INSTITUTO POLITECNICO NACIONAL

UPIICSA

PLAN DE MANTENIMIENTO DEL SISTEMA

Aplicación Móvil de Gestión de Tareas

Equipo 5 – Programación Móvil, Secuencia 6NM61

Profesor: Gustavo Martínez Vázquez

UPIICSA – Instituto Politécnico Nacional

Fecha de elaboración: 15 de mayo de 2025

CONTENIDO

OBJETIVO GENERAL

Garantizar la operatividad, estabilidad, escalabilidad, seguridad y evolución constante de la aplicación móvil WADI (Write & Do It), diseñada como un gestor de tareas y notas para Android, mediante la implementación de procesos sistemáticos de mantenimiento técnico, correctivo, evolutivo, adaptativo y preventivo.

OBJETIVOS PARTICULARES

- Establecer las directrices para el mantenimiento correctivo, preventivo, adaptativo y evolutivo del sistema.
- Asegurar la continuidad operativa del sistema incluso ante cambios tecnológicos o necesidades del usuario.
- Documentar procedimientos técnicos para corrección de errores, ajustes funcionales, pruebas de calidad, y control de versiones.
- Aplicar protocolos de respaldo, recuperación y pruebas automatizadas que reduzcan el riesgo de pérdida de información.
- Optimizar el rendimiento de la app mediante actualizaciones constantes en código, librerías y arquitectura.
- Proveer una estructura clara de roles y herramientas a utilizar para facilitar la ejecución del mantenimiento.
- Establecer indicadores medibles para la evaluación continua de la calidad del sistema y la eficacia del mantenimiento aplicado.
- Integrar flujos de trabajo DevOps para automatizar pruebas, construcción y despliegue de versiones.

ALCANCE DEL DOCUMENTO

Este plan contempla el mantenimiento completo del sistema WADI, incluyendo:

- Base de datos local (SQLite) y sincronización en la nube (Firebase Firestore).
- Componentes de interfaz desarrollados con Flutter 3.10.
- Sistema de autenticación mediante Firebase Auth.
- Funcionalidades principales: tareas, notas, multimedia, geolocalización, recordatorios y estructura jerárquica.
- Historial de cambios, recuperación de versiones y personalización de la interfaz.
- Notificaciones locales y sincronizadas.
- Gestión avanzada de cambios y restauración de datos.
- Uso exclusivo en dispositivos Android (versión mínima 8.0).

No contempla, por ahora:

- Versión web de WADI.
- Colaboración multiusuario en tiempo real.

- Implementación de Inteligencia Artificial para sugerencias automatizadas.
- Integración con otras plataformas (Google Calendar, Trello, etc.).

Usuarios esperados: Estudiantes, freelancers, trabajadores independientes, pequeños equipos de trabajo y público general con necesidad de organización avanzada offline y sincronizada.

TIPOS DE MANTENIMIENTO

WADI requiere un enfoque estructurado para cuatro tipos principales de mantenimiento, alineados con estándares profesionales:

Tipo de Mantenimiento	Propósito	Ejemplos Típicos	Herramientas	Frecuencia
Correctivo	**Resolver errores o fallos en ejecución.**	**Cierre de app al editar una tarea; botones que no responden.**	**Android Logcat, Crashlytics, GitHub Issues.**	**Inmediato tras detección.**
Preventivo	**Evitar errores futuros optimizando código y dependencias.**	**Advertencias en compilación, obsolescencia de plugins.**	**Linter, Detekt, dependabot.**	**Mensual.**
Evolutivo	**Agregar nuevas funciones a partir de necesidades.**	**Agregar subtareas o modo oscuro.**	**Figma, Firebase Remote Config, Flutter DevTools.**	**Trimestral o bajo demanda.**
Adaptativo	**Ajustar la app a nuevos entornos o tecnologías.**	**Soporte para nuevas versiones de Android, migración de Firebase.**	**Android SDK, Firebase CLI.**	**Cada cambio de entorno.**

A continuación, se describen detalladamente cada uno de ellos.

MANTENIMIENTO CORRECTIVO

Este mantenimiento se activa cuando se detectan errores funcionales o visuales que impiden o dificultan el uso normal de la app. Las tareas aquí incluyen:

- Análisis de logs generados por el sistema.
- Revisión de reportes en Firebase Crashlytics.
- Identificación del entorno en que ocurre el fallo (versión Android, dispositivo, actividad).
- Desarrollo y prueba del fix correspondiente.
- Aplicación del parche en rama hotfix y merge posterior a main con documentación del incidente.

Ejemplos reales detectados durante pruebas del equipo:

- La aplicación se cierra inesperadamente al hacer una “limpieza” total de la base de datos local.
- El botón “Nueva Página” permanece congelado si el usuario no tiene conexión a internet activa.
- Al cambiar el orden de páginas anidadas, se pierde la sincronización con Firestore.
- La interfaz presenta glitches visuales cuando se carga una imagen desde galería en dispositivos de gama baja.

Este mantenimiento se considera crítico y debe ser ejecutado bajo un protocolo rápido de acción.

MANTENIMIENTO PREVENTIVO

Este mantenimiento se aplica de forma periódica, incluso si no hay fallos visibles, con el objetivo de anticiparse a errores potenciales, asegurar la estabilidad del sistema y prolongar su vida útil.

Actividades clave:

- Revisión estática del código fuente con herramientas como Lint, Detekt y Flutter Analyze.
- Actualización de dependencias (paquetes de Flutter y plugins de Firebase) para evitar vulnerabilidades.
- Refactorización de módulos con código duplicado, complejidad alta o malas prácticas detectadas.
- Pruebas automatizadas de regresión con integración continua (GitHub Actions / Firebase Test Lab).
- Limpieza del repositorio de ramas obsoletas, archivos temporales y assets no utilizados.
- Evaluación del rendimiento en dispositivos de gama baja o media.

Casos detectados en fases previas:

- La librería cloud_firestore usada originalmente tenía una incompatibilidad con Android 13. Fue prevenida al revisar notas de versión.
- Algunas tareas sin título podían ser almacenadas debido a falta de validaciones, detectado por pruebas unitarias preventivas.

El mantenimiento preventivo debe ser registrado en bitácoras mensuales y comunicado al coordinador técnico para su aprobación antes de aplicación a producción.

MANTENIMIENTO EVOLUTIVO

Su propósito es ampliar las funcionalidades y adaptarse a las necesidades cambiantes de los usuarios, ya sea por mejoras propuestas, tendencias del mercado o análisis del feedback recolectado.

Actividades posibles:

- Adición de nuevas funcionalidades:
    - Subtareas dentro de una página.

    - Integración de grabación de audio como adjunto.
    - Historial visual de progreso por tarea.
- Mejora de la interfaz:
    - Rediseño de íconos, navegación, o tema oscuro automático.
    - Transiciones más suaves, animaciones y feedback háptico.
- Personalización avanzada:
    - Perfiles de usuario configurables.
    - Nuevas vistas para tareas: modo lista, tablero, calendario.
- Incorporación de análisis de uso:
    - Telemetría no intrusiva para saber qué funciones usan más los usuarios.

Ejemplos considerados:

- Varios usuarios reportaron necesidad de guardar enlaces o insertar videos: puede incorporarse un bloque multimedia HTML embebido.
- Algunas pruebas con estudiantes revelaron que prefieren crear checklists rápidas desde la pantalla de inicio: se propone un widget flotante.

Este mantenimiento puede planificarse tras cada ciclo de validación de entregables y pruebas con usuarios externos, preferentemente una vez cada trimestre.

MANTENIMIENTO ADAPTATIVO

Este tipo se ejecuta cuando hay cambios en el entorno que afectan el comportamiento de la app. Suele estar vinculado a actualizaciones del sistema operativo, herramientas de desarrollo o servicios externos (Firebase, Android SDK).

Situaciones típicas:

- Nuevas versiones de Android: Cambios en permisos, APIs, comportamiento de servicios en segundo plano.
- Cambios en SDK de Firebase: Reestructuración de reglas de seguridad, nuevos módulos de autenticación.
- Actualización en dependencias críticas como provider, riverpod, sqflite, firebase_core.

Acciones específicas:

- Refactorizar código deprecated por el SDK.
- Validar que las políticas de seguridad en Firestore sigan funcionando correctamente.
- Adaptar la app al nuevo modelo de almacenamiento en Android (por ejemplo, acceso restringido a carpetas externas).
- Recompilar y probar en dispositivos con versiones nuevas del sistema operativo.

Casos reales que podrían ocurrir:

- Google podría requerir el uso de targetSdkVersion 34 en la Play Store, obligando a cambiar cómo se manejan los permisos.
- Firebase puede forzar migración de Cloud Functions a una versión nueva del runtime.

Este mantenimiento es esencial para garantizar la continuidad operativa en entornos actualizados y debe ejecutarse antes de cada despliegue mayor.

PLANIFICACIÓN Y FRECUENCIA DE MANTENIMIENTO

A continuación, se presenta la programación sugerida para los tipos de mantenimiento aplicables a WADI, distribuidos en función de su urgencia, periodicidad y tipo de intervención.

Tipo de Mantenimiento	Periodicidad	Responsable Principal	Interacción con usuarios	Tiempo de respuesta esperado
Correctivo	**Inmediato (bajo demanda)**	**Responsable Técnico de Backend y UI**	**Alta**	**24-72 horas (según severidad)**
Preventivo	**Mensual (cada día 28)**	**Todo el equipo técnico**	**Nula**	**48-72 horas por ejecución**
Evolutivo	**Trimestral / Bajo demanda**	**Coordinador de Proyecto**	**Alta (retroalimentación)**	**1 semana de desarrollo por funcionalidad**
Adaptativo	**Cada cambio de entorno (SDK/API)**	**Coordinador Técnico + QA**	**Media**	**3 días a 1 semana**

El cumplimiento de estas actividades debe ser auditado mediante bitácoras técnicas, actas de revisión y validación de pruebas.

PERSONAL RESPONSABLE

El equipo de mantenimiento del sistema WADI está conformado por seis integrantes, distribuidos en roles estratégicos que cubren las áreas de programación, pruebas, diseño, documentación, soporte y gestión técnica.

Nombre Completo	Rol Principal	Funciones Específicas
Pérez Ibarra Manuel Sebastián	**Coordinador de mantenimiento técnico**	**Gestión del plan de mantenimiento, asignación de tareas, revisión de calidad, control de tiempos de respuesta y despliegue de parches críticos.**
Rodríguez Brenes Adriel	**Líder de interfaz y experiencia de usuario (UI/UX)**	**Encargado de rediseñar, corregir y mejorar la interfaz gráfica. Responsable de accesibilidad, estilos visuales y pruebas de usabilidad.**

Miguel Alarcón Adrián Manuel	**Responsable de backend y sincronización**	**Optimización de lógica offline y sincronización con Firebase Firestore. Encargado de compatibilidad con cambios de API.**
Flores Manzano Luis Jesús	**Encargado de control de versiones y documentación**	**Manejo de repositorios GitHub, control de ramas y bitácoras de mantenimiento. Encargado de mantener actualizado el historial técnico.**
Gutiérrez Sánchez César Axel	**Especialista en pruebas QA y validación**	**Responsable de ejecutar pruebas unitarias, automatizadas y funcionales. Detectar errores antes del despliegue de versiones.**
Ibarra Lima Julio Jair	**Desarrollador de funciones evolutivas y soporte**	**Encargado del desarrollo de nuevas funciones, mantenimiento evolutivo, y atención a peticiones surgidas por retroalimentación de usuarios.**

Todos los miembros participan en sesiones de revisión técnica quincenal, donde se presentan reportes de incidencias resueltas, bitácoras de mantenimiento, propuestas evolutivas y actualizaciones necesarias por cambios en entornos externos.

HERRAMIENTAS TECNOLÓGICAS UTILIZADAS

A continuación se enlistan las principales herramientas, entornos y servicios utilizados para el desarrollo, mantenimiento, prueba y control de versiones de la aplicación WADI:

Desarrollo y Entorno de Programación

- Flutter 3.10 y Dart 3.0: Framework y lenguaje principal para el desarrollo multiplataforma de la app.
- Android Studio: Entorno de desarrollo oficial con emuladores, herramientas de inspección visual y depuración.
- Visual Studio Code: Editor auxiliar para modificaciones ligeras, revisión de código y testing unitario.

Bases de Datos

- SQLite (mediante SQFlite o Floor): Base local persistente para operaciones offline.
- Firebase Firestore: Sincronización y respaldo de datos en la nube, integración con login y reglas de seguridad.

Pruebas y Validación

- Flutter Test / WidgetTester: Para pruebas de UI y funcionalidades individuales.
- Firebase Test Lab: Pruebas automatizadas en dispositivos físicos y virtuales.
- Emuladores y teléfonos reales: Verificación de compatibilidad en distintos modelos y versiones de Android.

Gestión de Proyecto y Versionamiento

- Git: Sistema de control de versiones distribuido.
- GitHub: Repositorio remoto colaborativo, con pull requests, issues, releases y branches por funcionalidad.
- GitHub Projects / Issues: Tablero de control para gestión de tareas de mantenimiento, bugs y evolución.

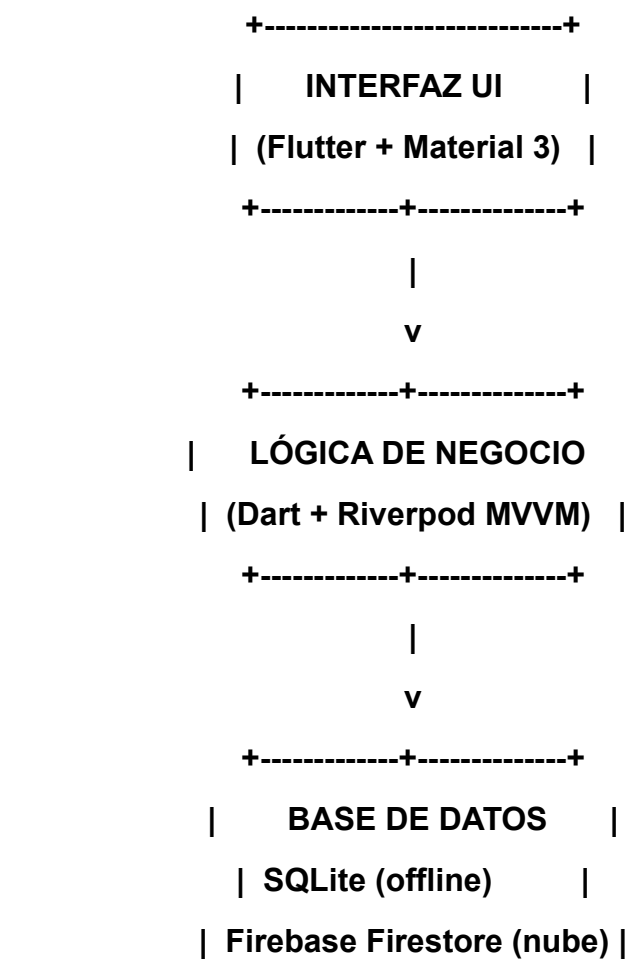
Diseño e Interfaz de Usuario

- Figma / Mockitt: Herramientas para diseño de mockups, prototipos y testeo de usabilidad.
- Material Design 3: Guía de estilo para interfaz Android.

Infraestructura en la Nube y Servicios Adicionales

- Firebase Authentication: Login y autenticación de usuarios.
- Firebase Cloud Messaging (FCM): Notificaciones push.
- Firebase Remote Config: Control dinámico de configuración sin necesidad de nueva versión.

RESUMEN GRÁFICO DEL STACK TECNOLÓGICO



ESTRATEGIA DE RESPALDO Y RECUPERACIÓN

Para evitar la pérdida de información crítica y garantizar la recuperación del sistema en caso de fallos, se implementa una política sistemática de respaldo automatizado y recuperación estructurada.

Políticas de Respaldo

1. **Backups del código fuente:**
    - Realizados con Git mediante commits firmados y ramas protegidas.
    - Push obligatorio a repositorio remoto (GitHub) una vez por día hábil.
2. **Exportación de APK:**
    - Cada nueva versión estable de la app genera un archivo .apk firmado.
    - Se almacena en carpeta compartida del equipo y en la nube (Google Drive).
3. **Documentación de cambios:**
    - Cada commit relevante debe incluir descripción técnica y referencia a bitácora de cambios.
    - Las bitácoras son revisadas y aprobadas quincenalmente por el Coordinador Técnico.
4. **Almacenamiento redundante:**
    - Google Drive compartido por el equipo.
    - Copia local de cada integrante sincronizada.
    - GitHub como respaldo central y seguro.

Procedimiento de Restauración (Ante corrupción de proyecto o pérdida crítica)

Caso A: Código dañado localmente pero versión válida en GitHub

1. Notificar al Coordinador Técnico y abrir issue en GitHub.
2. Eliminar localmente la carpeta afectada.
3. Clonar nuevamente el repositorio desde GitHub:

git clone https://github.com/equipo5/WADI.git

4. Cambiar a la rama estable (por ejemplo, main o release/v1.0.0):

git checkout main

5. Ejecutar flutter pub get para recuperar dependencias.
6. Ejecutar pruebas básicas con flutter test o abrir en Android Studio.
7. Confirmar funcionamiento y documentar el evento en la bitácora.

Caso B: Código dañado en GitHub y local

1. Buscar la copia más reciente del proyecto en la carpeta de respaldo local o Google Drive.
2. Validar que contenga el código fuente y pubspec.yaml funcional.
3. Restaurar manualmente desde esa carpeta (nueva ruta) y reiniciar entorno.
4. Configurar nuevamente las claves de API si se perdieron.
5. Ejecutar pruebas de validación y subir una nueva rama de respaldo (restore/YYYYMMDD).
6. Documentar todo el proceso en la bitácora y notificar al equipo.

Esta política garantiza que, incluso en escenarios críticos, el sistema WADI podrá ser restaurado en un plazo máximo de 24 horas hábiles, minimizando impacto en usuarios finales.

REGISTRO DE CAMBIOS Y CONTROL DE VERSIONES

Para mantener un desarrollo ordenado, reproducible y auditable, se implementa un control estricto del versionamiento y registro de cambios dentro del sistema WADI.

Convención de Versionado (SemVer)

Se adopta el estándar SemVer (Semantic Versioning), cuya sintaxis es:

vX.Y.Z

- X (Major): Versión mayor. Se incrementa cuando se realizan cambios incompatibles en la API o en la estructura del sistema.
- Y (Minor): Versión menor. Se incrementa cuando se agregan nuevas funcionalidades compatibles.
- Z (Patch): Parche. Se incrementa cuando se corrigen errores sin añadir nuevas funcionalidades.

Ejemplo:

- v1.0.0: Primer lanzamiento estable.
- v1.1.0: Se añade soporte para subtareas.
- v1.1.2: Se corrige bug de visualización de imágenes.

Flujo de Ramas en Git

- main: Rama estable, protegida, solo recibe merges aprobados de versiones probadas.
- develop: Rama de desarrollo, donde se integran las funcionalidades completadas por el equipo.
- feature/nombre-funcion: Ramas individuales para desarrollo de funciones específicas.
- hotfix/descripcion: Ramas para corregir errores críticos detectados en producción.
- release/x.y.z: Ramas para preparar versiones específicas antes del despliegue.

Registro de Cambios (CHANGELOG.md)

Se mantiene un archivo CHANGELOG.md actualizado con la siguiente estructura:

[1.1.0] - 2025-06-01

Agregado

- Funcionalidad de subtareas en páginas jerárquicas.

Corregido

- Error en animación de entrada para tareas con imágenes.

- Bug de sincronización con Firestore en modo offline prolongado.

Cada cambio realizado debe documentarse en este archivo en paralelo con su commit en Git, y vincularse con el número de Issue correspondiente.

Validación de Versiones

Antes de marcar una versión como estable (vX.Y.Z), debe:

- Superar todas las pruebas unitarias, de integración y de UI.
- Ser revisada por al menos 2 integrantes del equipo.
- Contar con respaldo (git tag) en el repositorio.
- Generar el correspondiente .apk firmado.

Este sistema garantiza trazabilidad, orden y calidad continua en el ciclo de vida de WADI.

INDICADORES DE RENDIMIENTO Y CALIDAD

La evaluación continua del rendimiento del sistema WADI y la calidad de su mantenimiento se fundamenta en indicadores técnicos y de gestión. Se clasifican en dos grandes grupos: indicadores del sistema y del proceso de mantenimiento.

Indicadores del Sistema WADI

Indicador	Descripción	Meta / Valor Esperado	Método de Medición
Tiempo promedio de respuesta UI	Tiempo entre la interacción del usuario y la ejecución de la acción.	< 250 ms	Herramientas de profiling en Flutter
Consumo de memoria en ejecución	Uso de RAM en dispositivos de gama media.	< 200 MB	Android Profiler
Tasa de retención a 7 días	Porcentaje de usuarios que continúan usando la app después de 7 días.	> 60%	Firebase Analytics
Tasa de finalización de tareas	Proporción de tareas creadas vs. completadas por los usuarios.	> 75%	Telemetría interna (no invasiva)
Caídas de aplicación por semana	Número de errores críticos (crashes) registrados.	0 o máximo 1	Firebase Crashlytics
Tiempo de sincronización completa	Tiempo promedio de sync local-nube sin errores.	< 1.5 segundos	Logs de sincronización interna

Indicadores del Proceso de Mantenimiento

Indicador	Descripción	Meta / Frecuencia Esperada	Herramienta / Proceso
Tiempo de respuesta ante bugs	Tiempo entre reporte de bug y su corrección confirmada.	24-72 horas según prioridad	GitHub Issues, Bitácora Técnica
Porcentaje de cobertura de pruebas	Cobertura de código alcanzada con pruebas automatizadas.	> 75%	Flutter Test Coverage, CI
Número de regresiones por versión	Incidencias que reaparecen después de ser corregidas.	0	Registro en changelog, QA
Cumplimiento de plan mensual	Número de tareas de mantenimiento ejecutadas vs. planificadas.	≥ 90%	Bitácora mensual
Frecuencia de actualizaciones	Número de versiones menores o parches generados por trimestre.	≥ 3	Historial de versiones GitHub

Revisión de dependencias	**Detección y actualización de librerías obsoletas.**	**Cada 30 días**	**Dependabot, Flutter pub outdated**

Estos indicadores serán auditados al cierre de cada ciclo mensual y reportados en las reuniones internas de revisión técnica. Las métricas permiten medir la eficiencia del mantenimiento y la satisfacción técnica de los usuarios.

PLAN DE ESCALAMIENTO Y SOPORTE POSTERIOR

El sistema WADI establece un proceso estructurado para la atención, gestión y escalamiento de incidentes que no pueden ser resueltos en el primer nivel técnico, así como para el soporte posterior a entregas estables.

Niveles de Escalamiento

Nivel 1 – Atención Inicial / QA / Soporte General

- Recepción del error o incidente mediante bitácora o canal interno.
- Validación del entorno: versión de app, dispositivo, pasos para reproducirlo.
- Clasificación por severidad (leve, media, crítica).
- Si se resuelve en este nivel, se documenta y cierra el caso.

Nivel 2 – Programador Especializado o Líder de Módulo

- Análisis de logs, revisión de código afectado.
- Identificación del componente exacto involucrado.
- Desarrollo y prueba de solución técnica.
- Si se requiere cambio estructural o el fallo persiste, escalar a nivel 3.

Nivel 3 – Coordinación Técnica o Revisión Global

- Análisis conjunto del equipo.
- Revisión de arquitectura, decisiones de diseño o dependencias externas.
- Rediseño parcial si es necesario.
- Plan de contingencia si el problema impacta el rendimiento general.

Esquema Jerárquico de Escalamiento Técnico

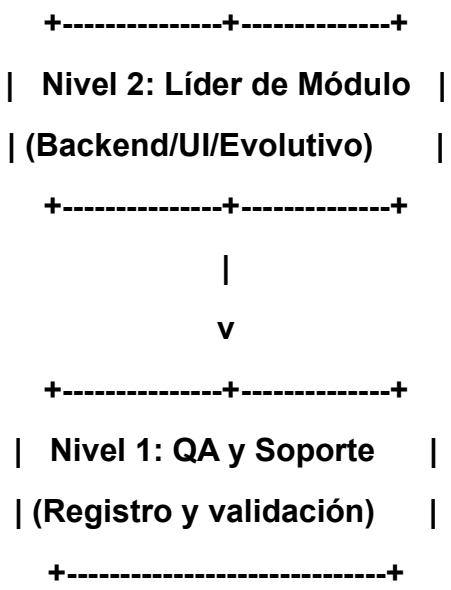
+----------------------------+

| Coordinador Técnico (Manuel P.)|

+---------------+--------------+

|

v



Cada caso que llegue al nivel 3 debe generar una entrada formal en la bitácora de mantenimiento, y, si se decide rediseño o cambio mayor, generar una nueva versión siguiendo la convención SemVer.

Soporte Posterior a Entregas Estables

- Período de observación: 14 días posteriores a cada release estable para detección de fallos no previstos.
- Canales de recepción de errores: Formulario interno, issues en GitHub, bitácora compartida.
- Reunión de revisión: Se programa al día 15 post-release para evaluación de desempeño y retroalimentación de usuarios.
- Documentación: Todo incidente surgido en este período debe documentarse y clasificarse en el changelog.

Este plan garantiza que todos los errores sean trazables, resueltos con responsabilidad y que ninguna anomalía permanezca sin respuesta técnica adecuada.

PROTOCOLOS DE COMUNICACIÓN DE INCIDENCIAS

Toda anomalía, fallo, bug o comportamiento inesperado debe ser reportado formalmente mediante protocolos establecidos, con el fin de garantizar su trazabilidad, priorización y resolución efectiva.

Canales oficiales de comunicación

1. Bitácora de Incidencias compartida (Google Docs o Notion): Formato estructurado de acceso común.
2. GitHub Issues: Para fallos técnicos complejos vinculados a código o dependencias.
3. Reuniones quincenales de seguimiento: Donde se revisan los reportes recibidos, se actualiza su estado y se reasignan tareas.
4. Mensajería interna (Discord/Telegram): Canal informal para consultas rápidas, que deben ser formalizadas luego en la bitácora.

Flujo de reporte

1. Usuario o integrante detecta el fallo.
2. Se registra el incidente en el formato oficial.
3. Se asigna prioridad (baja, media, alta, crítica).
4. El Coordinador Técnico asigna responsable para revisión.
5. Se documenta el avance y resolución.
6. Se actualiza la bitácora y changelog si aplica.

Plantilla oficial de reporte de incidencias

REPORTE DE INCIDENCIA

***Fecha:** [dd/mm/aaaa]*

***Reportado por:** [Nombre completo]*

***Versión de la app:** vX.Y.Z*

***Dispositivo / Versión de Android:** [Ej. Moto G8 - Android 11]*

***Título del error:** [Resumen breve del fallo]*

Descripción detallada del problema:

- ¿Qué se esperaba que ocurriera?:

- ¿Qué ocurrió realmente?:

- Pasos para reproducirlo:

1. ...

2. ...

3. ...

***Capturas o logs (si aplica):** [Adjuntar imágenes o log de consola]*

***Prioridad estimada:** Baja / Media / Alta / Crítica*

***Asignado a:** [Nombre o equipo]*

***Estado actual:** Abierto / En revisión / Solucionado / Cerrado*

Este formato debe estar disponible digitalmente en el repositorio del proyecto y también impreso en el expediente físico del equipo como referencia documental

PROCEDIMIENTO DE DOCUMENTACIÓN Y AUDITORÍA

Toda actividad técnica, correctiva, preventiva o evolutiva relacionada con el mantenimiento del sistema WADI debe quedar debidamente documentada, validada y archivada con fines de control, transparencia, trazabilidad y mejora continua.

Objetivos del proceso documental

- Establecer un repositorio organizado de bitácoras, reportes y versiones.
- Permitir auditorías internas o externas del estado del sistema.
- Asegurar la conservación de la memoria técnica del proyecto.
- Cumplir con las buenas prácticas de ingeniería del software.

Elementos obligatorios que deben documentarse

Documento / Evidencia	Responsable de creación	Medio de almacenamiento
Bitácora de mantenimiento mensual	**Todos (coordinado por Manuel)**	**Google Docs compartido**
Registro de incidencias (formato oficial)	**QA y usuarios**	**GitHub Issues + archivo digital**
Changelog de versiones (CHANGELOG.md)	**Desarrollador de versiones**	**Repositorio GitHub**
Plan de pruebas y resultados	**QA**	**Google Drive y respaldo impreso**
Reportes de reuniones quincenales	**Coordinador técnico**	**Carpeta de actas del equipo**
Archivos .apk firmados	**Responsable de release**	**Carpeta digital + almacenamiento local**

Frecuencia y responsable de auditoría

- La auditoría documental será semestral, con corte en junio y diciembre de cada año.
- Estará a cargo del Coordinador Técnico (Manuel P.) y validada por al menos un integrante externo al módulo auditado.

Proceso de auditoría semestral

1. Se recopilan todos los documentos obligatorios del semestre.

2. Se valida su integridad, coherencia y existencia física y digital.
3. Se revisa que el changelog refleje los commits realizados.
4. Se seleccionan 3 incidencias al azar y se verifica si fueron correctamente documentadas, asignadas, corregidas y cerradas.
5. Se genera un reporte de cumplimiento con observaciones y acciones pendientes.
6. Dicho reporte se presenta en reunión del equipo y se archiva en el expediente de proyecto.

Este procedimiento asegura que WADI evolucione con una base sólida, reproducible, y alineada a prácticas de calidad técnica profesional

ANEXOS

Anexo 1. Bitácora Técnica Modelo

Fecha	Tipo	Actividad realizada	Responsable	Resultado / Observaciones
28/05/25	**Correctivo**	**Fix de error en carga de imágenes**	**Luis J.**	**Versión estable v1.0.2, sin glitches**
03/06/25	**Evolutivo**	**Agregado de recordatorios visuales**	**Julio J.**	**Función operativa, pruebas exitosas**
10/06/25	**Preventivo**	**Refactorización de funciones con warnings**	**Miguel A.**	**Limpieza de código, sin errores nuevos**

Anexo 2. Guía Rápida de Restauración del Sistema

Caso: Corrupción o pérdida del proyecto local

1. Clonar nuevamente el repositorio desde GitHub.
2. Cambiar a la rama estable (main).
3. Ejecutar flutter pub get.
4. Validar funcionamiento y conectividad Firebase.
5. Confirmar versión estable (flutter run --release).
6. Registrar restauración en la bitácora técnica.

Caso: Fallo en sincronización Firebase o base local dañada

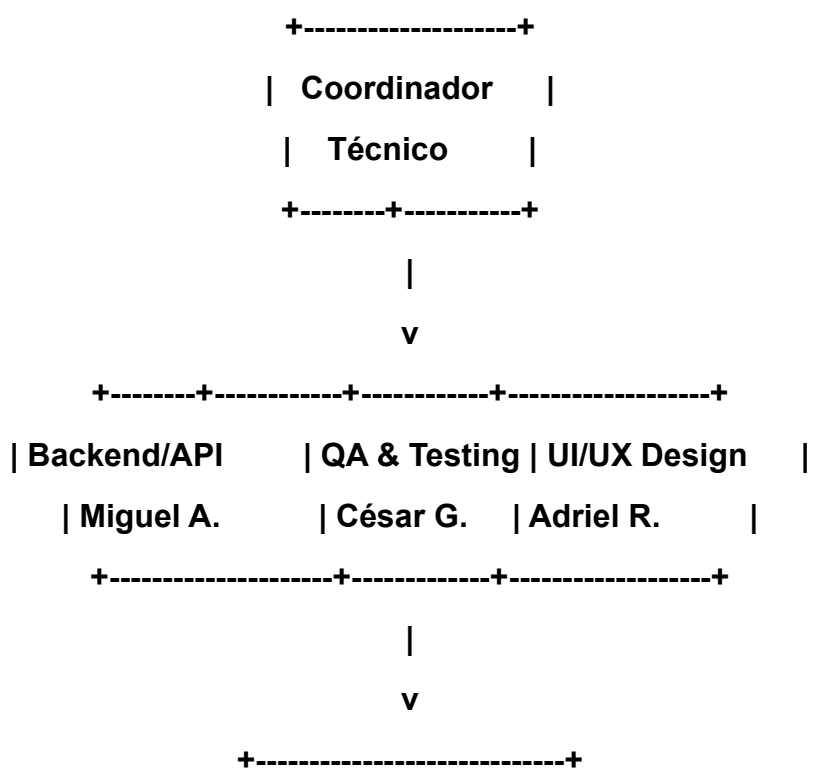
1. Limpiar datos de la aplicación desde configuración de Android.
2. Reinstalar versión más reciente desde archivo .apk firmado.
3. Validar acceso con usuario de prueba.
4. Consultar y restaurar Firestore desde Firebase Console si procede.
5. Informar al Coordinador Técnico para escalar si persiste.

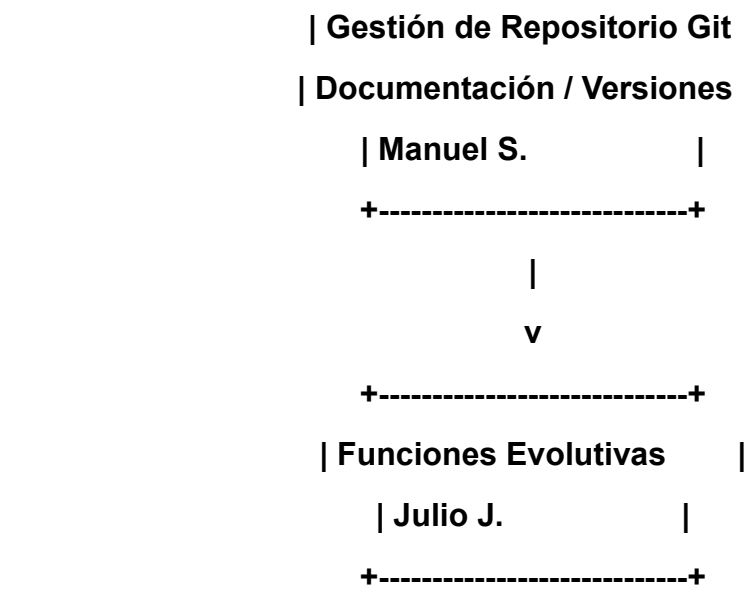
Anexo 3. Cronograma de Mantenimiento Semestral (Junio - Diciembre 2025)

Semana	Actividad Principal	Tipo	Responsable Principal
1 (3-7 junio)	Verificación de versiones y control de ramas	Preventivo	Pérez Ibarra M. Sebastián
2 (10-14 jun)	Fix de errores reportados por pruebas previas	Correctivo	Flores Manzano Luis
4 (24-28 jun)	Evaluación del rendimiento en gama media	Preventivo	Gutiérrez Sánchez César
6 (8-12 jul)	Prototipo de función: tareas recurrentes	Evolutivo	Ibarra Lima Julio
9 (29 jul - 2 ago)	Revisión de SDK y dependencias externas	Adaptativo	Miguel Alarcón Adrián
13 (26-30 ago)	Auditoría interna de documentación acumulada	Documental	Coordinador Técnico
17 (23-27 sep)	Preparación versión v1.2.0 y empaquetado	Release	Todo el equipo
22 (28 oct - 1 nov)	Revisión general y retroalimentación	QA / Revisión	Equipo completo

Este cronograma puede ajustarse dinámicamente según la evolución de funcionalidades y los incidentes detectados.

Anexo 4. Diagrama de Responsabilidades y Herramientas





Este conjunto de anexos respalda la trazabilidad, restauración y gestión técnica del mantenimiento, asegurando que el equipo WADI actúe con orden y estándares profesionales.