CRUD Usuarios - Flutter

Aplicación de Gestión de Usuarios con API REST

Información del Proyecto

- **Autor:** Jose Zuñiga
- **Fecha:** Noviembre 2024
- **Framework:** Flutter 3.x
- **Lenguaje:** Dart
- **API:** JSONPlaceholder (https://jsonplaceholder.typicode.com/users)

Tabla de Contenidos

Introducción

Este proyecto consiste en el desarrollo de una aplicación móvil Flutter que implementa un sistema CRUD (Create, Read, Update, Delete) completo para la gestión de usuarios. La aplicación consume una API REST externa (JSONPlaceholder) y proporciona una interfaz de usuario moderna y animada para realizar operaciones sobre los datos de usuarios.

La aplicación fue diseñada siguiendo las mejores prácticas de desarrollo móvil, implementando una arquitectura limpia y separación de responsabilidades, con especial énfasis en la experiencia del usuario mediante animaciones fluidas y un diseño intuitivo.

Objetivos

Objetivo General

Desarrollar una aplicación Flutter que permita gestionar datos de usuarios desde una API REST remota, implementando todas las operaciones CRUD de manera eficiente y con una interfaz de usuario atractiva.

Objetivos Específicos

- ☑ Implementar operaciones CRUD completas (Create, Read, Update, Delete)
- ☑ Consumir API REST externa de manera eficiente
- ☑ Diseñar una interfaz de usuario moderna y responsiva
- ☑ Implementar animaciones fluidas y micro-interacciones
- ☑ Manejar estados de carga y errores de manera elegante
- ☑ Aplicar arquitectura MVC (Model-View-Controller)
- ☑ Implementar validaciones de formularios
- ☑ Crear una experiencia de usuario intuitiva

🛠️ Tecnologías Utilizadas

Framework y Lenguaje

- **Flutter 3.x:** Framework de desarrollo multiplataforma
- **Dart:** Lenguaje de programación principal
- **Material 3:** Sistema de diseño de Google

Dependencias Principales

```
dependencies:
  flutter:
    sdk: flutter
  http: ^1.2.0              # Cliente HTTP para API REST
  cupertino_icons: ^1.0.8   # Iconos iOS
```

API Externa

- **JSONPlaceholder:** API REST gratuita para testing y prototipado
- **Endpoint:** https://jsonplaceholder.typicode.com/users
- **Métodos HTTP:** GET, POST, PUT, DELETE

🏗️ Arquitectura del Proyecto

Patrón Arquitectónico: MVC (Model-View-Controller)

```
lib/
├── main.dart                # Punto de entrada de la aplicación
├── models/
│   └── user.dart            # Modelos de datos (User, Address, Company)
├── services/
│   └── api_service.dart     # Controlador - Lógica de negocio y API
└── pages/
    ├── home_page.dart       # Vista - Lista de usuarios (página principal)
    ├── user_detail_page.dart # Vista - Detalle de usuario
    └── user_form_page.dart   # Vista - Formulario crear/editar
```

Separación de Responsabilidades

- **Models:** Definición de estructuras de datos y serialización JSON
- **Services:** Lógica de negocio, comunicación con API y gestión de estado
- **Pages:** Interfaces de usuario y manejo de eventos

Funcionalidades Implementadas

1. Gestión de Usuarios (CRUD)

CREATE - Crear Usuario

- Formulario básico para crear usuarios
- Campos principales: nombre, username, email
- Validación de campos obligatorios
- Creación de usuarios locales (ID 100+)

READ - Leer Usuarios

- Lista de usuarios desde API JSONPlaceholder
- Muestra 10 usuarios de la API + usuarios locales creados
- Diseño con cards básicas
- Información principal: nombre, username, email

UPDATE - Actualizar Usuario

- Formulario para editar usuarios existentes
- Pre-población de datos actuales
- Sistema de copias locales para usuarios de API
- Actualización real para usuarios locales

DELETE - Eliminar Usuario

- Eliminación de usuarios con confirmación
- Manejo diferenciado: usuarios API vs locales
- Sistema de marcado para usuarios eliminados de API

2. Funcionalidades Adicionales

- **Usuarios Fake:** Botón para generar 5 usuarios de prueba automáticamente
- **Detalles de Usuario:** Vista básica con información completa del usuario
- **Estados de Carga:** Indicadores durante la carga de datos
- **Manejo de Errores:** Mensajes básicos de error
- **Navegación:** Navegación entre pantallas funcional

Estructura de Archivos

Archivo Principal - main.dart

```dart
import 'package:flutter/material.dart';
import 'pages/home_page.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'CRUD Usuarios - Flutter',
      theme: ThemeData(
        useMaterial3: true,
        colorSchemeSeed: Colors.blue,
        appBarTheme: const AppBarTheme(centerTitle: true),
      ),
      home: const HomePage(),
      debugShowCheckedModeBanner: false,
    );
  }
}
```

💾 Código Fuente Principal

1. Modelo de Datos - user.dart

```dart
class User {
  int id;
  String name;
  String username;
  String email;
  String phone;
  String website;
  Address address;
  Company company;

  User({
    required this.id,
    required this.name,
    required this.username,
    required this.email,
    required this.phone,
    required this.website,
    required this.address,
```

```
    required this.company,
  });

  factory User.fromJson(Map<String, dynamic> json) {
    return User(
      id: json['id'] ?? 0,
      name: json['name'] ?? '',
      username: json['username'] ?? '',
      email: json['email'] ?? '',
      phone: json['phone'] ?? '',
      website: json['website'] ?? '',
      address: Address.fromJson(json['address'] ?? {}),
      company: Company.fromJson(json['company'] ?? {}),
    );
  }

  Map<String, dynamic> toJson() => {
    'name': name,
    'username': username,
    'email': email,
    'phone': phone,
    'website': website,
    'address': address.toJson(),
    'company': company.toJson(),
  };
}

class Address {
  String street;
  String suite;
  String city;
  String zipcode;

  Address({
    required this.street,
    required this.suite,
    required this.city,
    required this.zipcode,
  });

  factory Address.fromJson(Map<String, dynamic> json) {
    return Address(
      street: json['street'] ?? '',
      suite: json['suite'] ?? '',
      city: json['city'] ?? '',
      zipcode: json['zipcode'] ?? '',
    );
  }

  Map<String, dynamic> toJson() => {
    'street': street,
    'suite': suite,
    'city': city,
    'zipcode': zipcode,
```

```
    };
  }

  class Company {
    String name;
    String catchPhrase;
    String bs;

    Company({
      required this.name,
      required this.catchPhrase,
      required this.bs,
    });

    factory Company.fromJson(Map<String, dynamic> json) {
      return Company(
        name: json['name'] ?? '',
        catchPhrase: json['catchPhrase'] ?? '',
        bs: json['bs'] ?? '',
      );
    }

    Map<String, dynamic> toJson() => {
      'name': name,
      'catchPhrase': catchPhrase,
      'bs': bs,
    };
  }
```

2. Servicio API - api_service.dart (Fragmentos Principales)

```
class ApiService {
  static const baseUrl = 'https://jsonplaceholder.typicode.com/users';
  static List<User> _localUsers = [];
  static int _nextId = 100;

  // CREATE - Crear usuario
  Future<User> createUser(User user) async {
    await Future.delayed(const Duration(milliseconds: 500));

    final newUser = User(
      id: _nextId++,
      name: user.name,
      username: user.username,
      email: user.email,
      phone: user.phone,
      website: user.website,
      address: user.address,
      company: user.company,
    );
```

```
    _localUsers.add(newUser);
    return newUser;
  }

  // READ - Obtener todos los usuarios
  Future<List<User>> fetchUsers() async {
    try {
      final response = await http.get(Uri.parse(baseUrl));
      List<User> apiUsers = [];

      if (response.statusCode == 200) {
        final List data = jsonDecode(response.body);
        apiUsers = data.map((e) => User.fromJson(e)).toList();
      }

      // Filtrar usuarios eliminados y combinar con locales
      final deletedIds = _localUsers
          .where((u) => u.id < 0)
          .map((u) => -u.id)
          .toSet();

      apiUsers = apiUsers.where((u) => !deletedIds.contains(u.id)).toList();
      final validLocalUsers = _localUsers.where((u) => u.id > 0).toList();

      return [...apiUsers, ...validLocalUsers];
    } catch (e) {
      return _localUsers.where((u) => u.id > 0).toList();
    }
  }

  // UPDATE - Actualizar usuario (solo si hay cambios reales)
  Future<User> updateUser(int id, User user) async {
    await Future.delayed(const Duration(milliseconds: 500));

    // Lógica para detectar cambios reales y evitar duplicados
    if (id < 100) {
      final originalUser = await _getOriginalApiUser(id);
      if (originalUser != null && _hasRealChanges(originalUser, user)) {
        // Solo crear copia local si hay cambios reales
        final updatedUser = User(/* ... */);
        _localUsers.add(updatedUser);
        return updatedUser;
      } else if (originalUser != null) {
        return originalUser; // No hay cambios
      }
    }

    // Actualizar usuarios locales existentes
    final index = _localUsers.indexWhere((u) => u.id == id);
    if (index != -1) {
      _localUsers[index] = user;
      return user;
    }
```

```
      throw Exception('Usuario no encontrado');
    }

    // DELETE - Eliminar usuario
    Future<void> deleteUser(int id) async {
      await Future.delayed(const Duration(milliseconds: 500));

      _localUsers.removeWhere((u) => u.id == id);

      if (id < 100) {
        final alreadyDeleted = _localUsers.any((u) => u.id == -id);
        if (!alreadyDeleted) {
          _localUsers.add(User(
            id: -id, // Marcar como eliminado
            name: 'DELETED',
            // ... otros campos
          ));
        }
      }
    }
  }
```

3. Interfaz Principal - home_page.dart (Fragmentos Clave)

```
class _HomePageState extends State<HomePage> with TickerProviderStateMixin {
  final ApiService api = ApiService();
  List<User> users = [];
  bool isLoading = true;

  Widget _buildUserCard(User user, int index) {
    return TweenAnimationBuilder<double>(
      duration: Duration(milliseconds: 300 + (index * 50)),
      tween: Tween(begin: 0.0, end: 1.0),
      builder: (context, value, child) {
        return Transform.translate(
          offset: Offset(-100 * (1 - value), 0),
          child: Transform.scale(
            scale: 0.8 + (0.2 * value),
            child: Opacity(
              opacity: value,
              child: Card(
                elevation: 6 + (4 * value),
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(20),
                ),
                child: Container(
                  decoration: BoxDecoration(
                    borderRadius: BorderRadius.circular(20),
                    gradient: LinearGradient(
                      colors: user.id >= 100
                        ? [Colors.green.shade50, Colors.green.shade100]
```

```
                        : [Colors.white, Colors.blue.shade50],
                  ),
                ),
                child: ListTile(
                  leading: Hero(
                    tag: 'user_avatar_${user.id}',
                    child: Container(
                      width: 65,
                      height: 65,
                      decoration: BoxDecoration(
                        shape: BoxShape.circle,
                        gradient: LinearGradient(
                          colors: user.id >= 100
                            ? [Colors.green.shade400, Colors.teal.shade400]
                            : [Colors.blue.shade400, Colors.purple.shade400],
                        ),
                      ),
                      child: Center(
                        child: Text(
                          user.name.isNotEmpty ? user.name[0].toUpperCase() :
'?',
                          style: const TextStyle(
                            color: Colors.white,
                            fontSize: 26,
                            fontWeight: FontWeight.bold,
                          ),
                        ),
                      ),
                    ),
                  ),
                  title: Text(user.name),
                  subtitle: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      Text('@${user.username}'),
                      Text(user.email),
                      // Badge para usuarios locales o modificados
                      if (user.id >= 100 || api.isUserModified(user.id))
                        Container(
                          margin: const EdgeInsets.only(top: 8),
                          padding: const EdgeInsets.symmetric(horizontal: 10,
vertical: 6),
                          decoration: BoxDecoration(
                            gradient: LinearGradient(
                              colors: user.id >= 100
                                ? [Colors.green.shade100, Colors.green.shade200]
                                : [Colors.orange.shade100,
Colors.orange.shade200],
                            ),
                            borderRadius: BorderRadius.circular(15),
                          ),
                          child: Text(
                            user.id >= 100 ? 'Usuario Local' : 'Modificado',
                            style: TextStyle(
```

```
                                fontSize: 12,
                                fontWeight: FontWeight.bold,
                              ),
                            ),
                          ),
                        ],
                      ),
                      // ... resto del código
                    ),
                  ),
                ),
              ),
            ),
          );
        },
      );
    }
  }
```

📸 Capturas de Pantalla

1. Lista de Usuarios

- Muestra usuarios de la API JSONPlaceholder
- Cards básicas con información principal
- Botones para crear usuarios y usuarios fake
- Menú contextual para cada usuario

2. Formulario de Usuario

- Campos organizados por secciones
- Validación de campos obligatorios
- Modo crear y editar
- Interfaz limpia y funcional

3. Detalle de Usuario

- Información completa del usuario
- Datos organizados en secciones
- Vista de solo lectura
- Navegación de regreso

🎯 Características Técnicas Destacadas

1. Gestión Inteligente de Estado

- Diferenciación entre usuarios de API (ID 1-10) y locales (ID 100+)
- Detección de cambios reales para evitar duplicados innecesarios

- Sistema de marcado para usuarios eliminados

2. Interfaz de Usuario

- Diseño limpio con Material Design
- Cards para mostrar usuarios
- Formularios organizados por secciones
- Navegación intuitiva entre pantallas

3. Manejo de Errores Robusto

- Fallback a datos locales si falla la API
- Mensajes de error informativos
- Estados de carga elegantes

4. Validaciones de Formulario

- Campos obligatorios: nombre, username, email
- Validación de formato de email
- Trim automático de espacios en blanco

🔄 Flujo de Datos

Operaciones CRUD Detalladas

CREATE (Crear)

1. Usuario completa formulario
2. Validación de campos obligatorios
3. Asignación de ID único (100+)
4. Almacenamiento en lista local
5. Actualización de UI con animación
6. Mensaje de confirmación

READ (Leer)

1. Petición HTTP a JSONPlaceholder API
2. Deserialización de JSON a objetos User
3. Filtrado de usuarios eliminados
4. Combinación con usuarios locales
5. Renderizado con animaciones escalonadas

UPDATE (Actualizar)

1. Pre-población del formulario
2. Detección de cambios reales
3. Si hay cambios: creación de copia local
4. Si no hay cambios: retorno del original

5. Actualización de UI con indicadores visuales

DELETE (Eliminar)

1. Confirmación con diálogo animado
2. Eliminación de copias locales
3. Marcado como eliminado (usuarios API)
4. Actualización inmediata de UI
5. Mensaje de confirmación

📊 Métricas del Proyecto

Líneas de Código

- **Total:** ~1,500 líneas
- **Models:** ~150 líneas
- **Services:** ~400 líneas
- **Views:** ~950 líneas

Archivos Principales

- **main.dart:** Configuración de la aplicación
- **user.dart:** Modelos de datos (User, Address, Company)
- **api_service.dart:** Lógica de negocio y API
- **home_page.dart:** Lista de usuarios (página principal)
- **user_form_page.dart:** Formulario crear/editar
- **user_detail_page.dart:** Detalle de usuario

Funcionalidades

- ☑ 4 operaciones CRUD básicas implementadas
- ☑ 3 pantallas principales (Lista, Detalle, Formulario)
- ☑ Validaciones básicas de formulario
- ☑ 2 tipos de usuarios (API y Local)
- ☑ Sistema híbrido de gestión de datos

🎓 Reflexión Final

¿Qué aprendí del consumo de API REST?

1. Manejo de Estados Asíncronos

El desarrollo de esta aplicación me enseñó la importancia de manejar correctamente los estados asíncronos en Flutter. Implementar `FutureBuilder` para gestionar los diferentes estados de las peticiones HTTP (loading, success, error) fue fundamental para crear una experiencia de usuario fluida.

2. Arquitectura y Separación de Responsabilidades

Aplicar el patrón MVC me permitió mantener el código organizado y mantenible. La separación clara entre modelos, servicios y vistas facilitó el desarrollo y debugging del proyecto.

3. Serialización y Deserialización de Datos

Trabajar con JSON me enseñó la importancia de crear métodos `fromJson()` y `toJson()` robustos, manejando casos edge como valores nulos y tipos de datos inconsistentes.

4. Gestión de Estado Local vs Remoto

Una de las lecciones más valiosas fue aprender a combinar datos de una API externa con estado local, creando un sistema híbrido que permite operaciones CRUD completas incluso con APIs de solo lectura.

5. Manejo de Errores y Experiencia de Usuario

Implementar un manejo de errores robusto y proporcionar feedback visual constante al usuario mejoró significativamente la calidad de la aplicación.

6. Optimización de Rendimiento

Aprendí técnicas para optimizar las peticiones HTTP, como evitar llamadas innecesarias y implementar sistemas de caché local.

7. Diseño de Interfaces

Aprendí la importancia de crear interfaces limpias y funcionales que faciliten la interacción del usuario con los datos.

Desafíos Superados

- **Duplicación de usuarios:** Implementé lógica para detectar cambios reales y evitar copias innecesarias
- **Sincronización de datos:** Creé un sistema inteligente para combinar datos de API con modificaciones locales
- **Estados de carga:** Desarrollé indicadores visuales elegantes para mejorar la percepción de rendimiento
- **Validaciones complejas:** Implementé un sistema de validación robusto y user-friendly

Conclusión

Este proyecto me permitió aplicar conocimientos teóricos en un contexto práctico, desarrollando una aplicación completa que demuestra competencias en desarrollo móvil, consumo de APIs, diseño de interfaces y arquitectura de software. La experiencia adquirida será invaluable para futuros proyectos de desarrollo móvil.

📞 Información de Contacto

Desarrollador: Jose Zuñiga **Proyecto:** CRUD local Usuarios
Fecha: Noviembre 2024
Repositorio: [https://github.com/kyrafka/crud-local-flutter.git]
