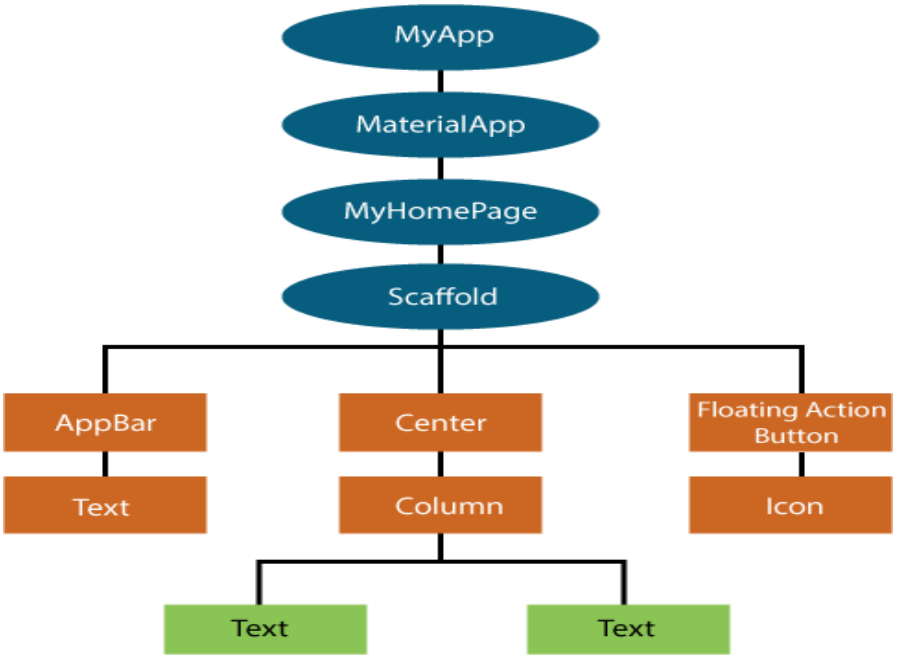
Widgets

- Flutter is Google's UI toolkit for crafting beautiful, natively compiled iOS and Android apps from a single code base.
- To build any application we start with widgets – The building block of flutter applications.
- Widgets describe what their view should look like given their current configuration and state. It includes a text widget, row widget, column widget, container widget, and many more.
- Widgets: Each element on a 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 an apps is a tree of widgets
- Widgets are building blocks which the developer, will compose together to make a UI. When learning Flutter, we will hear the phrase "everything is a widget" quite often.
- Every object in your Flutter application's UI is a widget. Structure is defined with widgets, styles are defined with widgets, even animations and routing is handled by widgets. And, widgets are just Dart classes that know how to describe their view

visual representation of the widget tree



Types of Widgets

- Types of Widget We can split the Flutter widget into two categories:
  - Visible (Output and Input)
  - Invisible (Layout and Control)

Visible widget

The visible widgets are related to the user input and output data. Some of the important types of this widget are:

- Text
- Button
- Image
- Icon

Invisible widget

The invisible widgets are related to the layout and control of widgets. It provides controlling how the widgets actually behave and how they will look onto the screen. Some of the important types of these widgets are:

- Column
- Row
- Center
- Padding
- Scaffold
- Stack

Widget Build Visualization

In Flutter, widgets can be grouped into multiple categories based on their features, as listed below

- Platform specific widgets
- Layout widgets
- State maintenance widgets
- Platform independent / basic widgets

Platform specific widgets

- Flutter has widgets specific to a particular platform - Android or iOS.
- Android specific widgets are designed in accordance with Material design guideline by Android OS. Android specific widgets are called as Material widgets.
- iOS specific widgets are designed in accordance with Human Interface Guidelines by Apple and they are called as Cupertino widgets.

Some of the most used material widgets are →

Scaffold	PopupMenuButton
AppBar	ButtonBar
BottomNavigationBar	TextField
TabBar	Checkbox
TabBarView	Radio
ListTile	Switch
RaisedButton	Slider
FloatingActionButton	Date & Time Pickers
FlatButton	SimpleDialog
IconButton	AlertDialog
DropdownButton	

Some of the most used Cupertino widgets are

CupertinoButton CupertinoPicker CupertinoDatePicker CupertinoTimerPicker CupertinoNavigationBar CupertinoTabBar CupertinoTabScaffold CupertinoTabView CupertinoTextField CupertinoDialog	CupertinoDialogAction CupertinoFullscreenDialogTransition CupertinoPageScaffold CupertinoPageTransition CupertinoActionSheet CupertinoActivityIndicator CupertinoAlertDialog CupertinoPopupSurface CupertinoSlider

Layout widgets

In Flutter, a widget can be created by composing one or more widgets. To compose multiple widgets into a single widget, Flutter provides large number of widgets with layout feature. For example, the child widget can be centered using Center widget.

Some of the popular layout widgets are as follows −

- Container − A rectangular box decorated using BoxDecoration widgets with background, border and shadow.
- Center − Center its child widget.
- Row − Arrange its children in the horizontal direction.
- Column − Arrange its children in the vertical direction.
- Stack − Arrange one above the another.

State maintenance widgets

- In Flutter, all widgets are either derived from StatelessWidget or StatefulWidget.
- Widget derived from StatelessWidget does not have any state information but it may contain widget derived from StatefulWidget.
- The dynamic nature of the application is through interactive behavior of the widgets and the state changes during interaction.
- For example, tapping a counter button will increase / decrease the internal state of the counter by one and reactive nature of the Flutter widget will auto re-render the widget using new state information.

Platform independent / basic widgets

- Flutter provides large number of basic widgets to create simple as well as complex user interface in a platform independent manner.

- some of the basic widgets
  - Text
  - Image
  - Icon

Text

- Text widget is used to display a piece of string. The style of the string can be set by using style property and TextStyle class. The sample code for this purpose is as follows −

```
Text('Hello World!', style: TextStyle(fontWeight: FontWeight.bold))
```

- Text widget has a special constructor, Text.rich, which accepts the child of type TextSpan to specify the string with different style. TextSpan widget is recursive in nature and it accepts TextSpan as its children.

The most important properties of the Text widget are as follows −

- **maxLines, int** − Maximum number of lines to show
- **overflow, TextOverFlow** − Specify how visual overflow is handled using TextOverFlow class
- **style, TextStyle** − Specify the style of the string using TextStyle class
- **textAlign, TextAlign** − Alignment of the text like right, left, justify, etc., using TextAlign class
- **textDirection, TextDirection** − Direction of text to flow, either left-to-right or right-to-left

Image

- Image widget is used to display an image in the application. Image widget provides different constructors to load images from multiple sources and they are as follows −

- Image − Generic image loader using ImageProvider
- Image.asset − Load image from flutter project's assets
- Image.file − Load image from system folder
- Image.memory − Load image from memory
- Image.Network − Load image from network

The easiest option to load and display an image in Flutter is by including the image as assets of the application and load it into the widget on demand

The most important properties of the Image widget are as follows −

- image, ImageProvider − Actual image to load
- width, double − Width of the image
- height, double − Height of the image
- alignment, AlignmentGeometry − How to align the image within its bounds

Icon

- Icon widget is used to display a glyph from a font described in IconData class. The code to load a simple email icon is as follows −
- Icon(Icons.email)

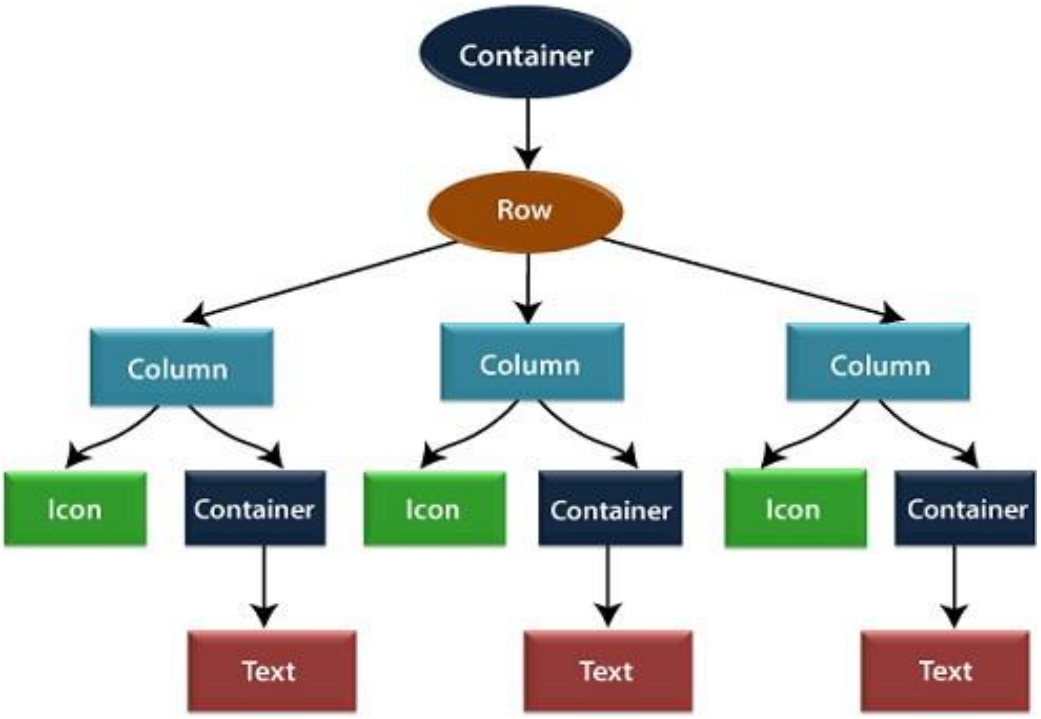
Introduction to Layouts

- The core concept of Flutter is widget, Flutter incorporates a user interface layout functionality into the widgets itself. Flutter provides quite a lot of specially designed widgets like Container, Center, Align, etc., only for the purpose of laying out the user interface.

- Widgets build by composing other widgets normally use layout widgets

Flutter allows us to create a layout by composing multiple widgets to build more complex widgets. For example, we can see the below image that shows three icons with a label under each one

In the second image, we can see the visual layout of the above image. This image shows a row of three columns, and these columns contain an icon and label.



Type of Layout Widgets

- Layout widgets can be grouped into two distinct category based on its child −
  - Widget supporting a single child
  - Widget supporting multiple child

Widget supporting a single child

- The single child layout widget is a type of widget, which can have only one child widget inside the parent layout widget. These widgets can also contain special layout functionality.
- Flutter provides us many single child widgets to make the app UI attractive. If we use these widgets appropriately, it can save our time and makes the app code more readable. The list of different types of single child widgets are:
- For example, Center widget just centers it child widget with respect to its parent widget and Container widget provides complete flexibility to place it child at any given place inside it using different option like padding, decoration, etc.
- Single child widgets are great options to create high quality widget having single functionality such as button, label, etc
- Most important single child layout widgets provided by Flutter −
- **Padding** − Used to arrange its child widget by the given padding. Here, padding can be provided by EdgeInsets class.
- **Align** − Align its child widget within itself using the value of alignment property. The value for alignment property can be provided by FractionalOffset class. The FractionalOffset class specifies the offsets in terms of a distance from the top left.
- Some of the possible values of offsets are as follows −
  - FractionalOffset(1.0, 0.0) represents the top right.
  - FractionalOffset(0.0, 1.0) represents the bottom left.
- FittedBox − It scales the child widget and then positions it according to the specified fit.
- AspectRatio − It attempts to size the child widget to the specified aspect ratio

basic layout widgets as specified below −

- **Container** − Generic, single child, box based container widget with alignment, padding, border and margin along with rich styling features.
- **Center** − Simple, Single child container widget, which centers its child widget

Multiple Child Widgets

- In this category, a given widget will have more than one child widgets and the layout of each widget is unique.
- For example, Row widget allows the laying out of its children in horizontal direction, whereas Column widget allows laying out of its children in vertical direction. By composing Row and Column, widget with any level of complexity can be built.
- Let us learn some of the frequently used widgets in this section.
  - Row − Allows to arrange its children in a horizontal manner.
  - Column − Allows to arrange its children in a vertical manner.
  - ListView − Allows to arrange its children as list.
  - GridView − Allows to arrange its children as gallery.
  - Expanded − Used to make the children of Row and Column widget to occupy the maximum possible area.
  - Table − Table based widget.
  - Flow − Flow based widget.
  - Stack − Stack based widget.

Introduction to Gestures

- Gestures are an interesting feature in Flutter that allows us to interact with the mobile app (or any touch-based device). Generally, gestures define any physical action or movement of a user in the intention of specific control of the mobile device.
- Gestures are as simple as tapping the screen of the mobile device to more complex actions used in gaming applications.

Some of the widely used gestures are mentioned here −

- **Tap** − Touching the surface of the device with fingertip for a short period and then releasing the fingertip.
- **Double Tap** − Tapping twice in a short time.
- **Drag** − Touching the surface of the device with fingertip and then moving the fingertip in a steady manner and then finally releasing the fingertip.
- **Flick** − Similar to dragging, but doing it in a speeder way.
- **Pinch** − Pinching the surface of the device using two fingers.
- **Spread/Zoom** − Opposite of pinching.
- **Panning** − Touching the surface of the device with fingertip and moving it in any direction without releasing the fingertip.

- Flutter provides an excellent support for all type of gestures through its exclusive widget, GestureDetector. GestureDetector is a non-visual widget primarily used for detecting the user's gesture.
- To identify a gesture targeted on a widget, the widget can be placed inside GestureDetector widget. GestureDetector will capture the gesture and dispatch multiple events based on the gesture

Some of the gestures and the corresponding events

• Tap
  • onTapDown
  • onTapUp
  • onTap
  • onTapCancel
• Double tap
  • onDoubleTap
 	
• Vertical drag
  • onVerticalDragStart
  • onVerticalDragUpdate
  • onVerticalDragEnd
	• Horizontal drag
	  • onHorizontalDragStart
	  • onHorizontalDragUpdate
	  • onHorizontalDragEnd
	• Pan
	  • onPanStart
	  • onPanUpdate
	  • onPanEnd
	• Long press
	  • onLongPress

Flutter also provides a low-level gesture detection mechanism through Listener widget. It will detect all user interactions and then dispatches the following events −

- PointerDownEvent
- PointerMoveEvent
- PointerUpEvent
- PointerCancelEvent

Flutter also provides a small set of widgets to do specific as well as advanced gestures. The widgets are listed below −

- Dismissible − Supports flick gesture to dismiss the widget.
- Draggable − Supports drag gesture to move the widget.
- LongPressDraggable − Supports drag gesture to move a widget, when its parent widget is also draggable.
- DragTarget − Accepts any Draggable widget
- IgnorePointer − Hides the widget and its children from the gesture detection process.
- AbsorbPointer − Stops the gesture detection process itself and so any overlapping widget also can not able to participate in the gesture detection process and hence, no event is raised.

Scrollable − Support scrolling of the content available inside the widget

MaterialApp

What is MaterialApp?

- At its core, MaterialApp is a container for your entire Flutter app. It sets up the app's main structure and provides essential features and services. Think of it as the foundation upon which you build your app's UI and functionality.

- The MaterialApp widget provides a wrapper around other Material Widgets. We can access all the other components and widgets provided by Flutter SDK.
- Text wedget, DropdownButton widget, AppBar widget, Scaffold widget, ListView widget, StatelessWidget, StatefulWidget, IconButton widget, TextField widget, Padding widget, ThemeData widget, etc. are the widgets that can be accessed using MaterialApp class

Scaffold

- Scaffold is a class in flutter which provides many widgets or we can say APIs like Drawer, Snack-Bar, Bottom-Navigation-Bar, Floating-Action-Button, App-Bar, etc. Scaffold will expand or occupy the whole device screen. It will occupy the available space. Scaffold will provide a framework to implement the basic material design layout of the application.

Properties of Scaffold Class:

- app-Bar: It displays a horizontal bar which mainly placed at the top of the Scaffold. appBar uses the widget AppBar which has its own properties like elevation, title, brightness, etc
- body: It will display the main or primary content in the Scaffold. It is below the appBar and under the floatingActionButton. The widgets inside the body are at the left-corner by default.
- floatingActionButton: FloatingActionButton is a button that is placed at the right bottom corner by default. FloatingActionButton is an icon button that floats over the content of the screen at a fixed place.

Row and Column

- Row and Column are the two most important and powerful widgets in Flutter. These widgets let you align children horizontally and vertically as per the requirement.
- To design any UI(User Interface) in a flutter, we need to arrange its content in the Row and Column manner so these Row and Column widgets are required when designing UI.

Row Widget

- This widget arranges its children in a horizontal direction on the screen. That is , 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.
- A row widget does not appear scrollable because it displays the widgets within the visible view.
- We can control how a row widget aligns its children based on our choice using the property **crossAxisAlignment** and **mainAxisAlignment**. The row's **cross-axis** will run **vertically**, and the **main axis** will run **horizontally**. See the below visual representation to understand it more clearly.

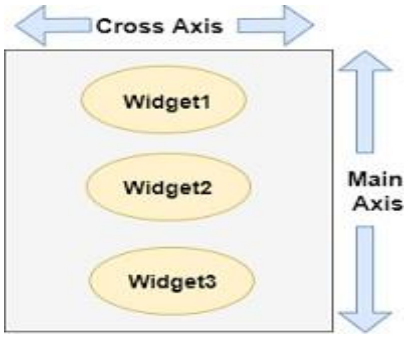
We can align the row's children widget with the help of the following properties:

- **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.

Column

- 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
- We can also control how a column widget aligns its children using the property mainAxisAlignment and crossAxisAlignment. The column's **cross-axis** will run **horizontally**, and the **main axis** will run **vertically**. The below visual representation explains it more clearly.



Text

- Flutter Text
- A Text is a widget in Flutter that allows us to display a string of text with a single line in our application. Depending on the layout constraints, we can break the string across multiple lines or might all be displayed on the same line. If we do not specify any styling to the text widget, it will use the closest DefaultTextStyle class style. This class does not have any explicit style. In this article, we are going to learn how to use a Text widget and how to style it in our application

  The following are the essential properties of the Text widget used in our application:

- **TextAlign**: It is used to specify how our text is aligned horizontally. It also controls the text location.
- **TextDirection**: It is used to determine how textAlign values control the layout of our text. Usually, we write text from left to right, but we can change it using this parameter.
- **Overflow**: It is used to determine when the text will not fit in the available space. It means we have specified more text than the available space.
- **TextScaleFactor**: It is used to determine the scaling to the text displayed by the Text widget. Suppose we have specified the text scale factor as 1.5, then our text will be 50 percent larger than the specified font size……

Center widget

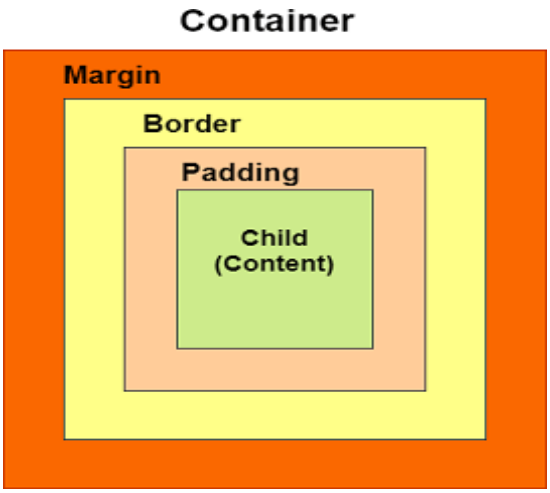
- A widget that centers its child within itself.
- This widget will be as big as possible if its dimensions are constrained and widthFactor and heightFactor are null. If a dimension is unconstrained and the corresponding size factor is null then the widget will match its child's size in that dimension.
- If a size factor is non-null then the corresponding dimension of this widget will be the product of the child's dimension and the size factor.
- For example if widthFactor is 2.0 then the width of this widget will always be twice its child's width**.**

Align class

- A widget that aligns its child within itself and optionally sizes itself based on the child's size.
- For example, to align a box at the bottom right, you would pass this box a tight constraint that is bigger than the child's natural size, with an alignment of Alignment.bottomRight.
- This widget will be as big as possible if its dimensions are constrained and widthFactor and heightFactor are null. If a dimension is unconstrained and the corresponding size factor is null then the widget will match its child's size in that dimension.
- if a size factor is non-null then the corresponding dimension of this widget will be the product of the child's dimension and the size factor. For example if widthFactor is 2.0 then the width of this widget will always be twice its child's width.

Flutter Container

- 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
- A container widget is same as tag in html. If this widget does not contain any child widget, it will fill the whole area on the screen automatically. Otherwise, it will wrap the child widget according to the specified height & width
- A basic container has a margin, border, and padding properties surrounding its child widget, as shown in the below image:



Properties of Container widget

1. **child**: This property is used to store the child widget of the container

2. **color**: This property is used to set the background color of the text. It also changes the background color of the entire container.

3. **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.

4. **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

5. **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

6. **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. In the below example, we are going to align its child into the bottom right position.

7. **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.

8. **transform**: The transform property allows developers to rotate the container. It can rotate the container in any direction, i.e., change the container coordinate in the parent widget. In the below example, we will rotate the container in the z-axis

9. **constraints**: This property is used when we want to add additional constraints to the child. It contains various constructors, such as tight, loose, expand, etc. Let's see how to use these constructors in our app:

Padding

- Padding widget in flutter does exactly what its name says, it adds padding or empty space around a widget or a bunch of widgets. We can apply padding around any widget by placing it as the child of the Padding widget. The size of the child widget inside padding is constrained by how much space is remaining after adding empty space around.
- The Padding widget adds empty space around any widget by using the abstract EdgeInsetsGeometry class.

Properties of Padding Widget:

- child: This property simply takes a widget as the object to display is inside the Padding widget on the screen.
- padding: This property holds the EdgeInsetsGeometry class as the object to add empty space around the widget.

Flutter 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.

Features of Buttons

- The standard features of a button in Flutter are given below:
- We can easily apply themes on buttons, shapes, color, animation, and behavior.
- We can also theme icons and text inside the button.

Buttons can be composed of different child widgets for different characteristics

Types of Flutter Buttons

- Following are the different types of button available in Flutter:
- Flat Button
- Raised Button
- Floating Button
- Drop Down Button
- Icon Button
- Inkwell Button
- PopupMenu Button
- Outline Button

Flutter Images

- When we 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.

How to display the image in Flutter

- 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
```

- Step 4: Finally, open themain.dart file and insert the code.
- Step 5: Now, run the app.

Flutter Icons

- 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

- Flutter icons widget has different properties for customizing the icons. These properties are explained below:

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.