TRABAJO DE GRADO
Opción Seminario-Diplomado.

Listado de asistencia

Corporación Universitaria Remington.
Ingeniería.
Ingeniería de Sistemas.

Lizandro Ancizar Lopez Lopez.
Jonatan Stick Campos Nuñez.
Opción de Trabajo de grado Seminario-Diplomado.
2025.

Tabla de contenido

Resumen

Este trabajo de grado presenta el diseño e implementación de una aplicación móvil para la gestión de estudiantes y la toma de asistencia en tiempo real en instituciones educativas. La aplicación fue desarrollada en Flutter y utiliza SQLite para almacenamiento local y una API en PHP para sincronización de datos con un servidor MySQL. La solución integra herramientas modernas como Ngrok para exponer servicios en entornos de desarrollo y Docker para estandarizar la configuración del servidor. Como resultado, se obtiene una herramienta multiplataforma, eficiente y de fácil uso que permite reducir errores en el llamado a lista y agilizar el reporte de asistencia.

Palabras clave

- Flutter
- SQLite
- MySQL
- API REST
- Asistencia

Pregunta orientadora de la búsqueda

¿Cómo desarrollar una aplicación móvil multiplataforma que permita gestionar estudiantes y registrar asistencia de manera dinámica, utilizando Flutter y bases de datos locales con sincronización a un servidor MySQL mediante API REST en PHP?

Metodología de Búsqueda de la Información

Para el desarrollo de este trabajo de grado se empleó un enfoque de investigación aplicada, con el objetivo de identificar las herramientas y tecnologías más adecuadas para la creación de una aplicación móvil multiplataforma. La metodología de búsqueda se estructuró en las siguientes fases:

1. **Definición de la pregunta de investigación**

   Se formuló la pregunta orientadora:

   ¿Cómo desarrollar una aplicación móvil multiplataforma que permita gestionar estudiantes y registrar asistencia en tiempo real, utilizando Flutter con base de datos local y sincronización remota vía API REST?

2. **Identificación de palabras clave**

   Se determinaron los términos más relevantes para las búsquedas:

   - "Flutter CRUD SQLite"
   - "Flutter REST API integration"
   - "Docker MySQL PHP development"
   - "Ngrok expose local server"
   - "mobile attendance system app"

3. **Fuentes de información consultadas**

   Se emplearon fuentes de información académica y técnica:

   - Documentación oficial de **Flutter**, **SQLite** y **MySQL**.
   - Guías y manuales de **PHP** y buenas prácticas de diseño de API REST.
   - Foros de discusión y comunidades de desarrolladores (Stack Overflow, Medium, Dev.to).
   - Artículos y publicaciones académicas relacionados con desarrollo de aplicaciones móviles educativas.

4. **Estrategia de búsqueda**
   - Uso de buscadores como Google Scholar y motores de búsqueda técnicos.
   - Revisión sistemática de tutoriales y ejemplos de implementación para validar la viabilidad de Flutter en entornos de producción.
   - Pruebas controladas con **Ngrok** y **Docker** para evaluar escenarios de despliegue y conexión en redes externas.

5. **Validación y selección de información**
   - Se priorizaron fuentes confiables, actualizadas y con documentación oficial.

- Se realizaron prototipos de prueba para verificar la aplicabilidad de la información encontrada.
- La selección final se centró en herramientas que ofrecieran estabilidad, facilidad de uso y soporte comunitario.

Sustentación teórica de la pregunta

El uso de aplicaciones móviles para la gestión de procesos educativos ha incrementado de forma significativa en la última década, debido a la masificación de los dispositivos inteligentes y la necesidad de digitalizar los procedimientos institucionales. Según Al-Musharraf y Al-Khateeb (2021), las aplicaciones móviles en entornos académicos permiten mejorar la interacción estudiante-docente, optimizar tiempos y ofrecer trazabilidad en la información.

Flutter, el framework de desarrollo multiplataforma creado por Google, ha emergido como una solución eficiente para construir aplicaciones nativas desde un único código fuente. Diversos estudios destacan que su arquitectura basada en widgets y el uso del lenguaje Dart favorecen un ciclo de desarrollo ágil y una experiencia de usuario consistente (Soto et al., 2023). Esta característica lo convierte en una herramienta adecuada para proyectos académicos que requieren rapidez de desarrollo y despliegue en múltiples plataformas.

Para el manejo de datos locales, se seleccionó SQLite como motor de base de datos embebido. SQLite es ampliamente reconocido por su ligereza y confiabilidad, siendo utilizado incluso en navegadores, sistemas operativos y aplicaciones de alta concurrencia (Owens & Allen, 2020). Su implementación permite que la aplicación funcione en modo offline, una característica relevante en entornos donde la conectividad no siempre está garantizada.

En cuanto al almacenamiento centralizado, se utilizó MySQL junto con una API REST desarrollada en PHP. Las API REST facilitan la interoperabilidad entre sistemas, ofreciendo un modelo escalable y estándar de comunicación cliente-servidor (Fielding, 2000). Esto permite mantener la información sincronizada en tiempo real y disponible para otros módulos de gestión institucional.

Adicionalmente, se emplearon herramientas de soporte como **Ngrok** para exponer el servidor local durante las fases de prueba, lo que permitió la validación desde dispositivos físicos sin necesidad de un despliegue permanente, y **Docker** para estandarizar el entorno de ejecución, garantizando que las pruebas fueran reproducibles y evitando conflictos de configuración (Merkel, 2014).

De esta forma, el proyecto responde a la pregunta orientadora, integrando tecnologías modernas, multiplataforma y escalables, con una base teórica sólida que sustenta su aplicabilidad en el contexto académico.

Diagramas y modelos

Diagrama de casos de uso

Representa cómo interactúan los actores (docente, sistema) con la aplicación.

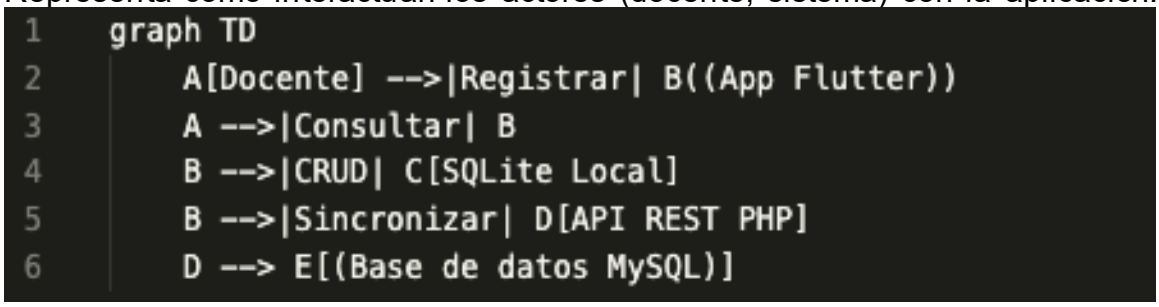

Figura 1. Diagrama de casos de uso.

Descripción:

- El docente puede **gestionar estudiantes** (CRUD).
- Puede **tomar asistencia** desde la app.
- Los datos se guardan en **SQLite** localmente y se sincronizan con **MySQL** a través de la API REST.

Diagrama de arquitectura

Muestra la estructura general de la solución.

```
flowchart TB
    subgraph Dispositivo Móvil
        A[Flutter App] --> B[SQLite DB]
    end

    subgraph Servidor
        C[API REST en PHP] --> D[(MySQL)]
    end

    A <--> C
```

Figura 2. Diagrama de arquitectura.

Descripción:

- La app maneja datos localmente con **SQLite** para funcionar offline.
- Cada nuevo estudiante se envía al servidor vía **HTTP POST**.
- El servidor guarda en **MySQL**, permitiendo reportes centralizados.

Modelo Entidad-Relación (simplificado)

Representa la tabla principal estudiantes y su estructura.

```
erDiagram
    ESTUDIANTES {
        INT id PK
        VARCHAR nombre
        VARCHAR codigo
        TINYINT activo
    }
    ASISTENCIA {
        INT id PK
        DATE fecha
        INT estudiante_id FK
        ENUM estado
    }
    ESTUDIANTES ||--o{ ASISTENCIA : registra
```

Figura 3. Diagrama de arquitectura.

Descripción:

- La tabla estudiantes guarda información básica.
- La tabla asistencia guarda cada registro de presencia/ausencia con fecha y referencia al estudiante.

Diagrama de Flujo del Llamado a Lista

Ayuda a explicar el proceso dinámico de asistencia.

```
flowchart TD
    A[Inicio de Pantalla Llamado a Lista] --> B[Cargar estudiantes desde SQLite]
    B --> C[Mostrar lista con botones Presente/Ausente]
    C --> D{Marcar asistencia?}
    D -->|Sí| E[Actualizar estado en mapa de asistencia]
    D -->|No| C
    E --> F[Guardar asistencia]
    F --> G[Mostrar gráfico de resultados]
    G --> H[Enviar datos al servidor vía API]
    H --> I[Fin]
```

Figura 4. Diagrama de flujo de llamado a lista.

Descripción del **Diagrama de Flujo del Llamado a Lista**:

El flujo inicia cuando el docente abre la pantalla de llamado a lista: la app **carga los estudiantes desde SQLite** y presenta una lista interactiva donde cada tarjeta permite marcar **Presente** o **Ausente** con feedback visual inmediato (cambio de color). Cada selección **actualiza un mapa de asistencia en memoria**. Al pulsar **Guardar**, el sistema calcula los totales, **muestra un gráfico** (presentes/ausentes/sin marcar) y **envía el resumen por HTTP** a la API (JSON con fecha, totales y detalle por estudiante). Si la red falla, se informa al usuario; en caso exitoso, se confirma el registro y finaliza el proceso.

Etapas clave (mapeo con el diagrama):

- **A → B:** Apertura de pantalla y carga de estudiantes desde SQLite.
- **B → C:** Renderizado de la lista con controles de Presente/Ausente por estudiante.

- **C → D:** El usuario interactúa; decisión de marcar asistencia.
- **D → E:** Actualización del estado en el mapa de asistencia (en memoria) y feedback visual.
- **E → F:** Acción **Guardar**: consolidación de resultados.
- **F → G:** Visualización del **gráfico** de resumen (pie/ring).
- **G → H: Envío HTTP POST** a la API con el payload de asistencia.
- **H → I:** Respuesta de servidor, notificación al usuario y cierre del flujo.

Desarrollo de la solución

Configuración del Proyecto Flutter

Para iniciar el desarrollo, se creó un nuevo proyecto en Flutter, el cual es un framework multiplataforma que permite generar aplicaciones para Android e iOS desde un solo código fuente. Este paso se realizó en la terminal, ubicándose en el directorio de trabajo del proyecto y ejecutando el siguiente comando:

flutter create asistencia_app

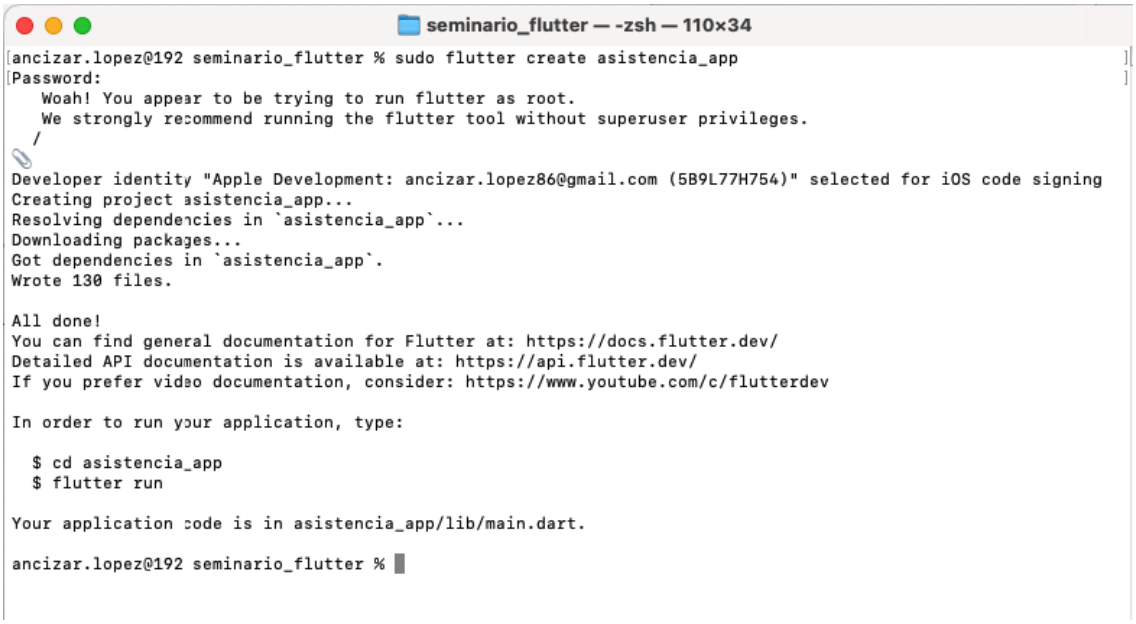

Figura 5. Ejecución comando para crear aplicación flutter.

1. Este comando genera de forma automática la estructura base de un proyecto Flutter, incluyendo:
2. Carpeta lib/ donde se escribirá el código principal en Dart.
3. Carpeta android/ y ios/ para la configuración nativa de cada plataforma.
4. Archivos de configuración como pubspec.yaml, donde se declaran las dependencias externas.
5. Código inicial en main.dart que permite ejecutar una app de ejemplo ("counter app") para verificar que el entorno funciona correctamente.

Estructura del Código Fuente

Para mantener el proyecto organizado y facilitar su escalabilidad, se definió una arquitectura de carpetas modular dentro del directorio lib/, siguiendo buenas prácticas de desarrollo en Flutter. La estructura propuesta fue la siguiente:

lib/
├── models/
├── services/
├── screens/
└── widgets/

Descripción de cada carpeta:

- **models/**
  Contiene las clases que representan la estructura de datos de la aplicación.

  Ejemplo: estudiante.dart, que define atributos como id, nombre, codigo, activo y métodos como toMap() y fromMap() para convertir objetos en registros de base de datos y viceversa.

- **services/**
  Incluye la lógica de acceso a datos y conexión con el servidor o base de datos local.

  Ejemplo: db_service.dart, encargado de inicializar SQLite, crear tablas, y realizar operaciones CRUD; y servicios para invocar la API REST en PHP vía HTTP.

- **screens/**

Agrupa las pantallas (vistas) de la aplicación, cada una en un archivo separado.

Ejemplo: estudiantes_screen.dart para el CRUD de estudiantes y lista_screen.dart para el llamado a lista.

- **widgets/**
  Contiene componentes reutilizables de UI, como formularios, botones personalizados, y tarjetas.

  Ejemplo: estudiante_form.dart que encapsula el formulario de registro/edición de estudiantes.

Beneficios de esta organización

- **Mantenibilidad:** Permite encontrar rápidamente la lógica de negocio, modelos y pantallas sin mezclar responsabilidades.
- **Escalabilidad:** Facilita agregar nuevos módulos (por ejemplo, asistencia, materias, cursos) sin afectar los existentes.
- **Reutilización:** Los widgets pueden ser reutilizados en distintas pantallas, reduciendo duplicación de código.
- **Buenas prácticas:** Sigue principios de arquitectura limpia y separación de capas, favoreciendo pruebas unitarias.

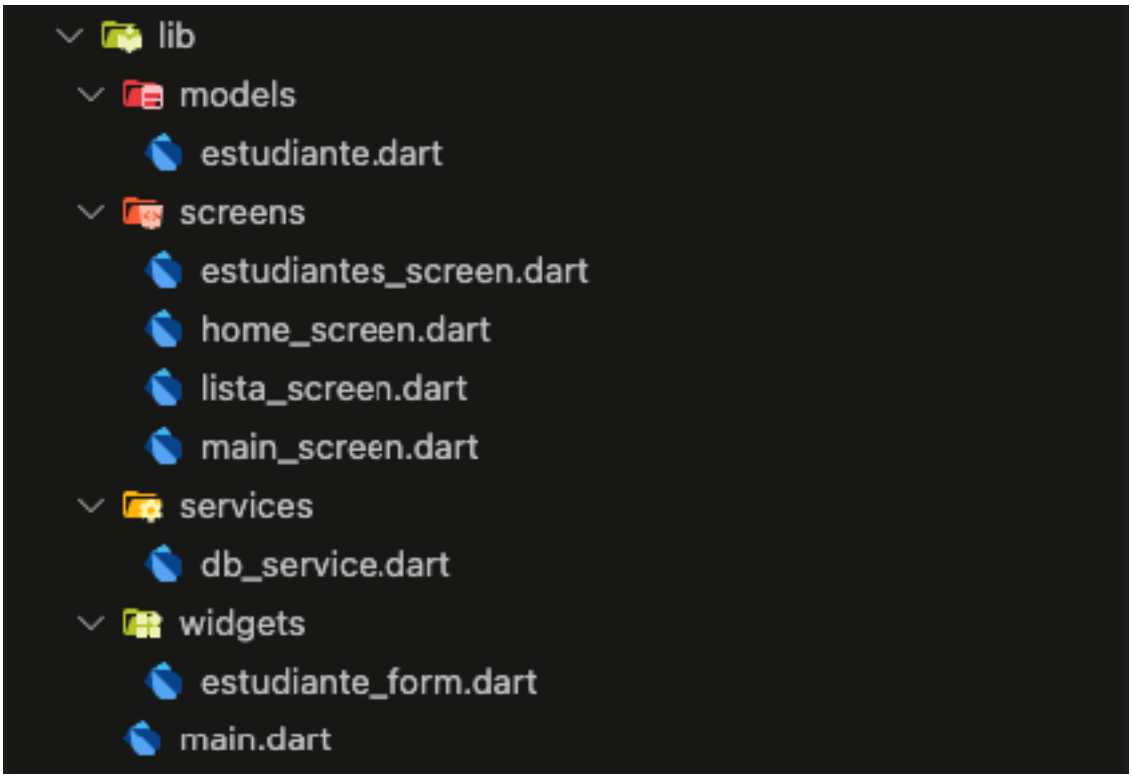

Figura 5. Distribución directorios flutter.

Diseño de la Base de Datos Local (SQLite)

Para permitir que la aplicación funcione sin conexión a internet (**offline-first**), se decidió utilizar **SQLite** como motor de base de datos local. SQLite es una base de datos embebida, ligera y confiable, ideal para aplicaciones móviles ya que no requiere un servidor dedicado y se ejecuta directamente en el dispositivo del usuario.

El diseño se centró en una tabla principal estudiantes, la cual almacena la información básica de cada estudiante para su gestión y para el proceso de llamado a lista.

Justificación del uso de SQLite

1. **Ligereza y velocidad:** SQLite es extremadamente eficiente para operaciones CRUD y no requiere configuración compleja.
2. **Compatibilidad:** Está soportado nativamente en Flutter a través de paquetes como sqflite.
3. **Modo offline:** Garantiza que el docente pueda gestionar estudiantes y registrar asistencia sin conexión a internet, sincronizando más tarde con el servidor.
4. **Simplicidad de despliegue:** No es necesario instalar un motor de base de datos adicional en el dispositivo.

```dart
class DBService {
  static Database? _db;

  static Future<Database> getDB() async {
    if (_db != null) return _db!;
    final dbPath = await getDatabasesPath();
    final path = join(dbPath, 'estudiantes.db');

    _db = await openDatabase(
      path,
      version: 1,
      onCreate: (db, version) async {
        await db.execute('''
          CREATE TABLE estudiantes(
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            nombre TEXT,
            codigo TEXT,
            activo INTEGER
          )
        ''');
      },
    );
    return _db!;
  }
```

Figura 6. Creación base de datos SQLite.

Desarrollo del CRUD de Estudiantes

Una de las funcionalidades clave de la aplicación es la gestión de estudiantes mediante operaciones **CRUD** (Create, Read, Update, Delete). Esto permite al

docente administrar el listado de alumnos que luego serán utilizados en el proceso de llamado a lista.

Create (Crear)

Para agregar un nuevo estudiante se implementó un **formulario modal** (EstudianteForm), mostrado mediante un AlertDialog.

Figura 7. Formulario ingreso de datos.

Este formulario utiliza TextFormField para capturar nombre y código, con validaciones para evitar registros vacíos.

Flujo de creación:

1. El docente pulsa el botón ➕ **Agregar**.
2. Se abre el AlertDialog con los campos de entrada.
3. Al guardar, se valida la información.
4. Si es correcta, se inserta el registro en SQLite mediante DBService.insertEstudiante().
5. Se llama a _enviarAlServidor() para enviar el registro vía HTTP POST a la API en PHP, sincronizando con MySQL.

Read (Leer)

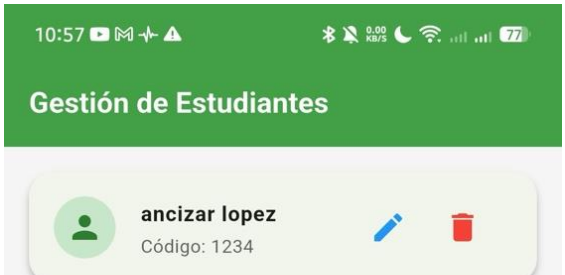

Figura 8. Pantalla listado de estudiantes.

El listado de estudiantes se implementó en estudiantes_screen.dart usando un ListView.separated con elementos Card para una presentación atractiva.

Cada estudiante muestra:

- **Nombre** en negrita.
- **Código** como subtítulo.
- **Botones de acción** para editar o eliminar.

Update (Actualizar)

La edición de un estudiante reutiliza el mismo EstudianteForm.

Al abrirlo en modo edición:

- Los campos se precargan con los datos actuales del estudiante.
- Al guardar, se ejecuta DBService.updateEstudiante() y se refresca la lista.

Delete (Eliminar)
Para evitar eliminaciones accidentales, se implementó un cuadro de confirmación antes de borrar:

1. El docente pulsa el ícono 🗑.
2. Aparece un AlertDialog de confirmación.
3. Si se confirma, se ejecuta DBService.deleteEstudiante() y se muestra un
4. SnackBar notificando la acción
5. La lista se recarga automáticamente.

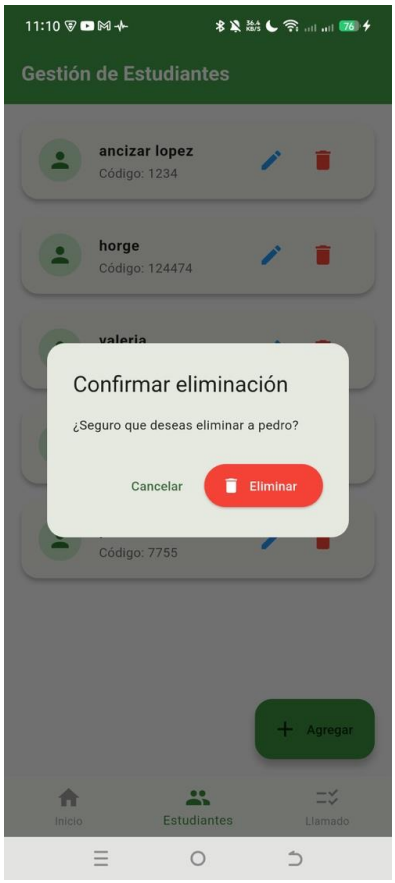

Figura 9. Confirmación de eliminación.

Pantalla de Llamado a Lista

La pantalla de llamado a lista es el corazón de la aplicación, donde el docente puede registrar de forma rápida y visual la asistencia de los estudiantes.

Diseño de la Pantalla

Se implementó en el archivo lista_screen.dart, utilizando un ListView dinámico que carga todos los estudiantes desde SQLite y genera una **tarjeta interactiva** (Card) para cada uno.

Cada tarjeta incluye:

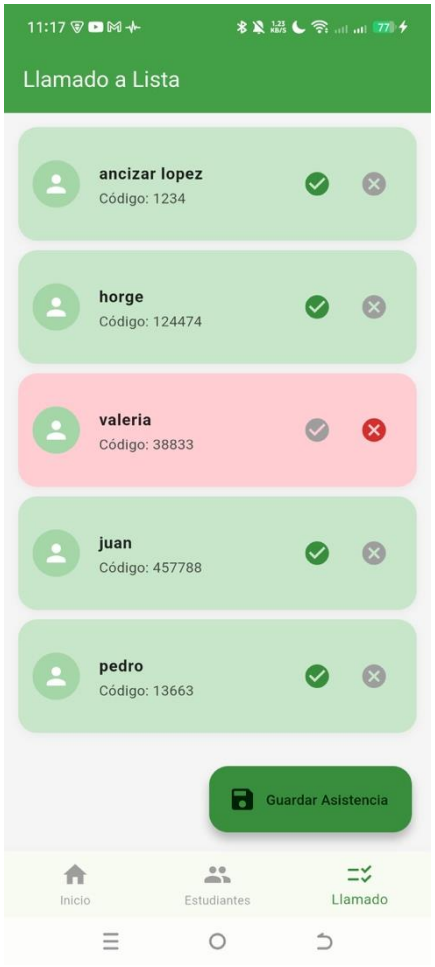

Figura 10. Pantalla para llamado de asistencia.

- **Nombre y código del estudiante.**
- **Ícono de persona** para identificación visual.
- **Botones de acción** para marcar **Presente** ✅ o **Ausente** ❌.

Interacción y Feedback Visual

- Al presionar el botón de **Presente**, la tarjeta cambia de color a **verde claro**.
- Al presionar **Ausente**, cambia a **rojo claro**.
- La selección se guarda en un Map<int,bool> en memoria para consolidar la asistencia al final.
- La lista se actualiza en tiempo real sin necesidad de recargar manualmente.

Este diseño ofrece una experiencia intuitiva, rápida y agradable para el usuario, reduciendo errores en el registro de asistencia

Gráfico de Asistencia

Al presionar el botón **Guardar Asistencia**, se ejecuta el método guardarAsistencia(), que:

1. Recorre el mapa de asistencia y calcula el total de presentes, ausentes y no marcados.

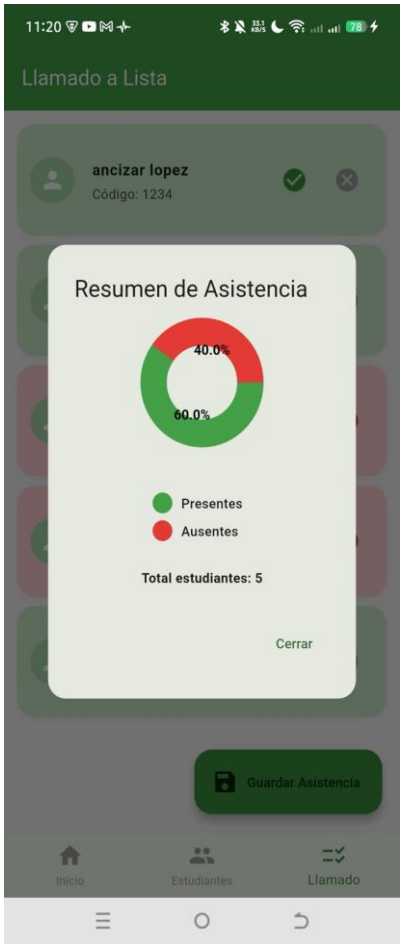

Figura 11. Grafíco circular de asistencia de estudiantes.

2. Muestra un AlertDialog con un **gráfico circular (PieChart)** representando los porcentajes de cada categoría.
3. Envía los datos al servidor vía HTTP POST en formato JSON para su almacenamiento centralizado.

Conclusiones

El desarrollo de la aplicación móvil para la gestión de estudiantes y llamado a lista demostró que es posible implementar una solución **eficiente, escalable y multiplataforma** usando tecnologías modernas de código abierto.

1. **Uso de Flutter y SQLite:**

   La elección de Flutter permitió crear una interfaz visual atractiva y consistente en Android e iOS desde un solo código fuente, optimizando tiempos de desarrollo. SQLite brindó un almacenamiento local robusto, garantizando que la aplicación funcione incluso sin conexión a internet (modo offline-first).

2. **Sincronización con servidor central (API REST + MySQL):**

   La integración con una API REST en PHP y la base de datos MySQL permitió mantener los datos centralizados, actualizados en tiempo real y listos para ser utilizados en reportes o sistemas institucionales más amplios.

3. **Infraestructura reproducible con Docker y pruebas con Ngrok:**

Docker facilitó la creación de un entorno controlado y portable para el servidor (Apache, PHP y MySQL), asegurando que la aplicación sea fácilmente desplegable en otros entornos. Ngrok permitió exponer el servidor en desarrollo para pruebas en dispositivos reales sin necesidad de un dominio público.

4. **Experiencia de usuario mejorada:**

   El uso de tarjetas interactivas, colores dinámicos y gráficos circulares hizo que el registro de asistencia sea **intuitivo, rápido y visualmente claro**, reduciendo errores humanos y mejorando la experiencia del docente.

5. **Buenas prácticas aplicadas:**

   La organización modular del código (models, services, screens, widgets), el uso de confirmaciones de eliminación, validaciones en formularios y feedback visual (SnackBars, diálogos) refuerzan un desarrollo limpio y mantenible.

Trabajo Futuro

- **Historial de asistencia:** Agregar un módulo para consultar reportes históricos de asistencia por estudiante, curso o fecha.
- **Soporte de múltiples grupos/cursos:** Permitir crear cursos y asignar estudiantes a cada grupo, para mejorar el filtrado.
- **Modo offline avanzado:** Implementar cola de sincronización que envíe los datos automáticamente cuando haya conexión a internet.
- **Autenticación de usuarios:** Integrar login de docentes para mantener seguridad y trazabilidad.
- **Exportación de datos:** Generar reportes en PDF o Excel directamente desde la aplicación para compartir con coordinación académica.

Referencias

- Al-Musharraf, A., & Al-Khateeb, H. M. (2021). *The effectiveness of mobile learning in higher education: A systematic review.* Education and Information Technologies, 26, 643–664. https://doi.org/10.1007/s10639-020-10240-6
- Soto, L., Jiménez, M., & Vargas, D. (2023). *Evaluación de frameworks multiplataforma para el desarrollo de aplicaciones móviles.* Revista Tecnología y Sociedad, 17(2), 45–58.

- Owens, M., & Allen, G. (2020). *The Definitive Guide to SQLite.* Apress.
- Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* (Doctoral dissertation, University of California, Irvine).
- Merkel, D. (2014). *Docker: lightweight Linux containers for consistent development and deployment.* Linux Journal, 2014(239), 2.