



INSTITUTO POLITECNICO
NACIONAL
IPN – UPIICSA



MAESTRO: Gustavo Martínez Vázquez

MATERIA: Programación móvil

SECUENCIA: 6NM61

INTEGRANTES DEL EQUIPO:

- Miguel Alarcon Adrian Manuel
- Ibarra Lima Julio Jair
- Flores Manzano Luis Jesus
- Rodríguez Brenes Adriel
- Pérez Ibarra Manuel Sebastián
- Gutiérrez Sánchez César Axel

FECHA: 15 de Mayo del 2025

CARRERA: Ingeniería en informática

PROYECTO: WADI (Write & Do It) – Gestor de Tareas y Productividad

VERSION DEL MANUAL TÉCNICO: 1.0

VERSION DEL APLICACIÓN: 1.0

ÍNDICE

1. Introducción
2. Objetivo del Manual
3. Descripción General del Proyecto WADI
4. Requisitos del Sistema
5. Configuración del Entorno de Desarrollo
7. Configuración del código fuente
6. Tecnologías Utilizadas
7. Arquitectura de Software (Clean Architecture)
8. Estructura del Proyecto
9. Base de Datos (Firebase)
10. Métodos de autenticación de cuentas (Firebase Auth)
11. Notificaciones Push (Firebase Cloud Messaging)
12. Gestión de Tareas y Componentes Personalizados
13. Diseño de Interfaz de Usuario (Material Design)
14. Funcionalidades de la App
15. Escalabilidad y Futuras Expansiones
16. Conclusiones y Recomendaciones

1. Introducción

En el presente documento se detalla el **Manual Técnico** de la aplicación móvil **WADI (Write And Do It)**, una herramienta desarrollada para facilitar la gestión inteligente de tareas personales o profesionales. Este manual está orientado a desarrolladores y personal técnico encargado de la implementación, mantenimiento y evolución de la aplicación. Se abordan aspectos clave como la arquitectura del sistema, configuración del entorno de desarrollo, estructura del código, tecnologías empleadas, pruebas, y posibles expansiones.

La aplicación WADI fue concebida como una solución modular, multiplataforma y extensible que permite a los usuarios organizar sus actividades mediante páginas dinámicas y bloques personalizables, integrando características como recordatorios, notas enriquecidas, adjuntos multimedia y sincronización en la nube.

2. Objetivo del Manual

El objetivo principal de este manual es proporcionar una guía completa y estructurada sobre el ciclo técnico de vida de la aplicación **WADI**, incluyendo:

- La configuración del entorno de desarrollo necesario para colaborar en el proyecto.
- Las tecnologías, librerías y herramientas utilizadas.
- La estructura del proyecto basada en **Clean Architecture**.
- Las buenas prácticas de programación adoptadas.
- Los procedimientos para compilar, ejecutar y desplegar la aplicación.
- La documentación de los módulos funcionales y las pruebas realizadas.
- La base para futuras expansiones o mantenimientos del sistema.

Este documento busca asegurar la continuidad del desarrollo del sistema, permitiendo a nuevos programadores integrarse rápidamente al proyecto, así como facilitar la operación del mismo en distintos entornos.

3. Descripción General del Proyecto WADI

WADI (Write And Do It) es una aplicación móvil multiplataforma desarrollada en **Flutter** que permite a los usuarios gestionar sus tareas de forma jerárquica y personalizada. Cada usuario puede crear múltiples "páginas", que funcionan como contenedores de información, y dentro de ellas agregar **bloques modulares** como:

- Listas de verificación (checklists).
- Texto enriquecido (notas).
- Recordatorios con notificaciones.
- Adjuntos multimedia (imágenes, audio).
- Vínculos y datos de ubicación.

WADI está diseñada para funcionar de forma **offline-first** con almacenamiento local en **SQLite**, mientras que también ofrece sincronización **en tiempo real** y **respaldo en la nube** usando **Firebase Firestore**. Su arquitectura basada en **Clean Architecture** garantiza una separación clara entre la lógica de dominio, la lógica de presentación y la infraestructura, permitiendo escalabilidad, testeo y mantenibilidad a largo plazo.

La interfaz de usuario está inspirada en herramientas como **Notion** y **Google Tasks**, ofreciendo una experiencia moderna, responsive y altamente personalizable con soporte para temas, colores y fuentes.



4. Requisitos del Sistema

Para el correcto desarrollo, ejecución y despliegue de la aplicación **WADI**, se requieren ciertos componentes de hardware y software tanto para los dispositivos de desarrollo como para los dispositivos móviles donde se ejecutará la aplicación. A continuación, se detallan los requisitos mínimos y recomendados.

4.1 Requisitos del entorno de desarrollo

Hardware mínimo recomendado:

- **Procesador:** Intel Core i5 o equivalente (mínimo); i7 o superior (recomendado)
- **Memoria RAM:** 8 GB (mínimo); 16 GB o más (recomendado para compilación rápida)
- **Almacenamiento:** 10 GB de espacio libre (mínimo); SSD recomendado
- **Sistema operativo:**
 - Windows 10 o superior
 - macOS 12 o superior
 - Linux (Ubuntu 20.04 LTS o superior)
- **Conectividad:** Conexión a internet estable para sincronización con Firebase y descarga de dependencias

Software necesario:

- Flutter SDK (versión estable)
- Android Studio o Visual Studio Code con extensiones de Flutter y Dart
- Java Development Kit (JDK) 11 o superior
- Android SDK y AVD Manager (para pruebas en emulador)
- Git (para control de versiones)
- Node.js (opcional, para herramientas adicionales de despliegue)

- Firebase CLI (opcional para pruebas de funciones en la nube)

4.2 Requisitos del dispositivo móvil (usuario final)

Android:

- **Versión mínima del sistema operativo:** Android 7.0 (Nougat)
- **Almacenamiento libre:** 100 MB o más
- **RAM:** 2 GB (mínimo); 3 GB o más recomendado
- **Permisos requeridos:**
 - Acceso a almacenamiento
 - Cámara
 - Notificaciones
 - Ubicación (opcional)
 - Internet

iOS:

- **Versión mínima del sistema operativo:** iOS 13.0
- **Dispositivos compatibles:** iPhone 7 en adelante
- **Permisos requeridos:**
 - Acceso a cámara
 - Notificaciones
 - Archivos
 - Ubicación (opcional)
 - Conexión a internet

4.3 Servicios y herramientas externas

- **Firebase (Google Cloud):**
 - Firestore (base de datos NoSQL en tiempo real)
 - Firebase Authentication (autenticación por email, Google, Apple)
 - Firebase Storage (archivos multimedia)
 - Firebase Messaging (notificaciones push)
- **SQLite (almacenamiento local)**
- **Dart Packages:**
 - provider, get_it, hive, flutter_local_notifications, image_picker, shared_preferences, entre otros.

5. Configuración del Entorno de Desarrollo

La correcta configuración del entorno de desarrollo es crucial para garantizar un flujo de trabajo estable, eficiente y libre de errores durante todo el ciclo de vida del proyecto **WADI**. A continuación, se describen los pasos realizados y las herramientas necesarias para el desarrollo local bajo el enfoque de **Clean Architecture** con **Flutter** como framework principal.

5.1 Instalación de Prerrequisitos

Previo a la creación del proyecto, se instalaron las herramientas necesarias:

- **Flutter SDK:** Instalado desde el canal estable, incluyendo las herramientas de línea de comandos flutter y dart.
- **Android Studio:** IDE utilizado como entorno principal de desarrollo, configurado con el plugin de Flutter y Dart.
- **Node.js y NPM:** Requerido para herramientas complementarias como Firebase CLI.

- **Git:** Para el control de versiones del proyecto.
- **Firebase CLI:** Herramienta de línea de comandos para conectar y administrar los servicios de Firebase.
- **Visual Studio 2022:** Con la carga de trabajo "**Desarrollo de escritorio con C++**" habilitada, necesaria para compilar correctamente Flutter en Windows.

5.2 Configuración de Variables de Entorno

Se configuraron las siguientes rutas en la variable de entorno PATH:

- Ruta al binario de Flutter (flutter/bin)
- Ruta al caché de paquetes Dart (Pub Cache)
- Ruta a los ejecutables de NPM
- Ruta a Firebase CLI (firebase-tools)

Esto aseguró que los comandos flutter, dart, firebase y flutterfire fueran reconocidos globalmente desde cualquier terminal.

5.3 Configuraciones específicas para Windows

- **Modo Desarrollador:** Activado para permitir enlaces simbólicos necesarios para Flutter.
- **Multiplataforma habilitada:** Aunque el enfoque es móvil, se dejaron listas las configuraciones para Flutter Desktop con Windows por si se desea extender en el futuro.
- **Visual Studio Toolchain:** Instalado correctamente para evitar errores de compilación relacionados con dependencias nativas en Windows.

5.4 Creación y Estructura del Proyecto Flutter

- **Identificador del Proyecto:** Se usó el dominio invertido com.wadi.app.

- **Ruta del Proyecto:** El proyecto se ubicó en C:\dev\wadi, asegurando una ruta limpia y libre de espacios o caracteres especiales, los cuales pueden interferir con Gradle.
- **Estructura de Clean Architecture:** Se definió la siguiente estructura de carpetas en lib/:
 - app/: Punto de entrada principal (UI y lógica de arranque)
 - core/: Lógica base (temas, constantes, helpers)
 - features/: Carpetas modulares por funcionalidad
 - shared/: Componentes reutilizables (widgets, servicios)
 - injection/: Configuración de dependencias e inyección

5.5 Integración con Firebase y Dependencias

- **Proyecto Firebase:** Creado desde la consola de Firebase.
- **Integración automatizada:** Se utilizó flutterfire configure para enlazar el proyecto Firebase con el proyecto Flutter, generando automáticamente firebase_options.dart.
- **Configuración Android automática:** Firebase configuró google-services.json y actualizó build.gradle nativo.

Dependencias clave configuradas en pubspec.yaml:

- **Firebase:**
 - firebase_core, firebase_auth, cloud_firestore, firebase_storage, firebase_messaging
- **Arquitectura y estado:**
 - flutter_riverpod, get_it, equatable, dartz
- **Persistencia local:**
 - sqflite, path_provider
- **Utilidades:**

- o image_picker, geolocator, shared_preferences, flutter_local_notifications

5.6 Configuración de la Plataforma Android

Archivo android/app/build.gradle.kts:

- minSdkVersion definido en 21 para asegurar compatibilidad con la mayoría de dispositivos Android y plugins.
- multiDexEnabled = true activado para permitir exceder el límite de métodos de una sola DEX (requisito para Firebase).

Archivo AndroidManifest.xml:

- Se añadieron permisos esenciales:
 - o INTERNET, CAMERA, ACCESS_FINE_LOCATION, READ_EXTERNAL_STORAGE, POST_NOTIFICATIONS
- Se agregó un <provider> para FileProvider, necesario para la funcionalidad de captura de imágenes.

Archivo res/xml/provider_paths.xml:

- Creado para declarar rutas seguras de acceso a archivos, indispensable para que image_picker funcione correctamente.

6. Configuración del Código Fuente – Carpeta lib/

6.1 Resumen General del Estado del Proyecto

Con la configuración de la carpeta android/ finalizada, el desarrollo del proyecto **WADI** ha entrado en una fase clave: la implementación del esqueleto principal de la aplicación y su primera funcionalidad crítica: la **autenticación de usuarios**.

El trabajo se ha centrado en la carpeta lib/, estructurada según los principios de **Clean Architecture**. Esta fase se enfoca en un desarrollo modular, desacoplado y escalable, asegurando que la app sea mantenible, extensible y fácil de testear.

6.2 pubspec.yaml – El Manifiesto del Proyecto

El archivo pubspec.yaml define las dependencias que permiten dar vida a la aplicación. Las principales son:

Firebase Suite

- firebase_core: Inicialización central del ecosistema Firebase.
- firebase_auth: Autenticación de usuarios (registro, login, sesión).
- cloud_firestore: Base de datos NoSQL para almacenar entidades como usuarios, páginas y componentes.

Gestión de Estado y Arquitectura

- flutter_riverpod: Gestión reactiva del estado e inyección de dependencias.
- dartz: Proporciona el tipo Either para manejo funcional de errores.
- equatable: Comparación por valor para evitar errores lógicos en entidades y estados.

Utilidades

- `intl`: Para internacionalización y formatos localizados (soporte `es_MX`).
- `image_picker`: Captura y selección de imágenes desde galería o cámara.
- `geolocator`: Obtención de ubicación GPS, útil para futuras funciones geográficas.

6.3 Estructura de Carpetas en lib/

6.3.1 core/ – Base Común y Transversal

Esta carpeta reúne el código utilizado globalmente en la app:

- **constants/**:
Define constantes compartidas (colores, cadenas, íconos, nombres de colecciones Firebase).
- **errors/**:
Manejo centralizado de errores:
 - `exceptions.dart`: Captura errores técnicos de origen (por ejemplo, Firebase).
 - `failure.dart`: Traduce errores técnicos a mensajes amigables y manejables.
- **types/**:
 - `enums.dart`: Define enums como `ComponentType` o `Priority`.
 - `typedefs.dart`: Alias de tipos como `ResultFuture<T>` para firmas limpias y coherentes.
- **utils/ y widgets/**:
 - Utilidades (`date_utils.dart`) y widgets reutilizables (`PrimaryButton`, `EmptyStateWidget`) usados en toda la app.

6.3.2 shared/ – Elementos Reutilizables con Identidad WADI

Componentes estilizados con la estética y funcionalidad únicas de WADI:

- **theme/:**
 - `text_styles.dart`: Define la tipografía usada.
 - `app_theme.dart`: Configura ThemeData claro y oscuro con colores personalizados y consistentes.
- **widgets/:**
 - Elementos visuales personalizados como `WadiTextField` y `WadiButton`.

6.3.3 features/auth/ – Módulo de Autenticación

- ◆ **domain/ – Lógica de Negocio**
 - `entities/`: Modelo de dominio puro `UserEntity`.
 - `repositories/`: Contrato `AuthRepository` que define acciones como `login/logout`.
 - `usecases/`: Casos de uso (`LoginUseCase`, `RegisterUseCase`, `LogoutUseCase`).
- ◆ **data/ – Implementación Técnica**
 - `models/`: `UserModel` (DTO) con métodos `fromJson/toJson`.
 - `datasources/`: `AuthRemoteDataSource` conecta con Firebase Auth y Firestore.
 - `repositories/`: `AuthRepositoryImpl` traduce los resultados en `Failure` o `Success`.
- ◆ **presentation/ – Interfaz de Usuario**
 - `state/`:
 - `AuthState`: Define estados (inicial, cargando, error).
 - `AuthController`: Controlador con `StateNotifier` que ejecuta los casos de uso.
 - `screens/`:

- LoginScreen y RegisterScreen: Escuchan el AuthController y reaccionan mostrando loading o errores con SnackBar.

6.3.4 injection/ – Inyección de Dependencias

Contiene el archivo `injection.dart`, encargado de registrar los Provider con **Riverpod**. Este módulo une todas las capas de la arquitectura de manera desacoplada.

- Ejemplo:
La UI no necesita instanciar directamente un AuthController, solo lo obtiene vía `ref.read(authControllerProvider)`, mejorando testabilidad y mantenibilidad.

6.3.5 main.dart y app/app.dart – Punto de Entrada

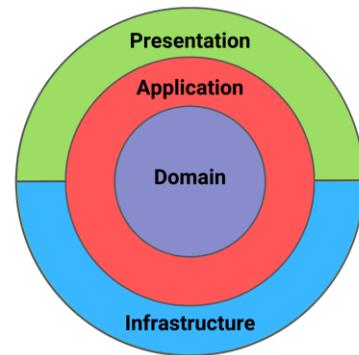
- **main.dart:**
 - Inicializa Firebase.
 - Envuelve la aplicación en un ProviderScope de Riverpod.
- **app/app.dart:**
 - Define WadiApp, que decide si mostrar la HomeScreen o LoginScreen según el estado del usuario.
 - Utiliza StreamProvider para reaccionar a cambios en tiempo real en el estado de autenticación.

7. Arquitectura de Software (Clean Architecture)

Estilo de Arquitectura Adoptado

WADI adopta una **arquitectura limpia (Clean Architecture)** en combinación con una **modularización por funcionalidades (feature-based modularization)**. Esta decisión estratégica permite que las distintas capas del sistema estén desacopladas, facilitando:

- El trabajo colaborativo entre desarrolladores.
- La escalabilidad del proyecto.
- La mantenibilidad del código a largo plazo.
- Una alta testabilidad en todas las capas.



Principales Capas

Se implementan tres capas bien definidas por módulo, siguiendo el patrón propuesto por la Clean Architecture:

- **Domain:**
Contiene la lógica de negocio pura, entidades (models sin dependencias externas) y los casos de uso. Esta capa no depende de ninguna otra.
- **Data:**
Gestiona el acceso a datos (remotos o locales). Implementa los repositorios definidos en el dominio y comunica con servicios como Firebase o bases de datos.
- **Presentation:**
Incluye la interfaz de usuario, el manejo del estado, y las pantallas visibles por el usuario. Se apoya en Riverpod para la gestión reactiva del estado.

8- Estructura de Carpetas del proyecto /lib

```
lib/
  └── main.dart          # Punto de entrada principal de la aplicación
  └── app/
    ├── app.dart          # Widget principal WadiApp
    └── router.dart        # Sistema de navegación y rutas

  └── core/               # Componentes base y utilidades transversales
    ├── constants/         # Paleta de colores, textos, íconos, constantes de
      └── Firebase
    ├── errors/             # Manejo y abstracción de excepciones y fallos
    ├── services/           # Servicios globales (notificaciones, autenticación,
      └── etc.)
    ├── utils/               # Funciones auxiliares como formatos de fecha,
      └── imágenes, validaciones
    ├── types/                # Tipos personalizados, enums, typedefs
    └── widgets/              # Componentes visuales reutilizables en toda la app

  └── features/            # Módulos funcionales organizados por dominio
    └── auth/                 # Módulo de autenticación (login, registro, logout)
      └── data/                  # Datasources, modelos y repositorio concreto
        └── domain/                # Entidad UserEntity, contrato AuthRepository,
          └── casos de uso
      └── presentation/          # Pantallas y estado del flujo de autenticación

    └── tasks/                  # Módulo de tareas (crear, ver, gestionar tareas)
    └── data/
```

```
| |   ├── domain/  
| |   └── presentation/  
  
|   ├── image_capture/      # Módulo para tomar, subir o mostrar imágenes  
|   ├── data/  
|   ├── domain/  
|   └── presentation/  
  
|   ├── settings/          # Módulo de configuración del usuario  
|   ├── data/  
|   ├── domain/  
|   └── presentation/  
  
|   └── ...                 # Otros módulos funcionales pueden añadirse aquí  
  
└── injection/             # Inyección de dependencias con get_it e injectable  
    ├── injection.dart  
    └── injection.config.dart  
  
└── shared/                # Componentes visuales compartidos con estilo  
WADI  
    ├── widgets/  
    └── theme/               # Tema visual (claro/oscuro, estilos de texto, etc.)
```

9. Base de Datos

La aplicación **WADI** utiliza una base de datos relacional para gestionar la información de usuarios, páginas, componentes modulares, multimedia, y configuración, garantizando integridad, seguridad y escalabilidad.

Se emplea una base de datos SQL (ej. MySQL o SQLite para local) estructurada en varias tablas relacionadas que reflejan la jerarquía y modularidad de la aplicación.

9.1 Estructura general de la base de datos

Tabla	Descripción
Usuario	Información de los usuarios con datos personales y seguridad (hash de contraseña y salt).
AssetsUsuario	Almacenamiento de archivos multimedia asociados a usuarios (imágenes, audio, documentos).
Pagina	Contenedores jerárquicos donde el usuario organiza sus bloques (tareas, notas, etc.).
Componente	Elementos modulares dentro de las páginas, que pueden ser listas, tareas, textos, imágenes, etc.
Tablas específicas	Información detallada según el tipo de componente (tareas, fechas, imágenes, texto, tablas).
Pagina_Compartida	Gestión de permisos para compartir páginas entre usuarios.
Historial_Cambios	Registro de acciones para auditoría y control de versiones.
Configuracion_Usuario	Preferencias personales del usuario (tema, idioma, notificaciones).

9.2 Diccionario de datos simplificado (ejemplo)

Tabla	Campo	Tipo	Descripción
Usuario	ID_Usuario	INT (PK)	Identificador único del usuario
	Nombre_Usuario	VARCHAR(50)	Nombre del usuario
	Correo_Usuario	VARCHAR(100)	Correo electrónico único
	Contraseña_Hash	TEXT	Contraseña cifrada
AssetsUsuario	ID_Asset	INT (PK)	Identificador del archivo multimedia
	Nombre_Asset	VARCHAR(255)	Nombre del archivo
	Tipo_Asset	ENUM	Tipo: imagen, video, documento, audio
Pagina	ID_Pagina	INT (PK)	Identificador de la página
	Nombre_Pagina	VARCHAR(100)	Nombre visible de la página
	ID_Usuario	INT (FK)	Propietario de la página
Componente	ID_Componente	INT (PK)	Identificador del componente
	Tipo_Componente	ENUM	Tipo: tarea, texto, imagen, etc.
	ID_Pagina	INT (FK)	Página donde se encuentra el componente
Componente_Tarea	Estado	BOOLEAN	Estado de la tarea (completada o no)
	Prioridad	ENUM	Prioridad asignada

Tabla	Campo	Tipo	Descripción
Pagina_Compartida	Permisos	ENUM	Nivel de acceso compartido

9.3 Relaciones principales

- Un **Usuario** puede tener muchas **Páginas**.
- Cada **Página** contiene múltiples **Componentes**.
- Los **Componentes** pueden tener jerarquía interna (componentes padres e hijos).
- Los archivos multimedia en **AssetsUsuario** se vinculan a componentes que los utilizan.
- Las páginas pueden ser compartidas con otros usuarios con diferentes niveles de permiso.
- El historial registra cambios importantes para auditoría.

9.4 Seguridad y rendimiento

- Las contraseñas se almacenan cifradas mediante hash + salt para mejorar la seguridad.
- Se implementan índices en campos clave para optimizar consultas frecuentes.
- Relaciones con integridad referencial para mantener la coherencia de los datos.

10. Módulo de Autenticación y Gestión de Cuentas

Se ha desarrollado un sistema de autenticación completo y robusto que va más allá de un simple login.

- **Métodos de Autenticación:**

- **Correo y Contraseña:** Implementación completa del flujo de registro y login.
- **Proveedores Externos:** Integración funcional con **Google Sign-In**. La estructura está preparada para añadir fácilmente otros proveedores como Apple.

- **Caché Local de Cuentas (AccountCacheService):**

- **Propósito:** Mejorar la experiencia de "cambiar de cuenta".
- **Implementación:** Se creó un servicio que, al ejecutarse, asegura la existencia de una tabla CachedUsers en la base de datos SQLite local.
- **Funcionalidad:** Después de cada inicio de sesión exitoso (sin importar el método), el AuthController invoca a este servicio para guardar o actualizar la información básica del usuario (ID, nombre, email y el **providerId** que indica cómo se autenticó, ej: 'google.com', 'password').

- **Módulo de Cambio de Cuenta (AccountSwitcherModal):**

- **Activación:** Se muestra al tocar el avatar del usuario en la HomeAppBar.
- **Visualización:** El modal lee la base de datos local a través del AccountCacheService y muestra una lista de todas las cuentas previamente guardadas.
- **Lógica de Cambio (switchAccount):** Al seleccionar una cuenta de la lista, el AuthController ejecuta la siguiente lógica:
 1. Cierra la sesión del usuario actual.
 2. Revisa el providerId de la cuenta seleccionada.

3. Llama al método de inicio de sesión correspondiente (`signInWithGoogle()` o `signInWithApple()`).
 4. Para cuentas 'password', simplemente cierra la sesión para que el router redirija a la pantalla de login, ya que un relogueo automático no es seguro.
- **Otras Acciones:** El modal también incluye opciones para "Cerrar sesión" y un placeholder para "Agregar otra cuenta" (que funcionalmente también cierra la sesión).

11. Notificaciones Push (Firebase Cloud Messaging)

11. Notificaciones Push

Las notificaciones push permiten a la aplicación enviar mensajes y alertas en tiempo real al usuario, incluso cuando la app está cerrada o en segundo plano. En una app de gestión de tareas, esto es crucial para recordar al usuario sobre tareas próximas o vencidas.

11.1 ¿Qué son las notificaciones push?

- Mensajes enviados desde un servidor o servicio externo que aparecen directamente en el dispositivo del usuario.
- No requieren que el usuario tenga la app abierta para recibirlas.
- Permiten mejorar la interacción y el compromiso con la app.

11.2 Firebase Cloud Messaging (FCM)

- Firebase Cloud Messaging es un servicio gratuito de Google que facilita el envío de notificaciones push a dispositivos Android, iOS y web.
- Es la plataforma más utilizada para gestionar notificaciones push en aplicaciones móviles.

11.3 Arquitectura de notificaciones en la app

- La app se conecta con Firebase para recibir un **token de dispositivo único**.
- El backend o el servicio de la app usa ese token para enviar mensajes personalizados.
- Firebase actúa como intermediario, gestionando la entrega eficiente de los mensajes.

11.4 Flujo de trabajo básico

1. Registro del dispositivo:

- La app solicita permiso para recibir notificaciones.
- Firebase genera y asigna un token único al dispositivo.

2. Envío de notificaciones:

- El servidor backend envía un mensaje a Firebase especificando el token del dispositivo y el contenido.

3. Recepción y muestra:

- Firebase entrega el mensaje al dispositivo.
- La app recibe la notificación y la muestra al usuario (banner, alerta o sonido).

11.5 Integración en la app

- Configurar Firebase en el proyecto (Android/iOS) con los archivos de configuración (google-services.json o GoogleService-Info.plist).
- Implementar la librería de Firebase Messaging.
- Manejar los permisos y el registro del token.
- Implementar la lógica para recibir y mostrar notificaciones cuando la app está en primer plano, segundo plano o cerrada.

11.6 Tipos de notificaciones en la app de gestión de tareas

- **Recordatorios de tareas:** Avisos previos a la hora programada para realizar una tarea.
- **Alertas de vencimiento:** Notificaciones cuando una tarea está próxima o pasada su fecha límite.
- **Actualizaciones del sistema:** Mensajes generales, por ejemplo, sobre nuevas funcionalidades o mantenimiento.

11.7 Personalización y acciones

- Las notificaciones pueden incluir botones de acción, como “Marcar como completada” o “Posponer”.
- Se pueden usar datos en el mensaje para abrir directamente una pantalla específica de la app.

11.8 Consideraciones y mejores prácticas

- Pedir permiso al usuario de manera clara y en el momento adecuado para aumentar la aceptación.
- No saturar con notificaciones para evitar molestia.
- Usar notificaciones programadas y condicionales para mejorar la relevancia.
- Implementar manejo de tokens expirados o inválidos.

11.9 Seguridad y privacidad

- Los tokens son únicos y deben protegerse para evitar envíos no autorizados.
- Los datos enviados deben respetar la privacidad del usuario, evitando información sensible en texto visible.

11.10 Monitoreo y análisis

- Firebase Console ofrece estadísticas sobre entregas, aperturas y conversiones de las notificaciones.
- Permite optimizar la estrategia de notificaciones basándose en el comportamiento de los usuarios.

12. Gestión de Tareas y Componentes Personalizados

12.1 Estructura Jerárquica de Gestión de Tareas

- **Páginas:** Cada usuario puede crear múltiples páginas, que actúan como contenedores temáticos o categóricos para organizar sus tareas y notas. Las páginas facilitan la agrupación lógica de la información.
- **Bloques Modulares:** Dentro de cada página, el usuario puede agregar diferentes tipos de bloques, que permiten construir contenido personalizado y dinámico.

12.2 Tipos de Bloques Modulares

WADI soporta los siguientes tipos de bloques, cada uno con funcionalidades específicas para enriquecer la gestión de tareas:

- **Listas de verificación (Checklists):** Permiten crear listas de tareas o sub-tareas con casillas para marcar progreso o finalización.
- **Texto enriquecido (Notas):** Bloques para agregar notas con formato, facilitando la inclusión de texto destacado, enlaces o listas.
- **Recordatorios con notificaciones:** Permiten asignar alertas temporizadas para recordar al usuario sobre tareas importantes, integrándose con el sistema de notificaciones push.
- **Adjuntos multimedia:** Soporte para insertar imágenes, grabaciones de audio u otros archivos, enriqueciendo el contexto de las tareas.
- **Vínculos y datos de ubicación:** Posibilidad de agregar enlaces a sitios web o referencias geográficas para una mejor contextualización.

12.3 Almacenamiento y Sincronización

- **Offline-first con SQLite:** La aplicación almacena toda la información localmente en una base de datos SQLite, garantizando acceso y funcionalidad sin conexión a internet.

- **Sincronización en tiempo real:** Utiliza Firebase Firestore para sincronizar los datos en la nube, permitiendo que los cambios realizados en un dispositivo se reflejen en otros, manteniendo la consistencia.
- **Respaldo y recuperación:** Gracias a la sincronización en la nube, los datos están respaldados y pueden recuperarse ante pérdida o cambio de dispositivo.

12.4 Arquitectura y Mantenibilidad

- WADI está diseñada bajo los principios de **Clean Architecture**, lo que asegura:
 - Separación clara entre la lógica de dominio (gestión de tareas y reglas de negocio), la lógica de presentación (interfaz de usuario) y la infraestructura (acceso a datos, servicios externos).
 - Facilita la escalabilidad, el testeo unitario y la mantenibilidad a largo plazo.
- Cada módulo de la app (páginas, bloques, sincronización) está desacoplado para permitir futuras extensiones y mejoras sin afectar otras partes.

12.5 Experiencia del Usuario y Personalización

- Los usuarios tienen total libertad para combinar diferentes bloques dentro de una misma página, creando así espacios de trabajo personalizados que se adaptan a sus necesidades y estilo.
- La interfaz intuitiva y modular facilita la rápida creación, edición y organización de tareas y contenidos multimedia

13. Diseño de Interfaz de Usuario (Material Design)

La interfaz de usuario (UI) de WADI está diseñada siguiendo los principios y componentes de **Material Design**, el sistema de diseño desarrollado por Google, que busca ofrecer una experiencia visual coherente, intuitiva y accesible en aplicaciones móviles multiplataforma.

13.1 Principios de Material Design aplicados en WADI

- **Claridad:** La UI prioriza la legibilidad y organización clara de la información para facilitar la interacción con las múltiples funcionalidades y bloques modulares.
- **Consistencia:** Uso coherente de colores, tipografías, iconografía y espacios, alineados con las guías oficiales de Material Design para asegurar familiaridad al usuario.
- **Jerarquía visual:** A través de tamaños, colores y sombras, se destacan elementos clave, como botones de acción, títulos y notificaciones importantes.
- **Retroalimentación:** La interfaz ofrece respuestas inmediatas a las acciones del usuario mediante animaciones sutiles, cambios de color y mensajes informativos.

13.2 Componentes Material Design utilizados

- **AppBar:** Barra superior que contiene el título de la pantalla, opciones de navegación y acciones contextuales.
- **Bottom Navigation Bar:** Navegación principal en la parte inferior que permite acceder a las secciones principales de la app.
- **Floating Action Button (FAB):** Botón flotante para acciones destacadas, como agregar una nueva página o tarea.
- **Cards:** Contenedores con sombra y bordes redondeados que organizan bloques modulares y elementos dentro de la UI.

- **Lists y Checkboxes:** Para la gestión de listas de verificación y elementos dentro de las páginas.
- **Dialogs y Snackbars:** Para interacciones temporales, confirmaciones y mensajes de feedback al usuario.

13.3 Personalización visual

- **Paleta de colores:** WADI utiliza una paleta de colores corporativos que aporta identidad visual y facilita la diferenciación entre estados e interacciones.
- **Tipografía:** Se usan fuentes legibles y modernas compatibles con Material Design, ajustando tamaños para títulos, subtítulos y cuerpo de texto.
- **Iconografía:** Íconos claros y reconocibles que apoyan la navegación y las acciones, siguiendo el estilo visual de Material Design.

13.4 Diseño adaptable y responsivo

- La UI está diseñada para adaptarse automáticamente a diferentes tamaños y orientaciones de pantalla, asegurando una experiencia óptima tanto en teléfonos como tablets.
- Se utilizan layouts flexibles y widgets responsivos que reacomodan el contenido según el espacio disponible.

13.5 Accesibilidad

- Se incorporan buenas prácticas de accesibilidad, como:
 - Contraste suficiente entre texto y fondo.
 - Soporte para lectores de pantalla.
 - Tamaños táctiles adecuados para botones y elementos interactivos.
 - Uso de etiquetas y descripciones accesibles.

14. Funcionalidades de la App

WADI (Write And Do It) es una aplicación móvil multiplataforma enfocada en la gestión eficiente y personalizada de tareas, que ofrece un conjunto robusto de funcionalidades para cubrir las necesidades de organización y productividad del usuario.

14.1 Creación y gestión de páginas

- Los usuarios pueden crear múltiples páginas que funcionan como contenedores para organizar tareas y notas por temas, proyectos o categorías.
- Permite editar, renombrar, duplicar y eliminar páginas según las necesidades del usuario.

14.2 Bloques modulares dentro de páginas

- Soporte para agregar distintos tipos de bloques que enriquecen el contenido:
 - Listas de verificación (checklists) para gestionar tareas y sub-tareas.
 - Texto enriquecido para notas detalladas con formato.
 - Recordatorios con alertas y notificaciones push.
 - Adjuntos multimedia como imágenes y audios.
 - Vínculos a sitios web y datos de ubicación geográfica.

14.3 Sistema de recordatorios y notificaciones

- Configuración de recordatorios temporizados para tareas importantes.
- Notificaciones push para alertar al usuario antes o en el momento de la tarea.
- Posibilidad de posponer o marcar tareas como completadas desde la notificación.

14.4 Gestión offline y sincronización en la nube

- Funciona offline con almacenamiento local en SQLite, asegurando acceso continuo sin conexión.
- Sincronización automática con Firebase Firestore para respaldo y actualización en tiempo real entre dispositivos.

14.5 Búsqueda y filtrado

- Herramientas para buscar tareas, notas o páginas por palabras clave.
- Filtrado avanzado por fecha, estado (completadas/pendientes) y tipo de bloque.

14.6 Personalización de la experiencia

- Interfaz modular que permite combinar libremente distintos bloques dentro de las páginas.
- Ajustes para configurar preferencias personales como temas, notificaciones y sincronización.

14.7 Seguridad y privacidad

- Autenticación segura para acceso a la cuenta del usuario.
- Encriptación y manejo responsable de datos almacenados y sincronizados.
- Control sobre permisos para acceso a multimedia y ubicación.

14.8 Soporte multimedia y enriquecimiento de contenido

- Inserción y reproducción de imágenes y grabaciones de audio dentro de las páginas.
- Inclusión de enlaces y mapas para referencias geográficas o recursos externos.

15. Escalabilidad y Futuras Expansiones

WADI está diseñada con una arquitectura flexible y modular que permite adaptarse y crecer conforme las necesidades de los usuarios y las tendencias tecnológicas evolucionen. Esta sección describe cómo la app soporta la escalabilidad y las posibles expansiones futuras.

15.1 Arquitectura modular y desacoplada

- La adopción de **Clean Architecture** permite que cada componente (gestión de páginas, bloques modulares, sincronización, UI, etc.) funcione de forma independiente y pueda evolucionar sin impactar otras partes.
- Facilita la incorporación de nuevas funcionalidades o módulos sin afectar la estabilidad existente.

15.2 Escalabilidad técnica

- La base de datos local (SQLite) y la sincronización en la nube (Firebase Firestore) están diseñadas para manejar un volumen creciente de datos y usuarios.
- Firebase ofrece escalabilidad automática y alta disponibilidad, soportando aumento en la cantidad de usuarios y dispositivos conectados.
- El sistema de notificaciones push puede ampliarse para manejar cargas mayores sin degradar la experiencia.

15.3 Futuras funcionalidades previstas

- **Integración con calendarios externos:** Sincronización con Google Calendar, Outlook, etc., para una gestión unificada de eventos y tareas.
- **Colaboración en tiempo real:** Permitir compartir páginas y bloques con otros usuarios para trabajo colaborativo.
- **Soporte para más tipos de bloques:** Incorporar nuevos formatos como tablas, diagramas, o integraciones con servicios externos.
- **Automatizaciones y AI:** Incorporar sugerencias inteligentes, recordatorios predictivos y análisis del comportamiento del usuario.
- **Multiplataforma ampliada:** Soporte para escritorio (Windows, macOS) y web para acceso desde cualquier dispositivo.

15.4 Mantenimiento y actualización continua

- El diseño modular facilita pruebas unitarias y de integración, asegurando calidad en las actualizaciones.
- Implementación de pipelines de integración continua (CI/CD) para desplegar mejoras rápidas y seguras.
- Monitoreo de rendimiento y errores para anticipar y resolver problemas antes de afectar a los usuarios.

15.5 Personalización y escalabilidad en UI

- La interfaz modular permite añadir nuevos componentes o modificar existentes sin romper la experiencia del usuario.
- Adaptabilidad a nuevas resoluciones, dispositivos y tecnologías de interacción.

16. Conclusión

El presente manual técnico de **WADI (Write And Do It)** ofrece una visión integral sobre el diseño, arquitectura, funcionalidades y componentes técnicos de la aplicación. Gracias a una estructura basada en Clean Architecture y modularización por funcionalidades, WADI garantiza una alta escalabilidad, mantenibilidad y calidad en la gestión personalizada de tareas.

La combinación de tecnologías como Flutter para el desarrollo multiplataforma, SQLite para almacenamiento local y Firebase para sincronización y notificaciones push permite ofrecer una experiencia robusta, flexible y segura para los usuarios, incluso en entornos offline.

Este manual servirá como guía fundamental para desarrolladores, testers y futuros colaboradores, facilitando el entendimiento del sistema y su evolución con funcionalidades futuras. Así, WADI está preparada para crecer, adaptarse a nuevas demandas y continuar mejorando la productividad y organización personal de sus usuarios.