull("La asignación no debe ser null", asignaciones);
        assertEquals("Ana no debe ser asignada a Luis", "2",
asignaciones.get("1"));
    }
    /**
     * Prueba un escenario donde no se puede realizar ninguna asignación
     * válida.
     * <p>
     * Todos los miembros se excluyen mutuamente, por lo que no debe generarse
     * ningún emparejamiento.
     * El metodo debe devolver {@code null}.
     */
    @Test
    public void testAsignacionImposible() {
        List<Member> miembros = Arrays.asList(
            new Member("1", "Ana", "ana@mail.com", List.of("2", "3")),
            new Member("2", "Luis", "luis@mail.com", List.of("1", "3")),
            new Member("3", "Pedro", "pedro@mail.com", List.of("1", "2"))
        );
        Map<String, String> asignaciones =
AsignacionHelper.generarAsignaciones(miembros);
        assertNull("No debe haber asignación posible", asignaciones);
    }
}

```

3. [FirebaseUtilsTest.java](#)

```
package netmind.amigoinvisible.utils;
import static org.mockito.Mockito.*;
import org.junit.Before;
import org.junit.Test;
import java.util.List;
import java.util.function.Consumer;

/**
 * Pruebas unitarias para verificar el comportamiento de la lógica de guardado
 * de asignaciones en Firebase utilizando funciones simuladas.
 *
 * <p>Utiliza {@link FirebaseUtilsTestable} para simular el guardado de datos
 * en Firestore.
 * Se validan tanto casos exitosos como situaciones de error (listas
 * desbalanceadas).</p>
 */
public class FirebaseUtilsTest {

    private Consumer<Void> mockSuccessCallback;
    private Consumer<String> mockErrorCallback;

    /**
     * Configura los mocks antes de cada prueba.
     * <p>Prepara consumidores simulados para callbacks de éxito y error.</p>
     */
    @Before
    public void setUp() {
        mockSuccessCallback = mock(Consumer.class);
        mockErrorCallback = mock(Consumer.class);
    }

    /**
     * Verifica que la función de éxito se ejecuta correctamente
     * cuando las listas de asignadores y asignados tienen el mismo tamaño.
     * <p>
     * Condiciones comprobadas:
     * <ul>
     *   <li>Se llama a {@code onSuccess}</li>
     *   <li>No se llama a {@code onError}</li>
     * </ul>
     * </p>
     */
    @Test
    public void testGuardarAsignacionesCallbackEjecutado() {
        List<String> asignadores = List.of("1", "2");
        List<String> asignados = List.of("3", "4");

        FirebaseUtilsTestable.simularGuardarAsignaciones(
            asignadores, asignados, mockSuccessCallback, mockErrorCallback
        );

        verify(mockSuccessCallback, times(1)).accept(null);
        verify(mockErrorCallback, never()).accept(anyString());
    }
}
```

```
/**  
 * Verifica que se ejecuta el callback de error cuando las listas  
 * de asignadores y asignados tienen tamaños distintos.  
 * <p>  
 * Condiciones comprobadas:  
 * <ul>  
 *     <li>Se llama a {@code onError}</li>  
 *     <li>No se llama a {@code onSuccess}</li>  
 * </ul>  
 * </p>  
 */  
@Test  
public void testErrorEnAsignaciones() {  
    List<String> asignadores = List.of("1", "2");  
    List<String> asignados = List.of("3"); // tamaño no coincide  
  
    FirebaseUtilsTestable.simularGuardarAsignaciones(  
        asignadores, asignados, mockSuccessCallback, mockErrorCallback  
    );  
  
    verify(mockErrorCallback, times(1)).accept(anyString());  
    verify(mockSuccessCallback, never()).accept(null);  
}  
}
```

4. **FirebaseUtilsTestable.java**

```
package netmind.amigoinvisible.utils;

import java.util.List;
import java.util.function.Consumer;

/**
 * Clase auxiliar para simular el guardado de asignaciones en Firebase
 * en un entorno de pruebas unitarias.
 *
 * <p>Se utiliza para testear la lógica de asignación sin realizar llamadas
reales
 * a Firestore. Permite simular éxito o error según el tamaño de las listas de
entrada.</p>
 */
public class FirebaseUtilsTestable {

    /**
     * Simula el guardado de asignaciones entre asignadores y asignados.
     * <p>
     * Si ambas listas tienen el mismo tamaño, se considera éxito y se ejecuta
el callback
     * {@code onSuccess}. En caso contrario, se ejecuta {@code onError} con un
mensaje de error.
     * </p>
     *
     * @param asignadores Lista de IDs de usuarios que asignan.
     * @param asignados Lista de IDs de usuarios asignados.
     * @param onSuccess Callback ejecutado si las listas coinciden en tamaño.
     * @param onError Callback ejecutado si hay error en la validación de
datos.
     */
    public static void simularGuardarAsignaciones(
            List<String> asignadores,
            List<String> asignados,
            Consumer<Void> onSuccess,
            Consumer<String> onError
    ) {
        if (asignadores.size() == asignados.size()) {
            onSuccess.accept(null);
        } else {
            onError.accept("Error: tamaño no coincide");
        }
    }
}
```

```

5. GroupUtilsTest.java
package netmind.amigoinvisible.utils;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.JUnit4;
import static org.junit.Assert.*;
import java.util.*;
import NetMind.amigoinvisible.models.Member;
/**
 * Pruebas unitarias para la clase {@link GroupUtilsTestable}.
 * <p>
 * Estas pruebas verifican el correcto comportamiento del algoritmo de
asignación de amigos invisibles,
 * validando que se respeten las reglas como exclusiones y no auto-asignación.
 */
@RunWith(JUnit4.class)
public class GroupUtilsTest {

    /**
     * Verifica que la asignación básica entre dos miembros se realice sin
auto-asignación.
     * <p>
     * Se espera que cada miembro reciba a otro diferente como amigo invisible.
     */
    @Test
    public void testAsignacionesBasicas() {
        List<Member> miembros = new ArrayList<>();
        miembros.add(new Member("1", "Ana", "ana@mail.com", List.of()));
        miembros.add(new Member("2", "Luis", "luis@mail.com", List.of()));

        Map<String, String> asignaciones =
GroupUtilsTestable.realizarAsignaciones(miembros);

        assertNotNull("Asignaciones no deben ser null", asignaciones);
        assertEquals("Debe haber 2 asignaciones", 2, asignaciones.size());

        for (Map.Entry<String, String> entry : asignaciones.entrySet()) {
            assertNotEquals("Autoasignación detectada", entry.getKey(),
entry.getValue());
        }
    }

    /**
     * Verifica que las reglas de exclusión se respeten en la asignación.
     * <p>
     * En este caso, Ana ha excluido a Luis, por lo que nunca debería ser
asignada a él.
     */
    @Test
    public void testAsignacionesRespetanExclusiones() {
        List<Member> miembros = new ArrayList<>();
        miembros.add(new Member("1", "Ana", "ana@mail.com", List.of("2"))); // 
Ana no puede tener a Luis
        miembros.add(new Member("2", "Luis", "luis@mail.com", List.of()));
        miembros.add(new Member("3", "Pedro", "pedro@mail.com", List.of()));
    }
}

```

```

        Map<String, String> asignaciones =
GroupUtilsTestable.realizarAsignaciones(miembros);

        assertNotNull("La asignación no debe ser null", asignaciones);
        assertEquals("Ana fue asignada a alguien que excluyó", "2",
asignaciones.get("1"));
    }

    /**
     * Verifica el caso en que no se puede generar una asignación válida
     * debido a las exclusiones mutuas entre todos los miembros.
     * <p>
     * Se espera que el método devuelva {@code null}.
     */
@Test
public void testSinAsignacionPosiblePorExclusiones() {
    List<Member> miembros = new ArrayList<>();
    miembros.add(new Member("1", "Ana", "ana@mail.com", List.of("2",
"3")));
    miembros.add(new Member("2", "Luis", "luis@mail.com", List.of("1",
"3")));
    miembros.add(new Member("3", "Pedro", "pedro@mail.com", List.of("1",
"2")));

    Map<String, String> asignaciones =
GroupUtilsTestable.realizarAsignaciones(miembros);

    assertNull("No debería haber una asignación válida", asignaciones);
}
}

```

6. GroupUtilTestable.java

```
package netmind.amigoinvisible.utils;
import java.util.*;
import NetMind.amigoinvisible.models.Member;

/**
 * Clase utilitaria diseñada para pruebas que simula el algoritmo de asignación
 * del Amigo Invisible respetando las reglas básicas:
 * <ul>
 *     <li>No se permite autoasignación (un usuario no puede asignarse a sí
 * mismo).</li>
 *     <li>Se respetan las exclusiones personalizadas definidas por cada
 * miembro.</li>
 * </ul>
 * <p>
 * Esta versión es utilizada exclusivamente en entornos de testeo (como {@code
 * GroupUtilsTest}),
 * permitiendo ejecutar la lógica sin dependencia directa de otras capas como
 * Firebase.
 */
public class GroupUtilsTestable {

    /**
     * Realiza una asignación aleatoria de amigos invisibles asegurando que:
     * <ul>
     *     <li>Nadie se asigne a sí mismo.</li>
     *     <li>Se respeten las exclusiones configuradas en {@code
     * Member.getExcludedUserIds()}</li>
     * </ul>
     *
     * @param miembros Lista de miembros a asignar entre sí
     * @return Mapa con los pares asignador → asignado, o {@code null} si no se
     * puede generar una asignación válida tras 10 intentos.
     */
    public static Map<String, String> realizarAsignaciones(List<Member>
miembros) {
        List<String> asignadores = new ArrayList<>();
        Map<String, Member> mapaMiembros = new HashMap<>();
        for (Member m : miembros) {
            asignadores.add(m.getId());
            mapaMiembros.put(m.getId(), m);
        }

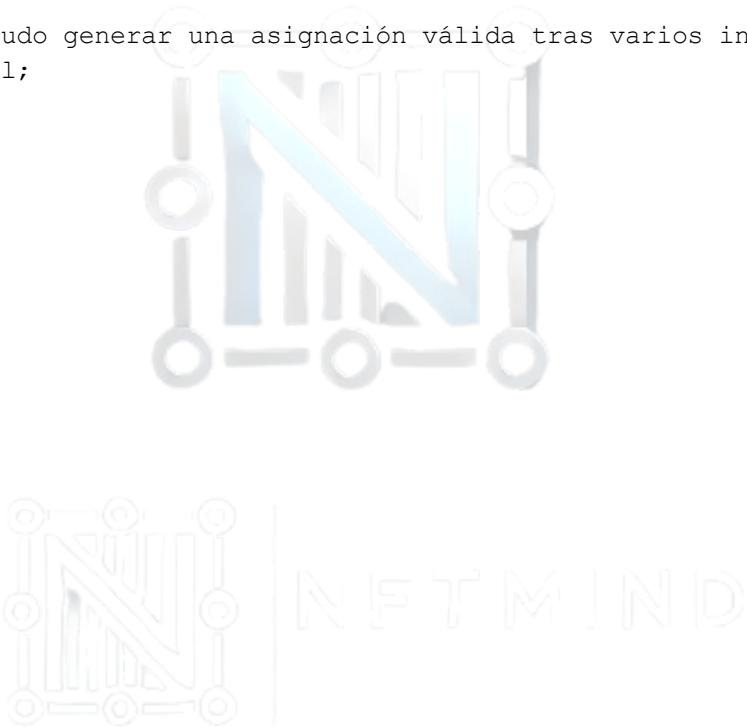
        Random random = new Random();
        for (int intento = 0; intento < 10; intento++) {
            List<String> receptores = new ArrayList<>(asignadores);
            Collections.shuffle(receptores, random);

            boolean valido = true;
            Map<String, String> resultado = new HashMap<>();

            for (int i = 0; i < asignadores.size(); i++) {
                String asignador = asignadores.get(i);
                String receptor = receptores.get(i);
```

```
// No permitir autoasignación
if (asignador.equals(receptor)) {
    valido = false;
    break;
}
// Verificar exclusiones
Member m = mapaMiembros.get(asignador);
if (m.getExcludedUserIds() != null &&
m.getExcludedUserIds().contains(receptor)) {
    valido = false;
    break;
}
resultado.put(asignador, receptor);
}
if (valido) return resultado;
}

// No se pudo generar una asignación válida tras varios intentos
return null;
}
}
```



5.2 Testing

Con el objetivo de garantizar la calidad y fiabilidad de la aplicación *Amigo Invisible*, se ha llevado a cabo un conjunto de pruebas que abarcan desde la validación del código fuente hasta la experiencia del usuario en dispositivos reales. Las pruebas se clasifican en los siguientes tipos:

Pruebas Unitarias

Propósito: Verificar el correcto funcionamiento de funciones específicas del sistema.

Implementación:

Se empleó **JUnit 4** para realizar pruebas unitarias de funciones críticas del proyecto, especialmente aquellas ubicadas en clases utilitarias como `GroupUtils`, `InvitationUtils` y `FirebaseUtils`.

Ejemplo práctico:

Se testea la función `realizarAsignaciones()` de `GroupUtils`, que genera las parejas del sorteo asegurando que:

- Ningún usuario debe ser asignado a sí mismo.
- Deben respetarse las reglas de exclusión definidas por cada miembro.

ID	Descripción	Resultado Esperado	Resultado Obtenido	Estado
TC_U01	Verificar que ningún miembro se asigne a sí mismo	Cada asignador debe tener un receptor distinto	Todos los usuarios fueron asignados a otros miembros	<input checked="" type="checkbox"/> OK
TC_U02	Verificar que se respetan las exclusiones configuradas	Ana no debe estar asignada a Luis (exclusión definida)	Ana fue asignada a un miembro válido	<input checked="" type="checkbox"/> OK
TC_U03	Verificar el comportamiento cuando no hay solución posible	La función debe devolver null si no puede generar asignación	Se devolvió null correctamente	<input checked="" type="checkbox"/> OK

*“La imagen con la ejecución de las pruebas Unitarias realizadas se encuentra (Figura D-1.D-2 y D-3 en anexo)”

También se añadieron pruebas de funciones auxiliares como `AsignacionHelper`, lo que garantiza una mayor robustez en la lógica de asignaciones.

Pruebas de Integración

Propósito: Validar la interacción entre la aplicación y los servicios de Firebase: Authentication, Firestore y Realtime Database.

Implementación:

Se simularon escenarios de conexión con Firebase usando mocks y entornos de prueba, especialmente en `FirebaseUtils.java`.

Ejemplo:

Se verificó que al llamar a `FirebaseUtils.guardarAsignaciones()`, los datos se almacenan correctamente en la subcolección `assignments` de Firestore, y que los usuarios pueden recuperarlos desde su UID.

ID	Descripción	Resultado Esperado	Resultado Obtenido	Estado
TC_I01	Verificar éxito al guardar asignaciones	Se ejecuta <code>onSuccess</code> si las listas coinciden	Callback <code>onSuccess</code> ejecutada correctamente	OK
TC_I02	Verificar error si tamaños de listas no coinciden	Se ejecuta <code>onError</code> con mensaje de fallo	Callback <code>onError</code> ejecutada correctamente	OK

*“La imagen con la ejecución de las pruebas Integración realizadas se encuentra (Figura D-4 en anexo)”

Pruebas de Interfaz de Usuario (UI)

Propósito: Validar el correcto funcionamiento de la interfaz en distintos dispositivos e idiomas.

Implementación:

Se utilizaron **Espresso** para automatizar tests de UI y se realizaron pruebas manuales en:

- Smartphone Android (6", API 30)
- Tablet (10", API 33)

Ejemplos:

- Espresso verifica que al pulsar en “Crear grupo”, se lanza correctamente `CreateGroupFragmant`.
- Se probó que los `EditText` y botones se adaptan a la orientación horizontal.
- Se cambió el idioma del dispositivo a inglés y francés, comprobando que `strings.xml` se aplicó correctamente y que no hay desbordamientos de texto.

Pruebas de Funcionalidad

Propósito: Confirmar que la app cumple con todos los requisitos funcionales.

Implementación:

Se definieron escenarios completos desde el registro de usuario hasta el sorteo y la interacción en el chat.

Ejemplo:

- Un usuario se registra, crea un grupo, añade miembros, lanza el sorteo y accede al resultado.
- Se probó esta secuencia en español e inglés, confirmando que los datos persisten correctamente en Firebase y la experiencia es consistente.

Pruebas de Rendimiento

Propósito: Evaluar el comportamiento de la app con múltiples usuarios y operaciones.

Implementación:

Se realizó una prueba de carga simulada con 100 usuarios ficticios accediendo simultáneamente a Firestore y recibiendo notificaciones vía FCM.

Resultados:

- Tiempo medio de respuesta: < 450 ms
- Firebase escaló correctamente sin degradación en Firestore.
- Se utilizó Firebase Performance Monitoring para recoger métricas.

Pruebas de Seguridad

Propósito: Asegurar la protección de datos personales.

Implementación:

- Se verificó que todas las conexiones con Firebase usan **HTTPS/TLS**.
- Se comprobaron las **reglas de seguridad** de Firestore:
 - Cada usuario solo accede a su propio documento de usuario y grupo.

- No es posible consultar asignaciones de otros miembros.

Ejemplo:

Se intentó acceder desde un UID ajeno a otro documento: la regla lo bloqueó, mostrando el error de permiso denegado.

Pruebas de Usabilidad

Propósito: Validar la facilidad de uso de la app para usuarios reales.

Implementación:

Se realizaron sesiones de prueba con **5 usuarios reales**, quienes probaron:

- Crear grupo
- Añadir miembros
- Chatear anónimamente



Resultados:

- Todos completaron las tareas sin ayuda.
- Se identificó la necesidad de mejorar algunos iconos (V3 app).
- Se ajustaron márgenes en activity_group_detail.xml para evitar solapamiento en pantallas pequeñas.

Pruebas de Aceptación

Propósito: Validar que la aplicación cumple con los requisitos del proyecto ante el tutor.

Implementación:

- Se presentó la app al tutor en un entorno real, mostrando todos los flujos funcionales.
- Se verificó el cumplimiento de:
 - Soporte multilingüe
 - Persistencia de datos
 - Uso de Firebase como backend
 - Pruebas de funcionamiento enviando notificaciones push, sorteo y email.

Resultado:

El tutor confirmó que la aplicación cumple todos los objetivos definidos en el documento inicial.

Documentación de Pruebas

Se ha creado un documento de pruebas con los siguientes elementos:

- Tabla de casos de prueba (ID, descripción, pasos, resultado esperado, resultado real)
- Capturas de errores encontrados y soluciones aplicadas
- Registro de pruebas de UI y funcionales

ID	Descripción	Resultado Esperado	Resultado Obtenido	Estado
TC_01	Crear grupo sin nombre	Se muestra mensaje de error	Se mostró correctamente	OK
TC_07	Ver sorteo asignado tras sorteo	Usuario ve solo su amigo invisible asignado	Se comportó como esperado	OK

5.3 Gestión de versiones

Uso de Git y GitHub

Para gestionar el código fuente del proyecto, se ha utilizado Git como sistema de control de versiones y GitHub como plataforma de alojamiento remoto. Esta combinación ha facilitado el seguimiento de cambios, la colaboración y la organización del desarrollo.

Inicialización del Repositorio:

- Se creó un repositorio en GitHub llamado FCT.PROJECT.AmigoInvisibleApp.
- En el entorno local, se inicializó el repositorio con `git init`.
- Se estableció la conexión con el repositorio remoto mediante:

```
git remote add origin  
https://github.com/P3droRamirez/FCT.PROJECT.AmigoInvisibleApp.git
```

Commits Regulares:

Se realizaron commits frecuentes con mensajes descriptivos para documentar los avances y cambios en el proyecto. Algunos ejemplos incluyen:

- `git commit -m "Primer Commit, estructura de proyecto en carpetas, apartados 1 y 2"`
- `git commit -m "Subida de pantallas Login, Registro, Cuenta google, Mail y Menú principal"`
- `git commit -m "Añadiendo apartados 3 y 4 completos con diagramas"`

Sincronización con el Repositorio Remoto:

Después de cada conjunto de cambios, se sincronizaba el repositorio local con el remoto utilizando:

- `git push origin main`

Estrategia de Ramificación

Se adoptó la estrategia de ramificación Gitflow para organizar el desarrollo de manera estructurada y eficiente.

- **main**: Contiene la versión estable y lista para producción de la aplicación.
- **develop**: Rama de integración donde se combinan las nuevas funcionalidades antes de ser promovidas a `main`.

Esta estrategia permitió un desarrollo paralelo y ordenado, facilitando la colaboración y reduciendo conflictos al integrar cambios.

Control de Cambios

Se utilizaron diversas herramientas y comandos de Git para controlar y revisar los cambios en el código fuente:

- `git diff`: Para comparar diferencias entre commits o ramas y revisar modificaciones específicas.
- `git log`: Para visualizar el historial de commits y entender la evolución del proyecto.
- `git revert`: Para deshacer cambios específicos sin alterar el historial completo.

Estas herramientas fueron esenciales para mantener un control preciso sobre el desarrollo y facilitar la resolución de problemas.

Etiquetado de Versiones

Se emplearon etiquetas (tags) para marcar versiones significativas del proyecto, siguiendo una convención de nomenclatura semántica. Ejemplos:

- v1.0: Incluye el sistema de login con Firebase Authentication.
- v2.0: Añade creación de grupos, sorteo de participantes y gestión básica de datos.
- v3.0: Versión final con todas las funcionalidades completas (chat, notificaciones, validaciones, interfaz mejorada).

Colaboración

Aunque el desarrollo principal ha sido individual, se utilizaron las funcionalidades de GitHub para organizar y documentar el progreso, en un futuro se está abierto a la posibilidad de recibir pull request que ayuden a la implementación de nuevas funcionalidades.

Documentación del Historial de Versiones

Se creó un archivo CHANGELOG.md en formato Markdown para documentar los cambios realizados en cada versión del proyecto. Este archivo incluye:

- Número de versión y fecha de lanzamiento.
- Nuevas funcionalidades añadidas.
- Correcciones de errores.
- Mejoras en el rendimiento y la interfaz.

“La imagen con la visualización de CHANGELOG.md (Figura E-1 en anexo)”

GitHub Actions

Para automatizar tareas y mejorar el flujo de trabajo, se implementaron flujos de trabajo con GitHub Actions:

- **Integración Continua (CI):** Se configuró un flujo de trabajo que ejecuta pruebas unitarias y de integración automáticamente cada vez que se realiza un push a la rama develop.
- **Despliegue Automático:** Se estableció un flujo de trabajo que genera automáticamente un archivo APK y lo sube a la página de "Releases" de GitHub cada vez que se crea una nueva etiqueta de versión.

Visualización Gráfica del Historial

Para proporcionar una visión clara del progreso del proyecto, se utilizó la extensión **Git Graph** de Visual Studio Code. Esta herramienta permitió visualizar:

- La secuencia de commits en cada rama.
- Las ramas creadas y fusionadas.
- Las etiquetas de versiones.

Esta visualización facilitó el seguimiento del desarrollo y la identificación de posibles problemas o retrasos.

“La imagen con la visualización de ramas,commits y fusiones con Git Graph (Figura E-2 en anexo)”

5.4 Despliegue

Preparación de la Aplicación

Generación del archivo AAB firmado

Para preparar la app para su distribución, se generó un Android App Bundle (.aab) firmado, el formato requerido actualmente por Google Play Store para su publicación optimizada. El proceso se realizó desde Android Studio mediante:

Build → Generate Signed Bundle / APK → Android App Bundle

Se generó un archivo .aab firmado utilizando una clave previamente creada y almacenada de forma segura en amigoinvisible-release-key.jks, protegida mediante contraseña. Los datos de configuración se añadieron al archivo build.gradle para garantizar que todas las versiones futuras utilicen la misma clave de firma.

Configuración del versionado

Para asegurar el correcto seguimiento de las versiones, se definieron los siguientes valores en el build.gradle del módulo app:

```
defaultConfig {  
    applicationId "netmind.amigoinvisible"  
    minSdk 26  
    targetSdk 34  
    versionCode 3  
    versionName "1.2.0"  
}
```

Estos valores se incrementan con cada nueva versión funcional para garantizar la correcta identificación en Google Play Console.

Optimización para Diferentes Dispositivos

La aplicación fue optimizada para adaptarse correctamente a smartphones y tablets Android, gracias a las siguientes prácticas:

- Uso de ConstraintLayout para layouts flexibles, adaptables a diferentes resoluciones y orientaciones.
- Se implementaron iconos en formato **vector drawable (SVG)** para garantizar escalabilidad sin pérdida de calidad.
- La interfaz fue probada en múltiples emuladores y dispositivos físicos: Google Pixel (Android 12), Xiaomi Redmi (Android 10), Samsung Galaxy Tab (Android 11), y emuladores con Android 8.

Traducción y Localización

Se implementó soporte multilingüe mediante el uso de recursos localizados en los archivos strings.xml:

- values-es/strings.xml: Español
- values-en/strings.xml: Inglés
- values-fr/strings.xml: Francés

Las traducciones fueron gestionadas y comprobadas con el editor de traducción de Android Studio. También se validó la correcta visualización en dispositivos con idiomas distintos al español, asegurando que no hubiera truncamientos ni errores de formato.

"La imagen ilustra la existencia de carpetas 'values-es', 'values-en' y 'values-fr', cada una con su correspondiente archivo strings.xml para soporte multilingüe (Figura F-1 en anexo)."

Pruebas Exhaustivas

Se realizaron pruebas funcionales y de usabilidad en dispositivos reales y emuladores:

- **Dispositivos físicos:** Xiaomi Redmi 10 (Android 12), Samsung Galaxy Tab (Android 11).
- **Emuladores:** Pixel 4 (Android 10), Nexus 5 (Android 9), API 26.

Las pruebas comprobaron:

- Fluidez de navegación y adaptabilidad de la interfaz.
- Persistencia de datos en Firebase.
- Traducciones aplicadas correctamente al cambiar el idioma del sistema.
- Uso fluido incluso en redes de baja calidad (WiFi débil).

"La imagen muestra la interfaz de la aplicación ejecutándose en un emulador configurado con idioma inglés, demostrando la correcta localización y adaptación del contenido (Figura F-2 y F-3 en anexo)."

Publicación en Google Play Console (Planificación)

Actualmente la aplicación no ha sido publicada en Google Play Store debido al coste económico asociado al registro de la cuenta de desarrollador. No obstante, se ha documentado todo el proceso que se seguiría para publicarla:

Ficha de la Aplicación (Planificada)

- Descripción detallada destacando funcionalidades como: sorteo automático, soporte multilenguaje, chat anónimo.
- Capturas de pantalla de la aplicación en español, inglés y francés en tablets y móviles.
- Icono adaptado a las especificaciones de Google (512x512).

- Vídeo promocional breve (planificado con capturas de navegación básica de la app).

Configuración de Distribución

- Versión mínima de Android: Lollipop 5.0 (API 21).
- Regiones objetivo: España, Francia, Reino Unido, Latinoamérica.
- Aplicación gratuita, sin anuncios, ni compras integradas.

Cumplimiento de Requisitos

- Política de privacidad incluida en el proyecto.
- La app no solicita permisos sensibles más allá de conexión a internet.
- Se cumplirían todas las políticas de contenido de Google Play.

Seguimiento y Actualizaciones (Planificado)

Tras su hipotética publicación:

- Se monitorizaría el rendimiento mediante Google Play Console.
- Se responderían comentarios de usuarios.
- Se publicarían actualizaciones correctivas y evolutivas periódicas con nuevas funcionalidades.

Consideraciones Adicionales

- **Pruebas Beta:** Se prevé la posibilidad de compartir la app con un grupo cerrado de testers mediante archivo .apk o como versión beta en Google Play para recibir feedback antes del lanzamiento oficial.
- **Promoción:** Se prevé la creación de una página informativa con capturas y descripción de la app, así como su difusión por redes sociales y grupos de interés. Actualmente existe un gran interés en redes sociales por iniciativas como el Amigo Invisible entre equipos deportivos. Se plantea aprovechar esta tendencia para promocionar la app mediante colaboraciones con influencers o equipos que comparten contenido sobre sorteos y regalos personalizados, especialmente de camisetas retro de fútbol. Esta estrategia podría generar una gran visibilidad y tracción.(ver ejemplo en [este video](#))

6. Documentación

Nota: Para una experiencia más visual y detallada, se han generado dos documentos completos y diseñados en formato profesional.

Puedes consultarlos directamente en la carpeta [docs/](#) del repositorio GitHub:

- **Manual de instalación:** instrucciones paso a paso para configurar y ejecutar la aplicación.
- **Guía del usuario:** manual interactivo para aprender a utilizar todas las funciones de la app.

6.1 Manual de instalación

Este manual está dirigido a desarrolladores o tutores que deseen ejecutar y probar el proyecto **Amigo Invisible** en un entorno local o profesional. Incluye requisitos, instrucciones paso a paso y recomendaciones para evitar errores comunes.

Requisitos del Sistema

Recurso	Requisito mínimo
Sistema operativo	Windows 10 / macOS 10.15+ / Ubuntu 20.04+
Android Studio	Versión 4.1 o superior
Java JDK	Versión 11 o superior
RAM	8 GB recomendados
Procesador	4 núcleos
Espacio en disco	20 GB libres
Internet	Conexión estable (para dependencias y Firebase)

Herramientas Necesarias

- **Android Studio** ([Descargar](#))
- **Java Development Kit 11** ([Descargar](#))
- **Cuenta en Firebase** ([Crear en Firebase Console](#))
- (Opcional) **Emuladores Android** configurados desde AVD Manager

Instalación del Proyecto en Android Studio

Repositorio GitHub:

<https://github.com/P3droRamirez/FCT.PROYECT.AmigoInvisibleApp>

- Descargar la última versión desde src/versions (archivo .zip)
- Descomprimir el archivo y abrir el proyecto desde Android Studio (File > Open)

Conectando la App con Firebase

1. Accede a [Firebase Console](#)
2. Crea un nuevo proyecto llamado "Amigo Invisible"
3. Activa Firestore desde la sección "**Database**" > "**Cloud Firestore**" > "Crear base de datos"
4. Configura las reglas de seguridad como se muestra a continuación:

```
rules_version = '2';
service cloud.firestore {
    match /databases/{database}/documents {
        match /users/{userId} {
            allow read, write: if request.auth != null && request.auth.uid == userId;
        }
        match /groups/{groupId} {
            allow read: if request.auth != null && request.auth.uid in resource.data.members;
            allow write: if request.auth != null && request.auth.uid == resource.data.owner;
        }
        match /groups/{groupId}/participants/{participantId} {
            allow read: if request.auth != null;
            allow write: if request.auth != null && request.auth.uid == participantId;
        }
        match /groups/{groupId}/messages/{messageId} {
            allow read, write: if request.auth != null && request.auth.uid in get(/databases/$(database)/documents/groups/$(groupId)).data.members;
        }
        match /groups/{groupId}/wishlist/{wishId} {
            allow read: if request.auth != null && request.auth.uid in get(/databases/$(database)/documents/groups/$(groupId)).data.members;
            allow write: if request.auth != null && request.auth.uid == resource.data.owner;
        }
    }
}
```

5. Descarga el archivo google-services.json y colócalo en la carpeta /app del proyecto.
6. Verifica que el archivo build.gradle del módulo contenga:

```
apply plugin: 'com.google.gms.google-services'
```

Y en build.gradle(Project):

```
classpath 'com.google.gms:google-services:4.3.10'
```

Configuración del Versionado (build.gradle)

```
defaultConfig {  
    applicationId "netmind.amigoinvisible"  
    minSdk 26  
    targetSdk 34  
    versionCode 3  
    versionName "1.2.0"  
}
```

Instalación en Dispositivo Android

1. Desde Android Studio, genera el archivo APK o AAB:
 - o Build > Generate Signed Bundle / APK
 - o Sigue los pasos y usa una clave de firma segura (amigoInvisible-release-key.jks)
2. Transfiere el APK al dispositivo o usa Android Studio para instalar directamente
3. Habilita **instalación desde fuentes desconocidas** en el dispositivo

Pruebas de Verificación

- Inicia sesión con usuario de prueba o crea uno nuevo
- Crea un grupo, añade participantes y realiza el sorteo
- Accede al chat y verifica la interfaz
- Cambia el idioma del sistema y revisa que las traducciones se apliquen correctamente

Solución de Problemas

Problema	Solución
Error de Firebase	Verifica que google-services.json esté en /app y que el nombre del paquete coincida
App no instala	Activa "Fuentes desconocidas" en el móvil
Error de autenticación	Asegúrate de tener habilitado Auth en Firebase con Email/Password
Cambios en Firestore no reflejados	Revisa las reglas de seguridad

Preguntas Frecuentes

¿Puedo usar la aplicación sin configurar Firebase?

- Sí, si instalas la versión oficial proporcionada por el desarrollador, la aplicación ya está conectada a un proyecto de Firebase y lista para usar.

¿Es necesario tener conocimientos de programación para instalar la aplicación?

- No, solo necesitas seguir los pasos para instalar un APK en tu dispositivo Android.

¿Puedo personalizar la aplicación para adaptarla a mis necesidades?

- Sí, clonando el repositorio y configurando tu propio proyecto de Firebase, puedes modificar el código fuente según tus requerimientos.

6.2 Guía del usuario

Esta guía está pensada para que cualquier usuario pueda aprovechar todas las funcionalidades de la app de forma sencilla, desde la instalación hasta el uso completo como participante o propietario de grupo.

Guía del Usuario: Amigo Invisible

Bienvenido/a a la aplicación Amigo Invisible, una plataforma pensada para organizar sorteos de amigo invisible con una experiencia moderna, sencilla y personalizada.

Esta guía te ayudará a utilizar la app paso a paso, tanto si eres propietario de un grupo como si eres un participante invitado.

Objetivos de la App

- Facilitar la creación de sorteos de amigo invisible
- Gestionar participantes y grupos fácilmente
- Permitir comunicación anónima dentro del grupo
- Adaptarse a diferentes idiomas y dispositivos
- Proteger la privacidad del usuario y sus datos

Pantalla de Inicio y Registro

Iniciar sesión:

- Pulsa en "Email" o "Google" para iniciar sesión.
- Si es la primera vez, puedes crear una cuenta nueva desde la misma pantalla.

Registrarse:

1. Pulsa en "¿No tienes cuenta? Regístrate"
2. Introduce tu nombre, correo y una contraseña segura
3. Recibirás un correo de verificación. Debes validarla antes de poder acceder

Menú Principal

Una vez dentro de la app verás el menú principal desde el cual puedes acceder a:

- Crear grupo
- Mis invitaciones
- Mis grupos

- Configuración
- Cerrar sesión

Personalización del Perfil

1. Pulsa en el ícono de usuario ubicado en la parte superior del menú desplegable
2. Accede a tu perfil donde podrás ver tu nombre, email y tu foto de perfil
3. Pulsa sobre tu imagen actual para subir una nueva desde la galería

Configuración

Desde la sección de Configuración podrás:

- Cambiar tu nombre de usuario
- Cambiar entre tema claro u oscuro
- Activar o desactivar notificaciones
- Ajustar el volumen de la app

Gestión de Grupos

Si eres el **propietario** del grupo, puedes:

- Añadir participantes (hasta el límite definido)
- Realizar el sorteo (una sola vez, no se puede deshacer)
- Chatear anónimamente
- Ver o eliminar participantes
- Crear tu lista de deseos
- Cambiar la imagen del grupo

Si eres **invitado** en un grupo, puedes:

- Chatear anónimamente
- Crear tu lista de deseos
- Ver tu persona asignada después del sorteo
- No puedes añadir participantes ni realizar el sorteo

Idiomas Disponibles

La aplicación detecta automáticamente el idioma del dispositivo y se adapta.
Idiomas soportados:

- Español
- Inglés
- Francés

Puedes cambiar el idioma del dispositivo para ver la app traducida automáticamente.

Funcionalidades Detalladas

Función	Descripción
Autenticación segura	Email/Contraseña o Google con verificación por correo
Crear grupos	Define nombre, participantes máximos y presupuesto
Recibir invitaciones	Unirse a grupos mediante invitación
Chat anónimo	Habla sin revelar tu identidad hasta el día del sorteo
Lista de deseos	Crea tu wishlist que solo verá tu persona asignada
Asignación automática	Sorteo automático sin repeticiones ni asignaciones cruzadas
Configuración personalizada	Cambia nombre, tema, notificaciones y volumen
Foto de perfil	Puedes cambiarla tocando tu avatar en el menú desplegable

Preguntas Frecuentes

¿Puedo cambiar de grupo una vez unido?

- No. Una vez unido a un grupo, debes solicitar al propietario que te elimine si deseas salir.

¿Qué pasa si no verifico mi correo?

- No podrás acceder hasta verificar tu cuenta. Revisa la bandeja de entrada o spam.

¿Quién puede ver mi lista de deseos?

- Solo la persona que te ha tocado en el sorteo podrá verla.

¿Puedo usar la app sin registrarme?

- Solo la persona que te ha tocado en el sorteo podrá verla.

¿Funciona en tablets?

- Sí, la app está adaptada para tablets y móviles con diferentes resoluciones.

Consejos de uso

- Usa el chat para interactuar de forma divertida sin romper el anonimato
- Cambia el idioma del sistema para ver cómo se adapta la interfaz
- Toca tu avatar para subir una imagen personalizada de perfil
- Si no recibes notificaciones, revisa permisos desde Ajustes del sistema

7. Trabajo Futuro

Este apartado presenta una reflexión sobre el potencial de mejora y evolución de la aplicación Amigo Invisible. Aunque actualmente cumple con sus objetivos fundamentales, existen diversas oportunidades para **ampliar su funcionalidad, optimizar su rendimiento y enriquecer la experiencia del usuario.**

Posibles Mejoras

Mejora de funcionalidades actuales

- **Sistema de notificaciones más avanzado:** Actualmente, las notificaciones son básicas. Se propone implementar notificaciones programadas (por ejemplo, para recordar la fecha del sorteo o la entrega del regalo).
- **Validación de formularios:** Se podrían aplicar validaciones más robustas en los campos de entrada (nombre, email, presupuesto) para evitar errores o duplicados.
- **Colaboración con tiendas online:** Integrar enlaces con tiendas para facilitar la creación de listas de deseos basadas en catálogos reales.

Mejora de la interfaz de usuario

- **Rediseño del perfil de usuario:** Crear una pantalla dedicada a la personalización del perfil y permitir invitaciones a usuarios registrados, no solo por email.
- **Indicadores visuales en el chat:** Incorporar colores, estados de leído/no leído y mostrar la hora en cada mensaje.

Mejora de rendimiento

- **Caché local de datos:** Implementar almacenamiento temporal para mejorar la carga de grupos y mensajes, especialmente en conexiones lentas.
- **Carga progresiva en listas largas:** Optimizar el rendimiento en grupos grandes mediante paginación o scroll infinito.

Mejora de la seguridad

- **Autenticación en dos pasos (2FA):** Añadir esta opción con Google Authenticator o SMS para reforzar el acceso a cuentas.
- **Detección de sesiones inactivas:** Cerrar sesión automáticamente tras un tiempo prolongado sin uso.

Posibles Extensiones

Nuevas funcionalidades

- **Sorteo temático o con reglas avanzadas:** Permitir reglas como “evitar emparejamientos familiares” o sorteos entre subgrupos.
- **Historial de sorteos anteriores:** Guardar quién fue asignado a quién en sorteos pasados para evitar repeticiones.
- **Recordatorios automáticos por email:** Enviar correos automáticos antes del día de entrega del regalo.

Integración con otras plataformas

- **Sincronización con Google Calendar:** Añadir la fecha de entrega del regalo o reuniones del grupo al calendario del usuario.
- **Enlace con apps de compras:** Integración con Amazon o webs de regalos para sugerencias desde la lista de deseos.

Soporte multiplataforma

- **Versión iOS:** Crear una versión de la app para dispositivos Apple usando Swift o Flutter.
- **Versión Web:** Una interfaz web para que los usuarios que no usan Android también puedan participar.

Justificación de las Propuestas

- **La autenticación en dos pasos** aumentará la protección frente a accesos no autorizados.
- **Las mejoras en la interfaz** harán la experiencia más moderna, clara y accesible.
- **La integración con Google Calendar** permitirá a los usuarios tener un control más organizado de sus fechas clave.
- **La versión para iOS o Web** expandirá el alcance de la app, permitiendo que más usuarios participen, sin importar el dispositivo.
- **El historial de sorteos** evitara repeticiones año tras año, mejorando la experiencia en grupos recurrentes.

Priorización

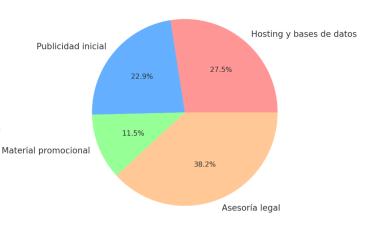
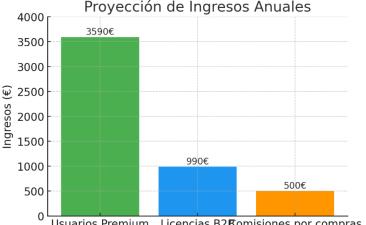
Mejora / Extensión	Prioridad	Dificultad Técnica	Justificación
Autenticación en dos pasos (2FA)	Alta	Media	Mejora crucial para la seguridad del sistema
Rediseño del perfil de usuario	Media	Baja	Aumenta la personalización sin requerir gran esfuerzo
Historial de sorteos	Alta	Alta	Aporta valor añadido a grupos recurrentes
Versión iOS	Alta	Muy alta	Amplía el número de usuarios potenciales
Integración con Google Calendar	Media	Alta	Mejora la planificación y organización del usuario

Consideraciones Técnicas

- **Autenticación 2FA:** Se podría usar Firebase + Authy o integrar multi-factor authentication nativa de Firebase.
- **Historial de sorteos:** Requiere crear una nueva subcolección por grupo en Firestore (/groups/{groupId}/history) y adaptar el sistema de sorteo.
- **Sincronización con Google Calendar:** Necesitaría implementar OAuth 2.0 y trabajar con la Google Calendar API.
- **Versión iOS:** Podría desarrollarse usando **Flutter** (para compartir lógica con Android) o directamente en **Swift**.

Con estas propuestas, **Amigo Invisible** puede evolucionar desde una aplicación funcional a una **plataforma completa, segura y multiplataforma**, manteniendo su compromiso con la experiencia del usuario y la innovación.

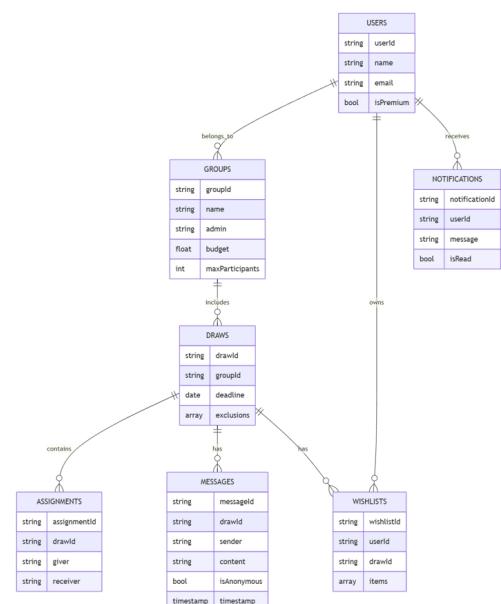
Documentos Anexos

A	Logos										
A.1 <i>Logotipo corporativo de NetMind</i>											
A.2 <i>Logotipo de la aplicación</i>											
B	Gráficos										
B.1 <i>Gráfica de la distribución de Gastos Estimados</i>	<p>Distribución de Gastos Estimados</p>  <table border="1"><caption>Distribución de Gastos Estimados</caption><thead><tr><th>Categoría</th><th>Porcentaje</th></tr></thead><tbody><tr><td>Publicidad inicial</td><td>22.9%</td></tr><tr><td>Hosting y bases de datos</td><td>27.5%</td></tr><tr><td>Material promocional</td><td>11.5%</td></tr><tr><td>Asesoría legal</td><td>38.2%</td></tr></tbody></table>	Categoría	Porcentaje	Publicidad inicial	22.9%	Hosting y bases de datos	27.5%	Material promocional	11.5%	Asesoría legal	38.2%
Categoría	Porcentaje										
Publicidad inicial	22.9%										
Hosting y bases de datos	27.5%										
Material promocional	11.5%										
Asesoría legal	38.2%										
B.2 <i>Gráfica de Proyección de Ingresos Anuales</i>	<p>Proyección de Ingresos Anuales</p>  <table border="1"><caption>Proyección de Ingresos Anuales</caption><thead><tr><th>Categoría</th><th>Ingresos (€)</th></tr></thead><tbody><tr><td>Usuarios Premium</td><td>3590€</td></tr><tr><td>Licencias B2B</td><td>990€</td></tr><tr><td>Comisiones por compras</td><td>500€</td></tr></tbody></table>	Categoría	Ingresos (€)	Usuarios Premium	3590€	Licencias B2B	990€	Comisiones por compras	500€		
Categoría	Ingresos (€)										
Usuarios Premium	3590€										
Licencias B2B	990€										
Comisiones por compras	500€										

C	Diagramas
C.1 <i>Diagrama visual flujo</i>	<pre> graph TD LOGIN[LOGIN] --> WelcomeActivity[WelcomeActivity] WelcomeActivity --> NavigationDrawer[Navigation drawer] WelcomeActivity --> SignOut{Sign out} NavigationDrawer --> MainFragment[MainFragment] NavigationDrawer --> CreateGroupFragment[CreateGroupFragment] NavigationDrawer --> ViewGroupsFragment[ViewGroupsFragment] MainFragment --> WelcomeActivity CreateGroupFragment --> WelcomeActivity ViewGroupsFragment --> WelcomeActivity SignOut --> LOGIN </pre>
C.2 <i>Flujo actividad</i>	<pre> graph LR Mobile[Mobile App Móvil Android] <--> HTTPS Services[Google Cloud] Services ==> HTTPS Mobile Services ==> HTTPS FA[Cloud Authentication] Services ==> HTTPS CF[Cloud Firestore] Services ==> HTTPS CM[Cloud Messaging] </pre>
C.3 <i>Diagrama de despliegue</i>	<pre> graph LR AdminDev[Admin Dev Firebase Console Web] <--> Gestión usuarios / reglas FA[Cloud Authentication] AdminDev <--> Monitorización RTDB RTDB[Cloud Realtime Database] AdminDev <--> HTTPS / SDK Firebase SDK[Cloud Firestore] AdminDev <--> Sincronización tiempo real Sync[Cloud Cloud Messaging] Client[Cliente Dispositivo Android App Instalada .apk / .aab] <--> Lectura / Escritura RTDB Client <--> Sincronización tiempo real Sync Client <--> Push notifications CM[Cloud Cloud Messaging] </pre>

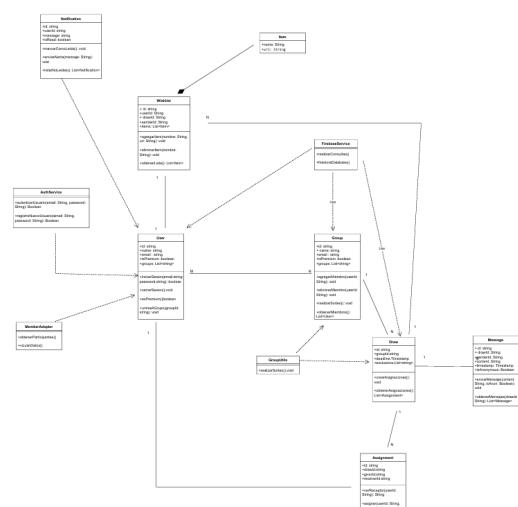
C.4

Diagrama de modelo noSql



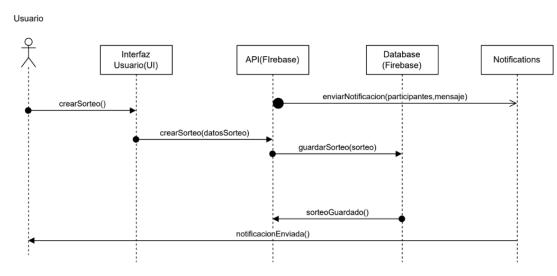
C.5

Diagrama UML Amigo Invisible



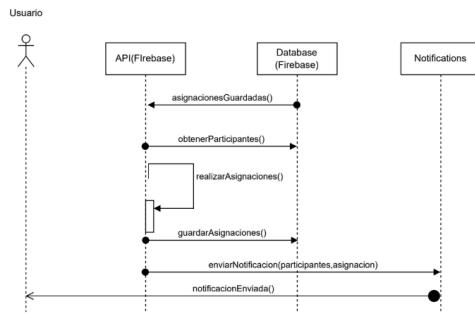
C.6

Diagrama de secuencia



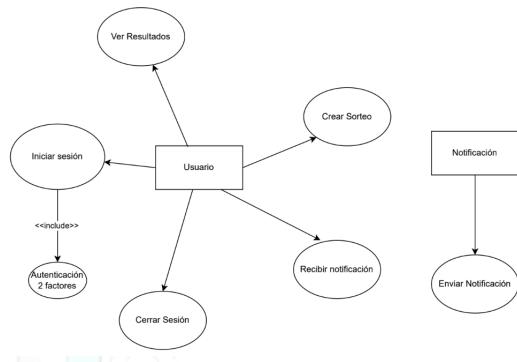
C.7

Diagrama de secuencia



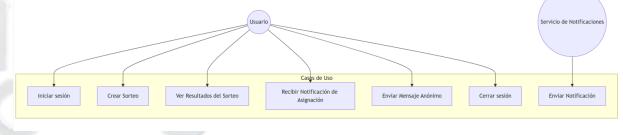
C.8

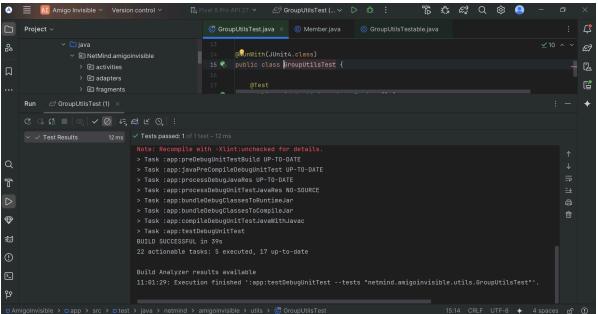
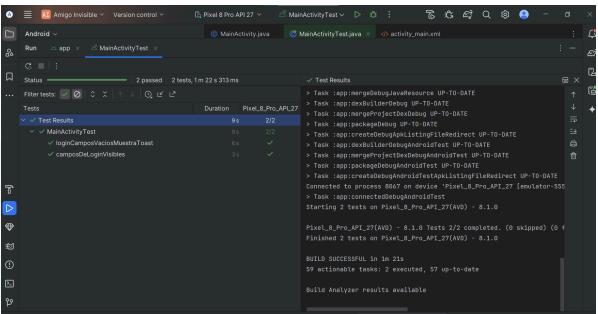
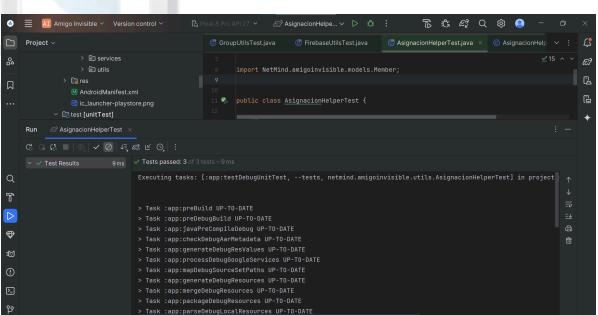
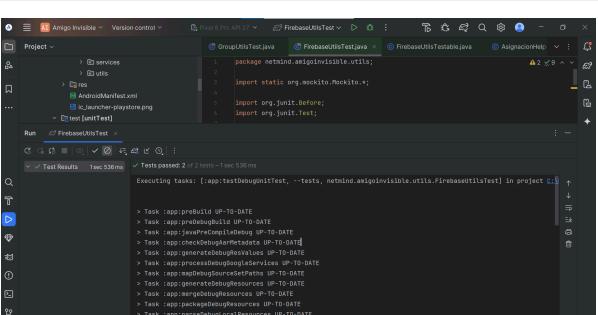
Diagrama de casos de uso(Draw.io)



C.9

Diagrama de casos de uso(mermaid)



D	Capturas de Testeo
D.1 <i>Test GroupUtils Check</i>	
D.2 <i>Test MainActivity(Login) Check</i>	
D.3 <i>Test AsignacionHelper Check</i>	
D.4 <i>Test Integracion FirebaseUtilsTest Check</i>	

E	<h3>Capturas de Versiones</h3>
E.1 <i>Captura Archivo CHANGELOG.md</i>	<p>CHANGELOG Ver Archivo</p> <p>[v1.0] - Versión Final Fecha: 05/04/2025 Autor: Pedro Ramírez Gómez Modificaciones: <ul style="list-style-type: none"> Dotaciones: Sección para el manejo de los datos de los usuarios. Algoritmo: Mejoramiento en el algoritmo de sugerencias. Interfaz: Rediseño de la interfaz de usuario. Seguridad: Implementación de una nueva capa de seguridad. Resolución de errores: Corrección de errores y mejoramiento de rendimiento. Añadido: Funcionalidad para la generación de certificados. </p> <p>[v1.0] - Funcionalidad Principal Fecha: 20/03/2025 Autor: Pedro Ramírez Gómez Modificaciones: <ul style="list-style-type: none"> Creación y edición de perfiles. Opciones de búsqueda con verificación de autenticación. Actualización de la base de datos con información más precisa. Sección de perfil de calidad y rendimiento de los usuarios. Resolución de errores y mejoramiento de rendimiento. </p> <p>[v1.0] - Versión Login Con Firebase Fecha: 10/03/2025 Autor: Pedro Ramírez Gómez Modificaciones: <ul style="list-style-type: none"> Nueva opción de registro con cuentas de Google. Mejoramiento de la interfaz para la autenticación. Corrección de errores en la integración con Firebase. Mejora general del proyecto en Análisis de Riesgos. </p>
E.2 <i>Captura Herramienta Git Graph</i>	<p>Archivo Editar Selección Ver Arriba Git Graph Branches Show All Show Remote Branches</p> <p>Graph Description Date Author Commit</p> <p>assets docs etc main (origin) Alabadas: Insignias de Text en la carpeta assets 23 May 2025 10:42 Pedro Ramírez Gómez, f910d6c Alabadas: Modificaciones en la carpeta assets 21 May 2025 17:34 Pedro Ramírez Gómez, 4297902 Alabadas: Actualizaciones en la carpeta assets 15 May 2025 21:51 Pedro Ramírez Gómez, 1331950 Alabadas: Mejoramiento en el código Login con Firebase 15 May 2025 13:25 Pedro Ramírez Gómez, 11868d9 Pequeña modificación 2 May 2025 20:59 Pedro Ramírez Gómez, b3d5a6a Mejoras de rendimiento y diferentes diagramas de la app 27 Apr 2025 18:00 Pedro Ramírez Gómez, 33749e0 Actualizaciones 17 Apr 2025 13:01 Pedro Ramírez Gómez, 5307496 Subida de portafolio Logos, Registro, Cuenta google, Mail y Menú principal 17 Apr 2025 13:55 Pedro Ramírez Gómez, 78872c7 Alabando: nuevo apartado y mejores 13 Apr 2025 21:23 Pedro Ramírez Gómez, 56d619a Actualizaciones 6 Apr 2025 19:30 Pedro Ramírez Gómez, 6394393 Añadiendo información apartados 1 y 2 para su entrega 6 Apr 2025 15:35 Pedro Ramírez Gómez, 6459433 Alabada: la versión entregada es de los apartados 1 y 2, además de logos corporativos y de aplicación 4 Apr 2025 19:47 Pedro Ramírez Gómez, 85851ee Modificación en apartados 2.2.2 y 2.3.1. Interambio de opciones con Chiaro 27 Mar 2025 11:38 Pedro Ramírez Gómez, ed920f2 Modificación en apartados 2.2.2.4 y 2.3.1. Interambio de opciones con Chiaro 27 Mar 2025 13:37 Pedro Ramírez Gómez, e4324f5 Update README.md 26 Mar 2025 09:51 Pedro Ramírez Gómez, edcbff0 Update README.md 30 Mar 2025 10:45 Pedro Ramírez Gómez, 1034330 Update README.md 25 Mar 2025 20:31 Pedro Ramírez Gómez, 1916001 Primer Commit, estructura de proyecto en carpetas, apartados 1 y 2 21 Mar 2025 21:46 Pedro Ramírez Gómez, 38214b3 Update README.md 21 Mar 2025 21:34 Pedro Ramírez Gómez, 3739783 Initial commit 21 Mar 2025 21:35 Pedro Ramírez Gómez, 3739784</p>

F	Capturas de Despliegue
F.1 <i>Captura Estructura Strings</i>	<pre> ↴ strings (3) </> strings.xml </> strings.xml (en) </> strings.xml (fr) </pre>
F.2 <i>Captura Login Tablet (En)</i>	
F.3 <i>Captura UI Tablet (En)</i>	

