Aplicativo "Planta Externa"

Autor
Stdnt. Alejandro Silva

Objetivo

El propósito de este documento es proporcionar una descripción técnica detallada y exhaustiva de la aplicación "Planta Externa". Se exponen las decisiones de arquitectura, el stack tecnológico, los patrones de diseño implementados y los flujos de datos subyacentes que gobiernan su funcionamiento. Este informe sirve como una referencia fundamental para el equipo de desarrollo, para facilitar futuros mantenimientos y escalabilidad, y para cualquier personal técnico que requiera un entendimiento profundo de la construcción, operación interna y lógica de negocio del software.

Arquitectura de la Aplicación

La aplicación sigue una arquitectura pragmática y modular, diseñada para ser robusta y mantenible. Está centrada en la gestión del estado por pantalla (State Management a nivel de widget) y una clara separación de responsabilidades para desacoplar la lógica de la interfaz.

Capa de Interfaz de Usuario (UI)

- **Basada en Widgets:** Construida enteramente con el framework de Flutter, la totalidad de la interfaz de usuario se compone de widgets. Cada pantalla (Planilla1, Planilla2, etc.) es un StatefulWidget. Se optó por esta clase de widget porque el contenido de los formularios debe cambiar dinámicamente en respuesta a la interacción del usuario. Un StatefulWidget mantiene un objeto State asociado, que conserva la información mutable (valores de los campos, selecciones, etc.) a lo largo del ciclo de vida del widget.
- **Reactiva:** La interfaz se adhiere al paradigma reactivo de Flutter. Las actualizaciones visuales son una consecuencia directa de los cambios en el estado. Una llamada explícita a setState() notifica al framework que los datos internos han cambiado, lo que provoca que se vuelva a ejecutar el método build() del widget y se reconstruya la interfaz con la información actualizada. Esto asegura que la UI siempre sea un reflejo fiel del estado actual de los datos.

Gestión de Estado y Datos

- **Estado Local (Widget-Level State):** La mayoría del estado se maneja localmente dentro del objeto State de cada pantalla (_FormularioPlantaExternaState, _ReportGeneratorScreenState, etc.). Este enfoque es deliberadamente simple y altamente eficiente para formularios, donde los datos son específicos de una pantalla y no necesitan ser compartidos en tiempo real con otros componentes no relacionados de la aplicación. Se evita así la complejidad de soluciones de gestión de estado globales (como BLoC o Provider) que serían innecesarias para este caso de uso.
- **Singleton para Persistencia de Sesión (FormDataManager):** Para persistir los datos si el usuario navega entre pantallas o cierra y vuelve a abrir la aplicación, se implementó el patrón de diseño Singleton. La clase FormDataManager se instancia una única vez durante el ciclo de vida de la aplicación y actúa como un único punto de verdad (Single Source of Truth) que almacena en memoria los modelos de datos de cada planilla. Esto permite que, al volver a una planilla, la función _loadSavedData pueda consultar este Singleton y recargar los datos previamente ingresados, proporcionando una experiencia de usuario fluida y sin pérdida de información.

Lógica de Negocio

- **Lógica de Formularios:** La validación de entradas, el renderizado condicional de campos y la gestión de los TextEditingController residen directamente en los objetos State de los widgets de cada pantalla. Esta cohesión es intencional: la lógica que controla un elemento de la UI vive junto al widget que lo define, simplificando el razonamiento sobre el comportamiento de cada formulario.
- **Lógica de Reportes Aislada (report_logic.dart):** La lógica más compleja y reutilizable, como la generación de la estructura del archivo .zip y la composición de los datos para el PDF de la Planilla 2, se ha extraído a un archivo separado (report_logic.dart). Esta separación de responsabilidades es una práctica clave de software limpio: el código de la pantalla se enfoca en la UI y la gestión de estado, mientras que la lógica de negocio pura se encuentra en un módulo independiente, lo que mejora la legibilidad, facilita las pruebas unitarias y el mantenimiento.
- **Lógica de Generación de PDF:** La creación de los documentos PDF se realiza directamente en las funciones de exportación (_exportarPDF, _generarPDF) utilizando las APIs del paquete pdf. Este paquete proporciona un conjunto de widgets (pw.Text, pw.Table, pw.Image) que permiten construir un documento de forma programática y declarativa, de manera similar a como se construye una interfaz en Flutter.

Persistencia y Gestión de Archivos

- **Gestión de Archivos Temporales Seguros:** Cuando un usuario selecciona una foto o un archivo, la aplicación utiliza el paquete path_provider para obtener acceso a un directorio temporal gestionado por el sistema operativo. El archivo original se copia inmediatamente a esta ubicación segura. Este paso es crucial para la robustez de la aplicación, ya que desacopla a la app de la ubicación original del archivo (ej. Galería). Si el usuario borra la foto original, la aplicación conserva su copia temporal y puede seguir funcionando sin errores.
- **Generación de Salida en Directorios Definidos por el Usuario:** Los archivos finales (PDF y ZIP) se ensamblan y se guardan en la ruta de descarga que el usuario configura explícitamente en la pantalla de bienvenida. Se utiliza el paquete file_picker con la función getDirectoryPath() para permitir al usuario seleccionar dicho directorio, respetando así sus preferencias de organización y almacenamiento.

Stack Tecnológico

La aplicación aprovecha un conjunto de herramientas y paquetes modernos y bien mantenidos del ecosistema Flutter para ofrecer una funcionalidad robusta.

- **Framework:** Flutter. Elegido por su capacidad para compilar a código nativo de alto rendimiento para múltiples plataformas desde una única base de código, asegurando consistencia y eficiencia en el desarrollo.
- **Lenguaje:** Dart. El lenguaje en el que se basa Flutter, caracterizado por su tipado fuerte, compilación AOT (Ahead-of-Time) y JIT (Just-in-Time), y su optimización para la construcción de interfaces de usuario.
- **Paquetes Clave (Dependencias):**
    - **UI y Componentes:** flutter/material.dart es la base de toda la interfaz, proporcionando el conjunto completo de widgets visuales que siguen las directrices de Material Design.
    - **Generación de PDF:** pdf para la creación programática de documentos PDF y printing para ofrecer funcionalidades de previsualización e impresión nativas.
    - **Mapas y Geolocalización:** flutter_map fue elegido por ser una solución de mapas versátil y de código abierto que utiliza proveedores como OpenStreetMap, evitando dependencias con servicios de pago. geolocator es el estándar de la comunidad para interactuar con el hardware GPS del dispositivo, y latlong2 proporciona utilidades para el manejo de coordenadas.
    - **Selección de Archivos:** file_picker es una solución completa que permite al usuario seleccionar archivos (imágenes, PDFs) y directorios, abstrayendo las complejidades de los selectores de archivos nativos de cada plataforma.
    - **Compresión:** archive y archive_io son paquetes que ofrecen APIs de bajo nivel para la manipulación de archivos comprimidos, permitiendo crear los archivos .zip de forma eficiente.
    - **Gestión del Sistema de Archivos:** path_provider es esencial para acceder a directorios estándar del sistema operativo de una manera agnóstica a la plataforma, como el directorio temporal o el de documentos.
    - **Peticiones** de **Red:** http es el paquete estándar para realizar peticiones HTTP, utilizado en este caso para descargar las teselas de los mapas estáticos desde el servicio de OpenStreetMap para el reporte de la Planilla 1.

Pasos Técnicos en la Construcción

El ciclo de vida del desarrollo de la aplicación se puede desglosar en los siguientes pasos técnicos fundamentales:

1. **Configuración del Entorno y Dependencias:** Se inició un proyecto Flutter estándar y se añadieron las dependencias clave mencionadas anteriormente en el archivo de configuración pubspec.yaml, resolviendo las versiones para asegurar la compatibilidad.
2. **Diseño y Estructura de la Interfaz de Usuario (UI):**
    - Se creó la WelcomePage como punto de entrada y navegador principal.
    - Se desarrollaron las tres pantallas de planillas utilizando una estructura estándar de Scaffold, AppBar para la navegación y SingleChildScrollView para asegurar que los formularios extensos sean desplazables y no causen overflow de píxeles.
    - Se implementaron los formularios utilizando Form para la gestión y validación, y una combinación de TextFormField, DropdownButtonFormField y Switch para la captura de los diferentes tipos de datos.
3. **Implementación de la Lógica de Formularios Dinámicos:**
    - La reactividad se logró utilizando setState dentro de los callbacks de los widgets (ej. onChanged). Una selección en un DropdownButtonFormField dispara una llamada a setState, que a su vez provoca que la lógica dentro del método build re-evalue las condiciones y renderice o elimine widgets del árbol según sea necesario.
    - Para gestionar la gran cantidad de campos de texto que aparecen y desaparecen, se utilizaron mapas de TextEditingController (ej. _napCampos, _fdtCamposNo). Esto permite asociar dinámicamente un controlador a cada campo de texto, facilitando la gestión de sus valores.
4. **Integración de Funcionalidades Nativas y de Hardware:**
    - **Geolocalización:** Se integró geolocator para solicitar permisos de ubicación y obtener la posición actual del dispositivo. Se creó un widget reutilizable (GeoField) que encapsula un TextFormField y un IconButton, promoviendo la reutilización de código y la separación de conceptos.
    - **Selección de Archivos:** Se implementaron funciones asíncronas que invocan a file_picker para abrir el selector de archivos nativo. La ruta del archivo seleccionado se guarda en el State del widget para su posterior procesamiento.
5. **Desarrollo del Módulo de Generación de Reportes:**
    - **PDF Programático:** Se crearon funciones asíncronas (ej. _exportarPDF) que construyen el objeto pw.Document del paquete pdf. La lógica interna añade páginas y widgets de forma secuencial, poblándolos con los datos almacenados en el State del formulario.

- **Captura de Widgets a Imagen (Técnica Avanzada):** Para el mapa de la Planilla 1, se utilizó una técnica avanzada para convertir un widget de Flutter en una imagen. Un RepaintBoundary con una GlobalKey se envuelve alrededor de un widget de flutter_map que se renderiza fuera de la pantalla (usando OverlayEntry). Una vez que el mapa ha cargado sus teselas, se utiliza la key para obtener el RenderRepaintBoundary y llamar a su método toImage(), que convierte el widget renderizado en una imagen en formato Uint8List. Esta imagen se inserta luego en el PDF.
- **Compresión a ZIP:** Se implementó la lógica para tomar el Uint8List del PDF generado y las rutas de los archivos de evidencia, y empaquetarlos en un único archivo .zip utilizando la librería archive, listo para ser guardado y enviado.

6. **Implementación de la Persistencia de Datos entre Sesiones:**
   - Se definió la clase FormDataManager con un constructor estático privado y una instancia estática para implementar el patrón Singleton.
   - Se implementaron las funciones _guardarDatos y `_loadSavedData