UNIT – 3

The Basics of Flutter

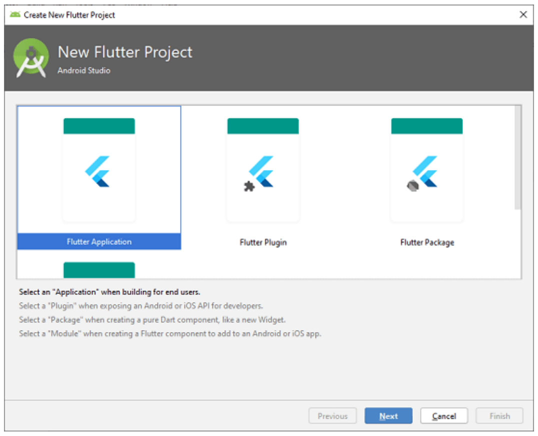
❖ Flutter First Application

➢ In this section, we are going to learn how to create a simple application in Android Studio to understand the basics of the Flutter application. To create Flutter application, do the following steps:

- Step 1: Open the Android Studio.

- Step 2: Create the Flutter project. To create a project, go to File-> New->New Flutter Project. The following screen helps to understand it more clearly.

- Step 3: In the next wizard, you need to choose the Flutter Application. For this, select Flutter Application-> click Next, as shown in the below screen.



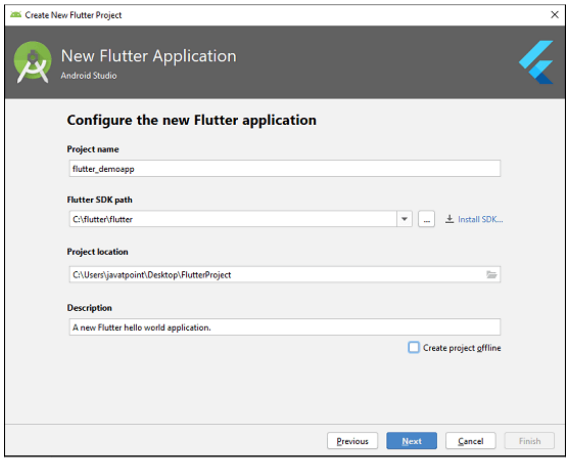
Select an "Application" when building for end users.
Select a "Plugin" when exposing an Android or iOS API for developers.
Select a "Package" when creating a pure Dart component, like a new Widget.
Select a "Module" when creating a Flutter component to add to an Android or iOS app.

- Step 4: Next, configure the application details as shown in the below screen and click on the Next button.

  Project Name: Write your Application Name.

  Flutter SDK Path: <path_to_flutter_sdk>

  Project Location: <path_to_project_folder>
  Descriptions: <A new Flutter hello world application>



Configure the new Flutter application

Project name: flutter_demoapp

Flutter SDK path: C:\flutter\flutter

Install SDK...

Project location: C:\Users\javatpoint\Desktop\FlutterProject

Description: A new Flutter hello world application.

☐ Create project offline

- Step 5: In the next wizard, you need to set the company domain name and click the Finish button.

➢ After clicking the Finish button, it will take some time to create a project. When the project is created, you will get a fully working Flutter application with minimal functionality.

- Step 6: Now, let us check the structure of the Flutter project application and its purpose. In the below image, you can see the various folders and components of the Flutter application structure, which are going to discuss here.

- Step 7: Open the main.dart file and replace the code with the following code snippets.

```
        class MyHomePage extends StatelessWidget {
       MyHomePage({Key key, this.title}) : super(key: key);
      // This widget is the home page of your application.
       final String title;

           @override
           Widget build(BuildContext context) {
            return Scaffold(
           appBar: AppBar(
           title: Text(this.title),
            ),
           body: Center(
           child: Text('Hello World'),
  ),
 );
 }
}
```

❖ <u>**Widgets**</u>

➢ Each element on the screen of the Flutter app is a widget. The view of the screen completely depends upon the choice and sequence of the widgets used to build the apps. And the structure of the code of apps is a tree of widgets.

➢ Types of Widgets
There are broadly two types of widgets in the flutter:

- Stateless Widget
- Stateful Widget

➢ **Stateless Widget**

- Stateless Widget is a type of widget which once built , then it’s properties and state can’t be changed. These widgets are immutable, once created can’t be modified.

- Don’t Store Realtime Data

- Its Static, Immutable, No State and Lightweight

- Examples: Display Text, Icons, Images, etc.

➢ **Stateful Widget**

- Stateful Widgets is a type of widget that can change state. It can maintain and update the appearance in the response to change in state.

- Store Realtime Data

- Its Dynamic, Mutable State, State Lifecycle

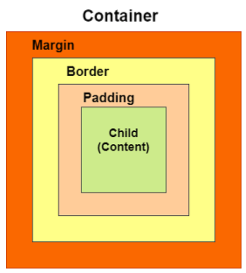
- Examples: Buttons, Sliders, Text Fields,

❖ <u>**Container**</u>

➢ The container in Flutter is a parent widget that can contain multiple child widgets and manage them efficiently through width, height, padding, background color, etc. It is a widget that combines common painting, positioning, and sizing of the child widgets. It is also a class to store one or more widgets and position them on the screen according to our needs. Generally, it is similar to a box for storing contents. It allows many attributes to the user for decorating its child widgets, such as using margin, which separates the container with other contents.

- **<u>Why we need a container widget in Flutter?</u>**

➢ If we have a widget that needs some background styling may be a color, shape, or size constraints, we may try to wrap it in a container widget. This widget helps us to compose, decorate, and position its child widgets. If we wrap our widgets in a container, then without using any parameters, we would not notice any difference in its appearance. But if we add any properties such as color, margin, padding, etc. in a container, we can style our widgets on the screen according to our needs.



❖ <u>Properties of Container widget</u>

- **child:** This property is used to store the child widget of the container. Suppose we have taken a Text widget as its child widget that can be shown in
- **color:** This property is used to set the background color of the text. It also changes the background color of the entire container.
- **height and width:** This property is used to set the container's height and width according to our needs. By default, the container always takes the space based on its child widget
- **margin:** This property is used to surround the empty space around the container. We can observe this by seeing white space around the container. Suppose we have used the EdgeInsets.all(25) that set the equal margin in all four directions, as shown in
- **padding:** This property is used to set the distance between the border of the container (all four directions) and its child widget. We can observe this by seeing the space between the container and the child widget. Here, we have used an EdgeInsets.all(35) that set the space between text and all four container directions
- **alignment:** This property is used to set the position of the child within the container. Flutter allows the user to align its element in various ways such as center, bottom, bottom center, topLeft, centerRight, left, right, and many more
- **decoration:** This property allows the developer to add decoration on the widget. It decorates or paint the widget behind the child. If we want to decorate or paint in front of a child, we need to use the forgroundDecoration parameter. The below image explains the difference between them where the foregroundDecoration covers the child and decoration paint behind the child.

❖ <u>**Column,Row**</u>

➢ we have learned to create a simple Flutter application and its basic styling to the widgets. Now, we are going to learn how to arrange the widgets in rows and columns on the screen. The rows and columns are not a single widget; they are two different widgets, namely Row and Column. Here, we will integrate these two widgets together because they have similar properties that help us understand them efficiently and quickly.

➢ **Row Widget**

- This widget arranges its children in a horizontal direction on the screen. In other words, it will expect child widgets in a horizontal array. If the child widgets need to fill the available horizontal space, we must wrap the children widgets in an Expanded widget.

➢ **We can align the row's children widget with the help of the following properties**:

start: It will place the children from the starting of the main axis.

end: It will place the children at the end of the main axis.

center: It will place the children in the middle of the main axis.

spaceBetween: It will place the free space between the children evenly.

spaceAround: It will place the free space between the children evenly and half of that space before and after the first and last children widget.

spaceEvenly: It will place the free space between the children evenly and before and after the first and last children widget

➢ <u>**Column**</u>

- This widget arranges its children in a vertical direction on the screen. In other words, it will expect a vertical array of children widgets. If the child widgets need to fill the available vertical space, we must wrap the children widgets in an Expanded widget.

- A column widget does not appear scrollable because it displays the widgets within the visible view. So it is considered wrong if we have more children in a column which will not fit in the available space. If we want to make a scrollable list of column widgets, we need to use the ListView Widget.

❖ Image

➢ In this section, we are going to see how we can display images in Flutter. When you create an app in Flutter, it includes both code and assets (resources). An asset is a file, which is bundled and deployed with the app and is accessible at runtime. The asset can include static data, configuration files, icons, and images. The Flutter supports many image formats, such as JPEG, WebP, PNG, GIF, animated WebP/GIF, BMP, and WBMP.

➢ Displaying images is the fundamental concept of most of the mobile apps. Flutter has an Image widget that allows displaying different types of images in the mobile application.

- **How to display the image in Flutter To display an image in Flutter, do the following steps:**

  Step 1: First, we need to create a new folder inside the root of the Flutter project and named it assets. We can also give it any other name if you want.

  Step 2: Next, inside this folder, add one image manually.

  Step 3: Update the pubspec.yaml file. Suppose the image name is tablet.png, then pubspec.yaml file is:

```
assets:
    - assets/tablet.png
    - assets/background.png
```

  If the assets folder contains more than one image, we can include it by specifying the directory name with the slash (/) character at the end.

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

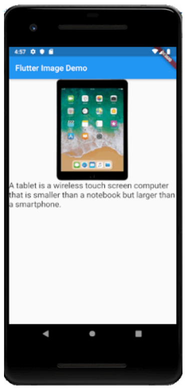
  Step 4: Finally, open themain.dart file and insert the following code.

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

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
            title: Text('Flutter Image Demo'),
        ),
                    body: Center(
                      child: Column(
                        children: <Widget>[
                          Image.asset('assets/tablet.png')
            Text(
'A tablet is a wireless touch screen computer that is smaller than a notebook but larger than a
smartphone.', style: TextStyle(fontSize: 20.0),
                          )
                        ],
                      ),
                    ),
                  ),
                );
              }
            }
```

Step 5: Now, run the app. You will get something like the screen below.



A tablet is a wireless touch screen computer that is smaller than a notebook but larger than a smartphone.

❖ <u>**Icon**</u>

- ➢ An icon is a graphic image representing an application or any specific entity containing meaning for the user. It can be selectable and non-selectable. For example, the company's logo is non-selectable. Sometimes it also contains a hyperlink to go to another page. It also acts as a sign in place of a detailed explanation of the actual entity.

- ➢ Flutter provides an Icon Widget to create icons in our applications. We can create icons in Flutter, either using inbuilt icons or with the custom icons. Flutter provides the list of all icons in the Icons class. In this article, we are going to learn how to use Flutter icons in the application**.**

- **Icon Widget Properties**

Property	Descriptions
icon	It is used to specify the icon name to display in the application. Generally, Flutter uses material design icons that are symbols for common actions and items.
color	It is used to specify the color of the icon.
size	It is used to specify the size of the icon in pixels. Usually, icons have equal height and width.
textDirection	It is used to specify to which direction the icon will be rendered.

❖ Buttons

➢ Buttons are the graphical control element that provides a user to trigger an event such as taking actions, making choices, searching things, and many more. They can be placed anywhere in our UI like dialogs, forms, cards, toolbars, etc.

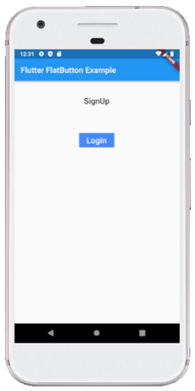
➢ Buttons are the Flutter widgets, which is a part of the material design library. Flutter provides several types of buttons that have different shapes, styles, and features.

❖ Types of Buttons

- Flat Button
- Raised Button
- Floating Button
- Drop Down Button
- Icon Button
- Inkwell Button
- PopupMenu Button
- Outline Button

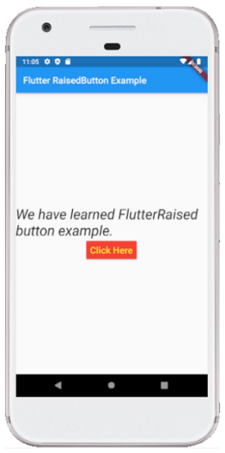
➢ Flat Button

- It is a text label button that does not have much decoration and displayed without any elevation. The flat button has two required properties that are: child and onPressed(). It is mostly used in toolbars, dialogs, or inline with other content. By default, the flat button has no color, and its text is black. But, we can use color to the button and text using color and textColor attributes, respectively.



➢ Raised Button

- It is a button, which is based on the material widget and has a rectangular body. It is similar to a flat button, but it has an elevation that will increases when the button is pressed. It adds dimension to the UI along Z-axis. It has several properties like text color, shape, padding, button color, the color of a button when disabled, animation time, elevation, etc.
- This button has two callback functions.
- onPressed(): It is triggered when the button is pressed.
- onLongPress(): It is triggered when the button is long pressed.

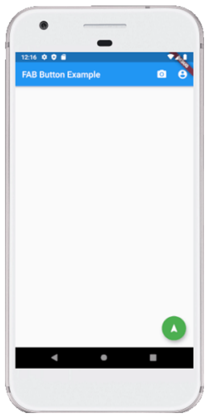

Flutter RaisedButton Example
We have learned FlutterRaised button example.
Click Here

➢ Floating Action Button

- A FAB button is a circular icon button that triggers the primary action in our application. It is the most used button in today's applications. We can use this button for adding, refreshing, or sharing the content. Flutter suggests using at most one FAB button per screen. There are two types of Floating Action Button:

FloatingActionButton: It creates a simple circular floating button with a child widget inside it. It must have a child parameter to display a widget.

FloatingActionButton.extended: It creates a wide floating button along with an icon and a label inside it. Instead of a child, it uses labels and icon parameters.



- **Drop Down Button**

- A drop-down button is used to create a nice overlay on the screen that allows the user to select any item from multiple options. Flutter allows a simple way to implement a drop-down box or drop-down button. This button shows the currently selected item and an arrow that opens a menu to select an item from multiple options.

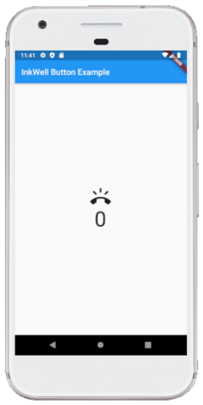
- Flutter provides a Dropdown Button widget to implement a drop-down list. We can place it anywhere in our app.

- **Icon Button**

- An IconButton is a picture printed on the Material widget. It is a useful widget that gives the Flutter UI a material design feel. We can also customize the look and feel of this button. In simple terms, it is an icon that reacts when the user will touch it.

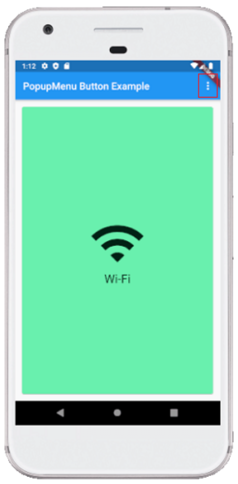
- **Inkwell Button**

- InkWell button is a material design concept, which is used for touch response. This widget comes under the Material widget where the ink reactions are actually painted. It creates the app UI interactive by adding gesture feedback. It is mainly used for adding splash ripple effect.



➢ **PopupMenu Button**

- It is a button that displays the menu when it is pressed and then calls the onSelected method the menu is dismissed. It is because the item from the multiple options is selected. This button contains a text and an image. It will mainly use with Settings menu to list all options. It helps in making a great user experience.



➢ **Outline Button**

- It is similar to the flat button, but it contains a thin grey rounded rectangle border. Its outline border is defined by the shape attribute.

➢ <u>TextField</u>

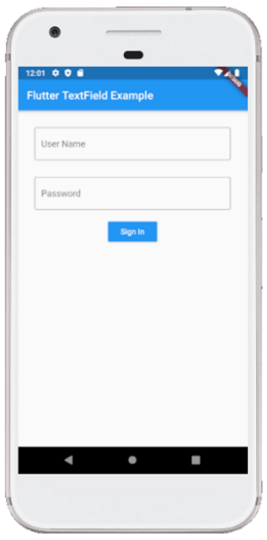
- A TextField or TextBox is an input element which holds the alphanumeric data, such as name, password, address, etc. It is a GUI control element that enables the user to enter text information using a programmable code. It can be of a single-line text field (when only one line of information is required) or multiple-line text field (when more than one line of information is required).

➢ demo example of TextFiled widget in Flutter

```
TextField (
  decoration: InputDecoration(
    border: InputBorder.none,
    labelText: 'Enter Name',
    hintText: 'Enter Your Name'
  ),
);
```

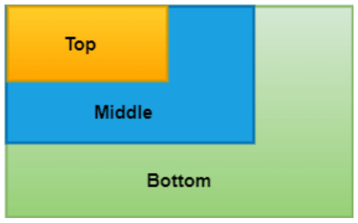
➢ common attributes used with the TextField widget

- **decoration:** It is used to show the decoration around TextField.
- **border:** It is used to create a default rounded rectangle border around TextField.
- **labelText:** It is used to show the label text on the selection of TextField.
- **hintText:** It is used to show the hint text inside TextField.
- **icon:** It is used to add icons directly to the TextField.



➢ <u>**Stack**</u>

- The stack is a widget in Flutter that contains a list of widgets and positions them on top of the other. In other words, the stack allows developers to overlap multiple widgets into a single screen and renders them from bottom to top. Hence, the first widget is the bottommost item, and the last widget is the topmost item



➢ <u>**Properties of the Stack Widget**</u>

- **alignment:** It determines how the children widgets are positioned in the stack. It can be top, bottom, center, center-right, etc.
- **textDirection:** It determines the text direction. It can draw the text either ltr (left to right) or rtl (right to the left).
- **fit:** It will control the size of non-positioned children widgets in the stack. It has three types: loose, expand and passthrough. The loose used to set the child widget small, the expand attribute makes the child widget as large as possible, and the passthrough set the child widget depending on its parent widget.
- **overflow:** It controls the children widgets, whether visible or clipped, when it's content overflowing outside the stack.
- **clipBehavior:** It determines whether the content will be clipped or not.