Aim: To design Flutter UI by including common widgets.

Theory:

Flutter is a UI toolkit that empowers developers to create cross-platform applications with interactive and aesthetically pleasing designs. The foundation of Flutter's UI design lies in its widget-based architecture. Widgets are classified into two main types: **Stateful Widgets** and **Stateless Widgets**, which govern the behavior and appearance of UI components.

1. Stateless Widgets

Stateless widgets are immutable, meaning their properties cannot change during runtime. These widgets do not store any state and are ideal for static UI components such as text, images, or icons. Examples include Text, Icon, and Container. They serve the purpose of displaying information that does not rely on dynamic updates.

2. Stateful Widgets

Stateful widgets, on the other hand, are mutable and maintain dynamic states. Their properties can change based on user interaction or application logic during runtime. Stateful widgets include an internal mechanism to manage state changes using State objects. Examples are Checkbox, Slider, and TextField. These widgets are crucial for creating interactive elements like forms, animations, or real-time data updates.

Common Widgets

- **Structural Widgets:** Container, Row, and Column are used to define the layout structure of the UI.
- **Interactive Widgets:** Button, Switch, Slider, and TextField allow user interactions.
- **Decorative Widgets:** Image, Icon, and Text enhance the visual appeal.
- **Input Widgets:** Widgets like TextField enable data input from users.
- **Gestures:** Widgets such as GestureDetector handle user interactions like taps, swipes, or pinches.

a. Hot Reload Feature

Flutter's *Hot Reload* enables developers to view changes in the UI instantly without restarting the application, significantly improving development efficiency.

b. Material Design and Cupertino Styles

Flutter supports both Google's Material Design and Apple's Cupertino styles, ensuring a consistent experience across platforms while retaining native appearance.

c. State Management

State management plays a vital role in handling dynamic updates. Popular approaches include:

- **Provider** and **Riverpod**: Simplify state management and dependency injection.
- **Bloc (Business Logic Component)**: Structures logic and states effectively for complex applications.

Code:

1. TextField - Searching the location
In my Crime Alert app, there's a feature that allows users to search for crime reports on the map using a search box to enter a specific location.

```
TextField(
          controller: searchController,
          onChanged: fetchLocationSuggestions,
          decoration: InputDecoration(
            labelText: 'Search Location',
            prefixIcon: Icon(Icons.search, color: Colors.red),
            border: OutlineInputBorder(),
            suffixIcon: _isSearching
                ? Padding(
                    padding: EdgeInsets.all(8.0),
                    child: CircularProgressIndicator(strokeWidth: 2),
                  )
                : null,
          ),
        ),
```

2. **Padding** : Adds space around content to prevent elements from touching screen edges.

```
Padding(
    padding: EdgeInsets.all(8.0),
child: CircularProgressIndicator(strokeWidth: 2),
     )
```

3. **Row** : Displays a bullet ("•") and the text side-by-side horizontally
4. **ElevatedButton**: A button with an icon and text (icon: calendar, label: date or "Select").

```
Row(
 children: [
          Expanded(
            child: ElevatedButton.icon(
              icon: Icon(Icons.calendar_today),
              label: Text(
```

```
          selectedDate == null
              ? 'Select Date'
              : '${selectedDate!.day}/${selectedDate!.month}/${selectedDate!.year}',
          overflow: TextOverflow.ellipsis,
        ),
        onPressed: () async {
          final DateTime? picked = await showDatePicker(
            context: context,
            initialDate: DateTime.now(),
            firstDate: DateTime(2000),
            lastDate: DateTime(2100),
          );
          if (picked != null) {
            setState(() => selectedDate = picked);
          }
        },
        style: ElevatedButton.styleFrom(
          backgroundColor: Colors.red,
          padding: EdgeInsets.symmetric(vertical: 12),
        ),
      ),
    ),
```

Output :