Módulo 5: Almacenamiento de Datos Locales en Aplicaciones Móviles

Lección 1: Tipos de Almacenamiento de Datos Locales en Aplicaciones Móviles con Flutter

Objetivos de la Lección

- Comprender las diferentes opciones para almacenar datos locales en una aplicación móvil creada con Flutter.
- Evaluar y seleccionar el tipo de almacenamiento adecuado según las necesidades de la aplicación.
- Conocer las ventajas y desventajas de cada método de almacenamiento.

Introducción a la Lección

El almacenamiento de datos locales es esencial en el desarrollo de aplicaciones móviles, ya que permite que las aplicaciones funcionen sin conexión a Internet y almacenen información relevante, como configuraciones de usuario, datos estructurados, o registros que se deben sincronizar posteriormente. En Flutter, existen diversas opciones para almacenar datos locales, y cada una es adecuada para diferentes propósitos.

Tipos de Almacenamiento en Flutter

1. **Para Datos Simples y Preferencias del Usuario: `shared_preferences`**
    - **`shared_preferences`** es una biblioteca de Flutter que se utiliza para almacenar datos simples, como configuraciones de usuario, preferencias y pequeños valores de cadenas o enteros. Utiliza almacenamiento basado en pares clave-valor, similar al almacenamiento local en un navegador web.
        - **Ejemplos de Uso**:
            1. Guardar el tema preferido (oscuro o claro) de la aplicación.
            2. Almacenar el nombre del usuario o el estado de inicio de sesión.
        - **Ventajas**:
            1. Fácil de usar y configurar.
            2. Ideal para datos pequeños y preferencias de usuario.
        - **Desventajas**:
            1. No es adecuado para almacenar datos complejos o estructurados.
            2. Limitado en términos de capacidad de almacenamiento.
2. **Para Datos Estructurados y Complejos: `sqflite` y `hive`**
    - **`sqflite`**: Es una biblioteca de Flutter que proporciona una base de datos SQL ligera para almacenar y consultar datos estructurados. Es útil para aplicaciones que necesitan realizar consultas complejas o manejar grandes cantidades de datos relacionados.
        - **Ejemplos de Uso**:
            1. Almacenar datos de una lista de tareas o un inventario.
            2. Gestionar información de usuarios con relaciones (por ejemplo, un sistema de contactos).

        - **Ventajas**:
            1. Soporta operaciones complejas de SQL y es eficiente para consultas estructuradas.
            2. Adecuado para aplicaciones que necesitan mantener la integridad de datos.
        - **Desventajas**:
            1. Requiere un conocimiento básico de SQL y la configuración es más compleja que `shared_preferences`.
    - **`hive`**:
        - Es un almacenamiento de base de datos ligero, sin esquemas y rápido. Utiliza almacenamiento basado en cajas (similar a tablas) y es fácil de usar para datos estructurados.
        - **Ejemplos de Uso**:
            1. Almacenar datos de usuario, como configuraciones, historial de búsqueda o listas de favoritos.
            2. Almacenar datos sin conexión, como registros de seguimiento en una aplicación de fitness.
        - **Ventajas**:
            1. Muy rápido y eficiente, incluso con grandes volúmenes de datos.
            2. Funciona sin conexión y es fácil de configurar en Flutter.
        - **Desventajas**:
            1. No es tan robusto como `sqflite` para consultas complejas.
            2. Puede no ser adecuado para aplicaciones con relaciones de datos complicadas.

3. Para un Enfoque Reactivo y ORM: `moor` (Drift)

- **`moor`** (ahora llamado **Drift**) es un paquete que proporciona un enfoque orientado a objetos (ORM) para bases de datos en Flutter. Permite

consultas reactivas, lo que significa que las interfaces de usuario se actualizan automáticamente cuando los datos cambian.

- **Ejemplos de Uso**:
    1. Aplicaciones que necesitan una sincronización de datos en tiempo real.
    2. Sistemas donde la actualización automática de datos es importante, como un chat o una aplicación de tareas.
- **Ventajas**:
    1. Soporte para consultas reactivas y la capacidad de manejar relaciones complejas de datos.
    2. Las consultas son más fáciles de escribir y mantener con un enfoque ORM.
- **Desventajas**:
    1. La configuración es más compleja y requiere un conocimiento más profundo del manejo de bases de datos.
    2. Puede ser excesivo para aplicaciones que solo necesitan almacenamiento simple.

Comparación de las Opciones de Almacenamiento

Criterio	**`shared_preferences`**	**`sqflite`**	**`hive`**	**`moor` (Drift)**
Tipo de Datos	Simples (clave-valor)	Estructurados (SQL)	Estructurados (sin esquema)	Estructurados (ORM)
Uso Principal	Preferencias de usuario	Consultas SQL complejas	Datos rápidos y sin conexión	Consultas reactivas
Ventajas	Fácil de usar	Robusto y bien soportado	Muy rápido y eficiente	Reactivo y orientado a objetos

Criterio	`shared_preferences`	`sqflite`	`hive`	`moor` (Drift)
Desventajas	No apto para datos complejos	Configuración compleja	Menos adecuado para datos relacionales	Configuración compleja y detallada
Ejemplos de Aplicaciones	Configuración de temas, estado de usuario	Gestión de inventarios, bases de datos de usuarios	Datos sin conexión, almacenamiento temporal	Aplicaciones de chat, seguimiento de tareas

Consideraciones para Seleccionar el Tipo de Almacenamiento

1. **Si necesitas almacenar datos simples o preferencias del usuario**, como configuraciones de la aplicación o el estado de inicio de sesión, utiliza `shared_preferences`.
2. **Si tu aplicación maneja datos complejos y necesita realizar consultas SQL**, como un sistema de inventario o una aplicación de gestión de tareas, considera `sqflite`.
3. **Para almacenamiento ligero, sin conexión y con acceso rápido a datos**, como en una aplicación de seguimiento de actividades o almacenamiento de caché, `hive` es una buena opción.
4. **Para aplicaciones que necesitan un enfoque reactivo y consultas ORM**, como un sistema de gestión de proyectos o una aplicación de chat, utiliza `moor` (Drift).

Resumen de la Lección

En esta lección, hemos explorado las diferentes formas de almacenar datos locales en aplicaciones móviles creadas con Flutter. Aprendimos que:

- **`shared_preferences`** es útil para datos simples y preferencias de usuario.

- **`sqflite`** es ideal para aplicaciones que necesitan manejar datos estructurados y realizar consultas SQL complejas.
- **`hive`** es una base de datos rápida y eficiente para almacenamiento sin conexión.
- **`moor` (Drift)** ofrece un enfoque reactivo y orientado a objetos para gestionar datos estructurados.

La elección del tipo de almacenamiento depende del objetivo y la complejidad de los datos de tu aplicación.

Actividad de la Lección

Esta actividad te permitirá evaluar y aplicar tus conocimientos sobre el almacenamiento de datos locales en Flutter, comprendiendo cómo seleccionar el método adecuado según el caso de uso de la aplicación.

Instrucciones:

1. Imagina que estás desarrollando una aplicación de tareas pendientes (to-do list). Identifica el tipo de almacenamiento local que utilizarías y explica por qué. Describe cómo almacenarías las tareas y sus estados (completadas o pendientes).
2. Selecciona un segundo ejemplo: una aplicación de configuración de usuario (donde los usuarios pueden elegir temas, idiomas y preferencias de notificación). Explica qué tipo de almacenamiento utilizarías y por qué.
3. Desarrolla un documento en formato PDF con tus contestaciones y envíalas a través del enlace provisto para esta actividad.