import 'package:path/path.dart';

import 'package:sqflite/sqflite.dart';

// Tu clase Transaction (Modelo de Datos) - Ya la tienes

class Transaction {

final int? id; // El ID será autogenerado por la base de datos

final String tipo;

final String nombre;

final double cantidad;

final DateTime fecha;

Transaction({

this.id,

required this.tipo,

required this.nombre,

required this.cantidad,

required this.fecha,

});

// Convertir un Transaction en un Map para guardar en la base de datos

Map<String, dynamic> toMap() {

return {

'id': id, // Si es null, SQLite lo autogenerará

'tipo': tipo,

'nombre': nombre,

'cantidad': cantidad,

'fecha':

fecha

.toIso8601String(), // Guardamos la fecha como String (formato estándar)

};

}

// Convertir un Map (de la base de datos) en un objeto Transaction

factory Transaction.fromMap(Map<String, dynamic> map) {

return Transaction(

id: map['id'],

tipo: map['tipo'],

nombre: map['nombre'],

cantidad: map['cantidad'],

fecha: DateTime.parse(

map['fecha'],

), // Convertir el String de vuelta a DateTime

);

}

}

// Tu clase DatabaseHelper (Gestor de Base de Datos) - Esta la agregarás

class DatabaseHelper {

// Usamos un Singleton para tener una única instancia del helper

static final DatabaseHelper \_instance = DatabaseHelper.\_internal();

factory DatabaseHelper() => \_instance;

DatabaseHelper.\_internal();

static Database? \_database;

// Obtener la base de datos. Si no existe, la inicializa.

Future<Database> get database async {

if (\_database != null) return \_database!;

\_database = await \_initDatabase();

return \_database!;

}

// Inicializar (abrir/crear) la base de datos

Future<Database> \_initDatabase() async {

// Obtener la ruta del directorio de documentos de la app

String documentsPath = await getDatabasesPath();

String path = join(

documentsPath,

'finances.db',

); // Nombre del archivo de tu base de datos

// Abrir la base de datos. La función onCreate se llama si no existe.

return await openDatabase(

path,

version: 1, // Versión de tu esquema de base de datos

onCreate: \_onCreate, // Función para crear tablas si la DB es nueva

);

}

// Función que se ejecuta solo la primera vez que se crea la base de datos

Future<void> \_onCreate(Database db, int version) async {

// Comando SQL para crear la tabla 'transactions'

await db.execute(

'CREATE TABLE transactions(id INTEGER PRIMARY KEY AUTOINCREMENT, tipo TEXT, nombre TEXT, cantidad REAL, fecha TEXT)',

);

}

// --- Métodos CRUD (Crear, Leer, Actualizar, Eliminar) ---

// Insertar una nueva transacción en la base de datos

Future<int> insertTransaction(Transaction transaction) async {

Database db = await database;

return await db.insert(

'transactions',

transaction.toMap(),

); // Usa el método toMap()

}

// Obtener todas las transacciones de la base de datos

Future<List<Transaction>> getTransactions() async {

Database db = await database;

final List<Map<String, dynamic>> maps = await db.query(

'transactions',

); // Consulta todas las filas

// Convertir la lista de Maps (de la DB) a una lista de objetos Transaction

return List.generate(maps.length, (i) {

return Transaction.fromMap(maps[i]); // Usa el método fromMap()

});

}

// Eliminar una transacción específica por su ID

Future<int> deleteTransaction(int id) async {

Database db = await database;

return await db.delete(

'transactions',

where:

'id = ?', // Condición: donde la columna 'id' sea igual al argumento

whereArgs: [id], // El valor para la condición '?'

);

}

// Eliminar transacciones por tipo (usado para borrar ingresos o gastos)

Future<int> deleteTransactionsByType(String tipo) async {

Database db = await database;

return await db.delete(

'transactions',

where:

'tipo = ?', // Condición: donde la columna 'tipo' sea igual al argumento

whereArgs: [tipo], // El valor para la condición '?' ('Ingreso' o 'Gasto')

);

}

//cabrera

// Puedes añadir un método close() si necesitas cerrar la base de datos explícitamente,

// pero a menudo no es necesario en apps Flutter ya que el sistema lo maneja.

// Future<void> close() async {

// Database db = await database;

// db.close();

// }

}