📄 Guía Completa de Flutter para Examen - DAD

> 🎯 **Objetivo de este documento:** Explicarte Flutter desde CERO, como si nunca hubieras programado en este framework. Todo está explicado paso a paso, con el "qué", "cómo", "por qué" y "cuándo" de cada concepto.

📑 Índice de Contenidos

15. Ejercicio 4: Formularios Accesibles
16. Ejercicio 5: Pokedex (API + GridView)

🛠 ANTES DE EMPEZAR: Comandos y Estructura de Flutter

La Función `main()` - El Punto de Entrada

¿Qué es `main()`?
Es la función **obligatoria** que Flutter ejecuta cuando arrancas la app. Es como el botón de encendido.

```
void main() {
  runApp(MiApp()); // Arranca Flutter con tu app
}
```

¿Qué hace `runApp()`?
Le dice a Flutter: “Toma este widget y conviértelo en una aplicación completa”.

```
void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        body: Center(child: Text("Hola Mundo")),
      ),
    ),
  );
}
```

Estructura mínima de cualquier app Flutter:

```
import 'package:flutter/material.dart'; // Importar Flutter
```

```dart
void main() => runApp(MiApp()); // Punto de entrada

class MiApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text("Mi App")),
        body: Center(child: Text("Contenido")),
      ),
    );
  }
}
```

Comandos Útiles de Flutter

1. Ejecutar la App

```
flutter run
```

- Ejecuta el archivo `lib/main.dart` por defecto
- Abre un menú interactivo en la terminal

Opciones útiles:

```
flutter run -d chrome       # Ejecutar en navegador Chrome
flutter run -d windows      # Ejecutar en Windows
flutter run -d <device-id>  # Ejecutar en un dispositivo específico
```

2. Ejecutar un Archivo Específico

Si tienes **varios archivos con `main()`** (como en los ejercicios):

```
flutter run -t lib/ejercicio1.dart
flutter run -t lib/carrito.dart
```

¿Por qué `-t`?
`-t` significa “target” (objetivo). Le dices a Flutter: “No ejecutes `main.dart`, ejecuta ESTE arc hivo”.

3. Ver Dispositivos Disponibles

```
flutter devices
```

Muestra todos los dispositivos/emuladores donde puedes ejecutar la app.

4. Hot Reload y Hot Restart

Mientras la app está corriendo:

- Presiona `r` → **Hot Reload** (recarga cambios sin perder el estado)
- Presiona `R` → **Hot Restart** (reinicia la app desde cero)
- Presiona `q` → **Quit** (cerrar la app)

¿Cuándo usar cada uno?

- **Hot Reload (`r`):** Cambios en UI (colores, textos, widgets)
- **Hot Restart (`R`):** Cambios en lógica (variables de estado, funciones)

5. Instalar Dependencias

Después de añadir un paquete en `pubspec.yaml`:

```
flutter pub get
```

6. Limpiar el Proyecto

Si algo no funciona bien:

```
flutter clean
flutter pub get
```

Estructura de un Proyecto Flutter

```
mi_proyecto/
├── lib/
│   ├── main.dart          ← Archivo principal (punto de entrada)
│   ├── ejercicio1.dart    ← Otros archivos con main() (ejercicios)
│   └── models/            ← Carpeta para modelos (opcional)
├── assets/                ← Imágenes, fuentes, etc.
├── pubspec.yaml           ← Configuración y dependencias
└── test/                  ← Tests (opcional)
```

Archivos clave:

`lib/main.dart`

El archivo que Flutter ejecuta por defecto con `flutter run`.

`pubspec.yaml`

Configuración del proyecto. Aquí añades dependencias:

```yaml
name: mi_proyecto
description: Mi app de Flutter

dependencies:
  flutter:
    sdk: flutter
  provider: ^6.0.0      # Gestión de estado
  http: ^1.1.0          # Peticiones HTTP
```

```yaml
flutter:
  assets:
    - assets/          # Carpeta de imágenes
```

Después de modificar `pubspec.yaml`, SIEMPRE ejecuta:

```
flutter pub get
```

Problema: Múltiples Archivos con `main()`

Escenario: Tienes estos archivos:

```
lib/
├── main.dart          (tiene main())
├── ejercicio1.dart    (tiene main())
├── ejercicio2.dart    (tiene main())
```

¿Qué pasa si haces `flutter run`?
Flutter ejecuta **solo** `lib/main.dart`.

¿Cómo ejecutar los ejercicios?

Opción 1: Usar `-t` (recomendado)

```
flutter run -t lib/ejercicio1.dart
flutter run -t lib/ejercicio2.dart
```

Opción 2: Copiar y pegar
Copia el código del ejercicio en `lib/main.dart` y ejecuta `flutter run`.

Opción 3: Comentar el main() que no uses

```
// En main.dart, comenta el main() original:
```

```dart
// void main() => runApp(MiApp());

// Y descomenta el del ejercicio:
void main() => runApp(CarritoApp());
```

Imports Comunes

```dart
// Material Design (widgets de Android/iOS)
import 'package:flutter/material.dart';

// Provider (gestión de estado)
import 'package:provider/provider.dart';

// HTTP (peticiones a APIs)
import 'package:http/http.dart' as http;

// Convertir JSON
import 'dart:convert';

// Async/Await
import 'dart:async';
```

¿Por qué `as http`?
Para evitar conflictos de nombres. Usas `http.get()` en lugar de solo `get()`.

Consejos para el Examen

1. **Si te dan código con `main()`:** Cópialo en `lib/main.dart` y ejecuta `flutter run`
2. **Si falta algún import:** Flutter te dirá cuál falta (mira el error)
3. **Si falta una dependencia:** Añádela en `pubspec.yaml` y ejecuta `flutter pub get`
4. **Si algo no funciona:** `flutter clean` + `flutter pub get` + reiniciar
5. **Para probar rápido:** Usa `flutter run -d chrome` (más rápido que emulador)

---

1. 🚀 ¿Qué es Flutter y Dart?

¿Qué es Flutter?

Flutter es un **framework** (conjunto de herramientas) creado por Google para hacer aplicaciones móviles (Android e iOS), web y escritorio **con un solo código**.

Analogía: Imagina que en lugar de construir dos casas diferentes (una para Android y otra para iOS), construyes una sola casa que mágicamente funciona en ambos terrenos.

¿Qué es Dart?

Dart es el **lenguaje de programación** que usa Flutter. Es similar a JavaScript, Java o C#, pero optimizado para crear interfaces de usuario.

¿Por qué Dart?

- Es rápido (compila a código nativo)
- Es fácil de aprender si sabes JavaScript o Java
- Tiene “null safety” (te protege de errores comunes)

---

2. 🔷 Conceptos Básicos de Dart

Variables y Tipos de Datos

Declaración de Variables

```
// String: texto
String nombre = "Ana";

// int: números enteros
int edad = 25;

// double: números decimales
```

```dart
double precio = 19.99;

// bool: verdadero o falso
bool estaActivo = true;

// var: Dart adivina el tipo automáticamente
var ciudad = "Madrid"; // Dart sabe que es String
```

`final` vs `const`

```dart
// final: Se asigna UNA VEZ y no cambia (pero se calcula en tiempo de ejecución)
final DateTime ahora = DateTime.now(); // Se calcula cuando se ejecuta

// const: Se asigna UNA VEZ y NUNCA cambia (se calcula en tiempo de compilación)
const double PI = 3.14159; // Valor fijo desde el principio
```

¿Cuándo usar cada uno?

- `final`: Cuando el valor se calcula al ejecutar (ej: fecha actual, resultado de una función)
- `const`: Cuando el valor es fijo y conocido desde antes (ej: colores, textos fijos)

Null Safety (El `?` y el `!`)

Dart te obliga a ser explícito sobre si una variable puede ser `null` (vacía) o no.

```dart
// SIN el ?: La variable DEBE tener un valor, no puede ser null
String nombre = "Ana"; // ✅ Correcto
String apellido; // ❌ ERROR: Falta inicializar

// CON el ?: La variable PUEDE ser null
String? apodo; // ✅ Correcto, puede ser null
apodo = "Anita"; // Ahora tiene valor
```

```
apodo = null; // También válido
```

Operadores útiles:

```
String? nombre;

// Operador ??: "Si es null, usa esto"
String saludo = nombre ?? "Invitado"; // Si nombre es null, usa "Invitado"

// Operador !: "Te prometo que NO es null"
print(nombre!.length); // ⚠ PELIGRO: Si nombre es null, la app explota
```

Listas (Arrays)

```
// Lista de Strings
List<String> frutas = ["Manzana", "Pera", "Plátano"];

// Acceder a elementos (empieza en 0)
print(frutas[0]); // "Manzana"

// Añadir elementos
frutas.add("Naranja");

// Longitud de la lista
print(frutas.length); // 4
```

Mapas (Diccionarios / Objetos JSON)

```
// Mapa: clave -> valor
Map<String, dynamic> persona = {
  'nombre': 'Ana',
  'edad': 25,
```

```
  'activo': true
};

// Acceder a valores
print(persona['nombre']); // "Ana"

// Añadir/modificar
persona['ciudad'] = 'Madrid';
```

Funciones

```
// Función que NO devuelve nada
void saludar(String nombre) {
  print("Hola, $nombre");
}

// Función que SÍ devuelve algo
int sumar(int a, int b) {
  return a + b;
}

// Función con parámetros opcionales
void mostrarInfo(String nombre, {int? edad}) {
  print("Nombre: $nombre");
  if (edad != null) {
    print("Edad: $edad");
  }
}

// Uso:
mostrarInfo("Ana"); // Solo nombre
mostrarInfo("Luis", edad: 30); // Nombre y edad
```

---

3. 🧩 Widgets: El Corazón de Flutter

¿Qué es un Widget?

TODO en Flutter es un widget. Un botón es un widget, un texto es un widget, incluso el e spacio vacío entre dos elementos es un widget.

Analogía: Piensa en widgets como piezas de LEGO. Cada pieza tiene una función, y las c ombinas para construir algo más grande.

Tipos de Widgets

1. Widgets Visuales

Los que ves en pantalla:

- `Text`: Muestra texto
- `Image`: Muestra imágenes
- `Icon`: Muestra iconos
- `ElevatedButton`: Un botón con fondo

2. Widgets de Layout

Los que organizan otros widgets:

- `Column`: Pone widgets uno debajo de otro (vertical)
- `Row`: Pone widgets uno al lado del otro (horizontal)
- `Container`: Una caja que puede tener color, bordes, padding, etc.
- `Scaffold`: La estructura base de una pantalla (con AppBar, body, etc.)

3. Widgets de Interacción

Los que responden a acciones del usuario:

- `GestureDetector`: Detecta toques, deslizamientos, etc.
- `TextField`: Campo de texto donde el usuario escribe
- `Checkbox`: Casilla de verificación

La Regla de Oro: `child` vs `children`

Esto es CRÍTICO para no confundirte:

```
// child (singular): El widget solo puede tener UN hijo
Container(
  child: Text("Solo puedo tener un widget aquí")
)

// children (plural): El widget puede tener VARIOS hijos (una lista)
Column(
  children: [
    Text("Primer widget"),
    Text("Segundo widget"),
    Text("Tercer widget"),
  ]
)
```

Widgets con `child`: Container, Center, Padding, Scaffold (body), ElevatedButton
Widgets con `children`: Column, Row, ListView, Stack

---

4. 🔄 StatelessWidget vs StatefulWidget

StatelessWidget (Sin Estado)

¿Qué es? Un widget que **NO cambia** después de crearse. Es estático.

¿Cuándo usarlo? Cuando la información que muestra NO va a cambiar (ej: un texto fijo, u n logo).

```
class MiPantalla extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
```

```
    return Scaffold(
      appBar: AppBar(title: Text("Pantalla Estática")),
      body: Center(
        child: Text("Este texto nunca cambiará"),
      ),
    );
  }
}
```

StatefulWidget (Con Estado)

¿Qué es? Un widget que **SÍ puede cambiar** después de crearse. Tiene “estado” (informaci ón que puede variar).

¿Cuándo usarlo? Cuando algo en la pantalla va a cambiar (ej: un contador, un formulario, una lista que se actualiza).

```
class Contador extends StatefulWidget {
  @override
  _ContadorState createState() => _ContadorState();
}

class _ContadorState extends State<Contador> {
  int numero = 0; // Esta es la variable de ESTADO

  void incrementar() {
    setState(() { // setState le dice a Flutter: "Redibuja la pantalla"
      numero++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
```

```
      appBar: AppBar(title: Text("Contador")),
      body: Center(
        child: Text("Número: $numero", style: TextStyle(fontSize: 30)),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: incrementar,
        child: Icon(Icons.add),
      ),
    );
  }
}
```

¿Por qué dos clases?

- `Contador` (StatefulWidget): Es la "cáscara" que no cambia
- `_ContadorState`: Es el "cerebro" que guarda el estado y puede cambiar

La función mágica: `setState()`

```
setState(() {
  // Aquí cambias las variables
  numero++;
});
```

¿Qué hace? Le dice a Flutter: "He cambiado algo, redibuja la pantalla con los nuevos valor es".

⚠ **IMPORTANTE:** Si cambias una variable SIN usar `setState()`, la pantalla NO se actualizar á.

---

5. 📐 Layouts Fundamentales

Los layouts son widgets que organizan otros widgets en la pantalla.

Scaffold: La Estructura Base

¿Qué es? El “esqueleto” de una pantalla. Casi TODAS tus pantallas empezarán con un Sc affold.

¿Qué tiene?

- `appBar`: La barra superior (título, botones)
- `body`: El contenido principal
- `floatingActionButton`: Botón flotante (opcional)
- `drawer`: Menú lateral (opcional)
- `bottomNavigationBar`: Barra inferior de navegación (opcional)

```
Scaffold(
  appBar: AppBar(
    title: Text("Mi App"),
    backgroundColor: Colors.blue,
  ),
  body: Center(
    child: Text("Contenido principal"),
  ),
  floatingActionButton: FloatingActionButton(
    onPressed: () {},
    child: Icon(Icons.add),
  ),
)
```

Column: Vertical

¿Qué hace? Pone widgets uno **DEBAJO** de otro.

```
Column(
  mainAxisAlignment: MainAxisAlignment.center, // Centrar verticalmente
  crossAxisAlignment: CrossAxisAlignment.start, // Alinear a la izquierda
  children: [
```

```
    Text("Elemento 1"),
    Text("Elemento 2"),
    Text("Elemento 3"),
  ],
)
```

Propiedades clave:

- `mainAxisAlignment`: Cómo se distribuyen en el eje principal (vertical en Column)
  - `MainAxisAlignment.start`: Arriba
  - `MainAxisAlignment.center`: Centro
  - `MainAxisAlignment.end`: Abajo
  - `MainAxisAlignment.spaceBetween`: Separados al máximo
- `crossAxisAlignment`: Cómo se alinean en el eje cruzado (horizontal en Column)
  - `CrossAxisAlignment.start`: Izquierda
  - `CrossAxisAlignment.center`: Centro
  - `CrossAxisAlignment.end`: Derecha

Row: Horizontal

¿Qué hace? Pone widgets uno al **LADO** del otro.

```
Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  children: [
    Icon(Icons.star),
    Text("5 estrellas"),
    Icon(Icons.favorite),
  ],
)
```

⚠ **Problema común:** Si pones mucho contenido en un Row, se saldrá de la pantalla (verás rayas amarillas y negras). **Solución:** Usa `Expanded`.

Expanded: El Salvador

¿Qué hace? Hace que un widget ocupe **todo el espacio disponible** dentro de un Row o Column.

```
Row(
  children: [
    Icon(Icons.person),
    Expanded( // Este Text ocupará todo el espacio sobrante
      child: Text("Este es un texto muy largo que normalmente se saldría..."),
    ),
    Icon(Icons.arrow_forward),
  ],
)
```

Container: La Caja Multiusos

¿Qué hace? Es como una caja que puedes decorar: darle color, bordes, sombras, tamaño, padding, margin…

```
Container(
  width: 200,
  height: 100,
  padding: EdgeInsets.all(16), // Espacio INTERNO
  margin: EdgeInsets.all(10), // Espacio EXTERNO
  decoration: BoxDecoration(
    color: Colors.blue[100],
    borderRadius: BorderRadius.circular(12), // Bordes redondeados
    border: Border.all(color: Colors.blue, width: 2),
    boxShadow: [
      BoxShadow(
        color: Colors.grey,
        blurRadius: 5,
        offset: Offset(2, 2),
```

```
      )
    ],
  ),
  child: Text("Caja decorada"),
)
```

Padding vs Margin:

- `padding`: Espacio DENTRO del Container (entre el borde y el contenido)
- `margin`: Espacio FUERA del Container (entre el Container y otros widgets)

SizedBox: El Espaciador Invisible

¿Qué hace? Crea espacio vacío. Muy útil para separar widgets.

```
Column(
  children: [
    Text("Texto 1"),
    SizedBox(height: 20), // 20 píxeles de espacio vertical
    Text("Texto 2"),
  ],
)
```

---

6. 🎨 Widgets Esenciales de UI

Text: Mostrar Texto

```
Text(
  "Hola Mundo",
  style: TextStyle(
    fontSize: 24,
    fontWeight: FontWeight.bold, // Negrita
    color: Colors.red,
```

```
    fontStyle: FontStyle.italic, // Cursiva
    decoration: TextDecoration.underline, // Subrayado
  ),
  textAlign: TextAlign.center, // Centrar el texto
  maxLines: 2, // Máximo 2 líneas
  overflow: TextOverflow.ellipsis, // Si no cabe, poner "..."
)
```

Image: Mostrar Imágenes

Desde Internet

```
Image.network(
  'https://ejemplo.com/foto.jpg',
  width: 200,
  height: 200,
  fit: BoxFit.cover, // Cómo ajustar la imagen
  loadingBuilder: (context, child, loadingProgress) {
    if (loadingProgress == null) return child;
    return CircularProgressIndicator(); // Mostrar spinner mientras carga
  },
)
```

Valores de `fit`:

- `BoxFit.cover`: Llena todo el espacio, recortando si es necesario
- `BoxFit.contain`: Muestra la imagen completa, dejando espacios si es necesario
- `BoxFit.fill`: Estira la imagen para llenar el espacio (puede deformar)

Desde Assets (archivos locales)

1. Crea una carpeta `assets/` en la raíz del proyecto
2. Pon tus imágenes ahí
3. En `pubspec.yaml`, añade:

```
flutter:
  assets:
    - assets/
```

4. Usa:

```
Image.asset('assets/logo.png')
```

Icon: Iconos Predefinidos

Flutter trae miles de iconos incluidos.

```
Icon(
  Icons.favorite, // Busca "Icons." para ver todos
  color: Colors.red,
  size: 40,
)
```

Iconos comunes:

- `Icons.home`, `Icons.settings`, `Icons.person`
- `Icons.add`, `Icons.delete`, `Icons.edit`
- `Icons.favorite`, `Icons.star`, `Icons.check`

Botones

ElevatedButton (Botón con Fondo)

```
ElevatedButton(
  onPressed: () {
    print("Botón pulsado");
  },
  child: Text("Púlsame"),
```

```
  style: ElevatedButton.styleFrom(
    backgroundColor: Colors.green, // Color de fondo
    foregroundColor: Colors.white, // Color del texto
    padding: EdgeInsets.symmetric(horizontal: 30, vertical: 15),
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(20),
    ),
  ),
)
```

TextButton (Botón Sin Fondo)

```
TextButton(
  onPressed: () {},
  child: Text("Cancelar"),
)
```

IconButton (Solo Icono)

```
IconButton(
  onPressed: () {},
  icon: Icon(Icons.delete),
  color: Colors.red,
)
```

¿Qué es `onPressed`?
Es una **función** que se ejecuta cuando pulsas el botón. Si pones `onPressed: null`, el botón se desactiva (aparece gris).

TextField: Campo de Texto

```
TextField(
  decoration: InputDecoration(
```

```
    labelText: "Nombre", // Etiqueta flotante
    hintText: "Escribe tu nombre", // Texto de ayuda
    prefixIcon: Icon(Icons.person), // Icono a la izquierda
    border: OutlineInputBorder(), // Borde
  ),
  onChanged: (texto) {
    print("Escribiste: $texto");
  },
)
```

Para leer el texto después:

```
final TextEditingController _controlador = TextEditingController();

TextField(
  controller: _controlador,
)

// Más tarde, para leer el texto:
String texto = _controlador.text;
```

Continuaré en la siguiente parte con Navegación, Formularios, Provider y los Ejercicios Re sueltos detallados…