Aim: To design Flutter UI by including common widgets.

Theory:

Flutter is a modern UI toolkit developed by Google that allows developers to build natively compiled applications for mobile, web, and desktop platforms from a single codebase. It provides a rich set of pre-built widgets and tools that make UI development fast, flexible, and expressive. One of the most fundamental principles in Flutter is that **everything is a widget**—from layout components to user interface elements and even invisible helpers like padding.

Widgets in Flutter are used to build the entire UI by composing smaller widgets into complex hierarchies. These widgets fall into different categories based on their role in the user interface.

Types of Flutter Widgets

Widgets in Flutter can be broadly categorized into five main types:

1. Structural Widgets

Structural widgets are responsible for defining the **basic structure** of a Flutter application. These are often the first widgets used when building a screen and include top-level components that provide visual scaffolding and layout organization.

Examples:

- **MaterialApp**: Wraps the entire app and applies Material Design throughout.
- **Scaffold**: Provides a framework for implementing standard app elements like the AppBar, body, floating action buttons, drawers, etc.
- **AppBar**: Represents a material design app bar, often used as the top navigation bar of the screen.
- **Container**: A versatile widget used for styling and positioning its child widgets with padding, margins, borders, background color, etc.

These widgets form the **foundation** upon which the UI is built.

2. Layout Widgets

Layout widgets are used to arrange and position other widgets within the user interface. They control how widgets are nested, spaced, aligned, and layered.

Examples:

- **Column**: Arranges child widgets vertically.
- **Row**: Arranges child widgets horizontally.
- **Stack**: Overlays widgets on top of each other (like layers), allowing complex UI designs.
- **Padding**: Adds space around a widget.
- **SizedBox**: Adds fixed space between widgets or defines a fixed width or height for a widget.

Layout widgets allow developers to **organize UI elements** precisely as needed, offering control over the visual composition.

3. Display Widgets

Display widgets are used to present static content such as text, images, or icons on the screen. These widgets are essential for conveying information to the user.

Examples:

- **Text**: Displays a string of text with customizable styles.
- **Image**: Displays an image from an asset, file, or network.
- **Icon**: Renders predefined icons from material or custom icon sets.

These widgets are typically **non-interactive** and focus purely on the visual representation of data.

4. Scrollable Widgets

Scrollable widgets are essential when the content exceeds the screen size and needs to be viewed through scrolling. They make the UI responsive and user-friendly on devices with limited screen space.

Examples:

- **ListView**: Displays a scrollable list of widgets. It can be built dynamically using builders for large datasets.
- **SingleChildScrollView**: Makes a single widget scrollable when content overflows vertically or horizontally.
- **GridView**: Displays content in a 2D scrollable grid layout.

These widgets are crucial for building **scrollable views** and **dynamic lists** in real-world applications.

5. Interactive Widgets

Interactive widgets are responsible for handling user input such as taps, gestures, and other actions. They enhance user engagement by providing interactive elements like buttons and feedback mechanisms.

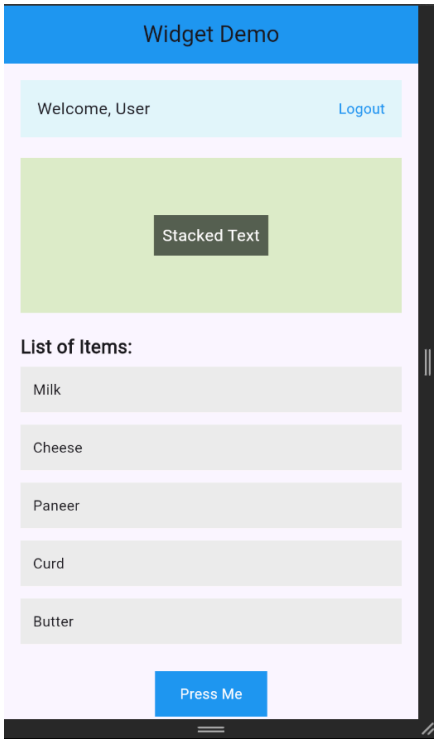
Examples:

- **GestureDetector**: Detects gestures like taps, double taps, long presses, etc., and triggers actions accordingly.
- **ElevatedButton**: A material design button with elevation and built-in onPress functionality.
- **TextButton**: A simple button without elevation or background.
- **SnackBar**: Provides brief messages at the bottom of the screen as a form of feedback or notification.

These widgets enable developers to **capture user interactions** and respond with appropriate UI behavior or actions.

Code (Github): https://github.com/Sairam-Vk-sudo/mplExp27/tree/main/exp2

Output:



Conclusion: Understanding the different types of widgets in Flutter is essential for designing effective and maintainable user interfaces. Each widget type serves a specific role—whether it's structuring the layout, displaying content, enabling interactivity, or handling overflow through scrolling. By combining these widgets effectively, developers can create flexible and scalable UIs tailored to a wide variety of applications.
In the provided experiment, all these widget types are demonstrated within a simple Flutter application, showcasing their usage in a real-world scenario. This hands-on practice reinforces the concept of widget-based UI design and prepares developers to build more complex applications using Flutter's powerful toolkit.