📓 Manual Técnico Completo - Rastreador de Gastos Flutter

🎯 Resumen para Desarrollador

Has creado una aplicación Flutter completa con soporte **multiplataforma** (móvil + web) que gestiona gastos personales usando una base de datos local.

🏗️ 1. ARQUITECTURA Y PATRONES IMPLEMENTADOS

📦 Estructura del Proyecto

```
lib/
├── models/                    # Modelos de datos (DTOs/POJOs)
│   ├── category.dart          # Categoría de gasto
│   └── expense.dart           # Gasto individual
├── database/                  # Capa de acceso a datos
│   └── database_helper.dart   # Singleton + Repository Pattern
├── screens/                   # UI/Vistas (MVC Pattern)
│   ├── dashboard_screen.dart
│   ├── add_expense_screen.dart
│   └── expenses_list_screen.dart
└── main.dart                  # Bootstrap de la aplicación
```

🎨 Patrones de Diseño Implementados:

1. **Singleton Pattern** (`DatabaseHelper`)
   - Una sola instancia de base de datos
   - Acceso global controlado
2. **Repository Pattern** (`DatabaseHelper`)
   - Abstrae la lógica de acceso a datos
   - Facilita testing y mantenimiento
3. **Factory Pattern** (`Category.fromMap()`, `Expense.fromMap()`)
   - Construye objetos desde Map<String, dynamic>
   - Necesario para SQLite/JSON serialization
4. **Immutable Objects** (copyWith methods)
   - State management más seguro
   - Evita modificaciones accidentales

---

2. BASE DE DATOS - MANEJO MULTIPLATAFORMA

⚠ PROBLEMA ORIGINAL:

SQLite (`sqflite`) NO funciona en **Flutter Web** por limitaciones del navegador.

☑ SOLUCIÓN IMPLEMENTADA:

Detección de plataforma con fallback a almacenamiento en memoria para web.

```
import 'package:flutter/foundation.dart' show kIsWeb;

if (kIsWeb) {
  // Usar listas en memoria para web
  static List<Category> _webCategories = [];
  static List<Expense> _webExpenses = [];
} else {
  // Usar SQLite para móvil
  Database db = await openDatabase(path);
}
```

🔧 Implementación Técnica:

Para Móvil (Android/iOS):

- SQLite con `sqflite` package
- Tablas relacionales con Foreign Keys
- Transacciones ACID

Para Web:

- Almacenamiento en memoria (Arrays)
- Simulación de auto-increment IDs
- Persistencia durante la sesión

📊 Schema de Base de Datos:

```
-- Tabla categories
CREATE TABLE categories(
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,              -- "Comida", "Transporte"
  icon TEXT NOT NULL,              -- "restaurant", "directions_car"
  color TEXT NOT NULL              -- "#FF6B6B", "#4ECDC4"
);

-- Tabla expenses
CREATE TABLE expenses(
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  amount REAL NOT NULL,            -- 25.50
```

```
  description TEXT NOT NULL,       -- "Almuerzo en restaurante"
  date TEXT NOT NULL,              -- "2025-11-27T14:30:00.000Z"
  category_id INTEGER NOT NULL,    -- FK a categories.id
  FOREIGN KEY (category_id) REFERENCES categories (id)
);
```

🎯 3. MODELOS DE DATOS (Data Classes)

📋 Características Implementadas:

```
class Category {
  final int? id;              // Nullable para nuevos objetos
  final String name;
  final String icon;
  final String color;

  // Constructor named parameters
  Category({this.id, required this.name, ...});

  // Serialización DB ↔ Objeto
  factory Category.fromMap(Map<String, dynamic> map) { ... }
  Map<String, dynamic> toMap() { ... }

  // Immutabilidad
  Category copyWith({int? id, String? name, ...}) { ... }

  // Debug & Comparación
  @override String toString() { ... }
  @override bool operator ==(Object other) { ... }
  @override int get hashCode { ... }
}
```

💡 ¿Por qué esta estructura?

- **Inmutable:** Evita bugs de estado
- **Nullable ID:** Permite objetos nuevos sin ID asignado
- **Serializable:** Compatible con JSON/SQLite
- **Debuggable:** toString() facilita desarrollo

🖩 4. INTERFAZ DE USUARIO (UI)

🎨 Flutter Widgets Utilizados:

Layout & Navigation:

```
Scaffold                    // Estructura básica de pantalla
AppBar                      // Barra superior
FloatingActionButton        // Botón flotante (FAB)
Navigator.push/pop          // Navegación entre pantallas
```

Formularios & Input:

```
Form + GlobalKey<FormState>  // Manejo de formularios
TextFormField                // Campos de texto con validación
DropdownButtonFormField      // Selector de categorías
DatePicker                   // Selector de fecha
```

Listas & Cards:

```
ListView.builder            // Lista dinámica eficiente
Card                        // Contenedores con elevación
ListTile                    // Elementos de lista estándar
RefreshIndicator           // Pull-to-refresh
```

Estado & Datos:

```
StatefulWidget             // Widgets con estado mutable
setState()                 // Actualizar UI
FutureBuilder             // UI basada en Future (async)
CircularProgressIndicator // Indicador de carga
```

🔧 Validaciones Implementadas:

```dart
validator: (value) {
  if (value == null || value.isEmpty) {
    return 'Campo requerido';
  }

  final amount = double.tryParse(value);
  if (amount == null || amount <= 0) {
    return 'Monto inválido';
  }

  return null; // Validación OK
}
```

5. GESTIÓN DE ESTADO

Patrón Implementado: setState() + StatefulWidget

```
class _DashboardScreenState extends State<DashboardScreen> {
  double _monthlyTotal = 0.0;        // Estado local
  List<Expense> _recentExpenses = []; // Estado local
  bool _isLoading = true;            // Estado de carga

  @override
  void initState() {
    super.initState();
    _loadDashboardData();    // Cargar datos al inicializar
  }

  Future<void> _loadDashboardData() async {
    setState(() => _isLoading = true);

    // Llamadas async a la base de datos
    _monthlyTotal = await _databaseHelper.getTotalExpensesByPeriod(...);
    _recentExpenses = await _databaseHelper.getExpenses();

    setState(() => _isLoading = false);  // Actualizar UI
  }
}
```

Comunicación Entre Pantallas:

```
// Navegación con resultado
final result = await Navigator.push(context, MaterialPageRoute(...));
if (result == true) {
  _loadDashboardData(); // Recargar datos si se agregó algo
}

// Al regresar de AddExpenseScreen
Navigator.of(context).pop(true); // Indica que se guardó algo
```

---

6. DEPENDENCIAS Y CONFIGURACIÓN

pubspec.yaml Explicado:

```
dependencies:
  flutter: sdk: flutter
  cupertino_icons: ^1.0.8     # Iconos iOS-style

  # Base de datos
```

```
  sqflite: ^2.3.0             # SQLite para móviles
  sqflite_common_ffi: ^2.3.0  # SQLite para desktop
  path: ^1.8.3                # Manejo de rutas de archivos

  # UI y formateo
  fl_chart: ^0.66.0           # Gráficos (futuro uso)
  intl: ^0.19.0               # Internacionalización y fechas

dev_dependencies:
  flutter_test: sdk: flutter
  flutter_lints: ^6.0.0      # Reglas de calidad de código
```

🔧 ¿Qué hace cada dependencia?

- **sqflite:** Base de datos SQLite para Android/iOS
- **sqflite_common_ffi:** Permite SQLite en desktop/web (con configuración adicional)
- **path:** Construye rutas de archivos de forma portable
- **intl:** Formateo de fechas, números, localización
- **fl_chart:** Gráficos para futuras estadísticas

🔍 7. DEBUGGING Y RESOLUCIÓN DE PROBLEMAS

❓ ¿Por qué el código se veía gris?

Causa: Errores de compilación por imports faltantes o incompatibilidades

Soluciones aplicadas:

1. ☑ Creamos `category.dart` faltante
2. ☑ Agregamos soporte multiplataforma en `database_helper.dart`
3. ☑ Instalamos dependencias correctas con `flutter pub get`

❓ ¿Por qué SQLite no funcionaba en web?

Causa: Los navegadores web no soportan SQLite nativo

Solución:

```
// Detección de plataforma
import 'package:flutter/foundation.dart' show kIsWeb;

if (kIsWeb) {
  // Usar almacenamiento en memoria
  static List<Category> _webCategories = [];
} else {
  // Usar SQLite real
  Database db = await openDatabase(path);
}
```

🎯 Logs de Debug Agregados:

```
print('✅ Datos web inicializados con ${_webCategories.length} categorías');
print('💰 Web: Agregado gasto ${newExpense.description} -
\$${newExpense.amount}');
print('📊 Web: Total del período: \$${total.toStringAsFixed(2)}');
```

🎓 8. CONCEPTOS DE FLUTTER APRENDIDOS

🔤 Dart Language Features:

- **Null Safety:** `int?`, `!`, `??`
- **Named Parameters:** `required`, optional
- **Factory Constructors:** `factory Category.fromMap()`
- **Async/Await:** `Future<void>`, `async`, `await`
- **Collections:** `List<T>`, `Map<String, dynamic>`

📱 Flutter Framework:

- **Widget Tree:** Stateless vs Stateful
- **Lifecycle:** `initState()`, `dispose()`
- **Navigation:** `Navigator.push/pop`
- **Forms:** `Form`, `GlobalKey`, validations
- **Async UI:** `FutureBuilder`, `CircularProgressIndicator`

🎨 Material Design:

- **Theme:** `ColorScheme`, `ThemeData`
- **Components:** `Card`, `ListTile`, `AppBar`, `FAB`
- **Icons:** `Icons.restaurant`, `Icons.add`
- **Colors:** `Colors.red[700]`, custom hex colors

🚀 9. CÓMO EJECUTAR Y PROBAR

💻 Ejecutar en Web:

```
flutter run -d web-server --web-port 8080
# Luego abrir: http://localhost:8080
```

📱 Ejecutar en Móvil:

```
flutter devices                    # Ver dispositivos disponibles
flutter run -d <device-id>         # Ejecutar en dispositivo específico
```

🔧 Comandos de Desarrollo:

```
flutter pub get                    # Instalar dependencias
flutter clean                      # Limpiar build cache
flutter analyze                    # Verificar código
flutter test                       # Ejecutar tests
```

📋 10. FUNCIONALIDADES IMPLEMENTADAS

☑ Dashboard (Pantalla Principal):

- Resumen mensual automático
- Lista de gastos recientes (últimos 5)
- Navegación a otras pantallas
- Pull-to-refresh para actualizar datos

☑ Agregar Gastos:

- Formulario con validaciones completas
- Selector de categoría con preview visual
- Selector de fecha con DatePicker
- Feedback de guardado exitoso/error

☑ Lista de Gastos:

- Vista completa ordenada por fecha
- Búsqueda en tiempo real por descripción
- Filtrado por categoría
- Eliminación con confirmación

☑ Base de Datos:

- 6 categorías predefinidas con iconos y colores
- CRUD completo para gastos y categorías
- Soporte multiplataforma (móvil + web)
- Validaciones de integridad referencial

🎯 11. PRÓXIMOS PASOS DE APRENDIZAJE

📊 Nivel Intermedio:

1. **State Management Avanzado:** Provider, Riverpod, o BLoC
2. **Persistencia Real en Web:** IndexedDB o almacenamiento local
3. **Testing:** Unit tests, Widget tests, Integration tests
4. **Arquitectura:** Clean Architecture, MVVM

🔥 Funcionalidades Adicionales:

1. **Gráficos:** Usar `fl_chart` para estadísticas visuales
2. **Export/Import:** CSV, JSON backup
3. **Categorías Custom:** CRUD completo de categorías
4. **Presupuestos:** Límites por categoría con notificaciones
5. **Sincronización:** Firebase o API REST

🎨 UI/UX Mejoras:

1. **Dark Mode:** Soporte para tema oscuro
2. **Animations:** Hero transitions, animaciones de lista
3. **Responsivo:** Adaptación a tablets y desktop
4. **Accesibilidad:** Screen readers, contraste

💡 12. NOTAS IMPORTANTES PARA DESARROLLO

🔐 Buenas Prácticas Aplicadas:

- ☑ Validación de formularios
- ☑ Manejo de errores con try-catch
- ☑ Loading states para mejor UX
- ☑ Inmutabilidad en modelos de datos
- ☑ Separación de responsabilidades (UI vs Logic vs Data)

⚠ Limitaciones Actuales:

- 🖩 **Web:** Datos no persisten al cerrar navegador
- 🔄 **Sync:** No hay sincronización entre dispositivos
- 📊 **Analytics:** No hay métricas de uso
- 🔐 **Auth:** No hay autenticación de usuarios

🎯 Arquitectura Escalable:

La estructura actual es perfecta para aprender y permite escalamiento hacia:

- **Backend APIs** (REST/GraphQL)
- **Authentication** (Firebase Auth, Auth0)
- **Cloud Storage** (Firebase Firestore, Supabase)
- **Advanced State Management** (BLoC, Riverpod)

🏆 CONCLUSIÓN

¡Felicidades! Has construido una aplicación Flutter completa y funcional que demuestra conceptos fundamentales de desarrollo móvil moderno.

Lo que dominas ahora:

- ☑ Arquitectura de apps Flutter

- ☑ Manejo de base de datos local
- ☑ Formularios y validaciones
- ☑ Navegación entre pantallas
- ☑ Gestión de estado básica
- ☑ Desarrollo multiplataforma

Tu próximo objetivo: Experimentar con las funcionalidades existentes, agregar nuevas características, y escalar hacia arquitecturas más complejas cuando estés listo.

📚 Este manual sirve como referencia técnica completa del proyecto. Guárdalo para futuras consultas y modificaciones.