**Name:Ajinkya Sunil Patil  
D15B/42  
Lab-mpl**

**Experiment 03: To Include Icons, Images, and Fonts in a Flutter App**

### **Objective:**

To understand the process of integrating icons, images, and custom fonts in a Flutter application, which are essential components in building a visually engaging and user-friendly interface.

### **Introduction:**

Flutter, a popular open-source UI framework, is known for its flexibility and ease of building cross-platform applications with a single codebase. It allows developers to create high-performance apps for mobile, web, and desktop. One of the key features of Flutter is its ability to include various UI elements such as icons, images, and custom fonts, which significantly enhance the overall user experience and improve the application's aesthetic appeal. By integrating these elements, developers can align the application's design with modern UI/UX standards, improving usability and visual coherence.

### **Theory:**

1. **Icons in Flutter:** Icons are crucial in communicating functionality and guiding users through the app interface. Flutter comes with a rich set of Material Design icons that are easy to implement. Additionally, it allows for the integration of custom icons from third-party libraries such as Font Awesome and Ionicons.  
   Icons can also be customized in terms of size, color, and positioning to match the overall theme and design of the app. The ability to use both built-in and custom icons offers great flexibility for designers to ensure consistency with branding and user preferences. Custom icons can be created as part of an icon font or as individual image files. Proper usage of icons can reduce the cognitive load on users, making navigation simpler and more intuitive.
2. **Images in Flutter:** Images are essential to create engaging, context-rich apps. Flutter allows the inclusion of images from multiple sources:
   * **Asset Images (Local Storage):** These are images bundled with the application itself. They are stored in the app’s project directory under the "assets" folder and referenced in the Flutter project configuration file (pubspec.yaml). Asset images are especially useful for app icons, logos, splash screens, and other elements that are static and always present in the app.
   * **Network Images (URL-based):** Flutter supports loading images directly from URLs, making it easy to integrate dynamic content, such as product images fetched from a web API. This is particularly useful for applications that display frequently updated content from remote sources, such as e-commerce apps, news apps, or social media platforms.
   * **Memory Images:** For more advanced use cases, images can be decoded and displayed directly from memory. This allows for the integration of images generated or processed at runtime, for example, images captured by the user’s camera or received through real-time communication channels.
3. Images can also be styled, resized, and transformed (e.g., rotated or scaled) within Flutter using the Image widget. This flexibility enables developers to optimize the layout and visual flow of their applications, ensuring that images appear sharp and well-positioned across various screen sizes.
4. **Fonts in Flutter:** Custom fonts are an important tool for creating visually distinctive applications. By using custom fonts, designers can align the app’s typography with the brand’s identity or simply enhance the readability and style of the text. Flutter provides support for custom fonts through two main methods:
   * **Google Fonts Integration:** Flutter offers a convenient way to access the extensive library of Google Fonts, making it easy to integrate modern, open-source fonts into the app. Google Fonts can be added via a simple package, offering a wide selection without the need to manually download or store font files.
   * **Manually Added Fonts:** Developers can also add their own fonts, such as .ttf or .otf files, to the app’s asset folder. These custom fonts must be declared in the pubspec.yaml file, ensuring that Flutter can access and use them in the application. Custom fonts provide a unique touch to an app’s design, allowing for a tailored user experience that reflects the app’s theme or purpose.
5. Additionally, Flutter allows for advanced typography features, such as setting specific fonts for different parts of the app (e.g., using a different font for headers and body text), adjusting font weight, size, letter spacing, and text alignment. This level of control ensures that developers can maintain a consistent visual identity across the application.

### **Steps:**

1. **Set up icons**:
   * Use Flutter’s built-in Material Icons or import third-party icon libraries to expand the icon set available in your app. Customize the appearance of icons to match the app’s theme.
2. **Add images**:
   * Store images in the assets folder and reference them in the app configuration (pubspec.yaml).
   * Use network images to display dynamic content fetched from the web.
   * Customize the display, layout, and resizing of images based on the app's design needs.
3. **Integrate custom fonts**:
   * Download and add font files (or use Google Fonts) and declare them in the pubspec.yaml file.
   * Apply the custom fonts to text widgets to improve the visual appeal and branding of the app.

### **Best Practices:**

* **Organize Assets Efficiently:** Place assets like icons, images, and fonts in well-structured directories and ensure they are properly referenced in pubspec.yaml. Maintaining an organized asset folder structure helps prevent errors and makes the project more maintainable.
* **Optimize Image Sizes:** Use appropriately sized images to prevent unnecessary memory usage and loading time. Images should be optimized for different device resolutions and screen densities to maintain high quality without slowing down the app.
* **Fallback Fonts:** When using custom fonts, provide a fallback font to avoid rendering issues in case the custom font fails to load or is unsupported on certain devices.
* **Testing on Multiple Devices:** It’s important to test the app on various screen sizes and platforms to ensure that icons, images, and fonts are rendered correctly and maintain their quality across devices.

### **Conclusion:**

Integrating icons, images, and fonts into a Flutter application is essential for creating visually engaging and user-friendly interfaces. Proper integration allows for consistent branding, enhanced UI aesthetics, and improved user experience. However, careful attention must be given to asset management, file paths, and project configuration. Misconfigurations in the pubspec.yaml file or incorrect asset paths can lead to common issues such as "asset not found" or "font loading errors." Through proper testing and management, these issues can be mitigated, resulting in a smooth, visually appealing application with rich multimedia content.

**#EXECUTION**

**Code:-**import 'package:flutter/material.dart';

import 'package:font\_awesome\_flutter/font\_awesome\_flutter.dart'; // ✅ Import FontAwesome Icons

import 'dart:math'; // ✅ Importing for EMI calculations

class LoanEmiCalculator extends StatefulWidget {

@override

\_LoanEmiCalculatorState createState() => \_LoanEmiCalculatorState();

}

class \_LoanEmiCalculatorState extends State<LoanEmiCalculator> {

final TextEditingController loanAmountController = TextEditingController();

final TextEditingController interestRateController = TextEditingController();

final TextEditingController tenureController = TextEditingController();

double emiResult = 0.0;

void calculateEMI() {

double P = double.tryParse(loanAmountController.text) ?? 0.0;

double annualRate = double.tryParse(interestRateController.text) ?? 0.0;

double tenureMonths = double.tryParse(tenureController.text) ?? 0.0;

if (P == 0 || annualRate == 0 || tenureMonths == 0) {

setState(() {

emiResult = 0.0;

});

return;

}

double r = (annualRate / 12) / 100; // Monthly interest rate

double n = tenureMonths; // Loan tenure in months

double emi = (P \* r \* pow((1 + r), n)) / (pow((1 + r), n) - 1);

setState(() {

emiResult = emi;

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text("Loan EMI Calculator"),

backgroundColor: Colors.grey[900],

),

backgroundColor: Colors.black,

body: SingleChildScrollView(

child: Padding(

padding: const EdgeInsets.all(20.0),

child: Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

// ✅ EMI Banner Image

Center(

child: Image.asset(

"assets/images/EMI-Calculator.png", // Make sure this image exists

height: 150,

fit: BoxFit.cover,

),

),

SizedBox(height: 20),

\_buildTextField(loanAmountController, "Loan Amount (₹)", FontAwesomeIcons.rupeeSign),

\_buildTextField(interestRateController, "Annual Interest Rate (%)", FontAwesomeIcons.percent),

\_buildTextField(tenureController, "Loan Tenure (Months)", FontAwesomeIcons.calendar),

SizedBox(height: 20),

Center(

child: ElevatedButton.icon(

onPressed: calculateEMI,

style: ElevatedButton.styleFrom(

backgroundColor: Colors.orange,

padding: EdgeInsets.symmetric(horizontal: 40, vertical: 12),

shape: RoundedRectangleBorder(

borderRadius: BorderRadius.circular(8),

),

),

icon: Icon(FontAwesomeIcons.chartLine, color: Colors.white), // ✅ EMI Calculation Icon

label: Text("Calculate EMI", style: TextStyle(fontSize: 18)),

),

),

SizedBox(height: 20),

Center(

child: Text(

"EMI: ₹${emiResult.toStringAsFixed(2)}",

style: TextStyle(fontSize: 24, color: Colors.white, fontWeight: FontWeight.bold),

),

),

],

),

),

),

);

}

Widget \_buildTextField(TextEditingController controller, String label, IconData icon) {

return Padding(

padding: const EdgeInsets.symmetric(vertical: 8.0),

child: TextField(

controller: controller,

keyboardType: TextInputType.number, // Ensures numeric input

style: TextStyle(color: Colors.white), // Text color white to match dark theme

decoration: InputDecoration(

labelText: label,

labelStyle: TextStyle(color: Colors.grey[400]), // Label text color

prefixIcon: Icon(icon, color: Colors.orange), // Icon color and icon from FontAwesome

filled: true,

fillColor: Colors.grey[850], // Background color of the input field

border: OutlineInputBorder(

borderRadius: BorderRadius.circular(8),

borderSide: BorderSide.none, // No border around the input

),

),

),

);

}

}

**Output:**  
