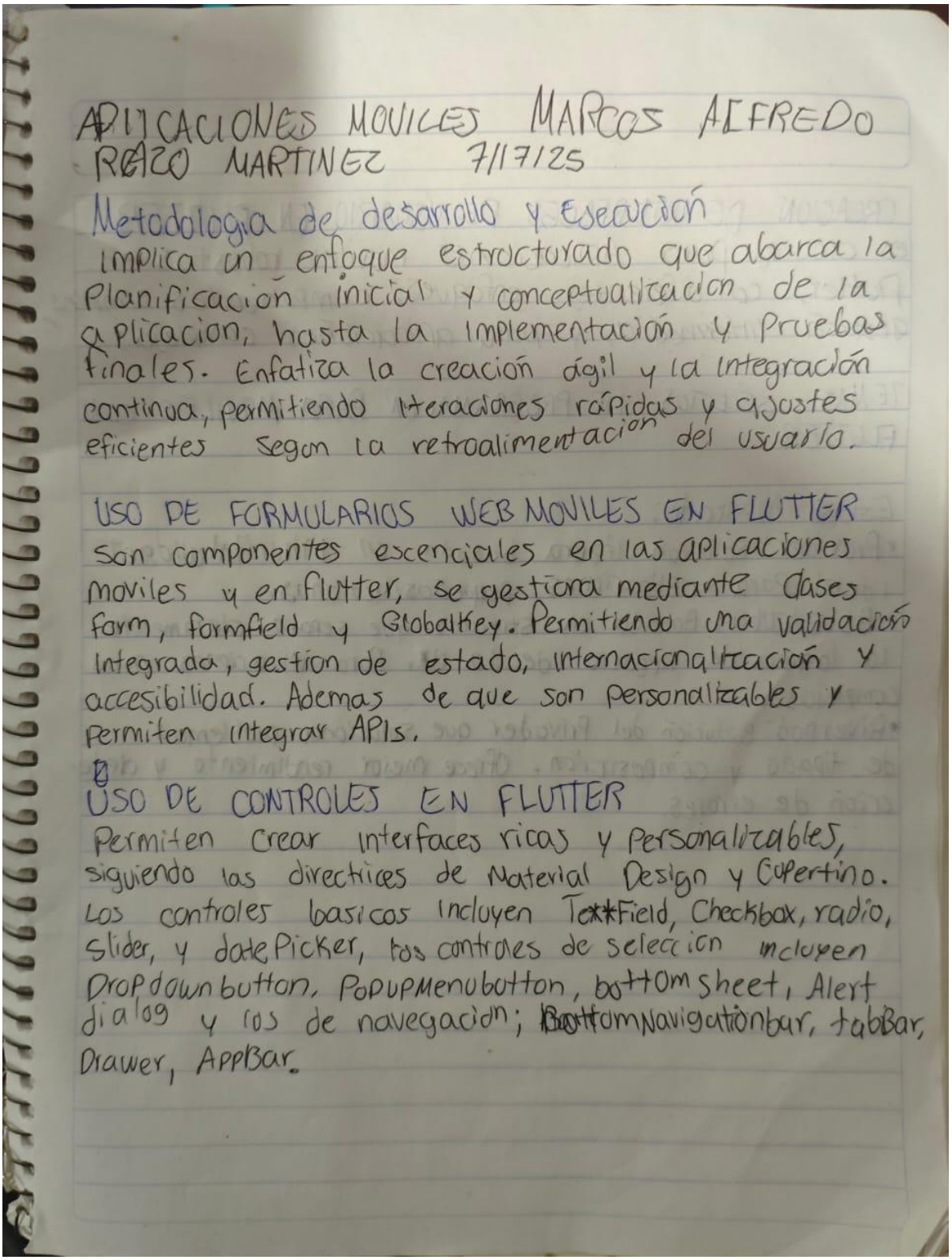
TEMAS DE LA UNIDAD 3 Y 4

UNIDAD 3



APLICACIONES MOVILES MARCOS ALFREDO RAZO MARTINEZ 7/17/25

Metodología de desarrollo y Ejecución

Implica un enfoque estructurado que abarca la planificación inicial y conceptualización de la aplicación, hasta la implementación y pruebas finales. Enfatiza la creación ágil y la integración continua, permitiendo iteraciones rápidas y ajustes eficientes según la retroalimentación del usuario.

USO DE FORMULARIOS WEB MOVILES EN FLUTTER

Son componentes esenciales en las aplicaciones moviles y en flutter, se gestiona mediante clases form, formfield y GlobalKey. Permitiendo una validación integrada, gestion de estado, internacionalización y accesibilidad. Ademas de que son personalizables y permiten integrar APIs.

USO DE CONTROLES EN FLUTTER

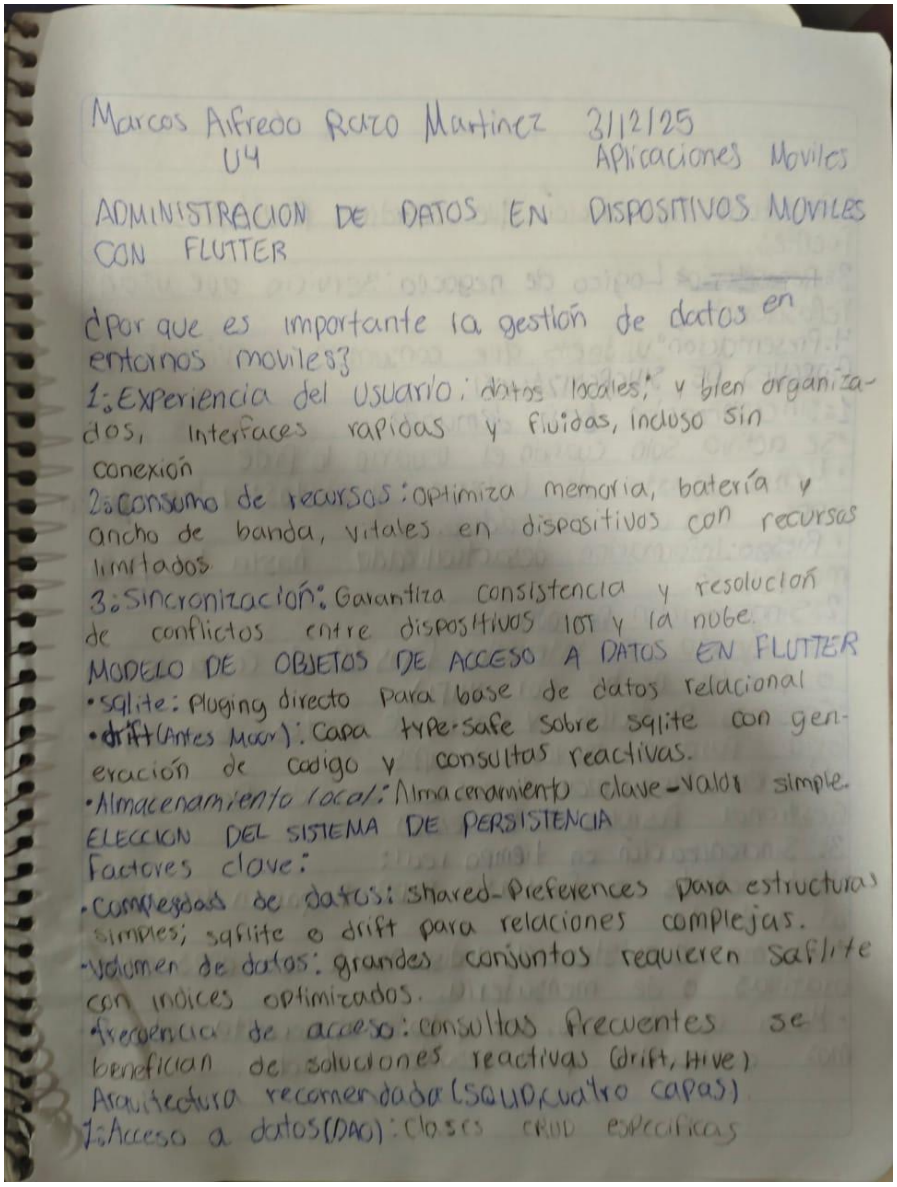
Permiten crear interfaces ricas y personalizables, siguiendo las directrices de Material Design y Cupertino. Los controles basicos incluyen TextField, Checkbox, radio, slider, y datePicker, los controles de seleccion incluyen Dropdownbutton, PopupMenubotton, bottomsheet, Alert dialog y los de navegación; BottomNavigationbar, tabBar, Drawer, AppBar.

CREACION DE INTERFACES DE USUARIO EN FLUTTER es crocial para Aplicaciones moviles robustas en flutter, con diferentes enfoques: componentes simples, gestión intermedia, y para aplicaciones complejas

TEMAS SELECTOS DE PROGRAMACION PARA MOVILES CON FLUTTER

Estado Avanzado:
- Provider: Solución ligera basada en inheritedwidget ideal para aplicaciones pequeñas/medianas.
- BLoC/Cubit: Basada en streams que separa claramente la lógica de negocio de la UI. Para aplicaciones mas complejas.
- Riverpod: Evolución del Privoder que soluciona problemas de tipado y composición. Ofrece mejor rendimiento y detección de errores.

UNIDAD 4



Marcos Alfredo Razo Martinez 3/12/25
U4 Aplicaciones Moviles

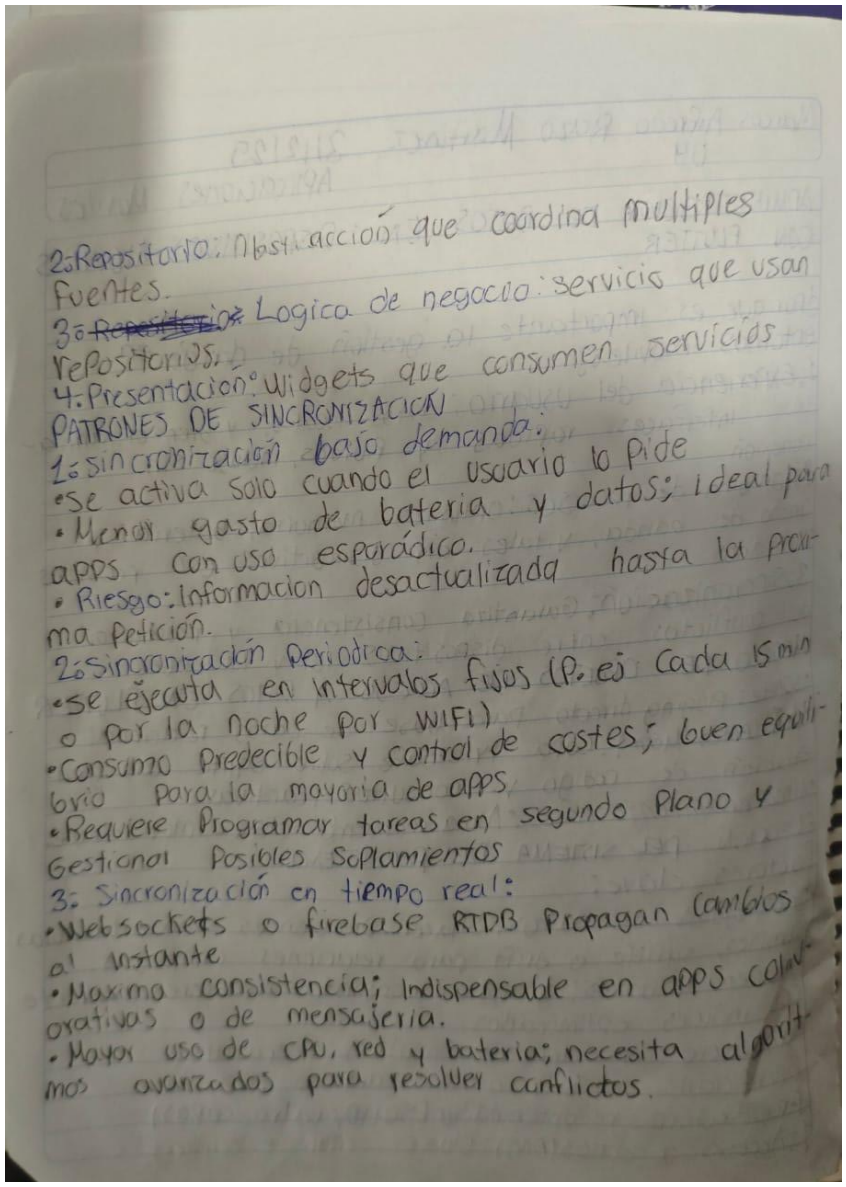
ADMINISTRACION DE DATOS EN DISPOSITIVOS MOVILES CON FLUTTER

¿Por que es importante la gestión de datos en entornos moviles?
1. Experiencia del usuario: datos locales, y bien organizados, interfaces rapidas y fluidas, incluso sin conexión
2. Consumo de recursos: optimiza memoria, batería y ancho de banda, vitales en dispositivos con recursos limitados
3. Sincronización: Garantiza consistencia y resolución de conflictos entre dispositivos IOT y la nube.

MODELO DE OBJETOS DE ACCESO A DATOS EN FLUTTER
- sqlite: Pluging directo para base de datos relacional
- drift (antes Moor): Capa type-safe sobre sqlite con generación de codigo y consultas reactivas.
- Almacenamiento local: Almacenamiento clave-valor simple.

ELECCION DEL SISTEMA DE PERSISTENCIA
Factores clave:
- Complejidad de datos: shared-Preferences para estructuras simples; sqflite o drift para relaciones complejas.
- Volumen de datos: grandes conjuntos requieren Saflite con indices optimizados.
- Frecuencia de acceso: consultas frecuentes se benefician de soluciones reactivas (drift, Hive)

Arquitectura recomendada (SOLID, cuatro capas)
1. Acceso a datos (DAO): Clases CRUD especificas



2. Repositorio: Abstracción que coordina multiples fuentes.
3. ~~Repositorio~~ Logica de negocio: servicios que usan repositorios.
4. Presentación: Widgets que consumen servicios
PATRONES DE SINCRONIZACIÓN
1. sincronización bajo demanda.
• se activa solo cuando el usuario lo pide
• Menor gasto de bateria y datos; ideal para apps con uso esporádico.
• Riesgo: Información desactualizada hasta la proxima petición.
2. Sincronización Periodica:
• se ejecuta en intervalos fijos (P. ej cada 15 min o por la noche por WIFI)
• Consumo predecible y control de costes; buen equilibrio para la mayoria de apps
• Requiere Programar tareas en segundo Plano y Gestionar Posibles Soplamientos
3. Sincronización en tiempo real:
• Websockets o firebase RTDB Propagan cambios al instante
• Maxima consistencia; Indispensable en apps colaborativas o de mensajeria.
• Mayor uso de CPU, red y bateria; necesita algoritmos avanzados para resolver conflictos.