CARRERA
INGENIERÍA EN DESARROLLO Y GESTIÓN DE SOFTWARE

TEMA:
INVESTIGACIÓN

ALUMNOS:
ALCANTAR MERINO JOSÉ MARÍA
BECERRA GUEVARA JUANITA RUBI
FABIAN CRUZ DIANA ELIZABETH
GÓMEZ ROSILLO FRANCISCO

CUATRIMESTRE:
DÉCIMO CUATRIMESTRE
DS03SV-24

Materia	Profesor
DESARROLLO MÓVIL INTEGRAL	HÉCTOR SALDAÑA BENÍTEZ

LUGAR: SAN JUAN DEL RIO, QRO. FECHA: 18/09/2025

ValetFlow QR – Selección y Justificación de Arquitectura

1. Contexto del proyecto

El sector de hospitalidad y entretenimiento continúa dependiendo de procesos manuales para el servicio de valet parking, principalmente mediante boletos de papel. Este método resulta problemático al generar pérdidas o deterioro de boletos, retrasos en la entrega del automóvil y poca transparencia hacia el cliente. Además, limita la capacidad administrativa de las empresas para monitorear su operación, conciliación de pagos y desempeño del personal.

ValetFlow QR surge como una solución digital que busca transformar este servicio a través de una aplicación móvil multiplataforma desarrollada en Flutter. El sistema eliminará el papel mediante el uso de códigos QR, incorporará registro fotográfico, notificaciones en tiempo real e integrará funciones futuras basadas en IA, IoT y chatbots para optimizar la operación.

- Tipo de aplicación: modelo B2B2C.
- Fuentes de datos: API REST y posible integración con Firestore.
- Necesidad de offline: media, con cache temporal.
- Notificaciones: Firebase Cloud Messaging (FCM).
- Crecimiento esperado: alto, con IA, IoT y CRM.
- Equipo: pequeño, experiencia intermedia en Flutter.

2. Comparativa de arquitecturas

Criterio	MVC	MVVM	Clean Architecture
Responsabilidades	Controlador coordina vista y modelo.	ViewModel gestiona estado y lógica; vista renderiza.	División estricta: dominio, casos de uso, datos.
Flujo de datos	Bidireccional (vista ↔ controlador ↔ modelo).	Unidireccional: ViewModel expone estado.	Regla de dependencia: dominio independiente.
Testabilidad	Baja: lógica mezclada con la vista.	Media: ViewModels probables con mocks.	Alta: capas independientes, pruebas unitarias/integración.
Complejidad	Baja, útil en apps pequeñas.	Media: balance modularidad-simplicidad.	Alta: requiere más esfuerzo inicial.
Curva de aprendizaje	Baja.	Media.	Alta.
Dependencia de frameworks	Alta, controladores ligados al framework.	Media, depende de librerías de estado.	Baja, lógica desacoplada.

3. Criterios de decisión

Se selecciona MVVM con inspiración en Clean Architecture, ya que esta combinación equilibra simplicidad, escalabilidad y testabilidad. Sus ventajas clave son:

- Mantenibilidad: separación entre UI y lógica de negocio.
- Escalabilidad: estructura modular en capas.
- Testabilidad: ViewModels y repositorios probables con mocks.
- Curva de aprendizaje: razonable para el equipo.
- Integración con Flutter: Riverpod como gestor de estado.

4. Riesgos y mitigaciones

1. ViewModels sobrecargados con lógica compleja → Mitigación: trasladar reglas a casos de uso.
2. Dependencia excesiva de Riverpod → Mitigación: encapsular lógica en repositorios.
3. Crecimiento desordenado con múltiples clientes → Mitigación: modularidad y separación de capas.

5. Estado en Flutter

La implementación se realizará con Riverpod y StateNotifier:

- ViewModels como StateNotifier para lógica y estado.
- Vistas escuchan el estado y renderizan.
- Repositorios abstraen fuentes de datos (REST, Firestore).
- Casos de uso para reglas de negocio más complejas.

Este enfoque asegura una aplicación modular, escalable y en línea con las mejores prácticas de Android Developers y los principios de Clean Architecture.

Referencias

Android Developers. (s. f.). *Guide to app architecture*. Google. Recuperado el 18 de septiembre de 2025, de https://developer.android.com/topic/architecture

Android Developers. (s. f.). *Architecture recommendations*. Google. Recuperado el 18 de septiembre de 2025, de https://developer.android.com/topic/architecture/recommendations

Fowler, M. (2006). *GUI architectures (MVC, MVP, MVVM)*. Martin Fowler. Recuperado el 18 de septiembre de 2025, de https://martinfowler.com/eaaDev/uiArchs.html

Martin, R. C. (2012, 13 de agosto). *The clean architecture*. The Clean Coder Blog. Recuperado el 18 de septiembre de 2025, de https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

Flutter Dev. (s. f.). *App architecture & state management*. Google. Recuperado el 18 de septiembre de 2025, de https://docs.flutter.dev/app-architecture/guide