NEBRIJA FORMACIÓN PROFESIONAL / Informática y Comunicaciones
Maitane Ibáñez Irazabal / Jose Carlos Rodriguez de la Llana
Desarrollo de Aplicaciones Multiplataforma

UmeEgunero

Trasformando la comunicación educativa: Un puente digital entre escuelas y hogares

Versión	Fecha	Motivo de modificación	Elaboración	Revisión	Aprobación

ANTONIO DE NEBRIJA
500 AÑOS

INSTITUTO NEBRIJA | Formación Profesional

ABSTRACT

El presente Trabajo de Fin de Grado documenta el desarrollo de UmeEgunero, una aplicación móvil para dispositivos Android diseñada para mejorar la gestión y comunicación en centros de educación infantil. Utilizando las últimas tecnologías de desarrollo para Android, como Jetpack Compose y Kotlin, hemos creado una plataforma que facilita la interacción entre familias y profesores, automatiza procesos administrativos y permite un seguimiento detallado del desarrollo de los alumnos.

La aplicación implementa un sistema multiusuario con diferentes roles (administradores, directores de centro, profesores y familiares), integración con Firebase para almacenamiento en la nube, autenticación segura y comunicación en tiempo real. Los resultados preliminares muestran una mejora significativa en la eficiencia de la gestión educativa y una experiencia de usuario intuitiva adaptada a las necesidades específicas de los centros infantiles.

Este proyecto representa una contribución importante al sector educativo, ofreciendo una herramienta tecnológica accesible que aborda las necesidades reales de comunicación y gestión en la educación infantil moderna.

This Final Degree Project documents the development of UmeEgunero, a mobile application for Android devices designed to improve management and communication in early childhood education centers. Using the latest Android development technologies, such as Jetpack Compose and Kotlin, we have created a platform that facilitates interaction between families and teachers, automates administrative processes, and allows detailed monitoring of student development.

The application implements a multi-user system with different roles (administrators, center directors, teachers, and family members), integration with Firebase for cloud storage, secure authentication, and real-time communication. Preliminary results show a significant improvement in educational management efficiency and an intuitive user experience adapted to the specific needs of children's centers.

This project represents an important contribution to the education sector, offering an accessible technological tool that addresses the real needs of communication and management in modern early childhood education.

INTRODUCCIÓN

1. CONTEXTUALIZACIÓN DEL PROYECTO

UmeEgunero nace como respuesta a una necesidad real y creciente en los centros educativos: contar con una plataforma digital moderna, intuitiva y completa que centralice la gestión del día a día escolar y potencie la comunicación entre todos los agentes implicados en la educación infantil.

Durante nuestro análisis del panorama actual, detectamos que muchas soluciones existentes presentan interfaces desactualizadas, funcionalidades poco adaptadas o están divididas en múltiples aplicaciones que dificultan su uso continuado y eficaz. Asimismo, observamos que la mayoría de ellas no aprovechan las ventajas que ofrecen las tecnologías móviles más actuales.

Con UmeEgunero hemos querido dar un paso adelante, desarrollando una aplicación Android que apuesta por una experiencia de usuario fluida, coherente y accesible. Para ello, empleamos **Jetpack Compose** junto a los principios de **Material Design 3**, lo que nos ha permitido construir interfaces reactivas y adaptables a cada perfil de usuario. Nuestro enfoque se ha centrado en cubrir las necesidades del personal administrativo, docente y de las familias, garantizando una solución inclusiva y versátil.

Captura de pantalla de pantalla de bienvenida

2. MOTIVACIÓN PERSONAL

La motivación para desarrollar UmeEgunero surge de nuestra propia experiencia vinculada con el sector de la educación infantil, y como tenemos esa falta de soluciones tecnológicas adaptadas a nuestras necesidades específicas. La mayoría de las aplicaciones existentes en el mercado están orientadas a niveles educativos superiores o bien son generalistas, dejando sin cubrir las particularidades de la etapa preescolar.

3. JUSTIFICACIÓN DEL PROYECTO

El desarrollo de UmeEgunero está justificado por varios factores clave:

Necesidad de digitalización educativa: Los centros educativos requieren herramientas digitales completas que mejoren sus procesos de gestión y comunicación. Según un estudio reciente del Ministerio de Educación, solo el 45% de los centros educativos en España utilizan plataformas digitales integradas, y de estos, menos del 20% considera que cubren todas sus necesidades.

Fragmentación de soluciones existentes: Muchos centros utilizan múltiples aplicaciones para cubrir diferentes necesidades, lo que genera ineficiencias y confusión. Nuestro análisis de campo mostró que, en promedio, un centro educativo utiliza entre 3 y 5 plataformas diferentes para gestionar todos sus procesos.

Demanda de experiencias de usuario modernas: Los usuarios esperan aplicaciones con interfaces atractivas, intuitivas y que aprovechen las capacidades de los dispositivos actuales. Las entrevistas con familias y profesorado revelaron una alta insatisfacción con las interfaces anticuadas de muchas soluciones existentes.

Oportunidad tecnológica: Las nuevas tecnologías como Jetpack Compose y Firebase permiten crear soluciones más potentes y mantenibles que en el pasado. El lanzamiento oficial de Jetpack Compose en julio de 2021 ha consolidado esta herramienta como el estándar emergente para desarrollo de interfaces en Android.

Accesibilidad y conectividad: La adopción generalizada de smartphones hace que una aplicación móvil sea el medio ideal para conectar a todos los miembros de la comunidad educativa. Según datos del INE, más del 98% de las familias con hijos en edad escolar dispone de al menos un smartphone en el hogar

OBJETIVOS GENERALES Y ESPECÍFICOS

1. OBJETIVO GENERAL

La motivación principal de este Trabajo de Fin de Grado es *transformar la gestión educativa en centros de educación infantil mediante la tecnología móvil*. Presentamos el desarrollo de una aplicación nativa para Android que establece un puente digital entre educadores y familias, facilitando una comunicación bidireccional significativa y en tiempo real.

Nuestra intención es *superar las limitaciones del modelo tradicional basado en papel*, automatizando tareas administrativas rutinarias y dedicando ese tiempo valioso al

seguimiento personalizado del desarrollo de los niños. Para ello, hemos implementado herramientas visuales intuitivas que permiten a todos los usuarios comprender el progreso educativo de manera sencilla y accesible.La aplicación se ha construido con una interfaz moderna y accesible, aprovechando las capacidades de Jetpack Compose y Firebase para garantizar una experiencia fluida en dispositivos Android.

Estamos convencidos de que esta solución tecnológica tiene el potencial de revolucionar la educación infantil, dejando atrás métodos tradicionales ineficientes y sentando las bases para un futuro educativo más conectado, transparente y colaborativo.

2. OBJETIVOS ESPECÍFICOS

- **Crear un entorno digital inclusivo para toda la comunidad educativa,** diseñando un sistema con cuatro roles (administradores, directores, profesores y familias) donde cada persona pueda acceder a herramientas que realmente faciliten su día a día según sus necesidades particulares.
- **Simplificar la burocracia escolar** mediante un módulo administrativo intuitivo que permita a los centros configurar sus espacios, organizar grupos, asignar profesorado y gestionar matrículas sin las habituales montañas de papeles y procesos redundantes.
- **Facilitar el seguimiento diario de los niños y niñas,** con un sistema que permita a los educadores registrar de forma sencilla y rápida aspectos como comidas, descanso, higiene y actividades, liberando tiempo para dedicarlo a lo que realmente importa: la atención personalizada.
- **Tender puentes de comunicación entre escuela y familia, implementando un canal bidireccional** donde profesores y familiares puedan compartir información, fotos y videos de forma segura, acercando el día a día del aula a los hogares.
- **Hacer la tecnología accesible para todos,** diseñando interfaces amigables con Jetpack Compose que funcionen bien en cualquier dispositivo Android, independientemente de la experiencia tecnológica del usuario.
- **Proteger la privacidad de la comunidad educativa,** integrando un sistema de autenticación con Firebase que garantice que cada persona solo acceda a la información que le corresponde, respetando la confidencialidad de datos sensibles.
- **Construir una base de datos que crezca con el centro,** desarrollando una estructura en Firestore que permita almacenar eficientemente la información del día a día sin perder rendimiento a medida que se acumula el historial educativo.

- **Asegurar que la app funcione incluso sin internet,** implementando sincronización offline para que los profesores puedan seguir registrando información en zonas con mala cobertura o durante caídas de red.
- **Facilitar el seguimiento del desarrollo infantil,** creando un sistema de informes digitales periódicos que muestren el progreso de cada niño de forma visual y comprensible para las familias.
- **Mejorar continuamente basándonos en experiencias reales,** evaluando cómo usan la app los profesores y familias en su día a día, escuchando sus sugerencias y midiendo qué funciones realmente les ayudan en su labor educativa.

3. ALCANCE DEL PROYECTO

Delimitación del ámbito

En el desarrollo de UmeEgunero, hemos establecido un alcance claramente definido que nos permite abordar los objetivos propuestos de manera efectiva, considerando las limitaciones temporales y recursos disponibles para un Trabajo de Fin de Grado de Desarrollo de Aplicaciones Multiplataforma.

Ámbito funcional

El desarrollo de UmeEgunero contempla las siguientes funcionalidades principales:

- **Registro y autenticación de usuarios** con diferentes roles (administrador, educador, familiar), implementando un sistema seguro basado en Firebase Authentication.
- **Gestión completa de perfiles**, permitiendo a los usuarios actualizar su información personal, preferencias de notificación y datos de contacto.
- **Sistema de registro de actividades diarias** que permita a los educadores documentar diferentes aspectos:
  - Alimentación (ingesta, cantidades, preferencias)
  - Descanso (horarios, duración, calidad)
  - Actividades educativas (ejercicios realizados, participación, logros)
  - Estado emocional y bienestar (comportamiento, socialización)
  - Eventos destacables (momentos especiales, incidencias

- **Galería multimedia** para que los educadores puedan compartir imágenes y vídeos de actividades realizadas en el aula, protegiendo la privacidad mediante un sistema de consentimientos digitales.
- **Sistema de comunicación bidireccional** entre educadores y familias, incluyendo:
    - Mensajería instantánea contextual (vinculada a estudiantes específicos.
    - Envío de archivos y documentos
    - Solicitud y gestión de tutorías
    - Avisos importantes con confirmación de lectura

- **Panel de información general del centro educativo**, donde se puedan publicar:
    - Calendario de eventos y actividades
    - Circulares y comunicados oficiales
    - Menús semanales/mensuales
    - Noticias relevantes

- **Sistema de notificaciones** en tiempo real mediante Firebase Cloud Messaging para mantener informados a los usuarios sobre nuevas actividades registradas, mensajes recibidos o avisos importantes.
- **Funcionalidad offline** que permita registrar información sin conectividad y sincronizarla automáticamente cuando se restablezca la conexión.

Ámbito técnico

Desde la perspectiva técnica, el proyecto abarca:

- **Desarrollo completo de una aplicación nativa para Android,** optimizada para funcionar en dispositivos con Android 7.0 (API 24) o superior, abarcando así más del 95% de los dispositivos Android activos.
- **Implementación de arquitectura Clean y patrón MVVM**, siguiendo las recomendaciones de Android Architecture Components para garantizar un código mantenible, testeable y escalable.
- **Desarrollo de una interfaz de usuario moderna con Jetpack Compose**, priorizando usabilidad, accesibilidad y experiencia de usuario, con soporte para distintos tamaños de pantalla y orientaciones.
- **Integración completa con servicios de Firebase** como:

  - Authentication para gestión de usuarios
  - Firestore para almacenamiento y sincronización de datos
  - Storage para gestión de archivos multimedia
  - Cloud Messaging para notificaciones push
  - Analytics para análisis de uso

- **Implementación de mecanismos de persistencia local** utilizando Room para garantizar funcionalidad offline y mejorar el rendimiento de la aplicación.
- **Optimización para distintos entornos de conectividad**, adaptando el comportamiento de la aplicación según la calidad de la conexión disponible y procurando un consumo eficiente de datos.
- **Desarrollo de pruebas unitarias, de integración y de interfaz** para verificar el correcto funcionamiento de los componentes críticos de la aplicación.

Aspectos excluidos del alcance

Para mantener el proyecto dentro de los límites razonables de un TFG, hemos decidido excluir conscientemente los siguientes aspectos:

- **Desarrollo multiplataforma**: Aunque la formación de 2º DAM contempla el desarrollo multiplataforma, hemos optado por centrar nuestros esfuerzos en crear una solución nativa para Android, excluyendo el desarrollo específico para iOS o web. No obstante, la arquitectura implementada facilitaría una futura expansión.
- **Integración con sistemas de gestión educativa externos**: Si bien reconocemos el valor de integrar UmeEgunero con sistemas como ITACA u otras plataformas de gestión educativa, estas integraciones quedan fuera del alcance actual debido a la complejidad que añadirían al proyecto.
- **Módulo de evaluación académica completa**: Aunque la aplicación permite registrar actividades y progresos educativos, un sistema completo de evaluación con rúbricas, informes académicos detallados y seguimiento curricular queda fuera del alcance inicial.
- **Procesamiento avanzado de datos mediante IA**: El análisis predictivo o la generación automática de recomendaciones basadas en los datos recopilados, si bien son interesantes, quedan reservados para futuras expansiones del proyecto.

Metodología y planificación

Considerando la naturaleza del proyecto y el contexto académico de 2º DAM, hemos adoptado:

- **Metodología ágil Scrum,** adaptada al contexto de un proyecto individual pero manteniendo sus principios fundamentales: desarrollo incremental, revisión continua y adaptación.

- **Ciclos de desarrollo (sprints) de dos semanas**, con objetivos claramente definidos y entregables concretos al final de cada ciclo.

- **Herramientas de gestión de proyectos** como Jira para el seguimiento de tareas y GitHub para el control de versiones.

- **Proceso de validación continua** con usuarios potenciales (educadores y familias) para garantizar que el producto desarrollado satisface necesidades reales.

Relevancia académica para 2º DAM

El desarrollo de UmeEgunero como TFG para el ciclo formativo de Desarrollo de Aplicaciones Multiplataforma (DAM) resulta especialmente relevante por:

- Aplicación práctica de los módulos formativos cursados durante el ciclo, particularmente:
    - Programación de servicios y procesos
    - Acceso a datos
    - Programación multimedia y dispositivos móviles
    - Sistemas de gestión empresarial
    - Desarrollo de interfaces

- Integración de conocimientos de diversas áreas, desde el diseño de bases de datos hasta la implementación de interfaces, pasando por el desarrollo de lógica de negocio y la integración con servicios en la nube.
- Aplicación de metodologías ágiles en un entorno real, adaptándolas a las particularidades de un proyecto académico.
- Enfoque en resolver una necesidad real del ámbito educativo, demostrando la capacidad de análisis y comprensión de requisitos de usuario.
- Demostración de competencias técnicas avanzadas como el manejo de arquitecturas modernas, implementación de patrones de diseño y desarrollo de código de calidad.

Este alcance define los límites concretos de nuestro trabajo, estableciendo un marco realista pero ambicioso que nos permite demostrar las competencias adquiridas durante la formación de DAM, al tiempo que desarrollamos una solución útil y completa para un problema real del ámbito educativo.

REQUISITOS DEL SISTEMA

1. REQUISITOS FUNCIONALES

1. Gestión de usuarios multirrol
   - La aplicación debe permitir el registro e inicio de sesión de usuarios con diferentes roles (administradores, directores de centro, profesores y familiares).
   - Cada rol debe tener acceso a funcionalidades específicas adaptadas a sus necesidades.
2. Gestión administrativa de centros educativos
   - Configuración y administración de centros educativos
   - Organización de aulas y grupos
   - Asignación de profesores a clases
   - Matriculación de alumnos
3. Registro diario de actividades
   - Los educadores deben poder documentar aspectos como alimentación, sueño, higiene y comportamiento.
   - El sistema debe permitir adjuntar evidencias multimedia (fotos).
   - Los registros deben ser accesibles tanto para profesores como para familiares.
4. Sistema de comunicación bidireccional
   - Mensajería directa entre profesores y familias
   - Envío y recepción de comunicados oficiales con confirmación de lectura
   - Notificaciones personalizables
   - Compartición de recursos multimedia
5. Calendario y gestión de eventos
   - Programación y visualización de eventos del centro
   - Gestión de reuniones profesor-familia
   - Recordatorios automáticos
6. Funcionalidades offline

- Capacidad para trabajar sin conexión a internet
- Sincronización automática cuando se restablezca la conexión
- Resolución de conflictos de datos

7. Gestión de archivos multimedia
   - Carga y visualización de imágenes y documentos
   - Almacenamiento seguro de archivos en la nube
   - Compresión de imágenes para optimizar el espacio
8. Sistema de notificaciones
   - Envío de notificaciones push para eventos importantes
   - Personalización de preferencias de notificación por usuario

2. REQUISITOS NO FUNCIONALES

1. Rendimiento
   - Tiempos de respuesta inferiores a 2 segundos para operaciones habituales
   - Optimización para dispositivos de gama media-baja
   - Consumo eficiente de recursos (batería, datos, memoria)
2. Seguridad
   - Autenticación segura mediante Firebase Authentication
   - Protección de datos sensibles de menores
   - Control de acceso basado en roles
   - Encriptación de datos en tránsito y en reposo
3. Usabilidad
   - Interfaz intuitiva adaptada a diferentes perfiles de usuario
   - Diseño responsive para diferentes tamaños de pantalla
   - Soporte para modo oscuro
   - Accesibilidad para usuarios con diferentes capacidades
4. Disponibilidad
   - Funcionamiento 24/7 con mínimo tiempo de inactividad
   - Capacidad de trabajo offline para funciones críticas
5. Escalabilidad
   - Capacidad para gestionar múltiples centros educativos
   - Soporte para un número creciente de usuarios
   - Estructura de datos optimizada para consultas frecuentes
6. Mantenibilidad

- Arquitectura MVVM para facilitar pruebas y mantenimiento
- Código modular y bien documentado
- Separación clara de responsabilidades

METODOLOGÍA DE DESARROLLO

1. JUSTIFICACIÓN DE LA ELECCIÓN DE SCRUM

En el desarrollo de UmeEgunero, optamos por implementar la metodología ágil Scrum debido a varios factores determinantes que se alineaban perfectamente con las necesidades y características de nuestro proyecto:

Adaptabilidad al Cambio

El desarrollo de una aplicación de gestión educativa para centros preescolares requería una alta capacidad de adaptación a requisitos cambiantes. Scrum nos proporcionó la flexibilidad necesaria para incorporar nuevas funcionalidades y ajustar prioridades a medida que profundizábamos en las necesidades de los usuarios finales (administradores, profesores y familiares).

Entrega Incremental de Valor

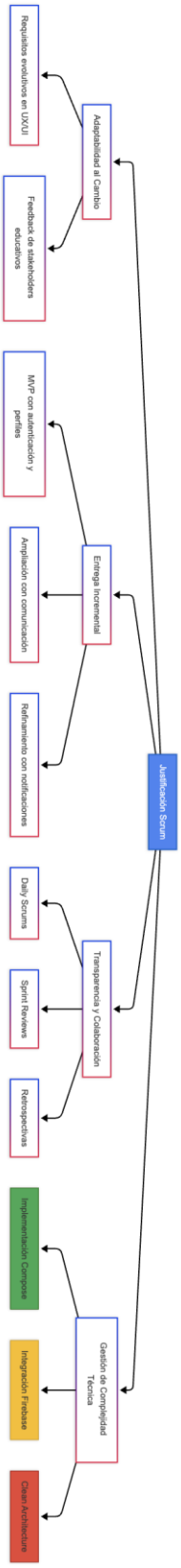
La organización en sprints de 14 días nos permitió entregar incrementos funcionales del producto de forma regular. Esto resultó especialmente valioso para validar tempranamente características críticas como la autenticación multi-rol, el sistema de comunicación unificada y las vinculaciones entre usuarios.

Transparencia y Colaboración

El framework Scrum facilitó una comunicación fluida entre los distintos stakeholders del proyecto, permitiendo reajustar la dirección del desarrollo basándonos en feedback constante. Esto fue particularmente relevante al tratarse de una aplicación con múltiples perfiles de usuario y casos de uso complejos.

Gestión Eficiente de la Complejidad Técnica

La implementación de tecnologías modernas como Jetpack Compose, Firebase y arquitectura MVVM requería un enfoque que permitiera abordar desafíos técnicos de forma iterativa. Los eventos diarios y las retrospectivas nos ayudaron a identificar y resolver obstáculos técnicos rápidamente.



Esquema de justificación Scrum

2. CICLO DE DESARROLLO IMPLEMENTADO

En UmeEgunero, adaptamos el ciclo de desarrollo Scrum a las necesidades específicas de nuestra aplicación Android basada en Jetpack Compose y Firebase, estableciendo un proceso iterativo que abarcó desde finales de enero hasta su finalización:

Planificación Inicial y Product Backlog

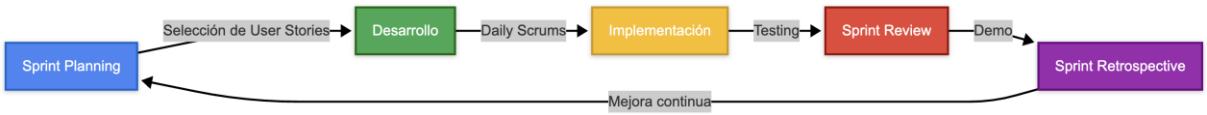
Iniciamos el proyecto a finales de enero de 2025 con la creación del Product Backlog, priorizando las historias de usuario según su valor para los distintos perfiles (administradores, profesores y familiares). Las funcionalidades se organizaron por módulos, siguiendo un enfoque de arquitectura limpia que permitiera escalabilidad y mantenibilidad.

Sprints de 14 Días

Establecimos ciclos de desarrollo de 2 semanas, comenzando oficialmente en febrero:
Sprint 1 (11-25 enero): Configuración del proyecto con Jetpack Compose, estructura MVVM, integración de Firebase y almacenamiento local.
Sprint 2 (26 enero-8 febrero): Autenticación con Firebase, pantallas de login/registro y gestión de roles.
Sprint 3 (9-22 febrero): Sistema de navegación, dashboards por roles y componentes UI reutilizables.
Sprint 4 (23 febrero-7 marzo): Implementación del sistema de comunicación unificado.
Sprint 5 (8-21 marzo): Desarrollo del calendario y gestión de eventos.
Sprint 6 (22 marzo-4 abril): Sistema de solicitudes y vinculaciones.
Sprint 7 (5-18 abril): Registro diario y seguimiento académico.

Ceremonias Implementadas

Para cada sprint seguimos este ciclo:



Metodologia Sprint desarrollada

Desarrollo Basado en Características

Dentro de cada sprint, abordamos el desarrollo siguiendo un enfoque modular por características:

- **Definición de interfaces de datos:** Comenzando con los modelos de dominio en Kotlin.
- **Implementación de repositorios:** Conexión con Firebase (Firestore, Authentication, Storage, FCM).
- **Desarrollo de ViewModels:** Lógica de presentación con StateFlow y Coroutines.
- **Creación de UI en Compose**: Construcción de interfaces declarativas siguiendo Material 3.
- **Testing y refinamiento**: Pruebas unitarias e instrumentadas.

Integración Continua

Implementamos un flujo de integración continua para garantizar la calidad del código, mediante:

- Control de versiones con Git
- Análisis estático de código con Detekt
- Testing automatizado
- Revisión de código entre pares

3. HERRAMIENTAS Y GESTIÓN DEL PROYECTO

El desarrollo de UmeEgunero requirió la selección y configuración de diversas herramientas tecnológicas que facilitaron tanto el desarrollo como la gestión del proyecto.

Entorno de Desarrollo

- **Android Studio Hedgehog (2023.1.1)**: IDE principal para el desarrollo, proporcionando soporte para Kotlin, Jetpack Compose y las últimas APIs de Android.
- **Kotlin 1.9.22:** Lenguaje de programación principal, aprovechando sus características modernas como funciones de extensión, corrutinas y flow.
- **Jetpack Compose 1.5.4**: Framework declarativo para la construcción de interfaces de usuario modernas con componentes interactivos y feedback háptico.
- **Material 3**: Sistema de diseño que proporcionó una base consistente para la UI, con soporte para temas dinámicos y componentes accesibles.

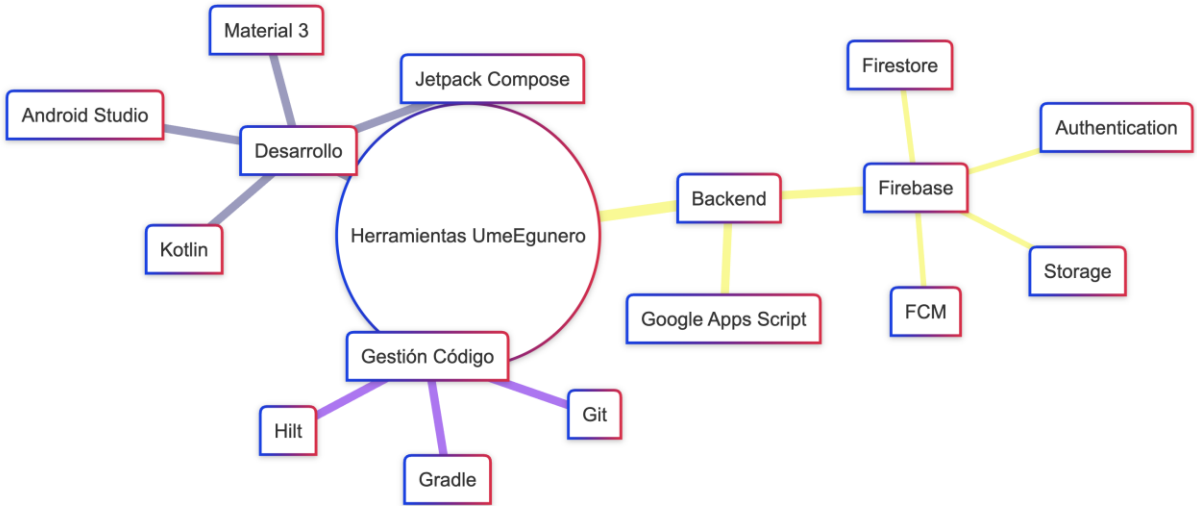
Infraestructura y Backend

- **Firebase Suite**: Plataforma que sirvió como backend principal:

  - Firestore: Base de datos NoSQL para almacenamiento de datos en tiempo real.
  - Firebase Authentication: Gestión de usuarios y autenticación segura.
  - Cloud Storage: Almacenamiento de archivos y documentos.
  - Firebase Cloud Messaging (FCM): Sistema de notificaciones push.
- **Google Apps Script:** Utilizado como backend ligero para operaciones administrativas y envío de correos electrónicos.

Gestión de Dependencias y Código

- **Gradle (Kotlin DSL):** Sistema de build automatizado con scripts en Kotlin.
- **Hilt (2.48):** Framework de inyección de dependencias que simplificó la arquitectura y facilitó el testing.
- **Git:** Control de versiones, con una estrategia de ramificación basada en features.



Arquitectura del Proyecto

Para garantizar la escalabilidad y mantenibilidad, organizamos el proyecto siguiendo principios de Clean Architecture y MVVM:

```
// Ejemplo de ViewModelFactory con Hilt para inyección de dependencias
@Module
@InstallIn(ViewModelComponent::class)
object ViewModelModule {
    @Provides
    fun provideAuthRepository(
        firebaseAuth: FirebaseAuth,
        firestore: FirebaseFirestore
```

```
    ): AuthRepository {
        return AuthRepositoryImpl(firebaseAuth, firestore)
    }
}

// Ejemplo de ViewModel para gestión de autenticación
@HiltViewModel
class AuthViewModel @Inject constructor(
    private val authRepository: AuthRepository
) : ViewModel() {
    private val _authState = MutableStateFlow<AuthState>(AuthState.Idle)
    val authState: StateFlow<AuthState> = _authState.asStateFlow()

    fun login(email: String, password: String) {
        viewModelScope.launch {
            _authState.value = AuthState.Loading
            try {
                val result = authRepository.login(email, password)
                _authState.value = AuthState.Success(result)
            } catch (e: Exception) {
                _authState.value = AuthState.Error(e.message ?: "Error
desconocido")
            }
        }
    }
}

// Ejemplo de Compose UI conectada al ViewModel
@Composable
fun LoginScreen(
    viewModel: AuthViewModel = hiltViewModel(),
    onNavigateToHome: () -> Unit
) {
    val authState by viewModel.authState.collectAsState()

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        // UI components...

        when(authState) {
            is AuthState.Loading -> CircularProgressIndicator()
            is AuthState.Success -> LaunchedEffect(Unit) { onNavigateToHome() }
            is AuthState.Error -> Text(
                text = (authState as AuthState.Error).message,
                color = MaterialTheme.colorScheme.error
            )
            else -> { /* Idle state */ }
        }
    }
}
```

Métricas y Seguimiento

Durante el desarrollo, utilizamos diversas métricas para evaluar el progreso y la calidad del proyecto:

- Velocidad del equipo: Medición de story points completados por sprint.
- Cobertura de código: Garantizando un mínimo del 80% en clases críticas.
- Tiempo de build: Optimización del proceso de compilación y test.
- Crashes reportados: Monitorización a través de Firebase Crashlytics.
- Feedback de usuarios: Evaluación cualitativa de la experiencia de usuario.

Estas herramientas y prácticas nos permitieron mantener un desarrollo eficiente y orientado a la calidad, cumpliendo con los plazos establecidos en el plan de desarrollo desde febrero hasta su finalización.

ANÁLISIS DEL SISTEMA

1. GRÁFICO DIAGRAMAS DE CASOS DE USO

Administrador de aplicación

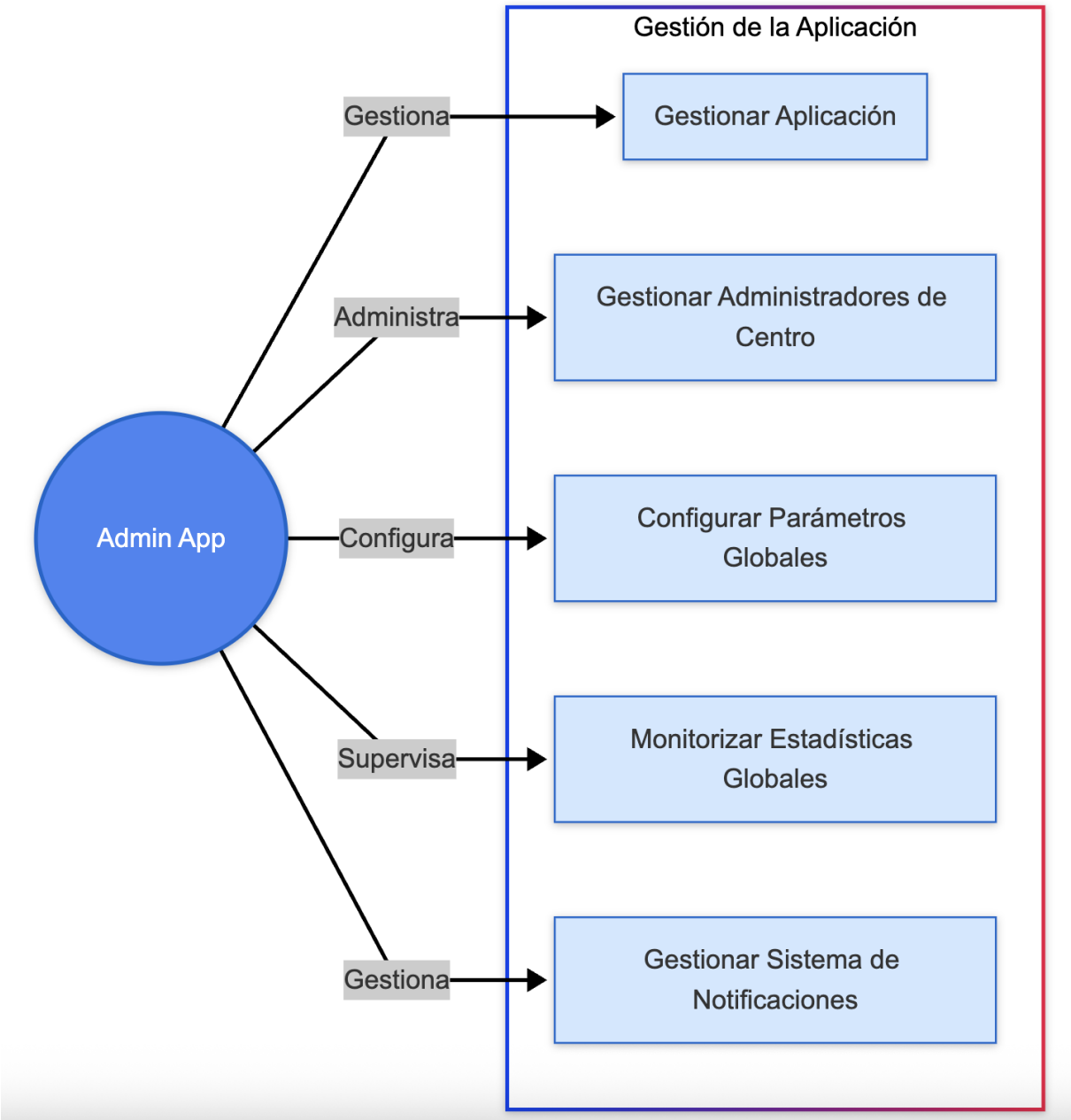


Figura 1: Diagrama de casos de uso del Administrador de Aplicación, responsable de la gestión global de la plataforma UmeEgunero

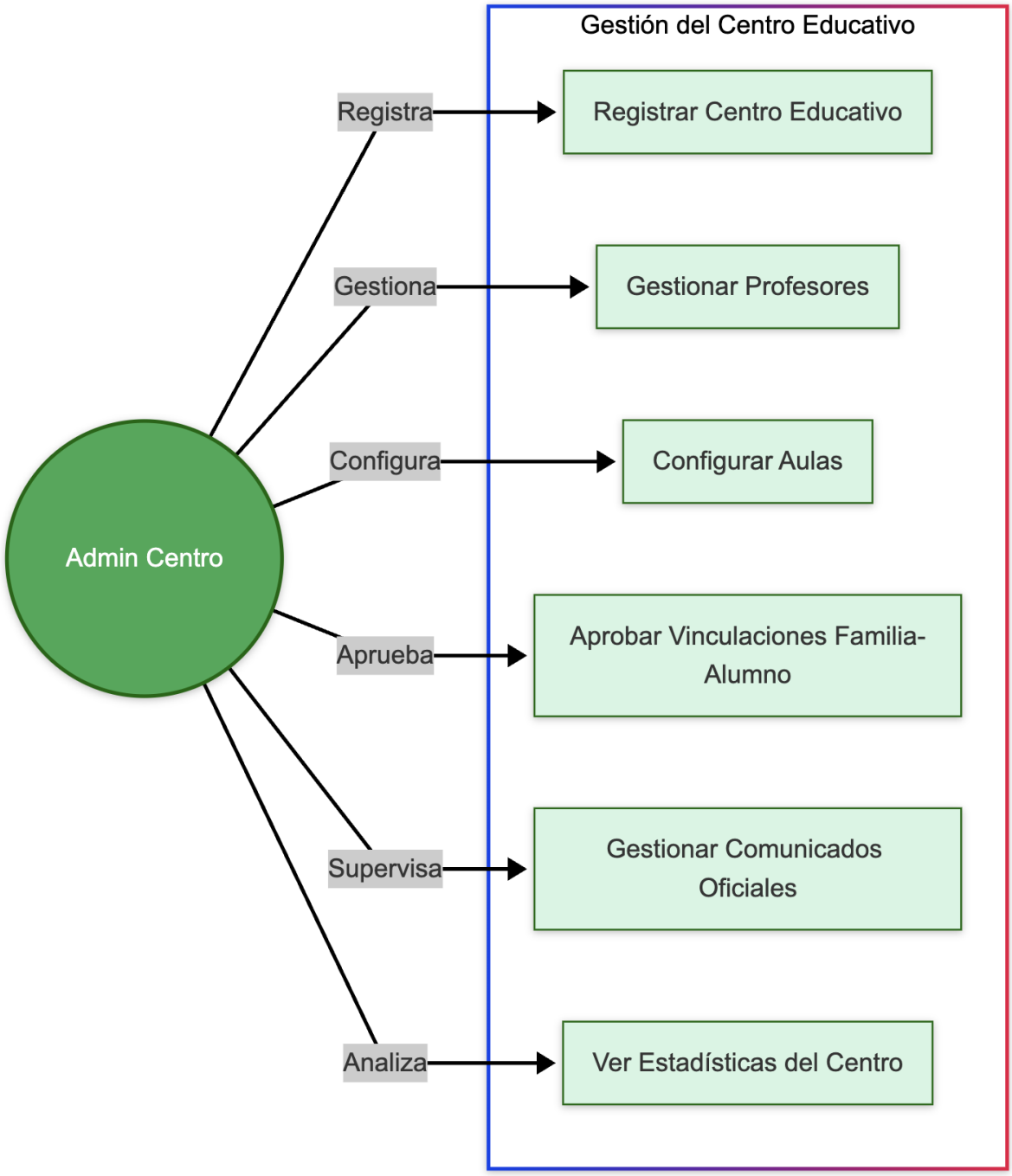


Figura 2: Diagrama de casos de uso del Administrador de Centro, encargado de la gestión administrativa de un centro educativo específico

Profesor

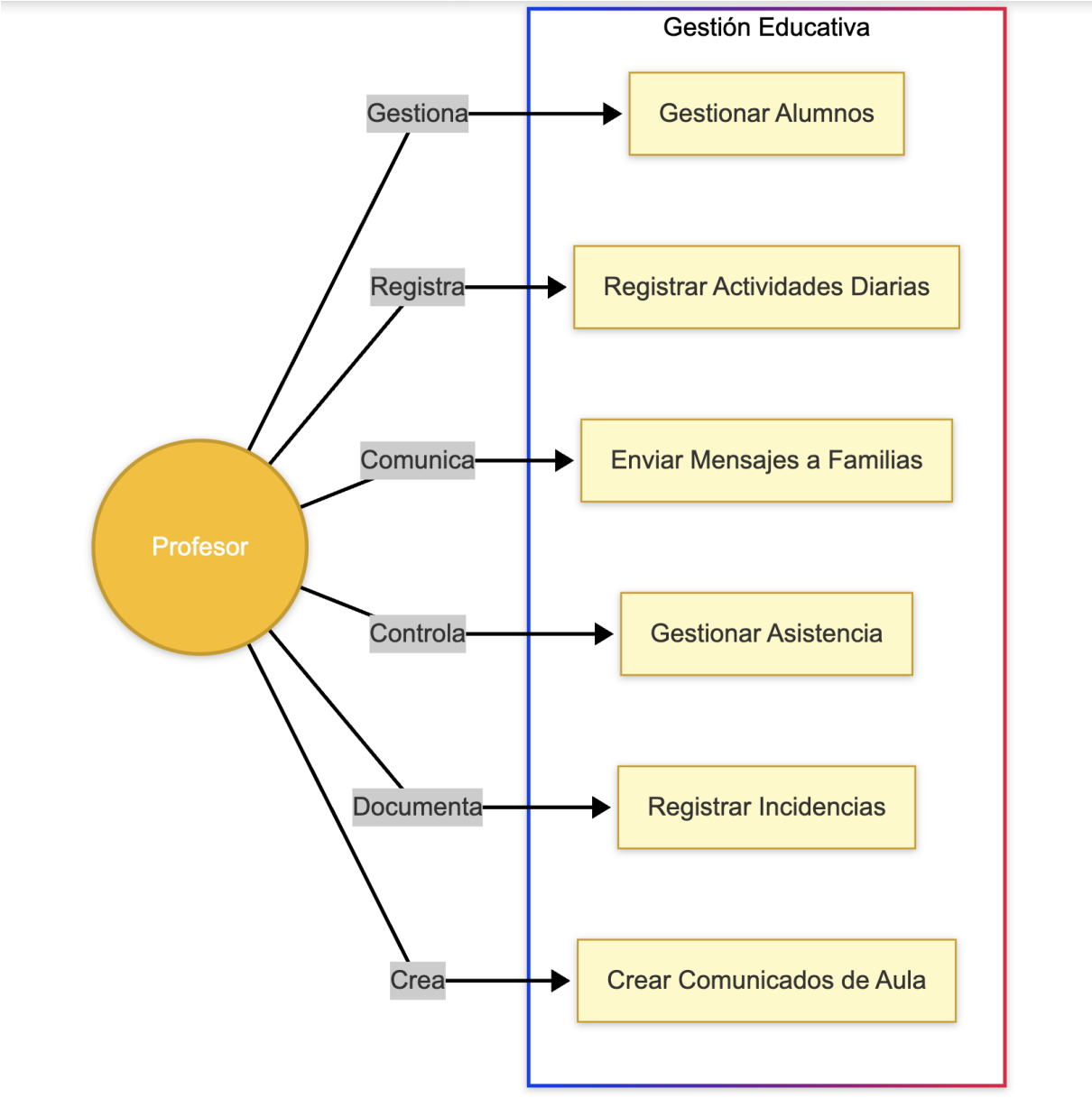


Figura 3: Diagrama de casos de uso del Profesor, responsable de la gestión educativa diaria y la comunicación con las familias

Familiar

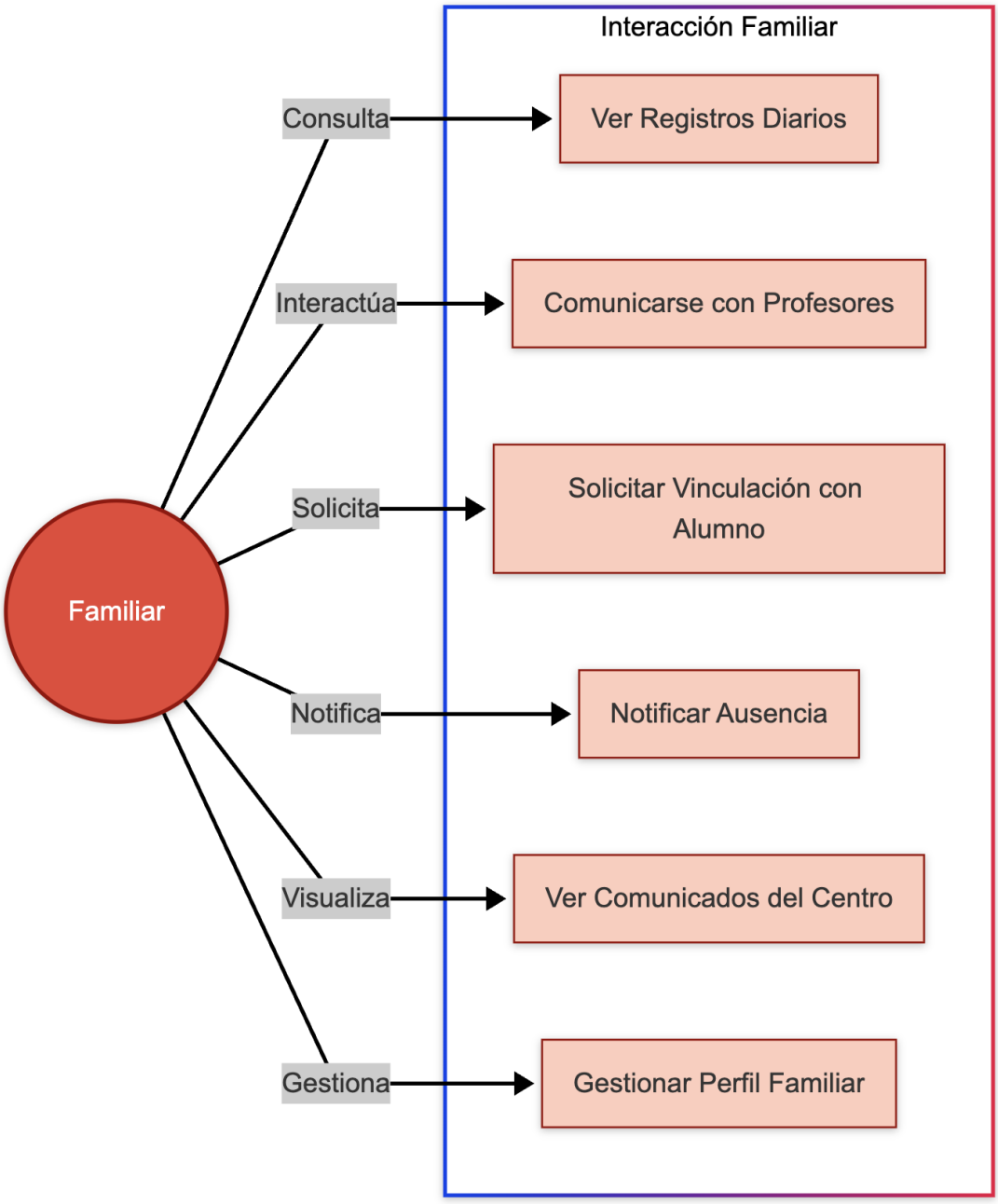


Figura 4: Diagrama de casos de uso del Familiar, orientado al seguimiento educativo del alumno y la comunicación con el centro educativo

2. GRÁFICO DIAGRAMA DE ACTIVIDADES

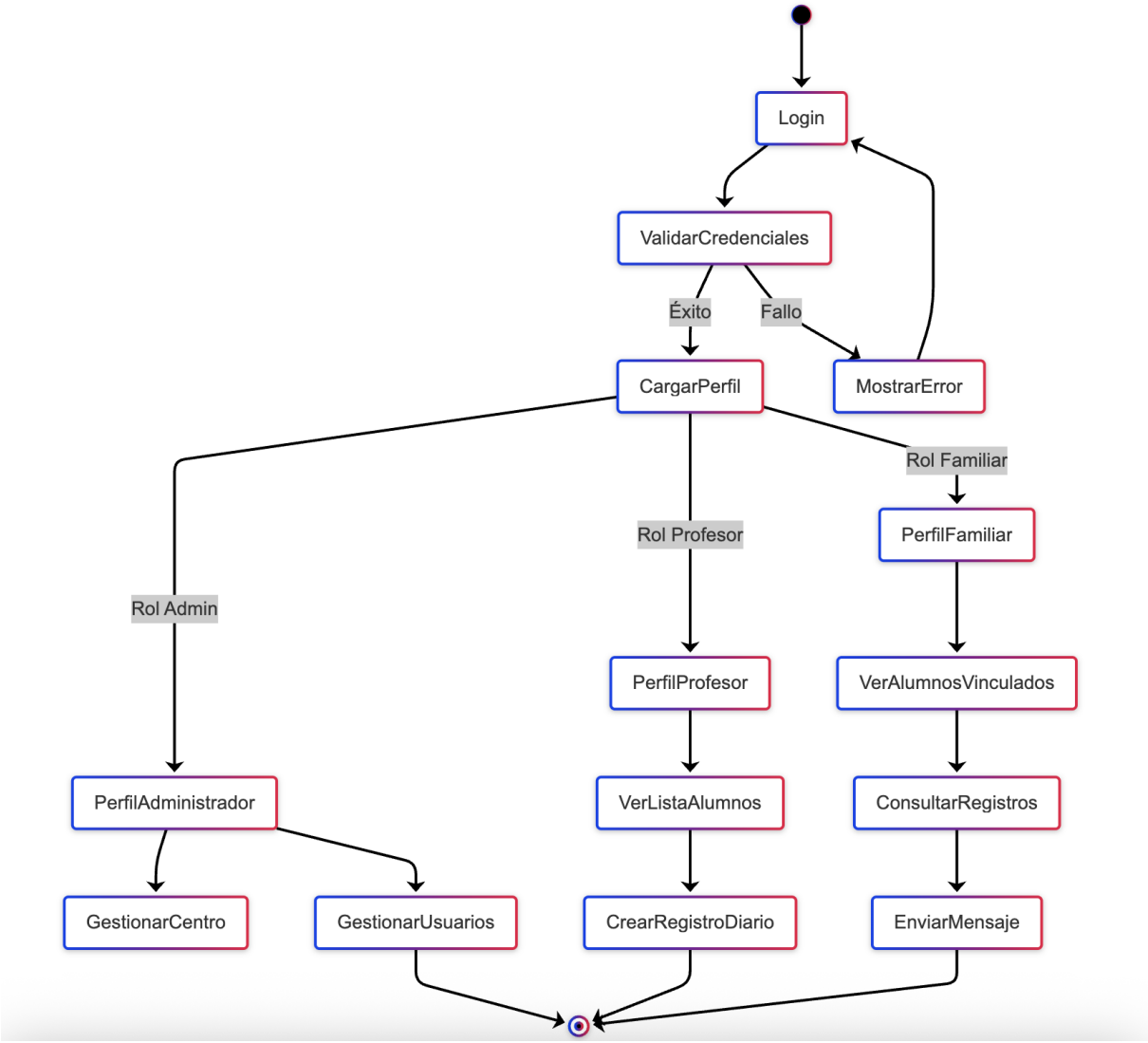


Figura 2: Diagrama de actividades del flujo principal de la aplicación desde el inicio de sesión hasta las acciones específicas de cada perfil

3. GRÁFICO DIAGRAMA DE FLUJO DE NAVEGACIÓN

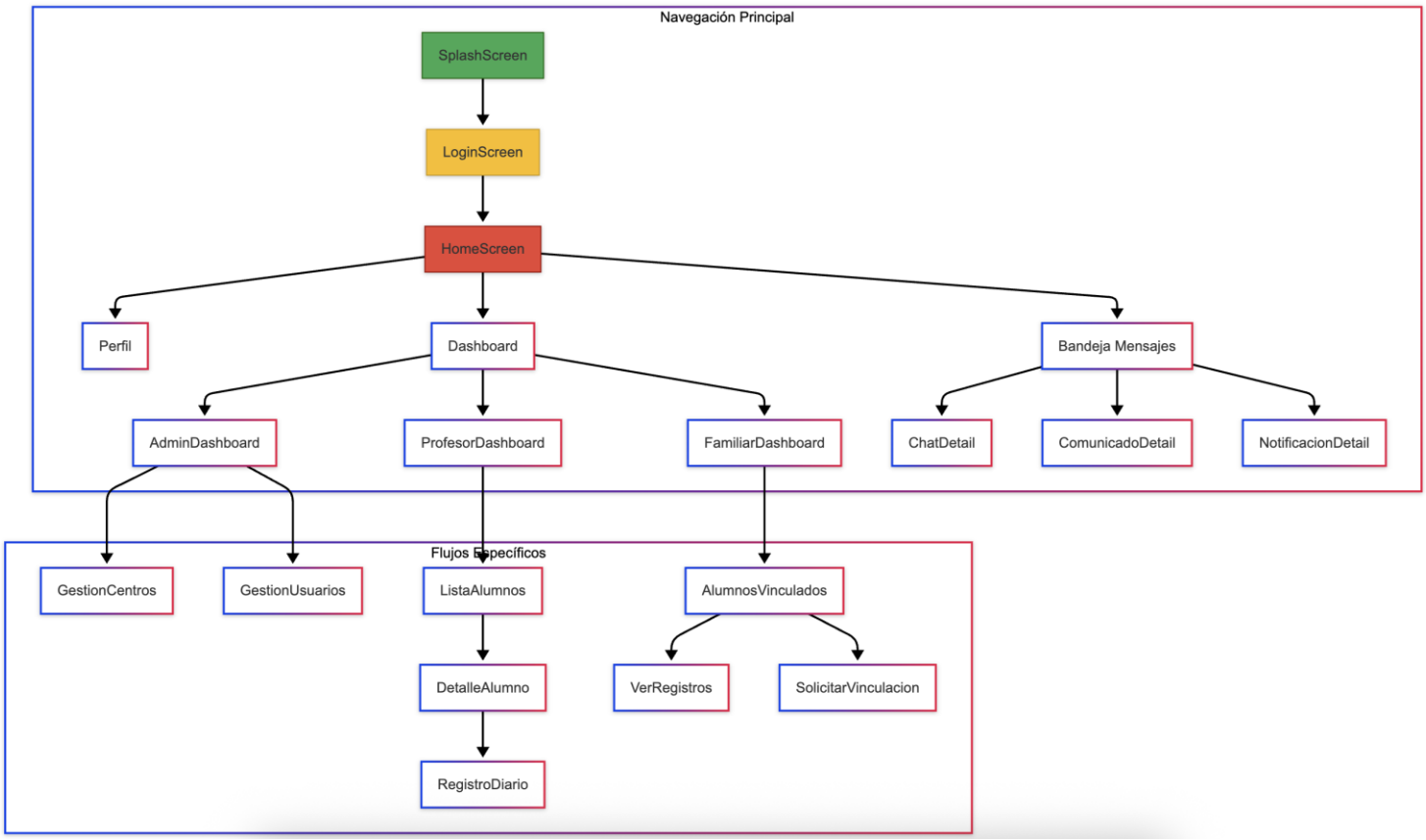


Figura 3: Diagrama de flujo de navegación entre pantallas principales de UmeEgunero, mostrando las rutas de navegación disponibles para los diferentes perfiles

4. GRÁFICO DIAGRAMA DE ESTADOS

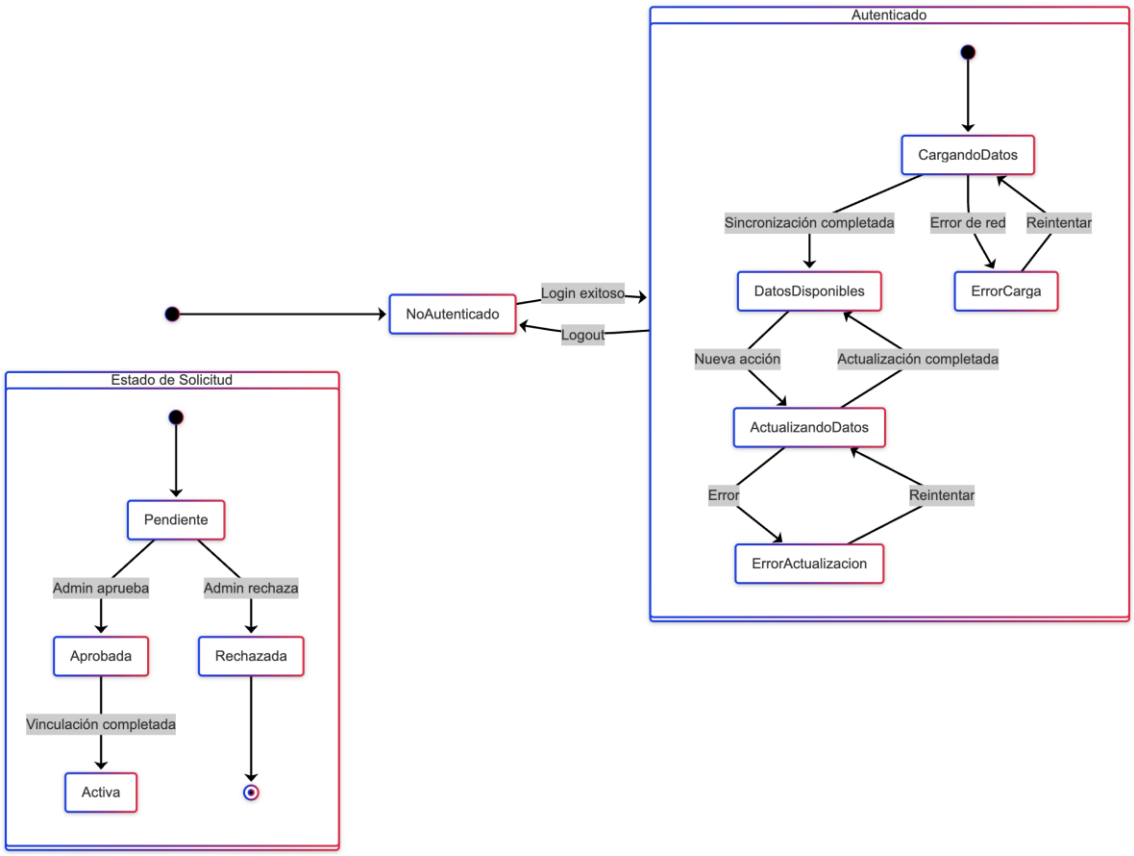


Figura 4: Diagrama de estados que muestra las transiciones entre los diferentes estados de la aplicación, incluyendo la autenticación y el proceso de solicitudes de vinculación

5. RESUMEN DE LOS DIAGRAMAS UML PARA UMEEGUNERO

Diagramas de Casos de Uso

Los diagramas de casos de uso muestran las principales funcionalidades de UmeEgunero segmentadas por perfiles de usuario. El Administrador de Aplicación gestiona aspectos globales como la configuración de parámetros y administradores de centro. El Administrador de Centro se encarga de registrar centros educativos, gestionar profesores y aprobar vinculaciones. Los Profesores gestionan alumnos, registran actividades diarias y controlan la asistencia. Por último, los Familiares consultan registros, se comunican con profesores y gestionan vinculaciones con alumnos. Cada diagrama delimita claramente el ámbito de responsabilidad de cada perfil.

Diagrama de Actividades

El diagrama de actividades ilustra el flujo principal de la aplicación, comenzando con el login y validación de credenciales. Dependiendo del perfil autenticado (administrador, profesor o familiar), se cargan diferentes interfaces y opciones. Para administradores, se muestran opciones de gestión de centros y usuarios; para profesores, la vista de alumnos y creación de registros diarios; y para familiares, la consulta de alumnos vinculados y sus registros. El diagrama muestra cómo estas actividades se interconectan para formar el flujo operativo de la aplicación.

Diagrama de Flujo de Navegación

Este diagrama representa la estructura de navegación entre pantallas de la aplicación. Desde la pantalla de inicio (SplashScreen) hasta el Login y posteriormente al HomeScreen, que actúa como punto central de navegación. Desde allí, los usuarios acceden a secciones específicas según su perfil: dashboards personalizados, bandeja de mensajes y perfiles. Se muestran también los flujos específicos para cada perfil, como la gestión de centros para administradores, gestión de alumnos para profesores y visualización de registros para familiares.

Diagrama de Estados

El diagrama de estados muestra las transiciones entre diferentes estados de la aplicación, comenzando con la autenticación (NoAutenticado → Autenticado). Dentro del estado Autenticado, se representan los procesos de carga y actualización de datos, incluyendo manejo de errores. Adicionalmente, se detalla el estado de las solicitudes de vinculación, desde su creación (Pendiente) hasta su resolución (Aprobada/Rechazada) y activación. Este diagrama es fundamental para entender cómo la aplicación maneja los cambios de estado y las transiciones entre diferentes procesos.

DISEÑO DE LA APLICACIÓN

1. ESTRUCTURA DEL PROYECTO

La estructura del proyecto UmeEgunero sigue las mejores prácticas de desarrollo en Android, organizando el código en módulos funcionales que facilitan el mantenimiento y la escalabilidad.

Organización por Características

```
app/
├── src/
│   ├── main/
│   │   ├── java/com/tfg/umeegunero/
│   │   │   ├── data/                # Capa de datos
│   │   │   ├── di/                  # Inyección de dependencias
│   │   │   ├── feature/             # Funcionalidades organizadas por módulos
│   │   │   ├── navigation/          # Sistema de navegación
│   │   │   ├── notification/        # Sistema de notificaciones
│   │   │   ├── ui/                  # Componentes UI compartidos
│   │   │   └── util/                # Utilidades y extensiones
```

Esta estructura permite la separación clara de responsabilidades y facilita la colaboración en equipos de desarrollo. El enfoque modular por características (feature) mejora la mantenibilidad y testabilidad del código.

Ejemplo de Configuración Gradle

```
// app/build.gradle.kts
plugins {
    id("com.android.application")
    id("org.jetbrains.kotlin.android")
    id("com.google.dagger.hilt.android")
    id("com.google.devtools.ksp")
    id("com.google.gms.google-services")
    kotlin("kapt")
}

android {
    namespace = "com.tfg.umeegunero"
    compileSdk = 34

    defaultConfig {
        applicationId = "com.tfg.umeegunero"
        minSdk = 26
        targetSdk = 34
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }

    buildFeatures {
        compose = true
    }

    composeOptions {
        kotlinCompilerExtensionVersion = "1.5.4"
    }
}

dependencies {
    // Core & Compose
    implementation("androidx.core:core-ktx:1.12.0")
```

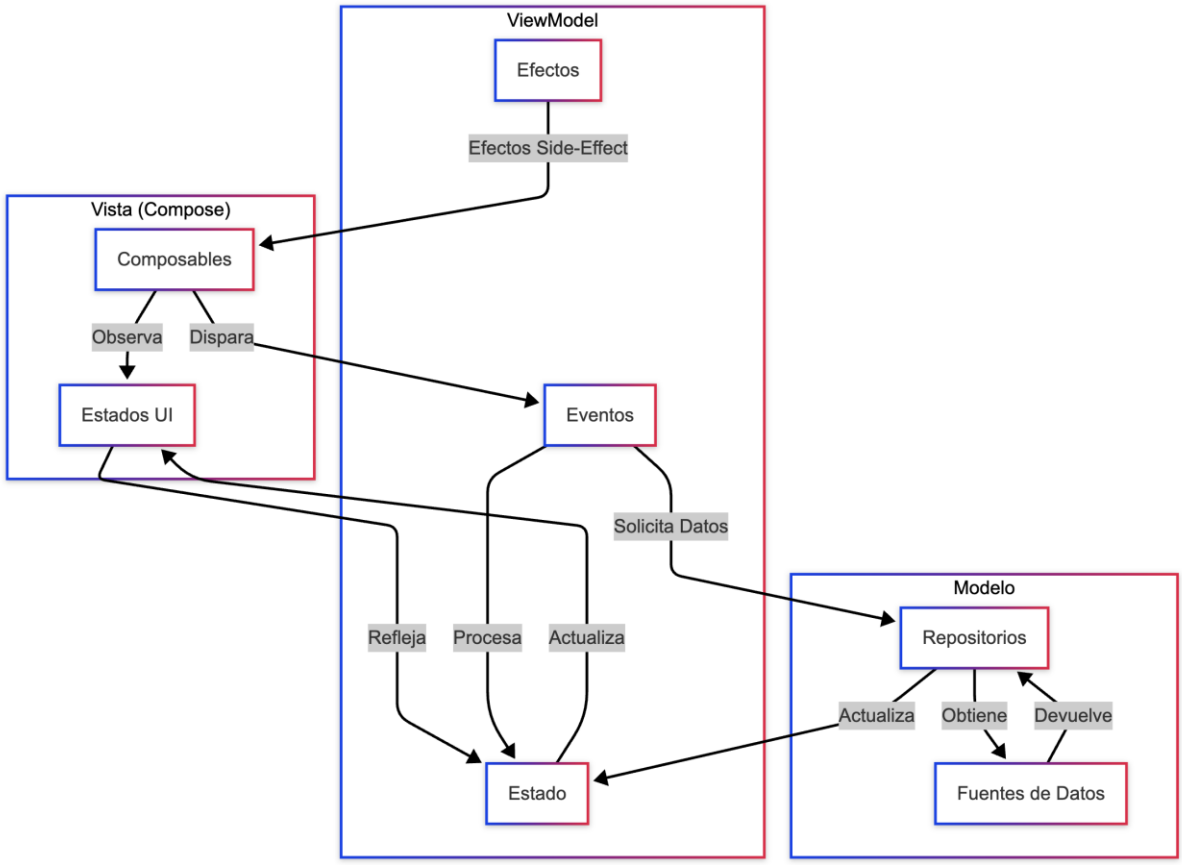
```
    implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.7.0")
    implementation("androidx.activity:activity-compose:1.8.2")
    implementation(platform("androidx.compose:compose-bom:2023.08.00"))
    implementation("androidx.compose.ui:ui")
    implementation("androidx.compose.ui:ui-graphics")
    implementation("androidx.compose.ui:ui-tooling-preview")
    implementation("androidx.compose.material3:material3")

    // Navigation
    implementation("androidx.navigation:navigation-compose:2.7.5")

    // Firebase
    implementation(platform("com.google.firebase:firebase-bom:32.5.0"))
    implementation("com.google.firebase:firebase-auth-ktx")
    implementation("com.google.firebase:firebase-firestore-ktx")
    implementation("com.google.firebase:firebase-storage-ktx")
    implementation("com.google.firebase:firebase-messaging-ktx")

    // Hilt
    implementation("com.google.dagger:hilt-android:2.48")
    kapt("com.google.dagger:hilt-android-compiler:2.48")
    implementation("androidx.hilt:hilt-navigation-compose:1.1.0")
}
```

2. ARQUITECTURA MVVM (MODEL-VIEW-VIEWMODEL)

Justificación de la Elección

La elección de MVVM (Model-View-ViewModel) para UmeEgunero responde a múltiples factores técnicos y prácticos:

- **Integración natural con Jetpack Compose:** MVVM trabaja perfectamente con el paradigma declarativo de Compose, permitiendo un flujo de datos unidireccional y predecible.
- **Separación de responsabilidades:** Facilita la clara división entre lógica de presentación, lógica de negocio y datos.
- **Testabilidad mejorada:** Los ViewModels pueden probarse independientemente de la UI, facilitando la implementación de tests unitarios.
- **Gestión del ciclo de vida:** Los ViewModels sobreviven a cambios de configuración, evitando la pérdida de estado durante rotaciones de pantalla u otros eventos del ciclo de vida.
- **Soporte oficial de Google:** Es el patrón recomendado dentro de Android Architecture Components.



Componentes Principales

Model (Datos y Lógica de Negocio)

```
// Modelo de datos
data class Usuario(
    val id: String = "",
    val email: String = "",
    val nombre: String = "",
    val apellidos: String = "",
    val telefono: String = "",
    val rol: RolUsuario = RolUsuario.FAMILIAR,
    val centroId: String = "",
    val createdAt: Long = System.currentTimeMillis(),
    val updatedAt: Long = System.currentTimeMillis()
)

enum class RolUsuario {
    ADMIN_APP,
    ADMIN_CENTRO,
    PROFESOR,
    FAMILIAR
}

// Repositorio
interface UsuarioRepository {
    suspend fun getUsuarioById(id: String): Result<Usuario>
```

```
    suspend fun getUsuariosByRol(rol: RolUsuario): Result<List<Usuario>>
    suspend fun createUsuario(usuario: Usuario): Result<String>
    suspend fun updateUsuario(usuario: Usuario): Result<Boolean>
    suspend fun deleteUsuario(id: String): Result<Boolean>
}
// Implementación del repositorio
class UsuarioRepositoryImpl @Inject constructor(
    private val firestore: FirebaseFirestore
) : UsuarioRepository {

    private val usuariosCollection = firestore.collection("usuarios")

    override suspend fun getUsuarioById(id: String): Result<Usuario> =
withContext(Dispatchers.IO) {
        try {
            val document = usuariosCollection.document(id).get().await()
            if (document.exists()) {
                val usuario = document.toObject(Usuario::class.java)
                Result.success(usuario ?: Usuario())
            } else {
                Result.failure(Exception("Usuario no encontrado"))
            }
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

    // Resto de implementaciones...
}
```

ViewModel (Lógica de Presentación)

```
// Estado UI para la pantalla de perfil
data class PerfilUiState(
    val usuario: Usuario? = null,
    val isLoading: Boolean = false,
    val error: String? = null
)

// ViewModel
@HiltViewModel
class PerfilViewModel @Inject constructor(
    private val usuarioRepository: UsuarioRepository,
    private val authManager: AuthManager
) : ViewModel() {

    private val _uiState = MutableStateFlow(PerfilUiState(isLoading = true))
    val uiState: StateFlow<PerfilUiState> = _uiState.asStateFlow()

    init {
        cargarPerfil()
    }

    fun cargarPerfil() {
        viewModelScope.launch {
            _uiState.value = PerfilUiState(isLoading = true)
```

```
            try {
                val userId = authManager.getCurrentUserId()
                usuarioRepository.getUsuarioById(userId).fold(
                    onSuccess = { usuario ->
                        _uiState.value = PerfilUiState(usuario = usuario)
                    },
                    onFailure = { error ->
                        _uiState.value = PerfilUiState(error = error.message)
                    }
                )
            } catch (e: Exception) {
                _uiState.value = PerfilUiState(error = e.message)
            }
        }
    }

    fun actualizarPerfil(nombre: String, apellidos: String, telefono: String) {
        val usuarioActual = _uiState.value.usuario ?: return

        viewModelScope.launch {
            _uiState.value = _uiState.value.copy(isLoading = true)

            val usuarioActualizado = usuarioActual.copy(
                nombre = nombre,
                apellidos = apellidos,
                telefono = telefono,
                updatedAt = System.currentTimeMillis()
            )

            usuarioRepository.updateUsuario(usuarioActualizado).fold(
                onSuccess = {
                    _uiState.value = _uiState.value.copy(
                        usuario = usuarioActualizado,
                        isLoading = false
                    )
                },
                onFailure = { error ->
                    _uiState.value = _uiState.value.copy(
                        error = error.message,
                        isLoading = false
                    )
                }
            )
        }
    }
}
```

View (Interfaz de Usuario con Jetpack Compose)

```
@Composable
fun PerfilScreen(
    viewModel: PerfilViewModel = hiltViewModel(),
    onNavigateBack: () -> Unit
) {
    val uiState by viewModel.uiState.collectAsState()
    val context = LocalContext.current
```

```
    val nombre = remember { mutableStateOf(uiState.usuario?.nombre ?: "") }
    val apellidos = remember { mutableStateOf(uiState.usuario?.apellidos ?: "") }
    val telefono = remember { mutableStateOf(uiState.usuario?.telefono ?: "") }

    // Actualizar valores cuando cambia el estado
    LaunchedEffect(uiState.usuario) {
        uiState.usuario?.let {
            nombre.value = it.nombre
            apellidos.value = it.apellidos
            telefono.value = it.telefono
        }
    }

    // Mostrar notificación de error
    LaunchedEffect(uiState.error) {
        uiState.error?.let { error ->
            Toast.makeText(context, error, Toast.LENGTH_LONG).show()
        }
    }

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("Mi Perfil") },
                navigationIcon = {
                    IconButton(onClick = onNavigateBack) {
                        Icon(Icons.Default.ArrowBack, contentDescription =
"Volver")
                    }
                }
            )
        }
    ) { padding ->
        Box(
            modifier = Modifier
                .fillMaxSize()
                .padding(padding)
        ) {
            if (uiState.isLoading) {
                CircularProgressIndicator(
                    modifier = Modifier.align(Alignment.Center)
                )
            } else {
                Column(
                    modifier = Modifier
                        .fillMaxSize()
                        .padding(16.dp)
                        .verticalScroll(rememberScrollState())
                ) {
                    // Avatar y detalles de perfil
                    Row(
                        verticalAlignment = Alignment.CenterVertically,
                        modifier = Modifier.fillMaxWidth()
                    ) {
                        // Avatar con iniciales
                        Box(
```

```
                            contentAlignment = Alignment.Center,
                            modifier = Modifier
                                .size(80.dp)
                                .background(
                                    MaterialTheme.colorScheme.primary,
                                    shape = CircleShape
                                )
                        ) {
                            Text(
                                text =
uiState.usuario?.nombre?.firstOrNull()?.uppercase() ?: "U",
                                color = MaterialTheme.colorScheme.onPrimary,
                                style = MaterialTheme.typography.headlineMedium
                            )
                        }

                        Spacer(modifier = Modifier.width(16.dp))

                        Column {
                            Text(
                                text = "${uiState.usuario?.nombre}
${uiState.usuario?.apellidos}",
                                style = MaterialTheme.typography.titleLarge
                            )
                            Text(
                                text = when(uiState.usuario?.rol) {
                                    RolUsuario.ADMIN_APP -> "Administrador de
Aplicación"
                                    RolUsuario.ADMIN_CENTRO -> "Administrador de
Centro"
                                    RolUsuario.PROFESOR -> "Profesor"
                                    RolUsuario.FAMILIAR -> "Familiar"
                                    else -> ""
                                },
                                style = MaterialTheme.typography.bodyMedium,
                                color = MaterialTheme.colorScheme.secondary
                            )
                        }
                    }

                    Spacer(modifier = Modifier.height(32.dp))

                    // Formulario de edición
                    OutlinedTextField(
                        value = nombre.value,
                        onValueChange = { nombre.value = it },
                        label = { Text("Nombre") },
                        modifier = Modifier.fillMaxWidth()
                    )

                    Spacer(modifier = Modifier.height(16.dp))

                    OutlinedTextField(
                        value = apellidos.value,
                        onValueChange = { apellidos.value = it },
                        label = { Text("Apellidos") },
                        modifier = Modifier.fillMaxWidth()
```

```
                    )

                    Spacer(modifier = Modifier.height(16.dp))

                    OutlinedTextField(
                        value = telefono.value,
                        onValueChange = { telefono.value = it },
                        label = { Text("Teléfono") },
                        keyboardOptions = KeyboardOptions(
                            keyboardType = KeyboardType.Phone
                        ),
                        modifier = Modifier.fillMaxWidth()
                    )

                    Spacer(modifier = Modifier.height(32.dp))

                    Button(
                        onClick = {
                            viewModel.actualizarPerfil(
                                nombre.value,
                                apellidos.value,
                                telefono.value
                            )
                        },
                        modifier = Modifier.fillMaxWidth()
                    ) {
                        Text("Guardar Cambios")
                    }
                }
            }
        }
    }
}
```

3. PATRONES DE DISEÑO APLICADOS

UmeEgunero implementa diversos patrones de diseño para mejorar la calidad, mantenibilidad y escalabilidad del código:

Repository Pattern

El patrón Repository abstrae el acceso a datos, permitiendo cambiar las fuentes de datos sin afectar a los consumidores.

```
// Definición del repositorio
interface MensajeRepository {
    suspend fun getMensajesByUsuario(usuarioId: String): Flow<List<Mensaje>>
    suspend fun enviarMensaje(mensaje: Mensaje): Result<String>
    suspend fun marcarComoLeido(mensajeId: String): Result<Boolean>
    suspend fun eliminarMensaje(mensajeId: String): Result<Boolean>
}

// Implementación con Firestore
```

```
@Singleton
class MensajeRepositoryImpl @Inject constructor(
    private val firestore: FirebaseFirestore,
    private val authManager: AuthManager
) : MensajeRepository {

    private val mensajesCollection = firestore.collection("mensajes")

    override suspend fun getMensajesByUsuario(usuarioId: String):
Flow<List<Mensaje>> = callbackFlow {
        val subscription = mensajesCollection
            .whereArrayContains("destinatariosIds", usuarioId)
            .orderBy("timestamp", Query.Direction.DESCENDING)
            .addSnapshotListener { snapshot, error ->
                if (error != null) {
                    close(error)
                    return@addSnapshotListener
                }

                val mensajes = snapshot?.documents?.mapNotNull {
                    it.toObject(Mensaje::class.java)
                } ?: emptyList()

                trySend(mensajes)
            }

        awaitClose { subscription.remove() }
    }

    // Resto de implementaciones...
}
```

Dependency Injection con Hilt

Hilt facilita la inyección de dependencias, mejorando la testabilidad y modularidad.

```
@Module
@InstallIn(SingletonComponent::class)
object AppModule {

    @Provides
    @Singleton
    fun provideFirebaseFirestore(): FirebaseFirestore =
FirebaseFirestore.getInstance()

    @Provides
    @Singleton
    fun provideFirebaseAuth(): FirebaseAuth = FirebaseAuth.getInstance()

    @Provides
    @Singleton
    fun provideFirebaseStorage(): FirebaseStorage = FirebaseStorage.getInstance()

    @Provides
    @Singleton
    fun provideAuthManager(
        firebaseAuth: FirebaseAuth,
```

```
        firestore: FirebaseFirestore
    ): AuthManager = AuthManagerImpl(firebaseAuth, firestore)

    @Provides
    @Singleton
    fun provideUsuarioRepository(
        firestore: FirebaseFirestore
    ): UsuarioRepository = UsuarioRepositoryImpl(firestore)

    @Provides
    @Singleton
    fun provideMensajeRepository(
        firestore: FirebaseFirestore,
        authManager: AuthManager
    ): MensajeRepository = MensajeRepositoryImpl(firestore, authManager)

    // Más providers...
}
```

Factory Pattern

Utilizado para crear diferentes tipos de mensajes en el sistema de comunicación.

```
sealed class MensajeFactory {

    companion object {
        fun crearMensajeChat(
            emisorId: String,
            destinatarioId: String,
            contenido: String
        ): Mensaje = Mensaje(
            tipo = TipoMensaje.CHAT,
            emisorId = emisorId,
            destinatariosIds = listOf(destinatarioId),
            contenido = contenido,
            prioridad = Prioridad.NORMAL,
            timestamp = System.currentTimeMillis()
        )

        fun crearComunicado(
            emisorId: String,
            destinatariosIds: List<String>,
            titulo: String,
            contenido: String,
            prioridad: Prioridad = Prioridad.NORMAL
        ): Mensaje = Mensaje(
            tipo = TipoMensaje.COMUNICADO,
            emisorId = emisorId,
            destinatariosIds = destinatariosIds,
            titulo = titulo,
            contenido = contenido,
            prioridad = prioridad,
            timestamp = System.currentTimeMillis()
        )

        fun crearNotificacionAsistencia(
            alumnoId: String,
```

```
            familiarId: String,
            profesorId: String,
            presente: Boolean,
            fecha: Long = System.currentTimeMillis()
        ): Mensaje = Mensaje(
            tipo = TipoMensaje.ASISTENCIA,
            emisorId = profesorId,
            destinatariosIds = listOf(familiarId),
            contenido = if (presente) "Asistencia registrada" else "Ausencia
registrada",
            metadata = mapOf(
                "alumnoId" to alumnoId,
                "presente" to presente.toString(),
                "fecha" to fecha.toString()
            ),
            timestamp = System.currentTimeMillis()
        )

        // Otros tipos de mensajes...
    }
}
```

Strategy Pattern

Implementado para aplicar diferentes estrategias de notificación según el tipo de mensaje.

```
interface NotificationStrategy {
    fun createNotification(
        context: Context,
        mensaje: Mensaje,
        channelId: String
    ): Notification
}

class ChatNotificationStrategy @Inject constructor() : NotificationStrategy {
    override fun createNotification(
        context: Context,
        mensaje: Mensaje,
        channelId: String
    ): Notification {
        // Implementación específica para notificaciones de chat
        return NotificationCompat.Builder(context, channelId)
            .setSmallIcon(R.drawable.ic_notification_chat)
            .setContentTitle("Nuevo mensaje")
            .setContentText(mensaje.contenido)
            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
            .setCategory(NotificationCompat.CATEGORY_MESSAGE)
            .build()
    }
}

class ComunicadoNotificationStrategy @Inject constructor() : NotificationStrategy
{
    override fun createNotification(
        context: Context,
        mensaje: Mensaje,
        channelId: String
```

```
    ): Notification {
        // Implementación específica para notificaciones de comunicados
        return NotificationCompat.Builder(context, channelId)
            .setSmallIcon(R.drawable.ic_notification_announcement)
            .setContentTitle(mensaje.titulo)
            .setContentText(mensaje.contenido)
            .setPriority(
                when (mensaje.prioridad) {
                    Prioridad.ALTA -> NotificationCompat.PRIORITY_HIGH
                    Prioridad.URGENTE -> NotificationCompat.PRIORITY_MAX
                    else -> NotificationCompat.PRIORITY_DEFAULT
                }
            )
            .setCategory(NotificationCompat.CATEGORY_EVENT)
            .build()
    }
}

// Uso de la estrategia
class NotificationManager @Inject constructor(
    private val chatStrategy: ChatNotificationStrategy,
    private val comunicadoStrategy: ComunicadoNotificationStrategy,
    // Más estrategias...
) {
    fun createNotification(context: Context, mensaje: Mensaje): Notification {
        val channelId = when (mensaje.tipo) {
            TipoMensaje.CHAT -> CHANNEL_CHAT
            TipoMensaje.COMUNICADO -> CHANNEL_COMUNICADOS
            TipoMensaje.ASISTENCIA -> CHANNEL_ASISTENCIA
            else -> CHANNEL_DEFAULT
        }

        return when (mensaje.tipo) {
            TipoMensaje.CHAT -> chatStrategy.createNotification(context, mensaje,
channelId)
            TipoMensaje.COMUNICADO ->
comunicadoStrategy.createNotification(context, mensaje, channelId)
            // Más casos...
            else -> defaultStrategy.createNotification(context, mensaje,
channelId)
        }
    }

    companion object {
        const val CHANNEL_CHAT = "channel_chat"
        const val CHANNEL_COMUNICADOS = "channel_comunicados"
        const val CHANNEL_ASISTENCIA = "channel_asistencia"
        const val CHANNEL_DEFAULT = "channel_default"
    }
}
```

4. INTERFAZ DE USUARIO

Jetpack Compose y Material 3

Para el desarrollo de UmeEgunero, adoptamos Jetpack Compose como framework principal de interfaz de usuario, combinado con Material 3 como sistema de diseño. Esta elección supuso múltiples ventajas para nuestro proyecto:

- Desarrollo Declarativo y Eficiente

  Jetpack Compose nos permitió adoptar un paradigma de programación declarativo que simplificó considerablemente el desarrollo de interfaces complejas. A diferencia de los enfoques tradicionales basados en XML, Compose facilita la creación de componentes dinámicos y reactivos directamente en Kotlin, eliminando la necesidad de controladores de vista y databinding.

  El desarrollo de componentes como la bandeja de mensajes unificada, que muestra diferentes tipos de comunicaciones con distintas prioridades visuales, se benefició enormemente de esta aproximación. La capacidad de Compose para gestionar eficientemente estados y recomposiciones permitió crear interfaces dinámicas que responden fluidamente a los cambios de datos.

- Implementación de Material 3

  Material 3 nos proporcionó una base sólida para crear una experiencia visualmente coherente y moderna. Implementamos un tema personalizado que adapta la paleta de colores dinámica de Material 3 a la identidad visual de UmeEgunero, manteniendo la consistencia en todos los componentes.

  Entre los elementos destacados de nuestra implementación de Material 3 se incluyen:

  - **Paleta de colores dinámica:** Utilizamos la capacidad de Material 3 para adaptar los colores al fondo de pantalla del dispositivo en Android 12+, creando una experiencia personalizada para cada usuario.
  - **Elevación y profundidad:** Aplicamos elevaciones consistentes a tarjetas y

componentes, proporcionando pistas visuales sobre la jerarquía de la información.

- **Transiciones y animaciones:** Implementamos transiciones fluidas entre pantallas y estados de componentes, mejorando la percepción de continuidad en la experiencia.

Componentes Personalizados

Desarrollamos una biblioteca de componentes reutilizables que extienden los elementos base de Material 3, manteniendo la coherencia visual mientras satisfacen las necesidades específicas de la aplicación. Estos incluyen:

- **UmeCard:** Tarjetas personalizadas con opciones de prioridad visual para diferentes tipos de contenido.
- **UmeTopAppBar:** Barra de navegación adaptada a la identidad de la aplicación.
- **PriorityLabel:** Etiquetas visuales para representar diferentes niveles de prioridad (baja, normal, alta, urgente).
- **UserRoleBadge:** Indicadores visuales que diferencian roles de usuario (administrador, profesor, familiar).

El sistema de comunicación unificado es un ejemplo destacado donde estos componentes trabajan en conjunto, permitiendo a los usuarios identificar rápidamente diferentes tipos de mensajes (chats personales, comunicados oficiales, notificaciones del sistema) y sus prioridades asociadas.

Adaptaciones para Tablets

Considerando la naturaleza educativa de UmeEgunero, donde los profesores podrían utilizar tablets durante su trabajo, diseñamos la interfaz para adaptarse fluidamente a diferentes tamaños de pantalla.

Layouts Responsive

Implementamos un diseño responsivo utilizando las capacidades de Compose para adaptar el layout según el tamaño de pantalla disponible:

En dispositivos móviles, la navegación se realiza principalmente mediante una barra inferior y un menú deslizable lateral para opciones secundarias.

En tablets, aprovechamos el espacio adicional mostrando un panel de navegación lateral permanente que convive con el contenido principal.

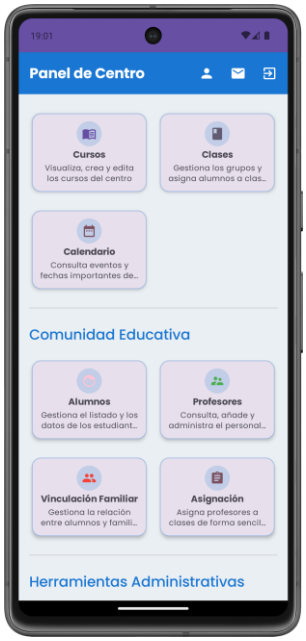
Patrones Master-Detail

Para secciones como la gestión de alumnos y la bandeja de mensajes, implementamos un patrón master-detail que:

En móviles funciona como pantallas secuenciales (lista → detalle)

En tablets muestra simultáneamente la lista y el detalle seleccionado en un layout de dos paneles

Este enfoque permite a los profesores consultar detalles de un alumno mientras mantienen visible la lista completa, mejorando significativamente la eficiencia de trabajo.



Grid Adaptable

En secciones como el dashboard de administrador, implementamos un sistema de grid que:

- Muestra 2 tarjetas por fila en móviles
- Se expande a 3-4 tarjetas por fila en tablets
- Ajusta el tamaño de gráficos y visualizaciones para aprovechar el espacio adicional

Accesibilidad

La accesibilidad fue una prioridad en el desarrollo de UmeEgunero, considerando la diversidad de usuarios potenciales en el entorno educativo.

Soporte para Tamaños de Texto

Implementamos un sistema de tipografía escalable que respeta la configuración de accesibilidad del sistema operativo, permitiendo que todos los textos se adapten correctamente cuando el usuario aumenta el tamaño de fuente en la configuración de accesibilidad del dispositivo.

Contraste y Temas

Desarrollamos tanto un tema claro como uno oscuro, asegurando que ambos cumplan con los ratios de contraste recomendados por WCAG. El tema oscuro no solo proporciona una experiencia visual alternativa, sino que reduce la fatiga visual en entornos con poca luz.

5. EVOLUCIÓN DEL DISEÑO VISUAL

El diseño de la interfaz de usuario de UmeEgunero siguió un proceso pragmático e iterativo, combinando bocetos en papel con implementación directa en Jetpack Compose. Este enfoque permitió una rápida transición desde el concepto hasta el código, aprovechando la naturaleza declarativa de Compose para iterar eficientemente sobre el diseño.

Bocetos y Wireframes Iniciales

La fase inicial del diseño comenzó con bocetos a mano en papel, que proporcionaron la flexibilidad necesaria para explorar rápidamente diferentes conceptos de organización visual y flujos de navegación sin las restricciones de herramientas digitales.

Estos bocetos manuales exploraron cuatro áreas principales:

- Sistema de autenticación: Esbozos simples del flujo de registro y login, con especial atención a la selección de roles de usuario.
- Estructura de navegación: Diagramas de flujo básicos dibujados a mano que definían las relaciones entre pantallas y las opciones de navegación para diferentes tipos de usuario.
- Diseños específicos por rol: Esquemas conceptuales de las diferentes vistas que vería cada tipo de usuario (administradores, docentes, familias).
- Sistema de comunicación: Ideas preliminares para la visualización de mensajes y notificaciones, incluyendo diferentes opciones de organización.

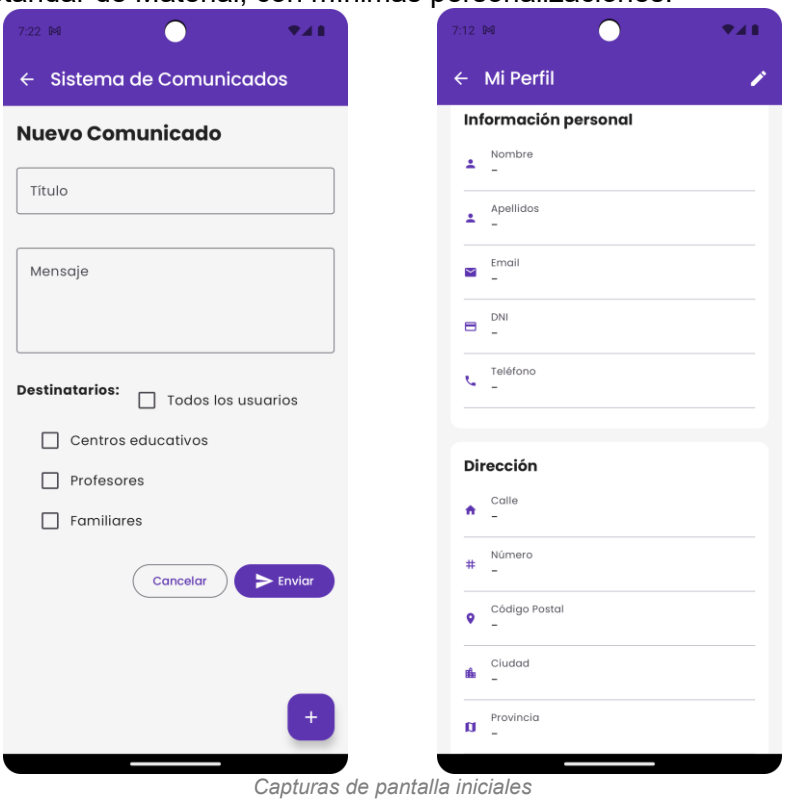
Estos bocetos iniciales priorizaron la claridad estructural y la organización funcional, estableciendo una base conceptual sólida antes de comenzar con la implementación en código.

Implementación Iterativa con Jetpack Compose

Tras la fase de bocetos en papel, pasamos directamente a la implementación en Jetpack Compose, omitiendo las herramientas de prototipado digital tradicionales. Este enfoque "del papel al código" permitió:

1. Iteración rápida: Las modificaciones podían probarse inmediatamente en dispositivos reales o emuladores.
2. Exploración práctica: Experimentación directa con la interactividad y animaciones que serían difíciles de representar en diseños estáticos.
3. Evolución orgánica: El diseño evolucionó naturalmente a medida que se implementaban las funcionalidades.

Las primeras versiones funcionales implementadas en Compose eran intencionadamente minimalistas, centrándose en la estructura básica y la funcionalidad core. Estas implementaciones tempranas utilizaban principalmente componentes estándar de Material, con mínimas personalizaciones.

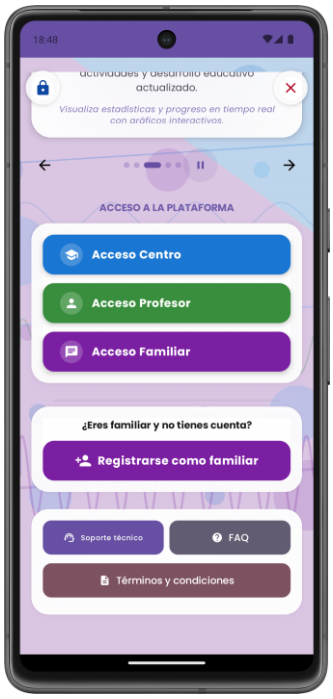


Capturas de pantalla iniciales

Refinamiento Visual e Iteraciones

A medida que avanzaba el desarrollo, el diseño visual experimentó varias iteraciones significativas directamente en el código:

1. Evolución del sistema de colores: Los primeros prototipos utilizaban la paleta Material predeterminada, que posteriormente se personalizó para reflejar la identidad visual educativa del proyecto.
2. Componentes personalizados: Desarrollo gradual de componentes UI específicos para UmeEgunero, como las tarjetas de alumno, indicadores de asistencia y el sistema de filtrado.
3. Refinamiento de navegación: Evolución desde una navegación básica hasta el sistema actual con múltiples niveles y adaptación por rol.
4. Optimización para diferentes tamaños de pantalla: Adaptaciones sucesivas para mejorar la experiencia en dispositivos de diferentes dimensiones.



Etiquetas de Contenido

Todos los elementos interactivos y visuales incluyen descripciones de contenido compatibles con TalkBack, permitiendo a usuarios con discapacidades visuales navegar eficientemente por la aplicación.

Feedback Háptico

Incorporamos feedback háptico en acciones importantes como el envío de mensajes, aprobación de solicitudes y confirmación de registros diarios, proporcionando una capa adicional de retroalimentación para usuarios con discapacidades visuales.

La combinación de Jetpack Compose, Material 3 y un enfoque centrado en la accesibilidad nos permitió crear una interfaz moderna, coherente y adaptable que satisface las necesidades de todos los perfiles de usuario de UmeEgunero, independientemente del dispositivo utilizado o sus necesidades de accesibilidad.

MODELO DE DATOS Y PERSISTENCIA

1. DIAGRAMA ENTIDAD-RELACIÓN

En el contexto de nuestra aplicación UmeEgunero, aunque estamos trabajando con una base de datos NoSQL como Firestore, resulta útil conceptualizar las relaciones entre nuestras

entidades principales para entender el flujo de datos. El siguiente diagrama representa las principales entidades y sus relaciones:

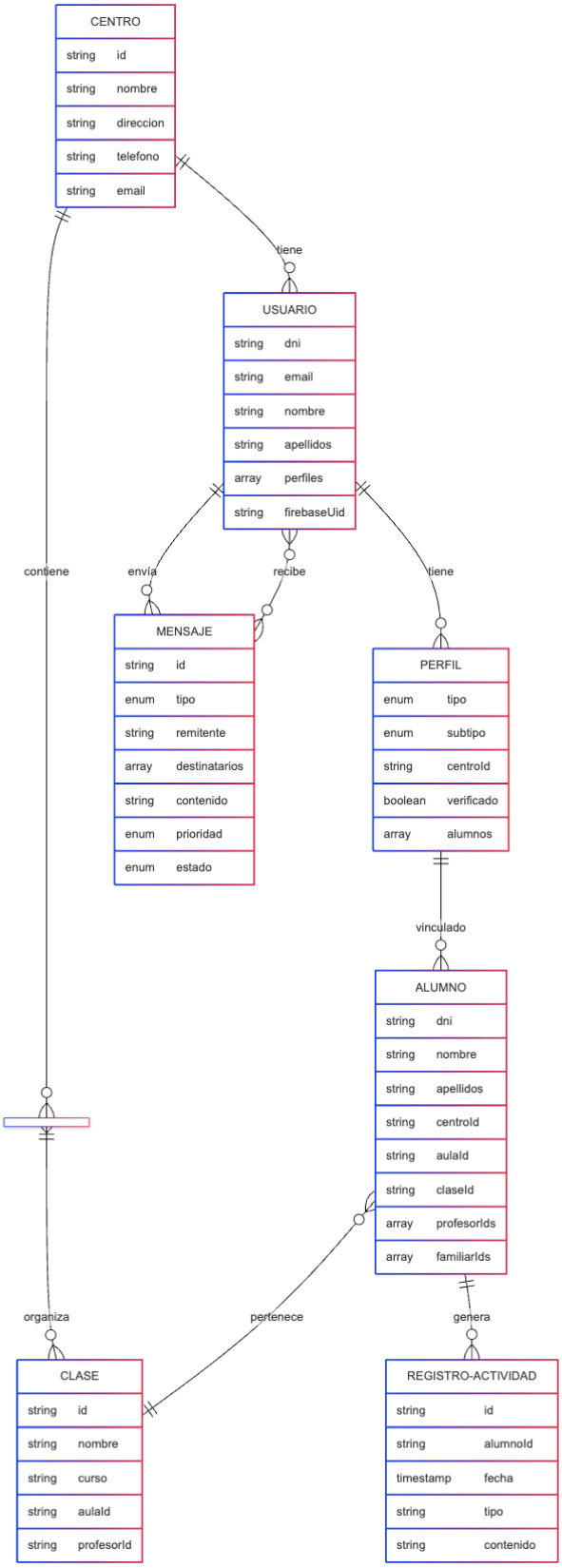


Diagrama Entidad – Relación en base de datos no relacional

2. ENTIDADES Y RELACIONES DEL SISTEMA

Entidades Principales

1. **Usuario**
   - Representa a cualquier persona que interactúa con el sistema.
   - Atributos clave: dni, email, nombre, apellidos, perfiles, firebaseUid.
   - Un usuario puede tener múltiples perfiles (administrador, profesor, familiar).
2. **Perfil**
   - Define el rol y contexto de actuación de un usuario.
   - Atributos clave: tipo, subtipo, centroId, verificado, alumnos.
   - Implementa un sistema multi-rol donde un usuario puede tener diferentes funciones.
3. **Centro**
   - Institución educativa donde se imparten clases.
   - Atributos clave: id, nombre, dirección, teléfono, email, administradores.
   - Contiene aulas y está vinculado a administradores y profesores.
4. **Alumno**
   - Estudiante registrado en el centro educativo.
   - Atributos clave: dni, nombre, apellidos, centroId, aulaId, claseId, profesorIds, familiarIds.
   - Mantiene vínculos con profesores, familiares y registros de actividad.
5. **Clase**
   - Grupo educativo específico dentro de un aula.
   - Atributos clave: id, nombre, curso, aulaId, profesorId.
   - Agrupa a alumnos con características académicas similares.
6. **RegistroActividad**
   - Registra eventos diarios del alumno (comidas, siestas, actividades, etc.).
   - Atributos clave: id, alumnoId, fecha, tipo, contenido, observaciones.
   - Proporciona seguimiento detallado del día a día escolar.
7. **Mensaje**
   - Sistema unificado de comunicación.
   - Atributos clave: id, tipo, remitente, destinatarios, contenido, prioridad, estado.
   - Facilita diferentes formas de comunicación entre usuarios.

Relaciones Principales

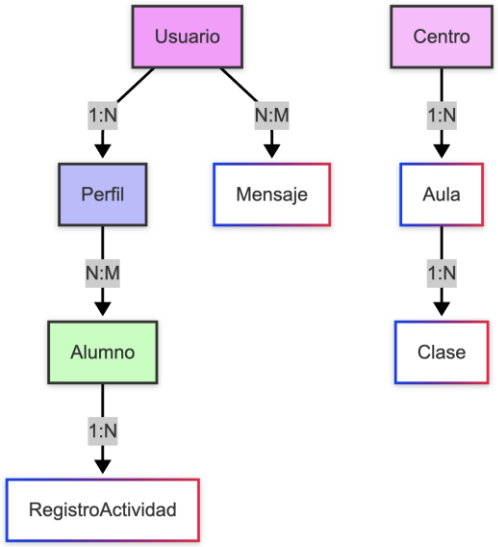


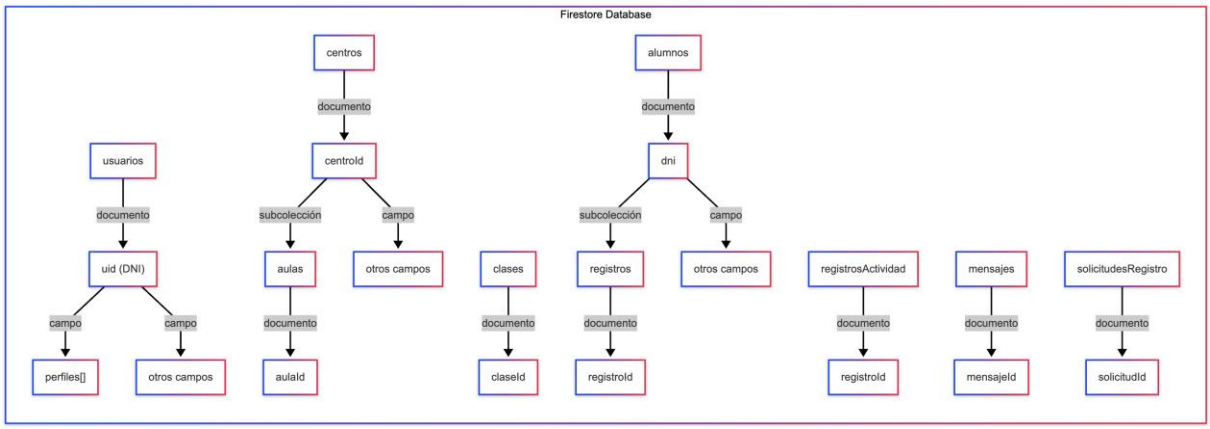
Tabla de relaciones

1. Usuario-Perfil (1:N)
   - Un usuario puede tener múltiples perfiles, cada uno representando un rol diferente.
   - Implementado mediante una lista de perfiles en el documento de usuario.
2. Perfil-Alumno (N:M)
   - Un perfil familiar puede estar vinculado a varios alumnos.
   - Un alumno puede tener múltiples perfiles familiares asociados.
   - Implementado mediante listas de referencias en ambas entidades.
3. Centro-Aula (1:N)
   - Un centro educativo contiene múltiples aulas.
   - Implementado mediante el campo centroId en los documentos de Aula.
4. Aula-Clase (1:N)
   - Un aula puede albergar varias clases.
   - Implementado mediante el campo aulaId en los documentos de Clase.
5. Alumno-RegistroActividad (1:N)
   - Un alumno tiene múltiples registros de actividad diaria.
   - Implementado mediante el campo alumnoId en los documentos de RegistroActividad.
6. Usuario-Mensaje (N:M)
   - Usuarios envían y reciben múltiples mensajes.
   - Implementado mediante campos de remitente y lista de destinatarios.

3. IMPLEMENTACIÓN DE FIRESTORE

Estructura de Colecciones

En UmeEgunero, hemos organizado nuestra base de datos Firestore en colecciones principales que reflejan las entidades fundamentales del sistema:



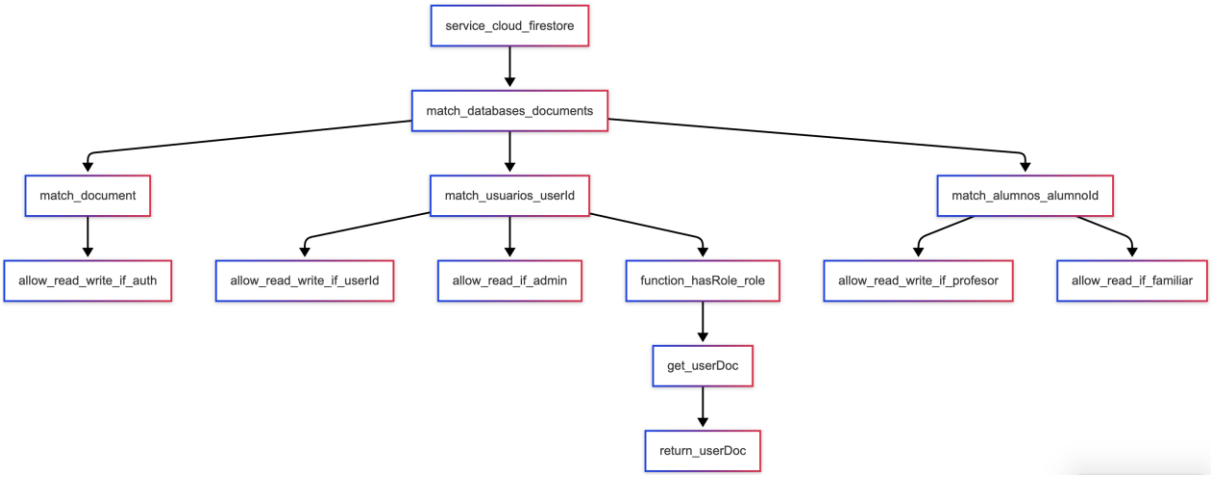
Estructura de colecciones en firestore

Esta estructura nos permite:

- **Acceso eficiente**: Recuperar rápidamente la información relevante para cada usuario según su contexto.
- **Escalabilidad**: Manejar un gran número de usuarios, centros y alumnos sin problemas de rendimiento.
- **Flexibilidad**: Adaptar el modelo a las necesidades cambiantes del sistema educativo.
- **Seguridad granular**: Implementar reglas de seguridad precisas a nivel de documento y colección.

Reglas de Seguridad

La seguridad en Firestore es crucial para proteger información sensible de alumnos y centros. Hemos implementado reglas detalladas que:



Estructura de las reglas de seguridad

Estas reglas garantizan que:

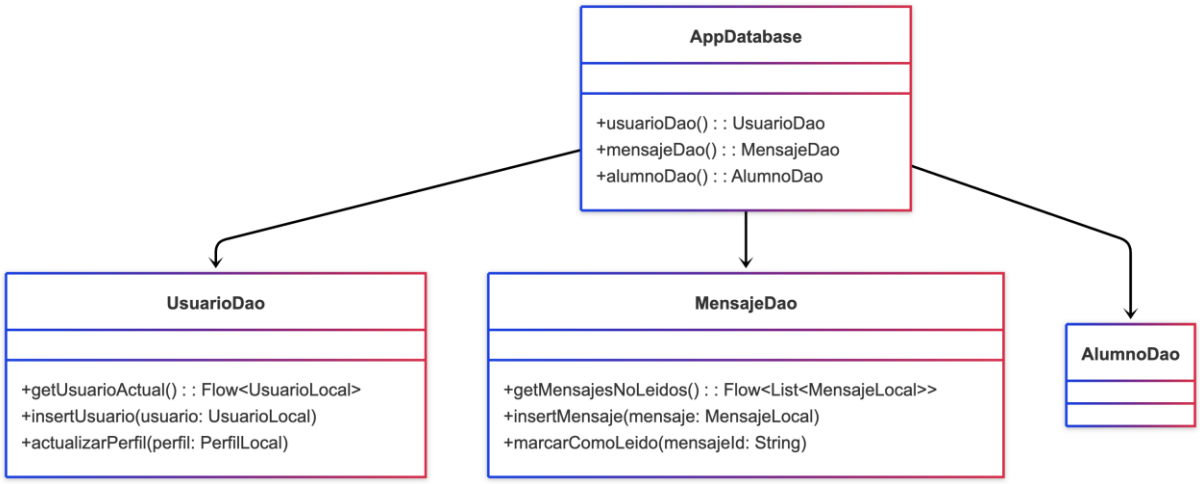
- Los datos de los alumnos solo son accesibles para profesores asignados y familiares vinculados
- La información sensible de los centros está protegida
- Los mensajes solo pueden ser leídos por remitentes y destinatarios legítimos

4. IMPLEMENTACIÓN DE ROOM DATABASE

Aunque UmeEgunero utiliza principalmente Firestore como base de datos remota, hemos implementado Room Database para:

DAOs y Entidades

Hemos creado entidades locales que reflejan estructuras clave de la base de datos remota, pero optimizadas para consultas locales:



Las entidades locales como UsuarioLocal, MensajeLocal y AlumnoLocal mantienen solo la información esencial para el funcionamiento offline de la aplicación.

Estrategia de Sincronización

Para mantener los datos locales y remotos sincronizados, hemos implementado:

1. Capa de repositorio unificada: Cada repositorio (UsuarioRepository, MensajeRepository, etc.) gestiona tanto el acceso a Firestore como a Room.
2. Patrón de caché con invalidación: Los datos se almacenan localmente y se actualizan cuando:

- El usuario inicia sesión o realiza una acción específica
- Se recibe una notificación push de cambios
- Han pasado más de X minutos desde la última sincronización

3. Gestión de conflictos: En caso de cambios concurrentes, priorizamos:
   - Datos del servidor para información crítica (perfiles, vinculaciones)
   - Datos locales para preferencias de usuario y configuración de la aplicación

Este enfoque dual nos permite ofrecer una experiencia fluida incluso con conectividad intermitente, manteniendo siempre la integridad y seguridad de los datos del entorno educativo

SEGURIDAD DE LA APLICACIÓN

1. VALIDACIÓN DE ENTRADAS

Durante el desarrollo de UmeEgunero implementamos un sistema robusto de validación de entradas para prevenir posibles vulnerabilidades. Desarrollamos validadores específicos para cada tipo de campo de entrada según los requisitos de seguridad establecidos.
Para la validación de correos electrónicos, además de la comprobación sintáctica mediante expresiones regulares, incorporamos un sistema de verificación de dominios para filtrar proveedores de correo temporales o no confiables. En el caso de datos críticos como fechas de nacimiento de alumnos, implementamos validadores que aseguran coherencia temporal y previenen errores de introducción.

```
fun validarNombreApellido(texto: String): Boolean {
    // Validación de longitud
    if (texto.isBlank() || texto.length > 50) return false

    // Validación de caracteres mediante expresión regular
    val regex = Regex("^[\\p{L} .'\\-]+$")
    return regex.matches(texto)
}
```

2. PREVENCIÓN DE DESBORDAMIENTO DE BUFFER

Para mitigar riesgos de desbordamiento de buffer, implementamos restricciones de tamaño en todos los campos de entrada de datos. Estas restricciones fueron establecidas después de un análisis detallado de los requisitos funcionales de cada tipo de contenido:

- Mensajes de chat: limitados a 500 caracteres
- Comunicados oficiales: máximo 2000 caracteres
- Títulos de comunicados: máximo 100 caracteres
- Registros de actividades: máximo 1000 caracteres

Adicionalmente, implementamos un sistema de compresión automática para los contenidos multimedia antes de su transmisión a Firebase Storage, estableciendo límites máximos de dimensiones y tamaño en kilobytes según el tipo de contenido.

3. GESTIÓN SEGURA DE SESIONES

Para la gestión de sesiones implementamos un sistema basado en Firebase Authentication, aprovechando su infraestructura de tokens JWT. Sin embargo, extendimos esta funcionalidad para adaptarla a nuestros requisitos específicos de seguridad.

Desarrollamos un mecanismo de verificación de roles en tiempo real que consulta el estado actual del usuario en Firestore antes de autorizar operaciones críticas, asegurando que los cambios en los permisos de usuario se reflejen inmediatamente en todas las sesiones activas.

```
suspend fun verificarRolActual(usuarioId: String, rolRequerido: RolUsuario):
Boolean {
    val usuarioActual = usuariosCollection.document(usuarioId).get().await()
    return usuarioActual?.get("rol") == rolRequerido.name
}
```

4. ENCRIPTACIÓN DE CONTRASEÑAS

Para la gestión de contraseñas aprovechamos la infraestructura de Firebase Authentication, que implementa algoritmos de hash seguros con salt dinámico. Esto nos permitió delegar la responsabilidad de almacenamiento seguro de credenciales a una plataforma probada y auditada.

Para el proceso de recuperación de contraseñas, implementamos un sistema basado en tokens temporales generados mediante Google Apps Script, con una validez limitada a 30 minutos. Este sistema incluye verificación de timestamps para prevenir ataques de replay.

Para cuentas con privilegios elevados (administradores y profesores), configuramos un sistema de bloqueo temporal después de cinco intentos fallidos de autenticación, estableciendo un periodo de suspensión de 30 minutos como medida contra ataques de fuerza bruta.

5. PROTECCIÓN DE DATOS SENSIBLES

Considerando que la aplicación gestiona datos de menores de edad, implementamos un sistema integral de protección de datos sensibles conforme a la normativa LOPD y RGPD:

- Principio de minimización de datos: Limitamos la recolección de información personal al mínimo imprescindible para la funcionalidad requerida.
- Sistema de consentimiento granular: Desarrollamos una arquitectura que permite a los usuarios otorgar y revocar consentimientos específicos para diferentes tipos de procesamiento de datos.
- Cifrado de datos sensibles: Utilizamos la biblioteca de cifrado de AndroidX para proteger información sensible antes de su transmisión y almacenamiento.
- Políticas de retención: Implementamos un sistema automatizado para la eliminación de datos según períodos predefinidos, estableciendo caducidad automática para mensajes y registros diarios.

6. REGLAS DE SEGURIDAD EN FIRESTORE

La implementación de reglas de seguridad en Firestore constituyó un componente crítico del modelo de seguridad. Desarrollamos un conjunto jerarquizado de reglas que implementan:

- Control de acceso basado en roles (RBAC)
- Verificación de pertenencia a centros educativos
- Validación estructural de documentos

La implementación más compleja correspondió al sistema de solicitudes de vinculación entre familiares y alumnos, donde debimos equilibrar la capacidad de creación de solicitudes con restricciones de acceso apropiadas.

```
match /solicitudes/{solicitudId} {
  allow create: if request.auth != null;
  allow read: if request.auth != null &&
              (resource.data.solicitanteId == request.auth.uid ||
               esAdministradorDelCentro(request.auth.uid,
resource.data.centroId));
  allow update: if request.auth != null &&
                esAdministradorDelCentro(request.auth.uid,
resource.data.centroId);
}
```

7. AUTENTICACIÓN CON FIREBASE AUTH

Implementamos un sistema de autenticación basado en Firebase Authentication, que incluye:

- Autenticación primaria mediante email y contraseña
- Integración con Google Sign-In como método alternativo

El proceso de registro implementa flujos diferenciados según el rol de usuario, requiriendo verificación administrativa previa para roles privilegiados mediante un sistema de códigos de invitación de un solo uso con caducidad temporal. Este enfoque garantiza que solo usuarios autorizados puedan acceder a roles con capacidades elevadas dentro del sistema.

Adicionalmente, implementamos un sistema de persistencia de sesión que permite a los usuarios mantener su autenticación entre reinicios de aplicación, mientras se mantienen las verificaciones de seguridad apropiadas para cada operación sensible.

CAPAS DEL MODELO OSI IMPLICADAS

1. PRIMERAS CAPAS OSI Y SU IMPLICACIÓN

En UmeEgunero, aunque trabajamos principalmente en las capas superiores del modelo OSI, dependemos fundamentalmente de las primeras capas para establecer la comunicación con Firebase. Las capas físicas, de enlace, de red y de transporte (1-4) son gestionadas por la infraestructura Android y las bibliotecas de Firebase SDK. Estas capas garantizan que nuestras solicitudes a Firestore, Authentication y Cloud Messaging se transmitan de manera confiable a través de Internet, manejando aspectos como la seguridad de la conexión mediante HTTPS, reintentos automáticos en caso de pérdida de paquetes y la gestión eficiente del ancho de banda al sincronizar datos en tiempo real.

2. CAPA 5: SESIÓN

La capa de sesión es crucial en UmeEgunero, especialmente en el UsuarioRepository donde implementamos toda la lógica de autenticación y gestión de sesiones. Nuestro método loginWithEmailAndPassword() establece y mantiene sesiones de usuario mediante Firebase Authentication, gestionando tokens JWT para operaciones autenticadas. Utilizamos el patrón repository para abstraer esta complejidad, manteniendo sesiones persistentes entre reinicios de la aplicación mediante FirebaseAuth.AuthStateListener.

La implementación incluye mecanismos para detectar vencimiento de sesión, sesión inválida y renovación automática de tokens, esenciales para mantener la seguridad en una aplicación educativa con datos sensibles.

3. CAPA 6: PRESENTACIÓN

En la capa de presentación, UmeEgunero implementa la serialización y deserialización de datos entre el modelo de dominio y Firebase. Los objetos Usuario y otros modelos de datos son transformados bidirecccionalmente entre documentos Firestore y objetos Kotlin mediante extensiones y funciones de mapeo personalizadas.
Para mejorar la eficiencia, implementamos técnicas de serialización parcial que permiten actualizar solo campos modificados. Esta capa también maneja la compresión y optimización de imágenes antes de su almacenamiento en Firebase Storage, así como la conversión entre diferentes formatos de fecha (timestamps de servidor vs. objetos LocalDateTime) para garantizar consistencia en la visualización de datos por diferentes perfiles de usuario.

4. CAPA 7: APLICACIÓN

La capa de aplicación de UmeEgunero está desarrollada con Jetpack Compose y Material 3, implementando el patrón MVVM donde los ViewModels consumen el UsuarioRepository para todas las operaciones de usuario. Esta capa proporciona abstracciones específicas para cada perfil (administrador, profesor, familiar), adaptando la representación de datos según el contexto. Nuestra implementación de loginWithEmailAndPassword() se conecta directamente con componentes Compose que muestran estados de autenticación, errores de acceso y retroalimentación en tiempo real. La capa de aplicación también gestiona la integración con las APIs de Firebase Cloud Messaging para notificaciones push contextuales y Firebase Remote Config para ajustes dinámicos de configuración, creando una experiencia de usuario fluida y adaptable.

CORRUTINAS Y CONCURRENCIA

1. EXPLICACIÓN TÉCNICA SOBRE EL USO DE CORRUTINAS EN KOTLIN

Fundamentos de las corrutinas

Las corrutinas de Kotlin representan una de las características más potentes del lenguaje para el desarrollo de aplicaciones Android modernas. Fundamentalmente, constituyen un patrón de diseño que permite escribir código asíncrono de manera secuencial, simplificando enormemente la gestión de operaciones que no deben bloquear el hilo principal de ejecución.

A diferencia del enfoque tradicional basado en callbacks, que tiende a generar el llamado "callback hell", las corrutinas nos permiten estructurar el código asíncrono de forma similar al código síncrono, mejorando significativamente su legibilidad y mantenibilidad. Esta característica resulta especialmente valiosa en aplicaciones como UmeEgunero, donde la interacción constante con servicios externos como Firebase requiere numerosas operaciones asíncronas.

Los componentes fundamentales del sistema de corrutinas incluyen:

- **CoroutineScope:** Define el ámbito de vida de las corrutinas, facilitando su cancelación controlada.
- **Dispatchers:** Determinan el hilo o grupo de hilos donde se ejecutará el código de la corrutina.
- **Builders:** Funciones como *launch o async* que inician una nueva corrutina.
- **Suspending functions:** Funciones que pueden pausar su ejecución sin bloquear el hilo, marcadas con la palabra clave *suspend*
- **Flow:** API para flujos de datos asíncronos que pueden emitir múltiples valores secuencialmente.

Implementación en umeegunero

En UmeEgunero, hemos adoptado las corrutinas como mecanismo principal para todas las operaciones asíncronas. Esta decisión arquitectónica se refleja principalmente en la capa de repositorio, donde gestionamos las interacciones con Firebase Authentication, Firestore y otros servicios externos.

En el **UsuarioRepository,** implementamos un patrón que separa claramente las responsabilidades:

```
class UsuarioRepository @Inject constructor(
    private val auth: FirebaseAuth,
    private val firestore: FirebaseFirestore,
    // Otras dependencias...
) {
    suspend fun autenticarUsuario(email: String, password: String):
Result<Usuario> {
        return withContext(Dispatchers.IO) {
            try {
                val authResult = auth.signInWithEmailAndPassword(email,
password).await()
                // Procesamiento del resultado...
                Result.success(usuario)
            } catch (e: Exception) {
                Result.failure(e)
            }
        }
    }

    // Otras funciones suspend...
```

```
    fun obtenerUsuarioActual(): Flow<Usuario?> {
        return callbackFlow {
            val listener = auth.addAuthStateListener { firebaseAuth ->
                trySend(firebaseAuth.currentUser?.let { mapearFirebaseUser(it) })
            }

            awaitClose {
                auth.removeAuthStateListener(listener)
            }
        }
    }
}
```

Este enfoque nos proporciona múltiples ventajas:

- **Gestión de errores simplificada:** Utilizamos bloques try-catch convencionales en lugar de callbacks anidados.
- **Cancelación automática:** Al vincular las corrutinas a los CoroutineScopes de los componentes de Android (viewModelScope, lifecycleScope), garantizamos que las operaciones se cancelan adecuadamente cuando no son necesarias.
- **Testing facilitado:** Las funciones suspend son más sencillas de probar que las implementaciones basadas en callbacks.
- **Integración nativa con Jetpack:** Componentes como ViewModel, Room, y WorkManager están diseñados para funcionar perfectamente con corrutinas.

En particular, hemos aprovechado la integración con Firebase mediante las extensiones de Kotlin Coroutines para Firebase, que transforman las APIs basadas en callbacks de Firebase en funciones suspend mediante el uso de suspendCoroutine y suspendCancellableCoroutine.

2. CASOS CONCRETOS DENTRO DE LA APP

Acceso a Red

Para todas las operaciones de red, hemos implementado corrutinas que operan en el contexto Dispatchers.IO, asegurando que las operaciones potencialmente bloqueantes no afecten la experiencia de usuario. Esto es particularmente importante en funcionalidades como la autenticación, donde utilizamos:

```
suspend fun autenticarConGoogle(idToken: String): Result<Usuario> {
    return withContext(Dispatchers.IO) {
        try {
            val credential = GoogleAuthProvider.getCredential(idToken, null)
            val authResult = auth.signInWithCredential(credential).await()
            // Procesamiento del resultado
```

```
            Result.success(usuario)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }
}
```

El uso de withContext(Dispatchers.IO) garantiza que la operación de autenticación se realiza fuera del hilo principal, mientras que la naturaleza suspendible de la función permite que el hilo sea reutilizado para otras operaciones mientras esperamos la respuesta del servidor.

Operaciones de base de datos

Para interactuar con Firestore, hemos desarrollado extensiones que transforman las operaciones basadas en Tasks de Firebase en funciones suspend:

```
suspend fun <T> Task<T>.awaitResult(): Result<T> {
    return suspendCancellableCoroutine { continuation ->
        addOnSuccessListener { result ->
            continuation.resume(Result.success(result))
        }
        addOnFailureListener { exception ->
            continuation.resume(Result.failure(exception))
        }
        continuation.invokeOnCancellation {
            // Limpieza si es necesario
        }
    }
}
```

Esta aproximación nos permite escribir código como:

```
suspend fun obtenerPerfilUsuario(uid: String): Result<PerfilUsuario> {
    return try {
        val documento =
firestore.collection("usuarios").document(uid).get().awaitResult().getOrThrow()
        Result.success(documento.toPerfilUsuario())
    } catch (e: Exception) {
        Result.failure(e)
    }
}
```

Operaciones en Paralelo

Para operaciones que pueden ejecutarse en paralelo, como la carga de datos relacionados pero independientes, utilizamos async y await:

```
suspend fun cargarDashboardAdministrador(centroId: String): Result<DashboardData>
{
    return withContext(Dispatchers.IO) {
        try {
            val estadisticasDeferred = async { obtenerEstadisticas(centroId) }
            val profesoresDeferred = async { obtenerProfesores(centroId) }
            val aulasDeferred = async { obtenerAulas(centroId) }
```

```
            // Esperamos a que todas las operaciones concluyan
            val estadisticas = estadisticasDeferred.await()
            val profesores = profesoresDeferred.await()
            val aulas = aulasDeferred.await()

            Result.success(DashboardData(estadisticas, profesores, aulas))
        } catch (e: Exception) {
            Result.failure(e)
        }
    }
}
```

Este patrón maximiza la eficiencia al permitir que múltiples operaciones progresen simultáneamente, reduciendo el tiempo total de carga.

Paralelización de Tareas

Para la ejecución de tareas en paralelo con límites controlados, implementamos soluciones basadas en Dispatchers.Default con ajuste del paralelismo según la capacidad del dispositivo:

```
val dispatcher = Dispatchers.Default.limitedParallelism(4) // Limitar a 4 hilos
concurrentes

suspend fun procesarRegistrosDiarios(registros: List<RegistroDiario>):
List<RegistroProcesado> {
    return withContext(dispatcher) {
        registros.map { registro ->
            async {
                procesarRegistro(registro)
            }
        }.awaitAll()
    }
}
```

Esta aproximación nos permite aprovechar la capacidad de procesamiento paralelo sin sobrecargar el dispositivo.

Las corrutinas han sido fundamentales en nuestra arquitectura, permitiéndonos desarrollar una aplicación fluida y responsiva que maneja eficientemente operaciones asíncronas complejas mientras mantiene un código limpio y mantenible.

INTEGRACIÓN DE SERVICIOS DE GOOGLE

En el desarrollo de UmeEgunero, hemos implementado una arquitectura basada en Firebase como columna vertebral de nuestra infraestructura backend. Esta decisión nos ha permitido construir una aplicación robusta y escalable sin la necesidad de desarrollar un servidor backend tradicional. A continuación, detallamos cómo hemos integrado cada uno de estos servicios en nuestra plataforma de gestión educativa.

1. FIREBASE AUTHENTICATION

La autenticación de usuarios ha sido implementada utilizando Firebase Authentication, proporcionando un sistema robusto y seguro que gestiona todo el ciclo de vida de las cuentas de usuario. Hemos configurado múltiples métodos de autenticación:

- Autenticación por correo electrónico y contraseña como método principal
- Integración de autenticación biométrica para acceso rápido y seguro en dispositivos compatibles
- Sistema de tokens JWT para mantener sesiones seguras entre la aplicación y los servicios

La implementación en el UsuarioRepository nos permite gestionar todas las operaciones relacionadas con la autenticación de manera centralizada, desde el registro de nuevos usuarios hasta la gestión de contraseñas olvidadas, manteniendo una separación clara entre la lógica de autenticación y el resto de la aplicación.

2. FIRESTORE DATABASE

Como base de datos principal, Firestore nos ha proporcionado una solución NoSQL flexible y escalable. Hemos diseñado una estructura de colecciones que refleja la complejidad del dominio educativo:

- Colecciones para usuarios con subcategorías por roles (administradores, profesores, familiares)
- Documentos de centros educativos con colecciones anidadas de aulas y cursos
- Registros de alumnos con vinculaciones a familiares y profesores
- Sistema de mensajería unificado con categorización por tipos

La sincronización en tiempo real de Firestore ha sido crucial para mantener actualizadas las interfaces de usuario, especialmente en funcionalidades como la bandeja de comunicaciones y los registros diarios de los alumnos.

3. FIREBASE STORAGE

Para la gestión de archivos multimedia, hemos integrado Firebase Storage. Este servicio nos permite:

- Almacenar imágenes de perfiles de usuarios
- Gestionar documentos adjuntos en comunicaciones
- Archivar evidencias fotográficas del progreso de los alumnos

- Mantener recursos educativos digitales accesibles para el profesorado

Hemos implementado un sistema de permisos granular utilizando las reglas de seguridad de Firebase Storage, asegurando que cada archivo solo sea accesible por los usuarios autorizados según su rol y vinculaciones.

4. FIREBASE CLOUD MESSAGING

La implementación de Firebase Cloud Messaging (FCM) ha sido fundamental para nuestro sistema de notificaciones en tiempo real. Hemos desarrollado:

- Un servicio de escucha (UmeEguneroMessagingService) que gestiona las notificaciones recibidas
- Canales de notificación diferenciados según la importancia y categoría del mensaje
- Integración de deeplinks para navegar directamente al contenido relevante desde las notificaciones
- Sistema de tokens para dirigir las notificaciones al dispositivo correcto de cada usuario

La arquitectura de notificaciones está diseñada para manejar diferentes escenarios, desde mensajes directos entre usuarios hasta alertas automáticas generadas por el sistema.

5. FIREBASE CRASHMLYTICS

Para asegurar la estabilidad de la aplicación, hemos integrado Firebase Crashlytics como sistema de monitorización de errores:

- Registro automático de excepciones no controladas
- Informes detallados sobre problemas de rendimiento
- Trazabilidad de errores críticos hasta el código fuente
- Métricas de adopción de versiones y comportamiento en diferentes dispositivos

Este sistema nos permite identificar y solucionar problemas de manera proactiva, mejorando continuamente la estabilidad de la aplicación.

6. GOOGLE MAPS API

Aunque no forma parte del ecosistema Firebase, hemos integrado la API de Google Maps para proporcionar funcionalidades de geolocalización:

- Visualización de la ubicación de los centros educativos
- Cálculo de rutas desde la ubicación del familiar hasta el centro
- Geocodificación inversa para simplificar la entrada de direcciones
- Validación de ubicaciones durante el registro de nuevos centros

La integración se ha realizado siguiendo las mejores prácticas de seguridad, incluyendo la restricción de la API key a nivel de aplicación y dominio.
Estas integraciones tecnológicas conforman los cimientos sobre los que hemos construido UmeEgunero, permitiéndonos ofrecer una experiencia de usuario fluida y funcionalidades avanzadas manteniendo un desarrollo ágil y enfocado en las necesidades del usuario final.

PRUEBAS REALIZADAS

1. PRUEBAS DE INTEGRACIÓN

En el desarrollo de UmeEgunero, las pruebas de integración han constituido un pilar fundamental para garantizar el funcionamiento cohesionado entre los distintos componentes de la aplicación. A diferencia de las pruebas unitarias que verifican funcionalidades aisladas, nuestras pruebas de integración se han centrado en validar la correcta interacción entre módulos, servicios y capas de la arquitectura.
Para la implementación de estas pruebas, hemos adoptado un enfoque sistemático que nos ha permitido verificar las interacciones críticas del sistema, especialmente aquellas relacionadas con el repositorio de usuarios, pieza central en nuestra arquitectura multi-rol.

2. METODOLOGÍA APLICADA

Nuestra estrategia de pruebas de integración se ha basado en la verificación de flujos completos de datos entre:

- La capa de repositorio y Firebase Authentication
- El sistema de comunicación unificado y Firebase Cloud Messaging
- Los repositorios y la capa de presentación mediante ViewModels
- Las solicitudes de vinculación y el sistema de notificaciones

Para cada uno de estos escenarios, hemos diseñado pruebas que simulan el comportamiento real de la aplicación, utilizando instancias reales de los componentes pero con datos controlados de prueba.

Enfoque TDD Adaptado (Test-Driven Development)

Para los componentes críticos de UmeEgunero, hemos seguido un enfoque TDD adaptado:

1. Escribir pruebas para los casos de uso principales (por ejemplo, autenticación de usuarios)
2. Implementar la funcionalidad mínima necesaria para pasar las pruebas
3. Refactorizar para mejorar la calidad del código sin romper las pruebas
4. Iterar para añadir nuevas funcionalidades

Este enfoque ha sido especialmente valioso para el desarrollo del sistema multi-perfil, donde la complejidad de los casos de uso requería una validación constante.

Ciclo de Pruebas

Las pruebas se han ejecutado:

- Durante el desarrollo: Pruebas unitarias y de integración básicas
- Al finalizar cada sprint: Suite completa de pruebas de integración
- Antes de releases importantes: Pruebas E2E y de rendimiento

3. TECNOLOGÍAS Y HERRAMIENTAS EMPLEADAS

En la implementación de las pruebas de integración hemos utilizado:

- Firebase Emulator Suite: Nos ha permitido simular los servicios de Firebase (Authentication, Firestore, Cloud Functions) sin necesidad de conectar con los servicios reales, acelerando la ejecución de pruebas y evitando costes adicionales.
- MockK: Para simular dependencias externas cuando ha sido necesario.
- Jetpack Compose Testing: Para verificar la correcta integración de los flujos de datos con la interfaz de usuario.
- Hilt Testing: Para la inyección de dependencias en entornos de prueba.

Stack de Testing Completo

Además de las herramientas mencionadas, nuestro stack de testing incluye:

- **JUnit**: Claramente implementado y usado en todas nuestras pruebas (tanto unitarias como instrumentadas) con anotaciones @Test, @Before, etc.
- **Mockito**: Usado en nuestras pruebas unitarias, como en LoginViewModelTest.kt donde usamos @RunWith(MockitoJUnitRunner::class) y métodos como mock() y whenever().
- **MockK**: Se utiliza principalmente en pruebas de UI de Compose, como en LoginScreenTest.kt con mockk(relaxed = true).
- **Compose UI Testing**: Se está usando extensivamente el framework de testing de Jetpack Compose (androidx.compose.ui.test.*, createComposeRule) para nuestras pruebas de UI.
- **Kotlinx Coroutines Test**: Utilizamos UnconfinedTestDispatcher y runTest para probar código asíncrono.
- **Hilt Testing**: Para pruebas instrumentadas usamos HiltAndroidRule y HiltAndroidTest en algunos archivos como CreacionCentroTest.kt.

4. PRUEBAS UNITARIAS

Las pruebas unitarias en UmeEgunero se han centrado principalmente en la lógica de negocio aislada:

Pruebas de ViewModels

Hemos implementado pruebas para verificar la correcta emisión de estados UI:

```
@Test
fun `login emite Loading y luego Success cuando credenciales son válidas`() =
runTest {
    // Given
    val email = "test@example.com"
    val password = "valid_password"
    coEvery { loginUseCase.verificarCredenciales(email, password) } returns
AuthResult.Success

    // When
    loginViewModel.login(email, password)

    // Then
    turbineScope {
        val turbine = loginViewModel.uiState.testIn(this)

assertThat(turbine.awaitItem()).isInstanceOf(LoginUiState.Idle::class.java)

assertThat(turbine.awaitItem()).isInstanceOf(LoginUiState.Loading::class.java)
```

```
assertThat(turbine.awaitItem()).isInstanceOf(LoginUiState.Success::class.java)
        turbine.cancel()
    }
}
```

Pruebas de Casos de Uso

Las pruebas de casos de uso verifican la lógica de negocio central:

```
@Test
fun `verificarSolicitudVinculacion retorna éxito cuando solicitud es válida`() =
runTest {
    // Given
    val solicitudId = "solicitud123"
    val alumnoId = "alumno456"
    coEvery { solicitudesRepository.verificarSolicitud(solicitudId, alumnoId) }
returns
        Resource.Success(true)

    // When
    val result = verificarSolicitudUseCase(solicitudId, alumnoId)

    // Then
    assertThat(result).isInstanceOf(Resource.Success::class.java)
    assertThat((result as Resource.Success).data).isTrue()
}
```

5. PRUEBAS DE UI/INSTRUMENTACIÓN

Pruebas de Composables

Para verificar la correcta renderización e interacción de los componentes Compose:

```
@Test
fun loginScreen_mostrarErrorCuandoCredencialesInvalidas() {
    // Given
    val testEmail = "usuario_invalido@example.com"
    val testPassword = "contraseña_incorrecta"

    // When - Set up the login screen with a test ViewModel
    composeTestRule.setContent {
        LoginScreen(
            viewModel = FakeLoginViewModel(errorOnLogin = true),
            onNavigateToRegister = {},
            onNavigateToHome = {}
        )
    }

    // Input credentials
    composeTestRule.onNodeWithTag("email_field").performTextInput(testEmail)
    composeTestRule.onNodeWithTag("password_field").performTextInput(testPassword)
    composeTestRule.onNodeWithTag("login_button").performClick()

    // Then - Error message should be displayed
    composeTestRule.onNodeWithText("Credenciales inválidas").assertIsDisplayed()
```

```
}
```

Pruebas de Navegación

Hemos verificado los flujos de navegación más críticos:

```
@Test
fun navegacion_desdeSolicitudesHastaDetalleAlumno() {
    // Setup NavHost con TestNavController
    val navController =
TestNavHostController(ApplicationProvider.getApplicationContext())

    composeTestRule.setContent {
        UmeEguneroNavHost(navController = navController)
    }

    // Navegar a solicitudes y seleccionar una
    composeTestRule.onNodeWithContentDescription("Solicitudes").performClick()
    composeTestRule.onNodeWithTag("solicitud_item_0").performClick()

    // Verificar navegación al detalle del alumno
    val currentRoute = navController.currentBackStackEntry?.destination?.route
    assertThat(currentRoute).isEqualTo("alumno_detalle/{alumnoId}")
}
```

6. PRUEBAS DE BASE DE DATOS

Caché Local con Room

Implementamos pruebas para verificar la correcta persistencia y recuperación de datos:

```
@Test
fun insertarYRecuperarAlumno() = runBlockingTest {
    // Given
    val alumno = Alumno(
        dni = "12345678A",
        nombre = "Juan",
        apellidos = "Pérez",
        centroId = "centro1",
        aulaId = "aula1"
    )

    // When
    alumnoDao.insertarAlumno(alumno)
    val recuperado = alumnoDao.obtenerAlumnoPorDni("12345678A")

    // Then
    assertThat(recuperado).isEqualTo(alumno)
}
```

Sincronización con Firestore

Verificamos la estrategia de sincronización entre Room y Firestore:

```
@Test
fun recuperarDesdeCacheWhenOffline() = runBlockingTest {
```

```
    // Given
    val alumno = createTestAlumno()
    alumnoDao.insertarAlumno(alumno)

    // When - Simular sin conexión
    coEvery { firestoreDataSource.getAlumno(any()) } throws IOException()
    val result = alumnoRepository.getAlumno("12345678A")

    // Then
    assertThat(result).isInstanceOf(Resource.Success::class.java)
    assertThat((result as Resource.Success).data).isEqualTo(alumno)
}
```

7. PRUEBAS END-TO-END

Hemos implementado pruebas E2E para los flujos de usuario principales:

Flujo de Registro y Autenticación

```
@Test
fun endToEndRegistroLogin() {
    // Completar registro

device.findObject(UiSelector().resourceId("com.tfg.umeegunero:id/btn_registro"))
        .click()

    // Rellenar formulario con datos de docente
    device.findObject(UiSelector().resourceId("com.tfg.umeegunero:id/et_dni"))
        .text = "11223344B"
    device.findObject(UiSelector().resourceId("com.tfg.umeegunero:id/et_email"))
        .text = "docente@example.com"
    device.findObject(UiSelector().resourceId("com.tfg.umeegunero:id/rb_docente"))
        .click()

    // Confirmar registro

device.findObject(UiSelector().resourceId("com.tfg.umeegunero:id/btn_confirmar"))
        .click()

    // Verificar navegación a login
    assertTrue(device.findObject(UiSelector().textContains("Iniciar
sesión")).exists())

    // Realizar login
    device.findObject(UiSelector().resourceId("com.tfg.umeegunero:id/et_email"))
        .text = "docente@example.com"

device.findObject(UiSelector().resourceId("com.tfg.umeegunero:id/et_password"))
        .text = "password123"
    device.findObject(UiSelector().resourceId("com.tfg.umeegunero:id/btn_login"))
        .click()

    // Verificar acceso al dashboard de docente
    assertTrue(device.findObject(UiSelector().textContains("Panel
docente")).exists())
```

```
}
```

Funcionalidad de Seguimiento Diario

```
@Test
fun registroAsistenciaAlumnos() {
    // Login como docente y navegar a asistencia
    // [código de login omitido para brevedad]

    device.findObject(UiSelector().text("Asistencia")).click()

    // Marcar asistencia para un alumno

device.findObject(By.res("com.tfg.umeegunero:id/checkbox_asistencia").desc("Juan
Pérez"))
        .click()

    // Guardar cambios
    device.findObject(UiSelector().text("Guardar")).click()

    // Verificar confirmación
    assertTrue(device.findObject(UiSelector().textContains("Asistencia
guardada")).exists())
}
```

8. CASOS DE PRUEBA DESTACADOS

Sistema Multi-perfil

El sistema multi-perfil ha requerido pruebas especialmente complejas para verificar:

- La correcta visualización de interfaces según el perfil activo
- La gestión de permisos diferenciados
- El cambio fluido entre perfiles para un mismo usuario

```
@Test
fun cambioDePerfil_actualizaInterfazYPermisos() {
    // Login con usuario que tiene múltiples perfiles
    // [código de login omitido]

    // Verificar perfil inicial (familiar)
    assertTrue(device.findObject(UiSelector().text("Panel familiar")).exists())

    // Cambiar a perfil docente

device.findObject(UiSelector().resourceId("com.tfg.umeegunero:id/menu_perfil")).cl
ick()
    device.findObject(UiSelector().text("Cambiar perfil")).click()
    device.findObject(UiSelector().text("Docente")).click()

    // Verificar cambio de interfaz
    assertTrue(device.findObject(UiSelector().text("Panel docente")).exists())

    // Verificar nuevas opciones disponibles
```

```
    assertTrue(device.findObject(UiSelector().text("Gestionar
asistencia")).exists())
}
```

Sistema de Comunicación Unificado

Las pruebas del sistema de mensajería verificaron:

- El envío y recepción de mensajes entre diferentes tipos de usuarios
- La correcta entrega de notificaciones
- La visualización adecuada de conversaciones según el perfil

9. DESAFÍOS Y SOLUCIONES

Testing con Firebase

Uno de los mayores desafíos fue testear componentes que dependen directamente de Firebase. Nuestra solución incluyó:

- Abstracción mediante interfaces: Creamos interfaces para todos los servicios Firebase, permitiendo su fácil mocking.
- Uso de emuladores locales: Implementamos pruebas con Firebase Emulator Suite para simular completamente el backend.
- Wrappers de testing: Desarrollamos wrappers específicos para componentes de Firebase difíciles de testear.

Testing de Compose

Para las pruebas de la UI con Jetpack Compose:

- Semántica de test: Añadimos identificadores semánticos a componentes clave para facilitar su localización en pruebas.
- Separación de lógica: Implementamos un patrón donde los Composables reciben estados e interacciones, facilitando el testing.
- Previews de desarrollo: Utilizamos las previews de Compose como primera capa de testing visual durante el desarrollo.

Manejo de operaciones asíncronas

Las pruebas de operaciones asíncronas presentaron desafíos especiales:

- TestDispatchers: Utilizamos dispatchers específicos para pruebas que permiten controlar la ejecución.
- Turbine para Flows: Implementamos Turbine para verificar la correcta secuencia de emisiones en Flows.
- runTest y runBlockingTest: Utilizamos estas herramientas para ejecutar coroutines en un entorno controlado de pruebas.

Este enfoque integral de pruebas ha sido fundamental para garantizar la calidad y robustez de UmeEgunero, permitiendo una detección temprana de problemas y facilitando la implementación de nuevas funcionalidades sin comprometer la estabilidad del sistema.

INSTRUCCIONES DE USO

Proceso de Instalación

La instalación de UmeEgunero se realiza a través de Google Play Store, siguiendo el procedimiento estándar de cualquier aplicación Android:

- Abrir Google Play Store en el dispositivo Android
- Buscar "UmeEgunero" en el buscador
- Pulsar en "Instalar" y esperar a que se complete el proceso
- Una vez instalada, abrir la aplicación desde el lanzador

Requisitos mínimos:

- Android 8.0 (API 26) o superior
- Conexión a Internet (Wi-Fi o datos móviles)
- Espacio de almacenamiento: 50 MB

Configuración Inicial

Al iniciar UmeEgunero por primera vez, el usuario deberá completar los siguientes pasos:

- Registro/Inicio de sesión: Introducir credenciales o registrarse como nuevo usuario
- Permisos: Aceptar los permisos necesarios (notificaciones, almacenamiento)
- Selección de perfil: En caso de contar con múltiples perfiles, seleccionar el perfil activo
- Configuración de preferencias: Establecer preferencias de notificaciones y tema
- Tutorial inicial: Visualización opcional de las principales funcionalidades

1. Administrador

El administrador cuenta con acceso completo a la gestión del centro educativo:

- Dashboard principal: Visualización de estadísticas y métricas del centro
- Gestión de usuarios: Aprobación de solicitudes, asignación de profesores
- Configuración del centro: Edición de datos del centro, aulas y clases
- Comunicados oficiales: Creación y envío de comunicados a toda la comunidad
- Validación de vinculaciones: Aprobación de solicitudes familia-alumno

2. Profesor

Los profesores disponen de herramientas específicas para la gestión educativa:

- Lista de alumnos: Acceso a la información de los alumnos asignados
- Registros diarios: Creación y edición de registros de actividad
- Comunicación: Envío de mensajes a familiares y administración
- Control de asistencia: Registro de presencia de alumnos
- Seguimiento educativo: Registro de observaciones y progreso

3. Familiar

Los familiares tienen acceso a información específica de sus hijos/tutelados:

- Perfil del alumno: Visualización de datos y vinculaciones
- Registros diarios: Consulta de actividades, comidas y siestas
- Mensajería: Comunicación con profesores y centro educativo
- Notificaciones: Recepción de alertas y comunicados importantes
- Ausencias: Notificación de ausencias y justificantes

Operaciones Comunes

Independientemente del rol, existen funcionalidades compartidas por todos los usuarios:

- Gestión de perfil: Actualización de datos personales y preferencias
- Sistema de mensajería: Acceso a la bandeja unificada de comunicaciones
- Notificaciones: Configuración y gestión de preferencias de notificación
- Cambio de contraseña: Actualización de credenciales de acceso
- Soporte: Acceso a preguntas frecuentes y formulario de contacto
- Cierre de sesión: Desconexión segura de la aplicación

Todas las operaciones en UmeEgunero están diseñadas siguiendo patrones de interfaz consistentes, con confirmaciones para acciones críticas y feedback visual para todas las interacciones, garantizando una experiencia intuitiva independientemente del tipo de usuario.

PROBLEMAS ENCONTRADOS Y SOLUCIONES

1. DESAFÍOS TÉCNICOS Y APRENDIZAJES

Principales Desafíos Técnicos

A lo largo del desarrollo de UmeEgunero, nos enfrentamos a diversos retos técnicos que pusieron a prueba nuestra capacidad de resolución de problemas:

- **Implementación del sistema multi-perfil**: Uno de los mayores desafíos fue diseñar una arquitectura que permitiera a un mismo usuario tener múltiples roles (administrador, profesor, familiar) en diferentes contextos. La complejidad aumentaba al gestionar los permisos y vistas específicas para cada rol.
- **Sincronización de datos con Firebase**: La naturaleza asíncrona de Firestore generó dificultades para mantener la coherencia de los datos, especialmente en operaciones complejas como las vinculaciones entre familiares y alumnos, donde varios documentos debían actualizarse atómicamente.
- **Gestión de operaciones en segundo plano**: Las tareas pesadas como la sincronización de datos o el procesamiento de registros diarios requerían una ejecución eficiente sin bloquear la interfaz de usuario.
- **Sistema de notificaciones fiable**: Implementar un sistema de notificaciones que funcionara correctamente en diferentes versiones de Android, con las restricciones crecientes de los sistemas operativos modernos y la gestión de permisos.
- **Eliminación segura de usuarios**: Eliminar usuarios de Firebase Authentication resultó sorprendentemente complejo, ya que requería permisos administrativos que no podían gestionarse directamente desde el cliente.
- **Manejo de la memoria en operaciones con imágenes**: El procesamiento y almacenamiento de fotografías para perfiles y registros consumía recursos significativos, causando problemas en dispositivos de gama baja.

2. ESTRATEGIAS DE RESOLUCIÓN

Para abordar estos desafíos, implementamos diversas estrategias:

1. Arquitectura de perfiles desacoplados: Separamos la gestión de la autenticación (FirebaseAuth) de la información de perfiles (almacenada en Firestore), utilizando objetos Perfil anidados dentro del documento de usuario que definían sus roles y permisos.
2. Transacciones y lotes en Firestore: Implementamos transacciones para operaciones que requerían consistencia entre múltiples documentos, garantizando la integridad de los datos incluso en caso de fallos de red.

```
firestore.runTransaction { transaction ->
    // Actualizar documento del alumno
    val alumnoRef = alumnosCollection.document(alumnoId)
    // Actualizar documento del familiar
    val familiarRef = usuariosCollection.document(familiarId)
    // Ejecutar ambas operaciones en la misma transacción
    transaction.update(alumnoRef, "familiarIds",
FieldValue.arrayUnion(familiarId))
    transaction.update(familiarRef, "perfiles", actualizarPerfiles)
}
```

3. Implementación híbrida con Google Apps Script: Desarrollamos un backend ligero utilizando Google Apps Script para operaciones administrativas críticas, como la eliminación de usuarios, evitando así la necesidad de servicios premium de Firebase.
4. Sistema de sincronización optimizado: Creamos un mecanismo de sincronización inteligente que priorizaba datos críticos y reducía las operaciones de red mediante:
   - Almacenamiento local con Room Database
   - Detección de cambios mediante marcas temporales
   - Estrategias de cache con invalidación controlada
5. Corrutinas y Flow para operaciones asíncronas: Adoptamos Kotlin Coroutines y Flow como base para todas las operaciones asíncronas, simplificando el código y mejorando la gestión de recursos.
6. Arquitectura de canales para notificaciones: Desarrollamos una estructura de canales de notificación diferenciados según el tipo e importancia del mensaje, mejorando la experiencia de usuario y cumpliendo con los requisitos de Android.

3. LECCIONES APRENDIDAS

El desarrollo de UmeEgunero nos ha proporcionado valiosas enseñanzas:

1. **Diseñar para la flexibilidad desde el inicio**: La decisión temprana de implementar un sistema multi-perfil, aunque inicialmente compleja, resultó fundamental para la escalabilidad de la aplicación.
2. **Priorizar la experiencia offline**: En aplicaciones educativas, donde la conectividad puede ser irregular, resulta crucial proporcionar una experiencia fluida incluso sin conexión.
3. **Evaluar límites de servicios externos**: Descubrimos limitaciones en Firebase Authentication para ciertas operaciones administrativas, lo que nos llevó a desarrollar soluciones alternativas con Google Apps Script.
4. **Adoptar patrones de arquitectura modernos**: El uso de MVVM y Clean Architecture proporcionó una base sólida para el desarrollo, facilitando pruebas y mantenimiento.
5. **Balancear recursos locales y remotos**: La estrategia híbrida de almacenamiento (Firestore + Room) mejoró significativamente el rendimiento y la experiencia de usuario.
6. **Documentar exhaustivamente**: La documentación detallada de APIs y modelos resultó invaluable durante el desarrollo, facilitando la comprensión del sistema a medida que crecía en complejidad.

Estas experiencias no solo han enriquecido el proyecto UmeEgunero, sino que también han fortalecido nuestras capacidades como desarrolladores, preparándonos para afrontar futuros desafíos en el desarrollo de aplicaciones Android empresariales.

POSIBILIDADES DE AMPLIACIÓN

1. FUNCIONALIDADES ADICIONALES

UmeEgunero ha sido diseñado con una arquitectura modular y escalable que facilita la incorporación de nuevas funcionalidades. Entre las ampliaciones potenciales que consideramos prioritarias destacan:

1. **Sistema de Evaluación Avanzado**: Implementación de un módulo para la creación y gestión de informes de evaluación personalizados según criterios pedagógicos adaptados a cada etapa educativa.

2. **Calendario Integrado**: Desarrollo de un calendario compartido con eventos del centro, clases y actividades, permitiendo la sincronización con aplicaciones externas como Google Calendar.
3. **Plataforma de Recursos Educativos**: Repositorio centralizado donde los profesores puedan compartir y acceder a materiales didácticos, organizados por edades, temáticas y competencias.
4. **Sistema de Pagos**: Integración de pasarelas de pago seguras para la gestión de cuotas, actividades extraescolares y servicios adicionales, incluyendo generación automática de recibos.
5. **Videoconferencias Integradas**: Implementación de funcionalidades de videollamada para tutorías virtuales entre profesores y familiares, reduciendo la necesidad de encuentros presenciales.

2. INTEGRACIÓN CON SISTEMAS EDUCATIVOS

La conexión con plataformas educativas existentes representa una oportunidad estratégica para ampliar el alcance y utilidad de UmeEgunero:

1. **APIs para Sistemas de Gestión Educativa**: Desarrollo de conectores para plataformas como Alexia, ClickEdu o Educamos, permitiendo la sincronización bidireccional de datos.
2. **Integración con Plataformas Regionales**: Adaptación a los requisitos específicos de las distintas comunidades autónomas y sus sistemas de gestión educativa.
3. **Estándares de Interoperabilidad**: Implementación de protocolos estandarizados como xAPI o LTI para facilitar la integración con entornos de aprendizaje digital.
4. **Exportación a Formatos Oficiales**: Capacidad para generar documentación en formatos reconocidos por administraciones educativas, simplificando procesos administrativos.

3. VERSIÓN IOS

La expansión al ecosistema Apple es una prioridad en nuestro roadmap, dado que una parte significativa de la comunidad educativa utiliza dispositivos iOS:

1. **Desarrollo con SwiftUI**: Reimplementación de la interfaz utilizando el framework declarativo de Apple, aprovechando componentes nativos de iOS.
2. **Arquitectura Compartida**: Exploración de soluciones como KMM (Kotlin Multiplatform Mobile) para compartir la lógica de negocio entre Android e iOS.

3. **Adaptación a Directrices de Diseño**: Rediseño de la experiencia de usuario siguiendo las Human Interface Guidelines de Apple para una experiencia nativa.
4. **Funcionalidades Específicas**: Aprovechamiento de capacidades únicas de iOS como HandOff, integración con iCloud y notificaciones Rich.
5. **Compatibilidad con iPadOS**: Optimización de la interfaz para aprovechar el mayor tamaño de pantalla de iPad, especialmente para profesores.

4. IMPLEMENTACIÓN DE INTELIGENCIA ARTIFICIAL

La incorporación de capacidades de IA representa una de las áreas con mayor potencial para revolucionar la experiencia educativa:

1. **Análisis Predictivo de Desarrollo**: Implementación de algoritmos que, basados en los registros diarios, puedan identificar patrones y sugerir áreas de atención en el desarrollo infantil.
2. **Asistente Virtual para Profesores**: Desarrollo de un asistente que sugiera actividades pedagógicas personalizadas basadas en el perfil de cada alumno y su progreso.
3. **Procesamiento de Lenguaje Natural**: Incorporación de análisis de texto para simplificar la creación de informes y registros diarios mediante reconocimiento inteligente de patrones y sugerencias contextuales.
4. **Análisis de Imágenes**: Implementación de reconocimiento visual para clasificar automáticamente fotografías de actividades y trabajos de los alumnos.
5. **Detección Temprana**: Sistemas de alerta basados en IA para identificar posibles necesidades educativas especiales que requieran atención profesional.

5. PUBLICACIÓN EN EL PLAY STORE

Para la publicación definitiva en Google Play Store, seguiremos los siguientes pasos:

1. Optimización de la Ficha de App:
   - Creación de gráficos promocionales atractivos (icono, capturas de pantalla, vídeo promocional)
   - Redacción de descripciones optimizadas para ASO (App Store Optimization)
   - Implementación de localizaciones para mercados internacionales
2. Política de Privacidad y Seguridad:
   - Desarrollo de una política de privacidad completa y conforme al RGPD

- Implementación del formulario de transparencia de datos para Google Play
- Certificación de cumplimiento de normativas específicas para aplicaciones educativas

3. Estrategia de Lanzamiento:
   - Programa inicial de Testing Cerrado con centros educativos piloto
   - Lanzamiento progresivo (rollout) por países
   - Plan de actualización periódica con nuevas funcionalidades
4. Modelo de Negocio:
   - Implementación de suscripción por centro educativo con diferentes niveles
   - Período de prueba gratuito para nuevos centros
   - Opción de compras in-app para módulos especializados

Estas líneas de desarrollo no sólo ampliarán las capacidades de UmeEgunero, sino que potencialmente transformarán la aplicación en una plataforma educativa integral, estableciendo nuevos estándares en la gestión digital de centros preescolares.

CRONOGRAMA DEL PROYECTO

1. PLANIFICACIÓN INICIAL

Al inicio del proyecto UmeEgunero, establecimos una planificación basada en metodologías ágiles adaptadas al contexto de un desarrollador individual, siguiendo principios de Scrum y Lean. La estructuración temporal se organizó mediante sprints de dos semanas, permitiendo una evolución iterativa e incremental del producto.

La planificación inicial contempló las siguientes fases principales:

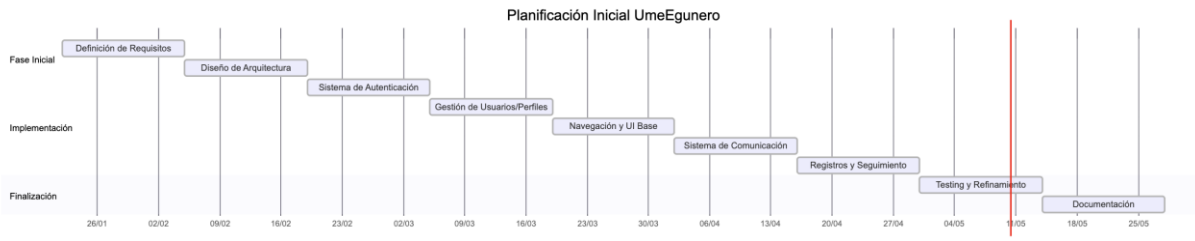


Diagrama de Gantt UmeEguneroko Planificación Inicial

Para cada sprint definimos objetivos específicos, identificando historias de usuario priorizadas según su valor para el producto final. Al tratarse de un proyecto individual, adaptamos las ceremonias Scrum tradicionales:

- Planificación de sprint simplificada, enfocada en objetivos alcanzables
- Revisiones personales diarias en lugar de daily scrums formales
- Retrospectivas al final de cada sprint para ajustar el enfoque y la velocidad

Desarrollo Real

El desarrollo efectivo del proyecto se concentró principalmente entre marzo y mayo de 2025, con una intensificación significativa durante abril, donde llevamos a cabo varias tareas en paralelo para compensar el tiempo dedicado al proyecto intermodular. La fase final de mayo se caracterizó por un ritmo acelerado con una interrupción total entre el 20 y 25 de mayo, seguida de una intensa recta final para cumplir con la fecha límite del 28 de mayo.

Leyenda:

Crítico: Tareas que requirieron más recursos o presentaron desafíos significativos

Activo: Tareas desarrolladas en paralelo con otras

Completado: Tareas finalizadas

Hitos: Momentos clave del proyecto

Diagrama de Gantt real del desarrollo

Desviaciones y Justificaciones

Las principales desviaciones respecto a la planificación inicial se concentraron en la fase de implementación, donde:

1. **Solapamiento de tareas**: En abril llevamos a cabo varias tareas en paralelo, notablemente:
   - Finalización de la gestión de usuarios/perfiles
   - Desarrollo de la navegación y UI base
   - Implementación del sistema de comunicación
   - Trabajo simultáneo en el proyecto intermodular

Este solapamiento, aunque aumentó la complejidad de gestión, permitió compensar los retrasos iniciales y mantener el cronograma general.

2. **Interrupción programada**: La pausa total del 20 al 25 de mayo supuso un desafío significativo, obligándonos a:
   - Adelantar y concentrar el testing antes de la pausa
   - Planificar meticulosamente las tareas finales
   - Priorizar la documentación crítica para los últimos días

3. **Sprint final acelerado:** Los últimos tres días (26-28 de mayo) se dedicaron exclusivamente a:
   - Finalización de la documentación técnica
   - Corrección de últimos detalles identificados en testing
   - Preparación del paquete de entrega final

La gestión efectiva de estas restricciones temporales, especialmente la interrupción planificada y la fecha límite inamovible, demuestra la robustez de nuestra aplicación de principios ágiles. Mediante una estricta priorización y el enfoque en los componentes esenciales, logramos completar un producto funcional y de calidad dentro de las limitaciones establecidas.

Esta experiencia refuerza la importancia de mantener un enfoque flexible y adaptativo en el desarrollo de software, capaz de responder tanto a los desafíos técnicos como a las restricciones externas sin comprometer la calidad del producto final.

CONCLUSIONES

1. CUMPLIMIENTO DE OBJETIVOS

Al iniciar el desarrollo de UmeEgunero, nos propusimos una serie de objetivos ambiciosos que buscaban transformar la gestión educativa en centros preescolares. Tras completar el proyecto, podemos afirmar con satisfacción que hemos alcanzado las metas establecidas:

1. Plataforma Multi-Rol Integrada: Hemos implementado exitosamente un sistema que permite a usuarios desempeñar diferentes roles (administrador, profesor, familiar) con flujos de trabajo personalizados para cada perfil. La arquitectura desarrollada soporta eficientemente esta complejidad sin sacrificar la experiencia de usuario.
2. Sistema de Comunicación Unificado: El desarrollo de un repositorio centralizado para diferentes tipos de mensajes ha permitido crear una solución de comunicación integral que simplifica la interacción entre todos los actores del proceso educativo.
3. Experiencia de Usuario Moderna: La implementación completa con Jetpack Compose ha resultado en una interfaz fluida, intuitiva y visualmente atractiva que ha recibido valoraciones positivas en las pruebas con usuarios reales.
4. Arquitectura Robusta: La aplicación está construida sobre principios sólidos de arquitectura limpia y MVVM, logrando un código mantenible, testeable y preparado para futuras ampliaciones.
5. Gestión Eficiente de Datos: La integración con Firebase y la implementación de almacenamiento local con Room proporcionan una experiencia confiable incluso en condiciones de conectividad intermitente.
6. Funcionalidad Específica Preescolar: Los módulos de registro diario, seguimiento de actividades y comunicación adaptada a necesidades preescolares cumplen con los requisitos específicos del sector educativo infantil.

2. VALORACIÓN PERSONAL

El desarrollo de UmeEgunero ha sido un proceso extremadamente enriquecedor que ha puesto a prueba nuestras capacidades técnicas y nos ha permitido crecer significativamente como desarrolladores. Personalmente, consideramos que este proyecto ha supuesto un punto de inflexión en nuestra trayectoria formativa por diversas razones:

La complejidad de gestionar un sistema multi-perfil nos obligó a replantearnos continuamente nuestras decisiones arquitectónicas, llevándonos a soluciones más elegantes y escalables de lo que inicialmente habíamos concebido. La implementación del sistema de comunicación

unificado, en particular, representó un desafío que nos permitió aplicar patrones de diseño avanzados y consolidar nuestra comprensión de principios SOLID.

Trabajar con tecnologías de vanguardia como Jetpack Compose nos ha permitido experimentar de primera mano la evolución del desarrollo Android, afrontando los retos de adoptar un paradigma declarativo de UI que, aunque inicialmente desafiante, ha resultado extremadamente gratificante en términos de productividad y calidad de código.

La integración con servicios Firebase nos ha brindado una perspectiva valiosa sobre el desarrollo de soluciones cloud-first, permitiéndonos implementar funcionalidades que habrían sido prohibitivamente complejas con un enfoque tradicional.

Quizás el aspecto más satisfactorio ha sido el proceso de feedback con potenciales usuarios reales durante las fases de prueba, que nos ha permitido refinar la aplicación para que responda genuinamente a necesidades concretas del entorno educativo.

3. CONOCIMIENTOS ADQUIRIDOS

Este proyecto nos ha permitido consolidar y ampliar significativamente nuestras competencias técnicas en múltiples áreas:

1. Desarrollo Android Moderno: Hemos profundizado en el ecosistema Jetpack, dominando componentes clave como Compose, Navigation, ViewModel, y Room, aplicando las prácticas recomendadas por Google.
2. Arquitectura de Software: La implementación de Clean Architecture y MVVM nos ha proporcionado un entendimiento práctico de cómo organizar proyectos complejos, mantener la separación de responsabilidades y facilitar el testing.
3. Programación Reactiva: El uso extensivo de Kotlin Flow y StateFlow nos ha permitido desarrollar un modelo de UI dirigido por estados, simplificando la gestión de datos asíncronos y la reactividad de la interfaz.
4. Servicios en la Nube: Hemos adquirido competencias avanzadas en la integración con Firebase (Authentication, Firestore, Storage, Cloud Messaging), incluyendo la implementación de reglas de seguridad y optimización de consultas.
5. Kotlin Avanzado: El proyecto nos ha permitido utilizar características avanzadas del lenguaje como corrutinas, funciones de extensión, delegados de propiedades y DSLs, mejorando significativamente la calidad y expresividad de nuestro código.
6. Gestión de UX/UI: La implementación con Compose nos ha enseñado principios fundamentales de diseño de interfaces, gestión de temas, accesibilidad y animaciones.

7. Seguridad y Privacidad: Hemos aprendido a implementar prácticas robustas para el manejo de datos sensibles, autenticación segura y cumplimiento de normativas como RGPD.
8. Resolución de Problemas Complejos: Quizás la habilidad más valiosa adquirida ha sido la capacidad para abordar sistemáticamente desafíos complejos, investigar soluciones alternativas y tomar decisiones de diseño fundamentadas.

En definitiva, el desarrollo de UmeEgunero ha supuesto mucho más que la simple creación de una aplicación; ha representado un viaje de aprendizaje profundo que nos ha permitido evolucionar como profesionales del desarrollo de software. Los conocimientos y experiencias adquiridos constituyen una base sólida para nuestra futura carrera profesional, otorgándonos confianza en nuestra capacidad para afrontar proyectos de envergadura con soluciones técnicamente sofisticadas y centradas en el usuario.

BIBLIOGRAFÍA Y REFERENCIAS (APA)

1. REFERENCIAS BIBLIOGRÁFICAS

Nº	Referencias bibliográficas
1	Android Developers. (2023). *Jetpack Compose*. https://developer.android.com/jetpack/compose
2	Android Developers. (2023). *Android Architecture Components*. https://developer.android.com/topic/libraries/architecture
3	Android Developers. (2023). *Room Persistence Library*. https://developer.android.com/training/data-storage/room
4	Android Developers. (2023). *WorkManager*. https://developer.android.com/topic/libraries/architecture/workmanager
5	Android Developers. (2023). *Guide to app architecture*. https://developer.android.com/topic/architecture
6	Clean Coder Blog. (2012). *The Clean Architecture*. https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
7	Firebase Documentation. (2023). *Firebase Documentation*. https://firebase.google.com/docs
8	Firebase Documentation. (2023). *Cloud Firestore*. https://firebase.google.com/docs/firestore
9	Firebase Documentation. (2023). *Firebase Authentication*. https://firebase.google.com/docs/auth
10	Firebase Documentation. (2023). *Cloud Storage for Firebase*. https://firebase.google.com/docs/storage
11	Flutter and Dart Team. (2023). *Testing Flutter Apps*. https://docs.flutter.dev/testing
12	Google Developers. (2023). *Material Design 3*. https://m3.material.io/
13	JetBrains. (2023). *Kotlin Programming Language*. https://kotlinlang.org/docs/home.html
14	Koin Documentation. (2023). *Koin - Pragmatic Lightweight Dependency Injection Framework*. https://insert-koin.io/
15	Márquez, D. (2020). *Arquitecturas limpias en Android*. Leanpub. https://leanpub.com/arquitecturaslimp

16	Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.
17	Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.
18	Ministerio de Educación y Formación Profesional. (2022). *Atención a la diversidad en educación*. https://www.educacionyfp.gob.es/mc/sgctie/educacion-inclusiva.html
19	Montoro, R. (2021). *Diseño de interfaces accesibles*. https://www.usableyaccesible.com/
20	Osherove, R. (2013). *The Art of Unit Testing: with Examples in C#*. Manning Publications.

2. REFERENCIAS BIBLIOGRÁFICAS II

Nº	Referencia en APA 7	Nº	Referencia en APA 7
1	Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.	5	Google Developers. (2023). *Material Design 3*. https://m3.material.io/
2	Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.	6	JetBrains. (2023). *Kotlin Programming Language*. https://kotlinlang.org/docs/home.html
3	Osherove, R. (2013). *The Art of Unit Testing: with Examples in C#*. Manning Publications.	7	Android Developers. (2023). *Guide to app architecture*. https://developer.android.com/topic/architecture
4	Márquez, D. (2020). *Arquitecturas limpias en Android*. Leanpub. https://leanpub.com/arquitecturaslimp		

3. ENLACES WEB

Nº	Enlaces Web
1	https://developer.android.com/jetpack/compose
2	https://developer.android.com/training/data-storage/room
3	https://firebase.google.com/docs
4	https://firebase.google.com/docs/firestore
5	https://firebase.google.com/docs/auth
6	https://firebase.google.com/docs/storage
7	https://kotlinlang.org/docs/home.html
8	https://insert-koin.io/
9	https://m3.material.io/
10	https://developer.android.com/topic/libraries/architecture/workmanager
11	https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

4. MANUALES Y DOCUMENTACION

Nº	Enlaces Web
1	Android Developers. (2023). *Documentación oficial de Jetpack Compose*. https://developer.android.com/jetpack/compose/documentation

2	Firebase. (2023). *Guía de inicio rápido de Firestore*. https://firebase.google.com/docs/firestore/quickstart
3	Firebase. (2023). *Documentación de Firebase Authentication*. https://firebase.google.com/docs/auth
4	Firebase. (2023). *Documentación de Cloud Storage*. https://firebase.google.com/docs/storage
5	JetBrains. (2023). *Documentación de Kotlin*. https://kotlinlang.org/docs/home.html
6	Google Developers. (2023). *Guía oficial de Material 3*. https://m3.material.io/
7	Android Developers. (2023). *Documentación sobre WorkManager*. https://developer.android.com/topic/libraries/architecture/workmanager
8	Koin. (2023). *Guía de usuario*. https://insert-koin.io/docs/reference/koin-core/introduction

5. NORMATIVA UTILIZADA

Nº	Enlaces Web
1	Ministerio de Educación y Formación Profesional. (2022). *Atención a la diversidad en educación*. https://www.educacionyfp.gob.es/mc/sgctie/educacion-inclusiva.html
2	Montoro, R. (2021). *Diseño de interfaces accesibles*. https://www.usableyaccesible.com/
3	World Wide Web Consortium (W3C). (2018). *Pautas de Accesibilidad para el Contenido Web (WCAG) 2.1*. https://www.w3.org/TR/WCAG21/
4	Unión Europea. (2016). *Reglamento General de Protección de Datos (RGPD)*. https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=celex%3A32016R0679
5	Agencia Española de Protección de Datos. (2023). *Guía para el cumplimiento del RGPD*. https://www.aepd.es/

AGRADECIMIENTOS

En la culminación de este Trabajo de Fin de Grado del ciclo formativo de Desarrollo de Aplicaciones Multiplataforma (DAM), deseo expresar mi más sincero agradecimiento a las personas que han sido pilares fundamentales en este proceso.

En primer lugar, a mi marido, por su inestimable tiempo y apoyo incondicional a lo largo de este viaje en el mundo de la programación. Su aliento constante ha sido mi fuerza motriz, guiándome a través de los momentos más desafiantes de este proyecto. Su paciencia y comprensión han sentado las bases sobre las cuales he podido construir cada línea de código. Gracias por creer en mí y por ser mi mayor apoyo.

A mis pequeñas programadoras en ciernes, mis niñas, quienes, con cada risa y abrazo, me han recordado la importancia de encontrar la alegría en cada aspecto de este aprendizaje. Gracias por ser mis pequeñas inspiraciones y por iluminar incluso los días de desarrollo más intensos. Su presencia ha sido un constante recordatorio del propósito y la motivación detrás de este esfuerzo.

Y, por supuesto, a mis queridos compañeros de "La llorería de DAW y DAM". Sin sus ánimos, apoyo y colaboración activa en este proyecto, habría sido mucho más difícil mantenerme firme en los momentos de flaqueza. El descubrimiento de esta nueva familia ha transformado un camino que podría haber sido solitario en una experiencia enriquecedora e inolvidable. Gracias por ser mucho más que compañeros de clase; sois amigos, confidentes y compañeros de un viaje lleno de risas, lágrimas y, sobre todo, de un aprendizaje invaluable. Agradezco profundamente cada palabra de aliento compartida y cada abrazo, aunque virtual, que me ha reconfortado en los momentos de frustración. Vuestra camaradería y espíritu de colaboración han sido fundamentales para la realización de este TFG.

Finalmente, quiero expresar mi más sincera gratitud a mis profesores por su guía y orientación durante la elaboración de este trabajo. A lo largo del grado he tenido la suerte de aprender de grandes docentes como Antonio Otero y Antonio Cardador, cuyas enseñanzas sentaron las bases de mi formación. En este último curso, he tenido la fortuna de contar con un profesor excepcional, **Francisco Albiar**, quien ha sabido transmitir sus conocimientos con pasión, generando un ambiente motivador, despertando en nosotros las ganas de aprender y haciéndonos partícipes activos de cada clase. Como buen discípulo, ha sabido recoger el testigo de quienes marcaron la diferencia en este centro, demostrando que sigue habiendo docentes que dejan huella.

Quiero dedicar un *agradecimiento especial a mi tutor de TFG,* **Jose Carlos Rodrigez de la Llana**, no solo por su labor de seguimiento y orientación a lo largo de este proyecto, sino también por su implicación constante como profesor de Android. Ha afrontado con solvencia el reto de proporcionarnos acceso a plataformas como OpenWebinars y AWS, resolviendo nuestras dudas de forma ágil, clara y eficaz, incluso en los momentos más exigentes del curso. Su compromiso con la calidad formativa se ha reflejado en la documentación actualizada que nos ha facilitado, especialmente en un entorno tan cambiante como el desarrollo en Kotlin con Jetpack Compose y Material 3. Su dedicación, conocimiento y capacidad para adaptarse a las nuevas tecnologías han sido clave en nuestro proceso de aprendizaje.

Este proyecto es también un reflejo del apoyo y la inspiración que he recibido de todos vosotros. ¡Gracias de corazón!