EXPERIMENT NO 4

Aim: To design a layout of flutter app using flutter widgets

Theory:

Flutter is an open-source UI framework developed by Google used to build fast, natively compiled applications for mobile, web, and desktop from a single codebase. It uses the Dart programming language and follows a widget-based architecture, where everything in the UI is a widget, making layouts highly customizable and flexible.

Flutter renders its own UI using the Skia graphics engine, which ensures high performance and a consistent look across platforms. Features like hot reload allow developers to see changes instantly, improving productivity, while rich built-in widgets and strong community support make Flutter a popular choice for modern app development.

Key Widgets Used in Flutter Forms

1. Form Widget

- Acts as a container for form fields
- Groups multiple input widgets together
- Helps in managing validation and submission

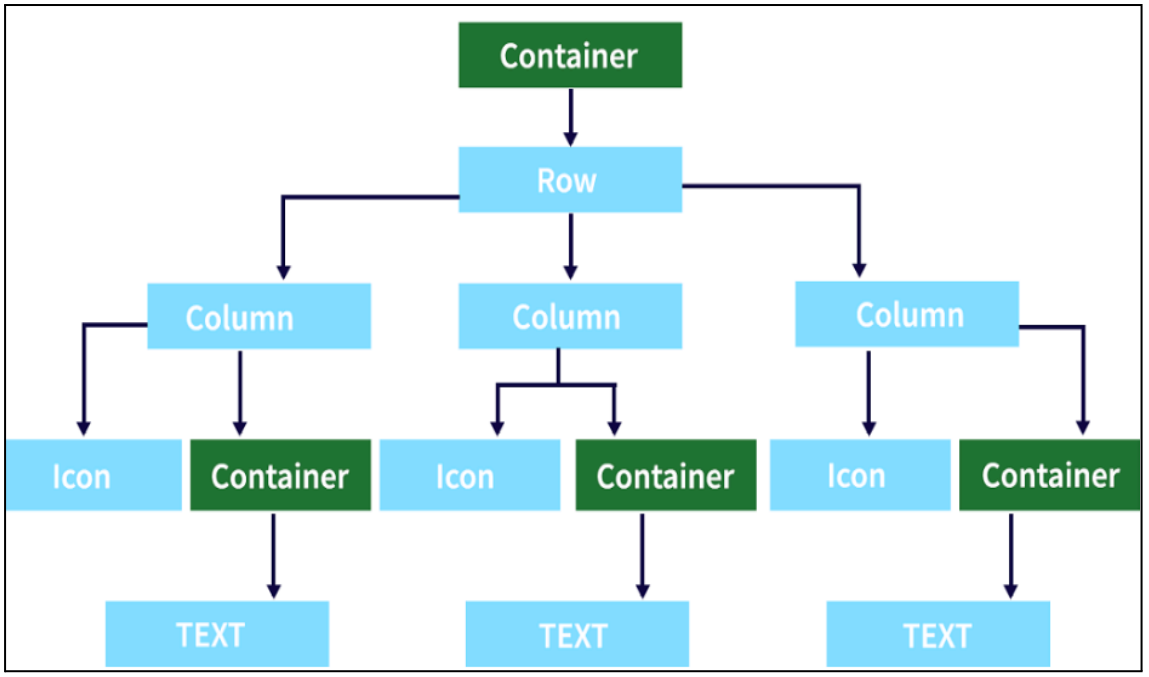
Flutter - Working with Layouts

Before talking about Layout in Flutter, there is just one thing to keep in mind that “Everything in Flutter is Widget". Meaning the core of the layout in any Flutter Application is the widget. Putting it simply, all the images, icons, labels and text, etc are technically widgets of different types and layouts. In this article, we will explore the concept of layouts in flutter in detail.

For a better understanding of the concept let's take a single example and break down those components for better understanding.

In the above image, you just saw a layout that is nothing but just a composition of few basic widgets.

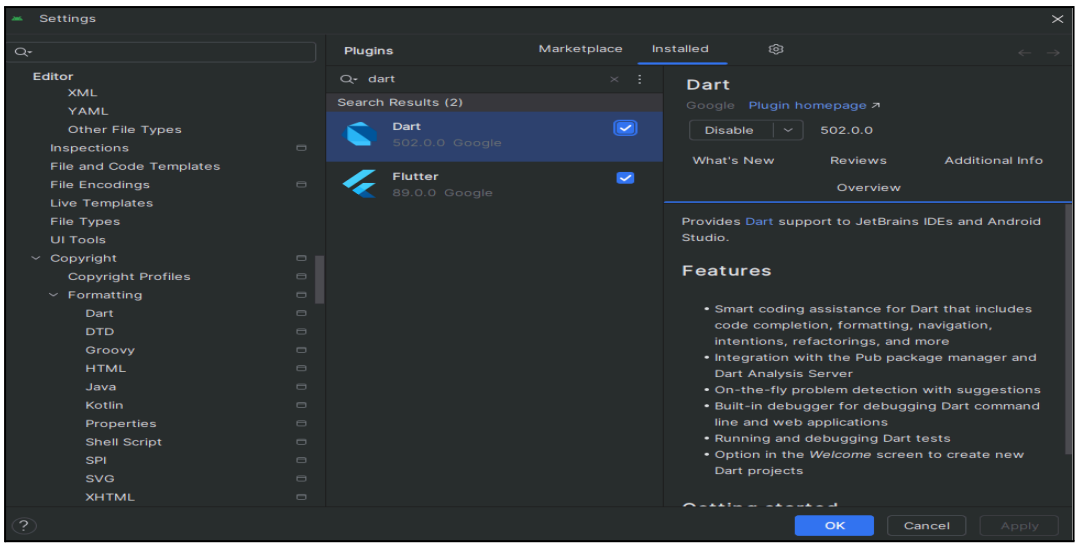
Now look at the above image here we just outline the layouts, look closely you can see inside a raw widget there are 3 column widgets and each column contains an icon and a label. Take a look at the below widget tree diagram.



Procedures:

STEP 1: Install Flutter & Dart Plugins in Android Studio

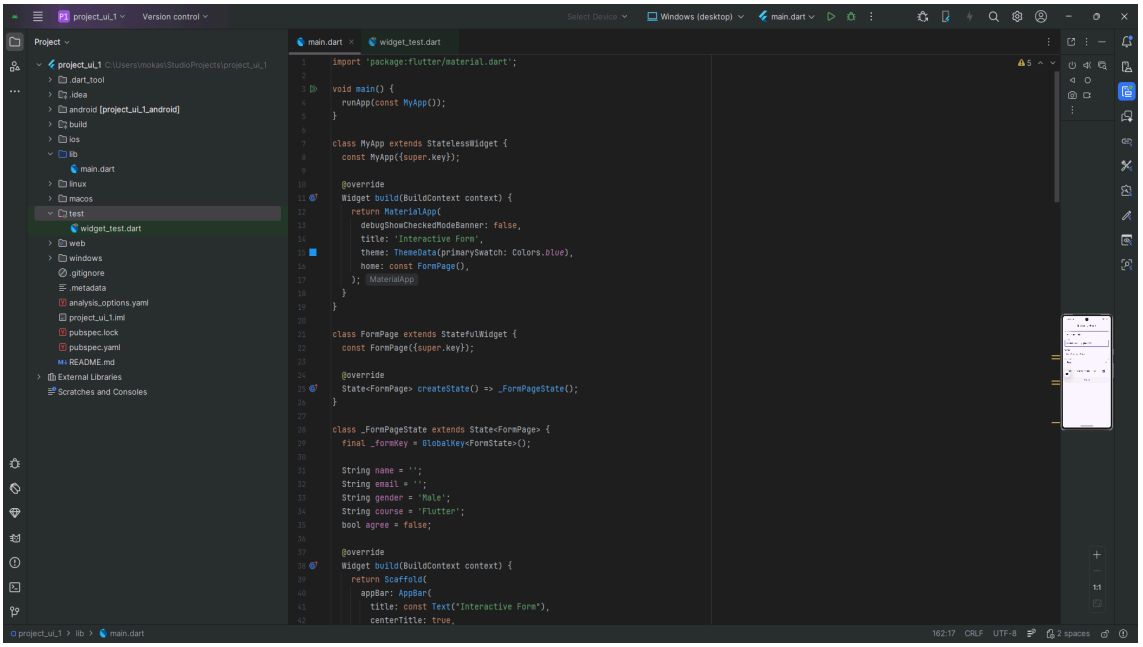
1. Open Android Studio
2. Go to Settings → Plugins
3. Search and Install- Flutter & Dart
4. Restart android studio



STEP 2: Create a New Flutter Project

1. Open Android Studio
2. Click New Flutter Project
3. Select Flutter Application
4. Choose Flutter SDK path (example):

   C:\flutter
5. Enter:
   - Project name: project_ui_1
   - Language: Dart
6. Click Finish



STEP 3: Code

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(primarySwatch: Colors.indigo),
      home: const DashboardScreen(),
    );
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Dashboard"),
        centerTitle: true,
```

```
),

// Drawer
drawer: Drawer(
  child: ListView(
    children: const [
      DrawerHeader(
        decoration: BoxDecoration(color: Colors.indigo),
        child: Text(
          "Flutter App",
          style: TextStyle(color: Colors.white, fontSize: 22),
        ),
      ),
      ListTile(leading: Icon(Icons.home), title: Text("Home")),
      ListTile(leading: Icon(Icons.person), title: Text("Profile")),
      ListTile(leading: Icon(Icons.settings), title: Text("Settings")),
    ],
  ),
),

body: SingleChildScrollView(
  child: Column(
    children: [

      // Stack section (Image + Text overlay)
      Stack(
        children: [
          Container(
            height: 180,
```

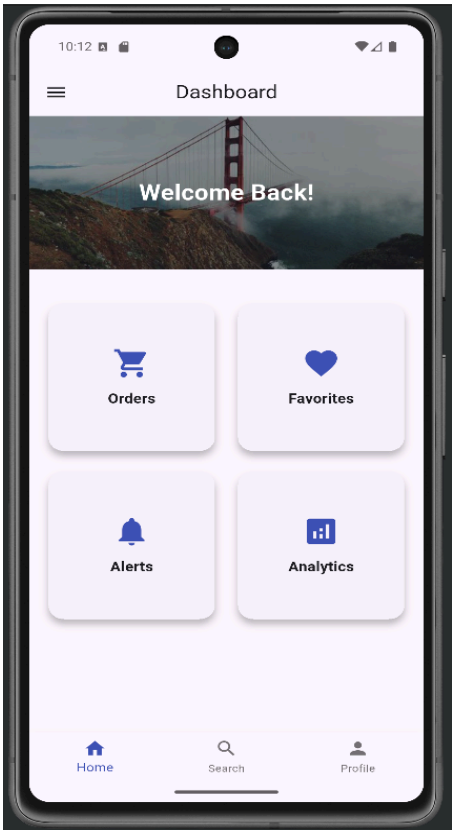
```
      decoration: const BoxDecoration(
        image: DecorationImage(
          image: NetworkImage(
            "https://picsum.photos/600/300",
          ),
          fit: BoxFit.cover,
        ),
      ),
    ),
    Container(
      height: 180,
      color: Colors.black.withOpacity(0.4),
      alignment: Alignment.center,
      child: const Text(
        "Welcome Back!",
        style: TextStyle(
          color: Colors.white,
          fontSize: 26,
          fontWeight: FontWeight.bold,
        ),
      ),
    ),
  ],
),

const SizedBox(height: 20),

// Grid Dashboard
Padding(
```

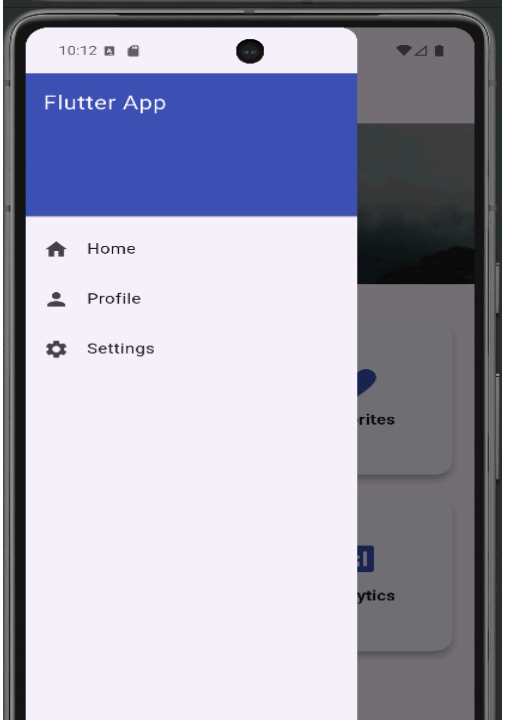
```
          padding: const EdgeInsets.all(16.0),
          child: GridView.count(
            crossAxisCount: 2,
            shrinkWrap: true,
            physics: const NeverScrollableScrollPhysics(),
            crossAxisSpacing: 16,
            mainAxisSpacing: 16,
            children: const [
              DashboardCard(icon: Icons.shopping_cart, title: "Orders"),
              DashboardCard(icon: Icons.favorite, title: "Favorites"),
              DashboardCard(icon: Icons.notifications, title: "Alerts"),
              DashboardCard(icon: Icons.analytics, title: "Analytics"),
            ],
          ),
        ),
      ],
    ),
  ),

  // Bottom Navigation
  bottomNavigationBar: BottomNavigationBar(
    selectedItemColor: Colors.indigo,
    items: const [
      BottomNavigationBarItem(icon: Icon(Icons.home), label: "Home"),
      BottomNavigationBarItem(icon: Icon(Icons.search), label: "Search"),
      BottomNavigationBarItem(icon: Icon(Icons.person), label: "Profile"),
    ],
  ),
);
```

```
  }
}

// Reusable Card Widget
class DashboardCard extends StatelessWidget {
  final IconData icon;
  final String title;

  const DashboardCard({
    super.key,
    required this.icon,
    required this.title,
  });

  @override
  Widget build(BuildContext context) {
    return Card(
      elevation: 6,
      shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Icon(icon, size: 40, color: Colors.indigo),
          const SizedBox(height: 10),
          Text(
            title,
            style: const TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
          ),
        ],
```

```
      ),
    );
  }
}
```

STEP 6: Output





Conclusion:

Designing a Flutter app layout using Flutter widgets provides a flexible and efficient way to create modern, responsive user interfaces. Since Flutter follows a widget-based architecture, every part of the UI—such as layout structure, styling, and alignment—is built by combining and nesting widgets like Row, Column, Container, Stack, and Scaffold. This approach makes UI development highly customizable, reusable, and easy to maintain. By using proper layout widgets, spacing, and responsive design techniques, developers can create visually appealing and consistent interfaces across different screen sizes and platforms. Overall, Flutter widgets simplify the process of UI design while delivering high performance and a smooth user experience.