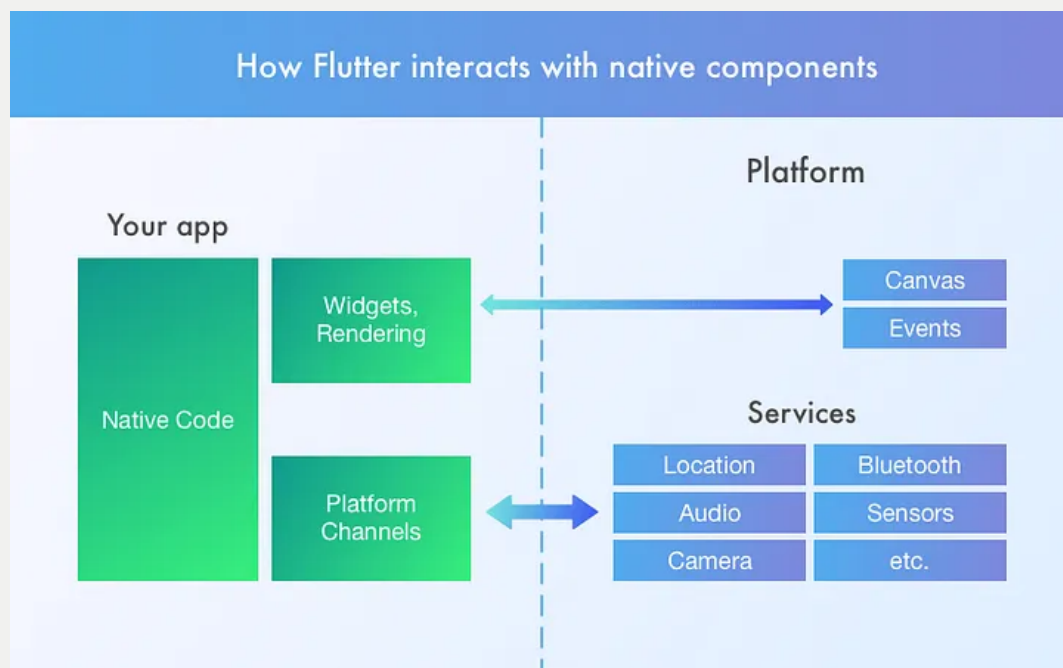


# Introduction to Flutter

## What is Flutter ?



Flutter is Google's UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase.



→ Flutter framework helps you code your app using DART programming language, this helps you to support

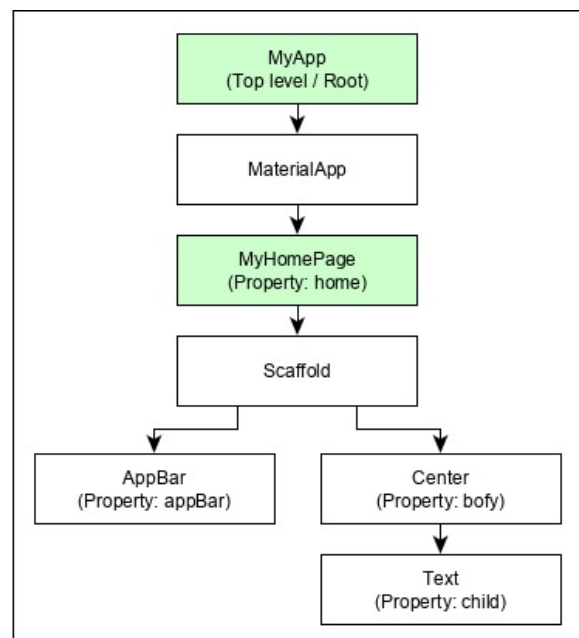
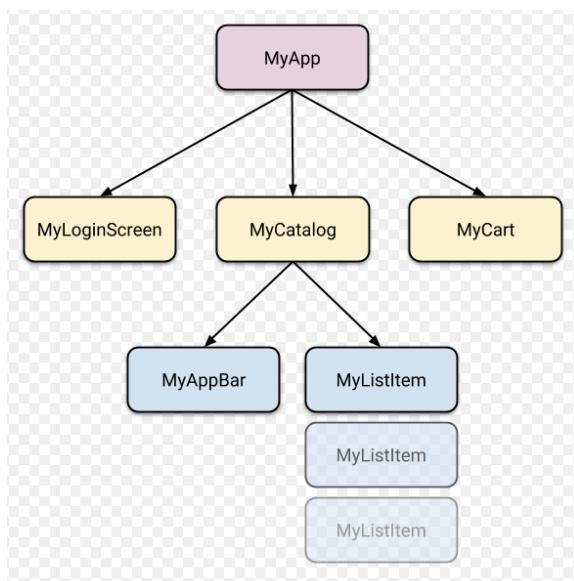
- Mobile OS : Android , iOS
- Web : Any web browser
- Desktop : It also support any kind desktop OS & resolution

```
void main() {
  runApp(MyApp);
}
```

```
class MyApp extends StatelessWidget {
  Widget build(BuildContext context) {
    return MaterialApp(home: Text('Hello!'))
  }
}
```

## Structure of Flutter

→ Flutter has a leveled , top to down tree structure. Every widget has parent widget and may have child widgets.



→ Understanding the Widgets

- Almost every element of your Flutter app is a widget. Widgets are basically user interface components used to create the user interface of the application.
- You'll use two fundamental types of widgets:
  - **Stateless**: Widgets that depend only upon their own configuration info, such as a static image in an image view.

```
class MyWidget extends StatelessWidget {
  final String userName;

  MyWidget(this.userName);

  @override
  Widget build(BuildContext context) {
    return DecorateBox(
      decoration: BoxDecoration(color: Colors.amber),
      child: Text(userName)
    );
  }
}
```

- *StatelessWidget* only requires a single method *build* to be implemented in its derived class.
- **Stateful:** Widgets that need to maintain dynamic information. They do so by interacting with a *State* object. It provides states management.

```
class MyWidget extends StatefulWidget {
  final String userName;

  MyWidget({this.userName});

  @override
  _MyWidget createState() => _MyWidget();
}

class _MyWidget extends State<MyWidget> {
  int appsCount = 1;

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () {
        setState(() {
```

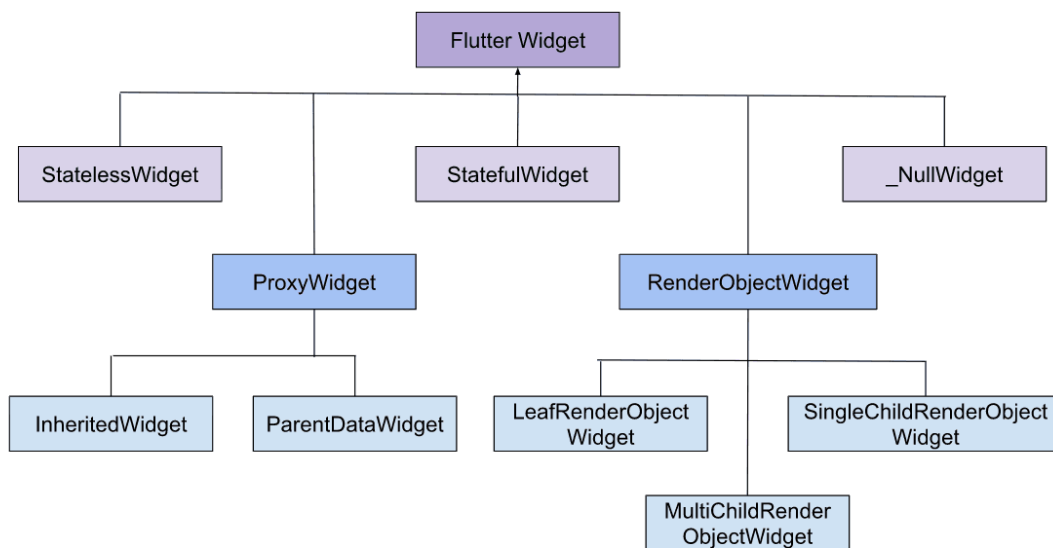
```

        appCount++;
    })
  },
  child: Text('The user ${widget.userName} has ${a
});}}

```

- In Flutter, all widgets are either derived from *StatelessWidget* or *StatefulWidget*.

## Widget Types



- **StatelessWidget:** The widget does not require a mutable state.
- **StatefulWidget:** The widget has a mutable state i.e state information can be read synchronously when it is built and it may change during the widget's lifetime. Therefore it's essential to ensure that the state is promptly notified when the change occurs using *setState*.
- **ProxyWidget:** The Widget has a child widget provided to it, instead of building a new widget. It is useful as a base class for other widgets like *InheritedWidget* and *ParentDataWidget*.
- **RenderObjectWidget:** It provides the configuration for the *RenderObjectElements*, which wrap *RenderObjects*, that provide the actual rendering of the application.

- **InheritedWidget:** It is a base class for widgets that efficiently propagates information down the tree. To obtain the nearest instance of a particular type of inherited widget from a build context, use *BuildContext.dependOnInheritedWidgetOfExactType*. When referenced in this way, it causes the consumer to rebuild when the inherited widget itself changes state.
- **ParentDataWidget:** It is a base class for a widget that hooks *ParentData* information to children of *RenderObjectWidgets*. It is used to provide a per-child configuration for *RenderObjectWidgets* with more than one child.
- **LeafRenderObjectWidget:** It is a superclass for *RenderObjectWidgets* that configure *RenderObject* subclasses that have no children.
- **SingleChildRenderObjectWidget:** It is a superclass for *RenderObjectWidgets* that configure *RenderObject* subclasses that have a single child slot.
- **MultiChildRenderObjectWidget:** It is a superclass for *RenderObjectWidgets* that configure *RenderObject* subclasses that have a single list of children.
- **\_NullWidget:** Zero cost widget. Use it when you need a placeholder.

## Most Used Widgets

### Animation And Motion Widgets

Flutter app development needs good animation. Flutter provides animation widgets without this animation it looks not good.

- **AnimatedAlign :** An animated version of the alignment that automatically changes the child's position over a given period when a given configuration changes
- **AnimatedBuilder :** General purpose widget for creating animations. To use Animated Builder, just create a widget and pass it to the builder function
- **AnimatedContainer :** A container that gradually changes its values over time

- **AnimatedCrossFade** : This widget provided animation between two children with different size
- **AnimatedDefaultTextStyle** : An animated version of the default text style that automatically transitions to the default default text style (text style to apply to offspring text widgets without specified style) when a given style changes
- **AnimatedListState**: The position for the scrolling container that animates when items are inserted or removed
- **AnimatedModalBarrier**: A widget that prevents the user from interacting with the widgets behind it.
- **AnimatedOpacity** : Animate opacity can help by merging the widget into a view or making it less prominent when something is not liked
- **AnimatedPositioned** : An animated version of the status that automatically changes the child's status to a given period when a given status changes
- **AnimatedSize** : Animated widget that automatically transfers the size of a given child over a given period of time
- **DecoratedBoxTransition**: An animated version of the decorated box that animates the different properties of its decor.
- **FadeTransition** : Animates the opacity of the widget.
- **Hero** : A widget that marks her child as a candidate for Hero Animation.
- **PositionedTransition** : An animated version of Positioned takes specific animations to move the child's position from start to finish during the life of the animation.
- **RotationTransition** : Animates the rotation of a widget.
- **ScaleTransition** : Animates the scale of the converted widget.
- **SizeTransition** : Animates its own size and aligns the child clips and clip.
- **SlideTransition**: The position of the widget corresponds to its normal position.

## Accessibility Widgets

- **ExcludeSemantics** : A widget that drops all the economics of its descendants. It's used for hidden sub widgets but would only be confusing. For example, Material Components hides the chip widget avatar because it does more work than the chip label.
- **MergeSemantics**: A widget that merges the economy of its dynasty.
- **Semantics**: A widget that notifies the widget tree with a description of the meaning of the widgets. Is provided a read only,enabled and label properties.

## Assets, Images, And Icon Widgets

- **AssetBundle**: Asset bundles include resources, such as images and strings, that can be used by the application. These resources are asynchronous so that they can be loaded transparently over the network (e.g. from the Network SetBundle) or from the local file system, without blocking the user interface on the application.
- **Icon**: Icon for Material Design.
- **Image**: A widget that displays an image.
- **RawImage**: Provided the widget in dart UI image direct.

## Async Widgets

- **FutureBuilder** : A widget based on the latest snapshots of interactions with the future that builds itself.
- **StreamBuilder** : Provide the widget in load API data from stream and visible from data in builder.

## Cupertino (iOS-style) Widgets

**Flutter app development** is cross-platform development, so to support iOS, flutter added cupertino widget in order to support iOS apps.

- **CupertinoActionSheet**: IOS-style model bottom action sheet for many people to choose an option.
- **CupertinoActivityIndicator**: IOS-style activity indicator. Displays a circular 'spinner'.
- **CupertinoAlertDialog**: Provide IOS-style alert dialog.

- **CupertinoButton** : Provide a button in IOS-style.
- **CupertinoContextMenu** : IOS-style full-screen modal root that opens when the child is long-pressed. Used to display actions related to your content.
- **CupertinoDatePicker** : Provide date or date and time picker in IOS-style.
- **CupertinoDialog** : Provide a dialog in IOS-style.
- **CupertinoNavigationBar** : IOS-style top navigation bar. Typically used with CupertinoPagescaffold.
- **CupertinoPageScaffold** : Basic iOS style page layout structure. Lays the navigation bar and content on the background.
- **CupertinoPageTransition** : Provides page transition animations in iOS-style.
- **CupertinoPicker**: IOS-style picker control. Provided to select items in list.
- **CupertinoPopupSurface**: A rounded rectangular widget like an iOS pop-up surface, such as an alert dialog or an action sheet.
- **CupertinoScrollbar** : Scrollbar that indicates which part of the currently scrollable widget is visible in IOS style.
- **CupertinoSegmentedControl** : An iOS-style split control. Used to select mutually exclusive options in the horizontal list.
- **CupertinoSlider**: Provided the value select from range.
- **CupertinoSwitch** : OS-style switch. Used to turn on / off the same settings mode.
- **CupertinoTabBar** : IOS-style bottom tab bar. Typically with CupertinoTabscaffold.
- **CupertinoTabView** : Root content of tabs that support navigation parallel between tabs. Typically with Cupertino Tabscaffold.
- **CupertinoTextField**: Provides text fields in iOS-style.

## Input Widgets



- **Form** : Optional container for grouping multiple form field widgets (e.g. textfield widgets) together.
- **FormField** : Single form field .This widget is current state in form field and update and validation error in ui.
- **RawKeyboardListener** : Widget that makes calls to a club back call whenever the user presses or releases a key on the keyboard

## Layout Widgets

Layout widgets are the most common in flutter app development. This is providing all types of facilities including managing content on screen using alignment, aspect ratio, center, padding etc properties.

- **Align** : widget that organizes its child within itself and sizes itself alternatively depending on the child's size.
- **AspectRatio**: A widget that tries to size the child in a certain aspect ratio.
- **Baseline** : A widget that locates its child according to the child's baseline.
- **Center** : A widget that keeps its child inside itself.
- **ConstrainedBox** : A widget that imposes an additional barrier on her child
- **Container** : A handy widget that combines general painting, positioning and resizing widgets.
- **CustomSingleChildLayout** : A widget that sends delegates to the layout of their child.
- **Expanded** : A widget used for extending the child to a row, cumulus or flax.
- **FittedBox** : Its position inside her child according to the scales and positions
- **FractionallySizedBox** : A widget that measures the fraction of the total available space of its available child.
- **IntrinsicHeight** : A widget that gives her child the size of a child's internal height.

- **IntrinsicWidth** : A widget that gives her child the size of a child's internal width.
- **LimitedBox** : A box that does not limit its size.
- **Offstage** : A widget that puts a child out of a tree, but without painting anything, without making the child available for a hit test, and without taking a room in the parent
- **OverflowBox** : A widget that can impose different barriers on her child than she gets from her parents, probably allows the child to overflow the parent.
- **Padding** : A widget that insets her child through a given padding and this widget flutter app developer is used in almost all apps
- **SizedBox** : Box x with specified size. If a child is granted, this widget accepts its child to a specific width and / or height (recognition of beliefs by the parents of this widget). If either width or height is null, this widget will take its own size to match the size of the child in that dimension.
- **Transform** : A widget that applies the conversion to her child before painting

## Multi-child Layout Widgets

- **Column** : This widget is common so flutter app developers uses it often. Layout the list of child widgets in the child direction.
- **CustomMultiChildLayout** : A widget that uses multiple children representative for size and position.
- **Flow** : A widget that implements a flow layout algorithm.
- **GridView** : The grid list also has a recurring pattern of cells arranged in a horizontal layout. The Grid View widget applies this component.
- **IndexedStack** : A stack that shows a child from a list of children
- **LayoutBuilder** : Creates a widget tree based on the size of the parent widget.
- **ListBody** : A widget that aligns its children sequentially with a given axis, forcing them into the parent dimension of the other axis.

- **ListView** : Scrollable, linear list of widgets. Listview is most used for scrolling the widget. She shows her children one scroll direction after another. In the cross axis, children are required to fill out a list view.
- **Row** : Layout the list of child widgets in the horizontal direction.
- **Stack** : This class is useful if you want to easily overlap multiple children, for example, if there is some text and an image, with a button attached at the top and bottom.
- **Table** : A widget that uses a table layout algorithm for its children.
- **Wrap** : A widget that shows her children in multiple horizontal or vert even runs.

## Sliver Widgets

- **CupertinoSliverNavigationBar**: IOS-style navigation bar with large iOS-style titles using sleeves.
- **CustomScrollView** : Scrollview which creates custom scroll effects using sliders.
- **SliverAppBar** : Content design application bar that integrates with CustomScrollView.
- **SliverChildBuilderDelegate** : A builder who supplies children for slippers using the builder's call lab back.
- **SliverChildListDelegate** : A representative who supplies children for Slavers using a clear list.
- **SliverGrid** : A sliver that holds multiple children in two dimensional configurations.
- **SliverList** : A sliver that holds multiple children in a linear array along the main axis.
- **SliverPadding** : A sleeve that applies padding to each side of the other sleeve
- **SliverPersistentHeader** : A sliver whose size changes when the silver scrolls to the edge of the viewport opposite the growth direction.
- **SliverToBoxAdapter** : A sliver that contains a single widget widget.

## Material Components Widgets

This widget is the most used of all apps so **Flutter app development company** expects the **flutter developer** to know the common widget like material components.

- **AppBar** : Material design application bar. An application bar includes toolbars and possibly other widgets, such as the TBbar and the FlexibleSpacebar.
- **BottomNavigationBar** : The navigation bars at the bottom make it easy to explore and switch high-level views in a single tap. The Bottom Navigation Bar Widget applies this component.
- **Drawer** : Provided a design in side a horizontally from edge the scaffold show navigation link in app.
- **MaterialApp** : A handy widget that wraps around many of the widgets typically needed for applications that apply material design.
- **Scaffold** : The basic material design implements the visual layout structure. This class provides a body, app bar and floating button.
- **TabBar** : Material design widget that displays a horizontal row of texts.
- **TabBarView** : A page view that displays a widget that corresponds to the currently selected tab. Especially used in conjunction with tabbars.

## Buttons

- **ButtonBar** : Horizontal alignment of buttons.
- **DropDownButton** : Provided the selected item in a particular list is a menu and select data.
- **FlatButton** : A flat button is a section printed on the Material Components widget that clicks an animated filling color.
- **FloatingActionButton** : The floating action button is a circular icon button that rotates on the content to promote the primary action in the application. Floating action buttons are most commonly used in the scaffold. floating action button area.
- **IconButton** : The icon button is a picture printed on a material widget that responds to touch by filling in color (ink).
- **OutlineButton** : A medium-load button for secondary actions that are important but not the primary action in the application.

- **PopupMenuButton:** Displays the menu when pressed and calls when the menu is fired when the item was selected.
- **RaisedButton :** A material design raised button. The Raised Button contains a rectangular piece of content that rotates across the interface.

## Input and selections

- **Checkbox :** Widget is provided to multiple items in one list. The Checkbox widget implements this component.
- **Date & Time Pickers :** This widget provides a date picker dialog and time picker dialog and the selected date and time is set and sent to the server.
- **Radio:** Widget is provided to a single item in a one list. The radio widget is used for only selected single values.
- **Slider :** Provided the user selects a range of values by moving the slider.
- **Switch :** The switch provided is on / off to tick the status of a settings option. The switch widget applies this component.
- **TextField :** Touching a text field moves the cursor and displays the keyboard. The textfield widget applies this component.

## Dialogs, alerts, and panels

- **AlertDialog :** Alerts are immediate interruptions that require acceptance that inform the user of the situation. The Alert dialog widget applies this component.
- **BottomSheet :** Apply the modal bottom sheet. You can call the show bottom sheet () to apply the continuous bottom sheet or the show modal bottom sheet ().
- **SimpleDialog :** Simple dialogs can provide additional details or actions about the list item. For example, they may display avatar symbols explaining subtext or orthogonal actions (such as adding an account).
- **SnackBar :** A lightweight message with an alternate action that is briefly displayed at the bottom of the screen.

## Information displays

- **Card :** Material design card. The cards have slightly rounded corners and shadows.

- **Chip** : Chip is provided as a complex entity in small blocks, such as contact.
- **CircularProgressIndicator** : A material design circular progress indicator, spins to indicate which application is busy.
- **DataTable** : Data tables display sets of raw data. They usually appear in desktop enterprise products. The datatable widget applies this component.
- **Icon** : Provided Material design icon.
- **Image** : Image is display widgets.
- **LinearProgressIndicator** : Is a progress bar a material design.

## Layout

- **Divider** : A horizontal line between two widget, with padding on both sides.
- **ListTile** : Provided an image with in list in tralling, title, subtitle and leading component.
- **Stepper** : Material design stepper widget that shows progress by sequence of steps

## Painting and effect widgets

Sometimes applications look and feel very good because the app uses custom paint that is defined inside **flutter programming language** already.

- **BackdropFilter** : A widget that applies a filter to existing painted material and then paints the child. This effect is relatively expensive, especially if the filter is non-localized, such as opacity.
- **ClipOval** : Child using an oval in clips widget.
- **ClipPath** : Child using path in clips widget.
- **ClipRect** : Child using a rectangle in clips widget.
- **CustomPaint** : Provides this widget a canvas on which to draw during the paint phase.
- **DecoratedBox** : Decorations before or after her child paints this widget.
- **FractionalTranslation** : A widget that applies a translation expressed as a fraction of B's size before painting it for her child.
- **Opacity** : This widget makes her child partially transparent.

- **RotatedBox** : That rotates her child by an integral number of quarter turns.
- **Transform** : A widget that applies the conversion to her child before painting.

## Scrolling Widgets

- **CustomScrollView** : Scrollview which creates custom scroll effects using sliders.
- **NestedScrollView** : Inside scrolling view, which may contain other scrolling views, with their scroll positions connected to each other
- **NotificationListener** : A widget that listens to subtle instructions that bounce the tree.
- **PageView** : works page by page work that scrollable list.
- **RefreshIndicator** : pull-to-refresh wrapper for scrollable objects in material design.
- **ScrollConfiguration** : Controls how scrollable widgets behave in the subtree.
- **Scrollable** : Scrollable applies a model of interaction to a scrollable widget, including gesture recognition, but not an opinion about how the viewport, which actually displays children, is created.
- **Scrollbar** : A material design scrollbar. scrollbar indicates in which part of scrollable widget is actually visible.
- **SingleChildScrollView** : A box in which a single widget can be scrolled.that all widget in set one column and his set in single child scrollview so all widget are scrollable.

## Styling Widgets

- **MediaQuery** : Is solve the problem in height or width in screen
- **Padding** : A widget that insets her child through a given left, right, top and bottom padding.
- **Theme** : Theme applies to descendant widgets. The theme describes the colors and typographic preferences of the application.

## Text Widgets

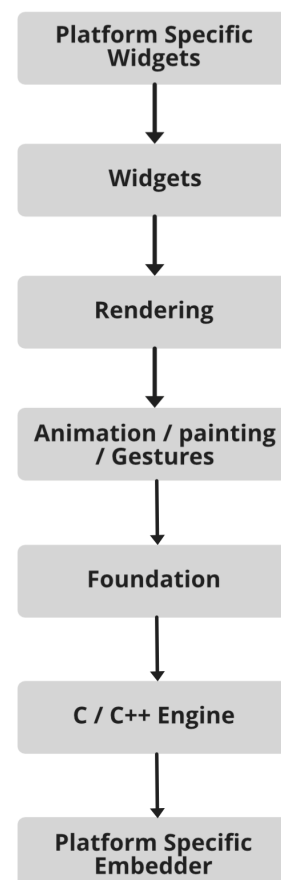
- **DefaultTextStyle** : This widget without explicit style and The text style to apply to descendant.
- **RichText** : This widget displays text in multiple sizes and different colors use text in a single sentence.
- **Text** : Single style run with text.

## Layers

→ The Flutter framework is categorized based on its complexity and establishes a hierarchy based on the decreasing level of these complexities. These categories are often called Layers. These layers are built on top of one another.

## Gesture

→ All physical form of interaction with a flutter application is done through pre-defined gestures. Gesture-Detectors are used for the same. It is an invisible widget that is used to process physical interaction with the flutter application. The interaction includes gestures like tapping, dragging, and swiping, etc. These features can be used to creatively enhance the user experiences of the app by making it perform desired actions based on simple gestures.





# Concept of State

→ The states are nothing but data objects. For the management of state in a Flutter application, Stateful-Widget is used. Re-rendering of widgets specific to the state occurs whenever the state changes. This also avoids the re-rendering of the entire application, every time the state of a widget changes.

## References

### Flutter Tutorial - GeeksforGeeks

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive

 <https://www.geeksforgeeks.org/flutter-tutorial/?ref=lbp>



### Flutter - Introduction to Layouts


Flutter - Introduction to Layouts - Since the core concept of Flutter is Everything is widget, Flutter incorporates a user interface layout functionality into the widgets itself. Flutter

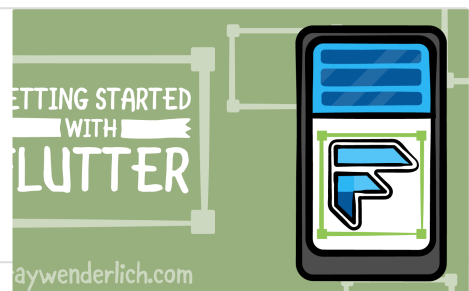
 [https://www.tutorialspoint.com/flutter/flutter\\_introduction\\_to\\_layouts.htm](https://www.tutorialspoint.com/flutter/flutter_introduction_to_layouts.htm)



### Getting Started With Flutter

Dive into the Flutter framework, which lets you build iOS, Android, web and desktop apps with a single codebase, by writing a cross-platform app using VS Code.

 <https://www.kodeco.com/24499516-getting-started-with-flutter/page/3>



## Author → Serhat Kumas

<https://www.linkedin.com/in/serhatkumas/>

## SerhatKumas - Overview

Computer engineering student who loves coding in different fields instead of focusing on a one spesific area. - SerhatKumas

 <https://github.com/SerhatKumas>

