Manual de usuario – AhorrApp

Elaborado por:

Santiago Peralta Castellanos
José Luis Ramírez Isaza
Sebastián Triana Guanay

Presentado a

Néstor German Bolívar Pulgarín

Asignatura:

Programación Orientada a Objetos

Universiada Nacional de Colombia
2025 – 2s
Bogotá, DC 202

Introducción

AhorrApp es una aplicación diseñada para proporcionar una estructura base orientada a la gestión de información, lógica de negocio y ejecución modular en Java utilizando Gradle como sistema de construcción. Su código fuente está disponible públicamente y puede ser extendido, modificado y adaptado según las necesidades del usuario o desarrollador. Este manual tiene como objetivo explicar de forma detallada todo lo relacionado con la instalación, estructura, funcionamiento y personalización de la aplicación.

Objetivos del manual

- Guiar al usuario en la instalación y ejecución del proyecto.
- Describir de manera detallada la estructura interna del repositorio.
- Explicar cómo modificar y extender AhorrApp.
- Presentar buenas prácticas y recomendaciones para desarrollo y mantenimiento.

Requisitos Previos

Antes de ejecutar AhorrApp es necesario asegurarse de contar con las siguientes herramientas:

- Java JDK: versión 17 o superior.
- Gradle: aunque el proyecto incluye wrapper, es recomendable tener Gradle configurado.
- Git: para clonar y mantener actualizado el repositorio.
- IDE recomendado: IntelliJ IDEA o Android Studio.

Instalación del Proyecto

Clonar el repositorio

Ejecuta en consola: git clone https://github.com/JosehhR/AhorrApp.git cd AhorrApp o desde Android studio con la opción clonar repositorio

Construir el proyecto

El repositorio incluye Gradle Wrapper, por lo que puedes compilar sin instalar Gradle: ./gradlew build o en Windows: gradlew.bat buil

Ejecución de la Aplicación

Dependiendo de la configuración del archivo build.gradle.kts, AhorrApp puede ejecutarse mediante: ./gradlew run Si se genera un archivo .jar, puedes ejecutarlo manualmente: java -jar build/libs/AhorrApp.jar

Estructura del Proyecto

El repositorio está dividido en varios directorios clave que permiten separar correctamente las responsabilidades del sistema. A continuación se presenta una descripción detallada de cada carpeta:

Carpeta raíz

Contiene los archivos principales de configuración:
• settings.gradle.kts: define el nombre del proyecto y módulos incluidos.
• build.gradle.kts: archivo central de construcción, dependencias y tareas.
• gradle.properties: parametrización del proyecto.
• .gitignore: archivos excluidos del control de versiones.
• gradlew / gradlew.bat: wrappers de Gradle para ejecutar tareas sin instalación previa.

Carpeta /app

Aquí se encuentra el código fuente principal. Su estructura es: app/ src/ main/ java/ (clases, paquetes y lógica de la app) main/resources/ test/java/ build.gradle.kts

Detalles Técnicos del Funcionamiento

Se presenta la forma en la que AhorrApp está diseñado para operar dentro del ecosistema Java + Gradle:

Sistema de construcción Gradle

AhorrApp utiliza Gradle en su versión Kotlin DSL. Esto significa que la configuración se hace mediante scripts .kts. Algunas características:

• Gestión de dependencias.
• Configuración de tareas personalizadas.
• Optimización del build.
• Integración con herramientas externas.

Arquitectura General y Tecnologías

El backend de AhorrApp se basa en un enfoque "Serverless" (sin servidor), utilizando la plataforma de Google Firebase como núcleo. Esto significa que no hay un servidor tradicional (como Node.js, Java Spring, etc.) que gestione la lógica. En su lugar, la lógica de negocio se ejecuta directamente en la aplicación cliente (Android), que se comunica de forma segura con los servicios de Firebase.

Las tecnologías principales son:

• Firebase Authentication: Para gestionar el registro, inicio de sesión y la identidad de los usuarios.

• Firebase Realtime Database: Una base de datos NoSQL en tiempo real que almacena toda la información de la aplicación.

Detalles de Operaciones Backend – Lógica Interna

1. La Preparación: Conexión al Backend (onCreate)

    • mAuth = FirebaseAuth.getInstance(); obtiene la conexión al servicio de autenticación.

    • mDatabase = FirebaseDatabase.getInstance().getReference(); referencia a la raíz de la base de datos.

    • usersRef = getReference("users"); referencia al nodo /users.

2. Primera Operación de Backend: Leer Usuarios (loadUsers)

Propósito: Obtener lista de usuarios para participantes.

    • addListenerForSingleValueEvent: lectura única del nodo /users.

    • onDataChange: recorre los usuarios y excluye al actual (!uid.equals(currentUser.getUid())).

    • Crea objetos User y activa el botón de añadir participante.

3. Segunda Operación de Backend: Escribir el Gasto (saveSharedExpense)

    • Disparador: botón Guardar.

    • Recolección de datos simples y participantes.

    • expenseId = push().getKey(): genera ID única.

    • Crea y llena un Map con name, value, ownerId, timestamp y participants.

    • Escritura: mDatabase.child("shared_expenses").child(expenseId).setValue(expense)

    • addOnCompleteListener: confirma éxito o error.

Flujo de Autenticación

1. Registro/Login: El usuario se autentica usando su email y contraseña a través de la `AuthActivity`.

2. Obtención de UID: Tras una autenticación exitosa, el SDK de Firebase proporciona un objeto `FirebaseUser que contiene el `uid` único del usuario.

3. Acceso a Datos:** Este `uid` se utiliza en todas las operaciones posteriores con la base de datos para asegurar que cada usuario solo acceda a su propia información.

Modelo de Datos en Firebase Realtime Database

La base de datos está estructurada como un gran árbol JSON. La forma en que se organizan los "nodos" (rutas) es crucial para la seguridad y la eficiencia.

a. `/users/{userId}`
Propósito: Almacenar información pública básica de cada usuario.

Estructura:

```json
"users": {
  "USER_ID_1": {
    "name": "Santiago Peralta",
    "email": "santiago@email.com"
  },
  "USER_ID_2": {
    "name": "Otro Usuario",
    "email": "otro@email.com"
  }
}
```

Lógica: Se utiliza principalmente para poblar las listas de selección de participantes en gastos y deudas compartidas.

b. /expenses/{userId}/{expenseId}
Propósito: Almacenar los gastos individuales de cada usuario.
Estructura:

```
json
"expenses": {
  "USER_ID_1": {
    "EXPENSE_ID_A": {
      "name": "Compra del supermercado",
      "value": 150000,
      "priority": "Alta",
      "timestamp": 1678886400000
    }
  }
}
```

Lógica de Seguridad: La información está **particionada por `userId`**. Las reglas de seguridad de Firebase refuerzan esto, permitiendo que un usuario solo pueda leer/escribir en la ruta /expenses/SU_PROPIO_UID.

c. /debts/{userId}/{debtId}
Propósito: Almacenar las deudas individuales.
Lógica: Funciona exactamente igual que los gastos individuales, particionado por `userId`.

d. `/shared_expenses/{expenseId}`
Propósito: Almacenar los gastos compartidos.
Estructura:

```
json
"shared_expenses": {
  "SHARED_EXP_ID_X": {
    "name": "Cena de equipo",
    "value": 250000,
    "ownerId": "USER_ID_1",
    "timestamp": 1678886400000,
    "participants": ["USER_ID_1", "USER_ID_2", "USER_ID_3"]
  }
}
```

Lógica de Seguridad: A diferencia de los gastos individuales, este nodo es global y plano. La seguridad se gestiona con reglas que verifican si el `uid` del usuario que intenta leer el dato está presente en la lista participantes.

e. /shared_debts/{debtId}
Propósito: Almacenar las deudas compartidas.
Lógica: Sigue la misma filosofía que los gastos compartidos (un nodo global y plano), pero con una estructura de datos más compleja que incluye porcentajes.

4. Lógica de la API (Operaciones de Base de Datos)

La "API" es el conjunto de métodos que proporciona el SDK de Firebase para Android.

Crear (Create): Se utiliza `databaseReference.push().getKey()` para generar un ID único y `setValue(object)` para escribir el dato.
Leer (Read): Se añade un `ValueEventListener` a una referencia para recibir los datos en tiempo real.
Actualizar (Update): Se utiliza `updateChildren(updatesMap)` para modificar campos específicos.
Borrar (Delete): Se llama a `reference.removeValue()`.

5. Seguridad (Basado en Reglas de Firebase)

Un backend de Firebase bien configurado depende de su archivo de reglas (`database.rules.json`) para ser seguro.

* **Autenticación Requerida:** Todas las rutas requieren que el usuario esté autenticado (`"auth != null"`).
* **Datos Individuales:** El usuario solo puede leer/escribir en `/expenses/{userId}` y `/debts/{userId}` si `$userId === auth.uid`.
* **Datos Compartidos:** El usuario puede leer un objeto en `/shared_expenses/{expenseId}` si su `auth.uid` está en la lista de `participants`.

Uso de AhorrApp

Actualmente AhorrApp funciona como base para futuras expansiones. Un uso típico incluye:

1. Ejecutar la aplicación.
2. Extender funcionalidades editando las clases dentro de /app/src/main/java.
3. Añadir dependencias necesarias en build.gradle.kts.
4. Construir y probar nuevas características

Buenas Prácticas de Desarrollo

• Mantener el código modular.
• Documentar cada clase y método relevante.
• Utilizar control de versiones con commits pequeños y claros.
• Escribir pruebas unitarias en /test.
• Mantener actualizado Gradle Wrapper.

Solución de Problemas

Error: No se reconoce 'gradlew'. Solución: Dar permisos de ejecución: chmod +x gradlew Error: Fallos en dependencias. Solución: Ejecutar: ./gradlew --refresh-dependencies Error: Java no reconocido. Solución: Instalar JDK y configurar JAVA_HOME.

Cómo Personalizar AhorrApp

AhorrApp está construido para ser extendido. Puedes modificar:
• Flujos de ejecución.
• Clases internas.
• Sistema de persistencia.
• Integración de APIs externas.
• Nuevos módulos completos.

Conclusiones

Este manual proporciona una guía completa para trabajar con AhorrApp, desde su instalación hasta su modificación. El repositorio está preparado para servir como base robusta para nuevas aplicaciones de gestión, finanzas personales, o cualquier tipo de sistema modular escrito en Java con Gradle.