# Informe Técnico: Vote App

Estudiantes: Cristian Chala, Constanza Garrido y Iván Castro

Fecha de Entrega:21-10-2025

Profesor: Sebastián Salazar

**1. Arquitectura y Estructura del Proyecto**

La aplicación se ha desarrollado siguiendo principios de **Arquitectura Limpia**, separando las responsabilidades en tres capas principales:

* **Capa de Presentación (presentation):** Contiene los Widgets de Flutter y la lógica de la UI. Se utiliza el patrón ConsumerWidget de Riverpod para construir una UI reactiva.
* **Capa de Datos (data):** Contiene el VotesRepository, responsable de comunicarse con la API REST a través de un ApiClient (basado en dio).
* **Capa Core (core):** Contiene lógica transversal como la configuración del cliente HTTP y el manejador de errores.

La estructura de carpetas dentro de lib/features refleja esta separación, agrupando los archivos por funcionalidad (auth, votes).

**2. Gestión de Estado con Riverpod**

Se ha elegido **Riverpod** como solución para la gestión de estado por su flexibilidad y seguridad de tipos. Los principales *providers* utilizados son:

* **FutureProvider:** Para operaciones asíncronas simples, como obtener la lista inicial de encuestas (votesProvider). Su método .when permite manejar de forma declarativa los estados de data, loading y error.
* **FutureProvider.family:** Para operaciones que dependen de un parámetro, como obtener el detalle (voteDetailProvider) o los resultados (voteResultsProvider) de una encuesta.
* **StateProvider:** Para gestionar estados simples de la UI, como la opción seleccionada en una votación (selectedOptionProvider).
* **AsyncNotifierProvider:** Para estados asíncronos complejos, como la paginación y búsqueda en la pantalla de resultados (pollResultsProvider).

**3. Autenticación y Manejo de JWT (Decisión de Diseño)**

El requisito inicial era implementar una gestión de sesión dinámica. Sin embargo, durante el desarrollo, surgieron desafíos técnicos significativos que llevaron a adoptar una solución alternativa para garantizar la funcionalidad del proyecto.

**Desafíos con el JWT Dinámico**

El principal obstáculo fue un error persistente ApiException: 10 (DEVELOPER\_ERROR) al intentar obtener un idToken de Google con el serverClientId de la API del profesor. Este error se debe a un conflicto de configuración entre el proyecto de Firebase del estudiante y el proyecto de Firebase donde está registrada la API del profesor. Google bloquea estas solicitudes "cross-project" por seguridad si no están explícitamente autorizadas.

Adicionalmente, se determinó que el uso del nombre de paquete por defecto (com.example.vote\_app) contribuía al problema, ya que Google restringe su uso en flujos de autenticación.

**Solución Implementada: JWT Estático**

Para superar este bloqueo y asegurar que la aplicación fuera 100% funcional, se optó por la siguiente estrategia:

1. **Login de Usuario (Firebase):** La autenticación del usuario se sigue manejando de forma segura a través de **Firebase Authentication** con el proveedor de Google. Esto asegura que solo usuarios autenticados puedan usar la app.
2. **Autenticación de API (JWT Estático):** Para comunicarse con la API del profesor, la aplicación carga al inicio un **JWT estático** proporcionado en la documentación del curso. Este token se almacena de forma segura en flutter\_secure\_storage y es inyectado por el ApiClient en cada petición a la API.

Esta es una solución pragmática que, si bien representa un compromiso de seguridad, permite cumplir con todos los demás requisitos funcionales de la rúbrica.

**4. Pruebas y Calidad del Software**

Para garantizar la calidad de la aplicación, se implementaron:

* **Pruebas de Widgets:** Se crearon pruebas para la pantalla VotesListPage que cubren los 4 estados de la UI (carga, datos, vacío y error), sobrescribiendo los *providers* de Riverpod para simular cada escenario.
* **Integración Continua (CI):** Se configuró un *workflow* de **GitHub Actions** que se ejecuta en cada push. Este *workflow* realiza análisis estático (flutter analyze), ejecuta las pruebas (flutter test) y compila un APK de debug.

**5. Decisiones de Diseño Clave**

* **Implementación de Historial de Usuario:** La aplicación consume directamente los *endpoints* de la API (GET /v1/me/votes con un *fallback* a /v1/users/me/votes) para obtener y mostrar el historial de votaciones del usuario autenticado. Esto proporciona una vista precisa y en tiempo real de la participación del usuario en las encuestas.
* **Manejo de Error 500 en Voto Duplicado:** Se detectó que la API devuelve un error 500 al intentar votar dos veces. Se implementó una lógica en VotesRepository que intercepta este error y lo traduce a un mensaje amigable para el usuario: "Ya has votado en esta encuesta."