INSTITUTO POLITECNICO NACIONAL

IPN - UPIICSA

MAESTRO: Gustavo Martínez Vázquez

MATERIA: Programación móvil

SECUENCIA: 6NM61

INTEGRANTES DEL EQUIPO:

- Flores Manzano Luis Jesús
- Gutiérrez Sánchez César Axel
- Ibarra Lima Julio Jair
- Miguel Alarcón Adrián Manuel
- Pérez Ibarra Manuel Sebastián
- Rodríguez Brenes Adriel

FECHA: 31 de mayo del 2025

CARRERA: Ingeniería en informática

ARQUITECTURA DE SOFTWARE

PERIODO – 2025

Documento Técnico: Arquitectura de Software de la Aplicación WADI

Introducción

La aplicación móvil **WADI** nace como una solución inteligente y personalizable para la gestión de tareas y contenido, diseñada para operar de forma fluida tanto en línea como sin conexión. Con una fuerte orientación a la experiencia de usuario y modularidad, WADI permite organizar información en "páginas" jerárquicas que contienen componentes dinámicos y reutilizables, como listas, notas, recordatorios, multimedia y más. Esta flexibilidad requiere una arquitectura de software robusta, escalable y bien organizada.

Este documento detalla la **arquitectura de software** adoptada, justificando sus decisiones de diseño, la estructura de carpetas y cómo se implementan los principios de Clean Architecture junto con la modularización por características (**feature-based modularization**).

Objetivo de la Arquitectura

- Proveer una **base estructurada y escalable** para el desarrollo de WADI.
- Separar claramente las responsabilidades mediante el uso de capas.
- Facilitar el mantenimiento, pruebas y extensión de funcionalidades.
- Permitir sincronización local/remota mediante SQLite y Firebase Firestore.
- Optimizar el rendimiento en dispositivos móviles.

Estilo de Arquitectura Adoptado

Clean Architecture + Modularización por Feature

Se optó por la **Clean Architecture** para garantizar la independencia entre la lógica de negocio, la fuente de datos y la interfaz de usuario. Esta se combina con un enfoque **modular por funcionalidades** (feature-based) que facilita el trabajo colaborativo, la escalabilidad y el mantenimiento del código.

Principales capas:

- **Domain:** lógica de negocio, entidades y casos de uso.
- **Data:** fuentes de datos locales y remotas, implementación de repositorios.
- **Presentation:** vistas, widgets y manejo del estado de UI.

Estructura de Carpetas de `lib/`

```
lib/
├── main.dart                      # Punto de entrada de la aplicación
├── app/                           # Inicialización general y enrutamiento
│   ├── app.dart
│   └── router.dart
│
├── core/                          # Funcionalidades compartidas entre
módulos
│   ├── constants/                 # Constantes globales (colores, íconos,
strings)
│   ├── errors/                    # Manejo de errores y excepciones
│   ├── services/                  # Servicios compartidos (auth, db,
notificaciones)
│   ├── utils/                     # Utilidades generales (fechas,
imágenes, etc.)
│   ├── types/                     # Tipos, enums, typedefs reutilizables
│   └── widgets/                   # Componentes visuales reutilizables
│
├── features/                      # Módulos funcionales independientes
│   ├── auth/                      # Autenticación y manejo de sesión
│   │   ├── data/                  # Datasources, modelos, repositorios
│   │   ├── domain/                # Entidades, repositorios y casos de uso
│   │   └── presentation/          # UI: pantallas, widgets, estados
│   │
│   ├── tasks/                     # Lógica relacionada con tareas y
páginas
│   │   ├── data/
│   │   ├── domain/
│   │   └── presentation/
│   │
│   ├── image_capture/             # Captura y almacenamiento de imágenes
│   │   ├── data/
│   │   ├── domain/
│   │   └── presentation/
│   │
│   ├── settings/                  # Preferencias y configuración del
usuario
│   │   ├── data/
│   │   ├── domain/
│   │   └── presentation/
│   └── ...                        # Otros módulos funcionales
│
├── injection/                     # Inyección de dependencias (get_it,
injectable)
│   ├── injection.dart
│   └── injection.config.dart
│
├── l10n/                          # Internacionalización (soporte
```

```
multilenguaje)
│   ├── app_en.arb
│   └── app_es.arb
│
└── shared/                     # Componentes visuales reutilizables
globalmente
    ├── widgets/
    └── theme/                  # Temas visuales y estilos tipográficos
```

Justificación de Carpetas y Componentes

`/app/`

Contiene la configuración general de la aplicación: inicialización y rutas.

`/core/`

Centraliza servicios, constantes y utilidades que no pertenecen a una funcionalidad específica pero son usadas ampliamente (por ejemplo, `auth_service`, `notification_service`, etc.).

`/features/`

Cada módulo funcional está contenido en su propia carpeta, siguiendo Clean Architecture internamente (`data`, `domain`, `presentation`). Esto permite que cada funcionalidad evolucione de forma aislada.

`/injection/`

Contiene la configuración de inyección de dependencias, fundamental para mantener bajo acoplamiento entre clases.

`/l10n/`

Soporta internacionalización mediante archivos `.arb`, permitiendo la traducción dinámica de la UI.

`/shared/`

Componentes visuales comunes como botones personalizados, estilos y temas visuales que pueden ser reutilizados entre módulos.

Ventajas de la Arquitectura Adoptada

- **Escalabilidad**: Permite añadir nuevas funcionalidades (features) sin afectar las existentes.
- **Testabilidad**: La separación de lógica facilita las pruebas unitarias y de integración.
- **Mantenibilidad**: Cada módulo es autocontenible, lo que reduce errores y conflictos.
- **Adaptabilidad**: Soporta múltiples fuentes de datos y es compatible con integración offline-online.

Conclusión

La arquitectura de WADI está diseñada para facilitar el desarrollo ágil, la colaboración en equipo y la escalabilidad. Su estructura basada en Clean Architecture con modularización por características permite que el equipo de desarrollo trabaje de forma organizada, eficiente y segura, cumpliendo con los estándares modernos del desarrollo móvil multiplataforma.

Gracias a esta arquitectura, WADI podrá evolucionar con nuevas funcionalidades como colaboración, historial avanzado, personalización estética y más, sin comprometer la estabilidad ni la calidad de la aplicación actual.