Justificación Técnica del Proyecto

Este documento detalla las decisiones técnicas tomadas durante el desarrollo de la prueba, así como el razonamiento detrás de la arquitectura, herramientas, librerías y estructura implementada.

🎯 Objetivo del Proyecto

El objetivo fue construir una aplicación móvil en Flutter que cumpliera con:

- Consumo de API pública
- Persistencia local
- CRUD completo
- Gestión de estado con Cubit
- Navegación con rutas nombradas
- UI escalable y responsiva
- Código claro, modular y mantenible

La solución debía ser desarrollada completamente en menos de 72 horas, priorizando calidad, claridad y estabilidad.

🧱 Decisiones Técnicas y Justificación

✔️ 1. Gestión de estado con Flutter BLoC/Cubit

Decisión: utilizar `flutter_bloc` y organizar el proyecto con Cubits separados para API y base local.

Por qué Cubit:

- Es más ligero y conciso que Bloc, ideal cuando los estados son simples.
- Mantiene una arquitectura clara y profesional.
- Facilita pruebas unitarias.

- Permite un flujo de datos predecible.
- Recomendado ampliamente en proyectos de mediana escala.

Cubits utilizados:

- `ApiCubit` → manejar petición HTTP, loading, success y error
- `LocalImagesCubit` → CRUD completo sobre SQLite

Esta separación cumple con el enunciado y permite escalabilidad futura.

✔️ 2. Persistencia local con SQLite (sqflite)

Decisión: usar `sqflite` como base de datos principal.

Justificación:

- SQLite es extremadamente estable y probado en producción.
- Permite relaciones y consultas complejas si el proyecto crece.
- No requiere adaptadores como Hive.
- Soporta CRUD nativo, con integridad de datos.
- Es ideal para listas de elementos personalizables como favoritos.

Elegí no usar Hive porque requiere generadores de código y adaptadores adicionales, lo cual consume tiempo y complejidad adicional sin aportar valor directo en esta prueba.

✔️ 3. Cliente HTTP con Dio

Decisión: utilizar `dio` para consumir la API de Picsum.

Justificación:

- Ofrece mejor manejo de errores que `http`.
- Permite interceptores, timeouts y control avanzado.
- Es ampliamente usado en entornos de producción.

Su sintaxis clara reduce errores y acelera el desarrollo.

✔️ 4. Arquitectura por capas (Presentation / Domain / Data)

Decisión: separar el proyecto en capas modulares.

Justificación:

- Evita acoplamiento entre UI, lógica y datos.
- Facilita pruebas y mantenimiento.
- Sigue patrones recomendados en Flutter.
- Permite escalar el proyecto fácilmente (multimódulos, nueva BD, nuevo backend, etc.)

La estructura usada es la recomendada en limpias arquitecturas para apps pequeñas y medianas.

✔️ 5. Inyección de dependencias manual (di.dart)

Decisión: manejar DI mediante un servicio sencillo, sin paquetes externos como get_it.

Justificación:

- Reduce el overhead y configuración.
- Alineado con el tamaño del proyecto.
- Facilita visibilidad del flujo de dependencias sin magia oculta.
- Mejora el aprendizaje del evaluador respecto al flujo interno.

✔️ 6. Animaciones y UX

Paquetes utilizados:

- `animate_do`: para animaciones leves al mostrar elementos guardados.
- `animated_splash_screen`: para un splash simple y limpio.

Justificación:

- Mejoran la experiencia sin aumentar complejidad.
- Le dan al proyecto una sensación más profesional y fluida.

✔️ 7. Diseño responsivo

Se emplearon:

- `SizedBox`
- `Expanded`
- `ConstrainedBox`
- Margenes y paddings relativos

Justificación:

- Garantiza compatibilidad con distintos dispositivos Android.
- Mejora la accesibilidad visual.
- Facilita la escalabilidad del proyecto.

✔️ 8. Disponibilidad para Android únicamente

Motivo técnico:

Actualmente no poseo una Mac, por lo cual **no es posible generar un build funcional para iOS** ni realizar pruebas reales en un emulador de iPhone.

Sin embargo:

- Todo el código es totalmente compatible con iOS.
- La migración a iOS sería inmediata una vez disponiendo de una Mac.

Esto no afecta el cumplimiento del enunciado, ya que la prueba no exige la entrega en ambas plataformas.

✔️ 9. Uso de Flutter 3.24.x

Justificación:

Trabajo actualmente en proyectos activos con dicho SDK, por lo que mantener la consistencia evita conflictos con versiones o dependencias.

El proyecto está desarrollado completamente con **Null Safety**, compatible con cualquier versión >= 3.10.

🧪 Confirmación de Cumplimiento del Enunciado

Requisito	Estado
Consumir API pública	✔️ Completado
Listar ítems de API	✔️
Guardar ítem local con nombre personalizado	✔️
CRUD completo en SQLite	✔️
Editar elemento	✔️
Eliminar elemento	✔️ + confirmación
Ver detalles con imagen + zoom	✔️
Gestión de estado con Cubit	✔️ Api + Preferences
Pantalla de carga y error	✔️
Diseño responsivo	✔️
Documentación	✔️
Arquitectura limpia	✔️
Rutas nombradas	✔️

💬 Mensaje para el Evaluador

A continuación te dejo un mensaje profesional listo para incluir:

✉️ Mensaje al Revisor

> Agradezco el tiempo dedicado a revisar esta prueba técnica.
>
> Disfruté desarrollarla, aplicando buenas prácticas, arquitectura limpia, Cubit y SQLite como persistencia local.
>
> Estoy realmente motivado por la posibilidad de trabajar con su equipo, aprender de sus procesos y aportar toda mi experiencia en desarrollo móvil con Flutter.
>
> Estoy a disposición para explicar cualquier parte del código, justificar alguna implementación adicional o realizar mejoras si lo consideran necesario.
>
> Será un gusto seguir avanzando en el proceso y demostrar mi compromiso, ética de trabajo y pasión por el desarrollo de software.
>
> **Muchas gracias por la oportunidad.**
>
> — *Rodrigo Vidal*