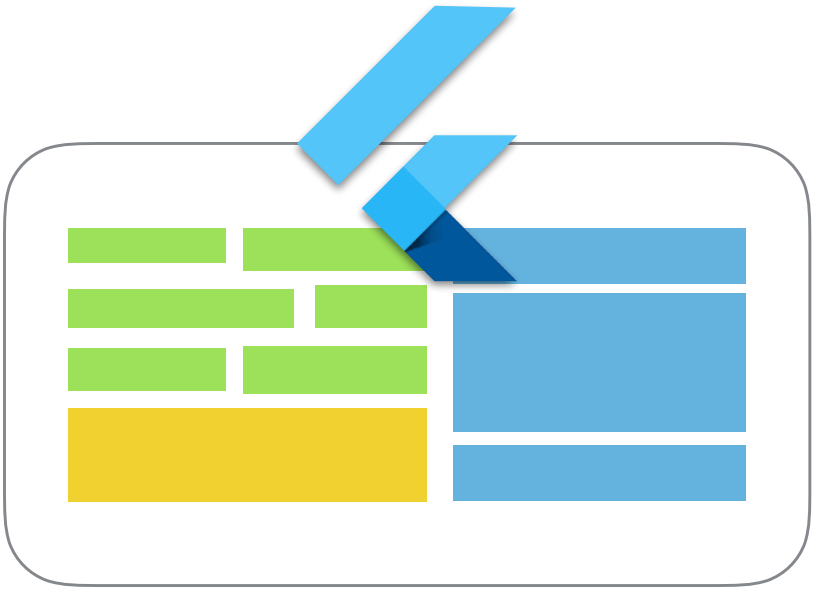
Chapter III

Layout & Widgets



1. MaterialApp Widget

A convenience widget that wraps a number of widgets that are commonly required for material design applications.

The MaterialApp configures the top-level Navigator to search for routes in the following order:

1. For the / route, the home property, if non-null, is used.
2. Otherwise, the routes table is used, if it has an entry for the route.
3. Otherwise, onGenerateRoute is called, if provided. It should return a non-null value for any valid route not handled by home and routes.
4. Finally if all else fails onUnknownRoute is called.

Eg.

```
MaterialApp(

  home:  …. return some widget
  routes: <String,Widget Function(BuildContext       context)> {
                    …. Map of String with BillderContext Function
               }
 onGenerateRoute: (RouteSettings settings) {
              … return some widget
              //Will execute when unregistered route is requested,
              //setting hold the requested name
              }

 onUnknownRoute: (RouteSettings settings) {
              … return some widget
              //Wil execute when onGenerateRoute return null
              }
  )
```

Eg .

```
MaterialApp(
  home: Scaffold(
    appBar: AppBar(
      title: const Text('Home'),
    ),
  ),

routes: <String, WidgetBuilder>{

     '/about': (BuildContext context) {
     return Scaffold(
       appBar: AppBar(
         title: const Text('About Route'),
       ),
     );
    }
  },
```

Note: You can put following instead of home properties for entry point of the App in routes table

```
'/': (BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Home Route'),
      ),
    );
  },
```

1. Theming In MaterialApp

```
theme: ThemeData(
        brightness: Brightness.dark,
        primaryColor: Colors.blueGrey
        ),
darkTheme:ThemeData(
        brightness: Brightness.dark,
        primaryColor: Colors.blue
        ),
```

2. Localization In MaterialApp

MaterialApp other properties: supportedLocales, localizationsDelegates and localeResolutionCallback are used for localization in app. For that, flutter_localizations package need to added in pubspec.yaml.

Eg flutter_localizations:
  sdk: flutter

2. Scaffold Widget

implements the basic material design visual layout structure.

This class provides APIs for showing drawers, snack bars, and bottom sheets.To display a snackbar or a persistent bottom sheet, obtain the ScaffoldState for the current BuildContext via Scaffold.of and use the ScaffoldState.showSnackBar and ScaffoldState.showBottomSheet functions.

```
Scaffold(
   backgroundColor: Colors.blue,
   appBar: AppBar(
      title: const Text('Sample Code'),
    ),
    body:
      Center(child:
            Text('Scaffolded App !')
      ),
    floatingActionButton: FloatingActionButton(
     onPressed: () { },
      child: const Icon(Icons.add),
    ),

 )
```

Ref:
https://api.flutter.dev/flutter/material/Scaffold-class.html

3. AppBar Widget

A material design app bar.

An app bar consists of a toolbar and potentially other widgets, such as a TabBar and a FlexibleSpaceBar.The AppBar displays the toolbar widgets, leading, title, and actions, above the bottom (if any). The bottom is usually used for a TabBar. If a flexibleSpace widget is specified then it is stacked behind the toolbar and the bottom widget.

```
AppBar(

title: const Text('AppBar Demo'),
      actions:  [
        IconButton(
          icon: const Icon(Icons.add),
          tooltip: 'Show Snackbar',
          onPressed: () {
            scaffoldKey.currentState.showSnackBar(snackBar);
          },
        ),
        IconButton(
          icon: const Icon(Icons.add_a_photo),
          tooltip: 'Next page',
          onPressed: () {
            openPage(context);
          },
        ),
      ],
    ),
```

Ref: https://api.flutter.dev/flutter/material/AppBar-class.html

4. Image Widget

A widget that displays an image. Several constructors are provided for the various ways that an image can be specified:

- new Image, for obtaining an image from an ImageProvider.
- new Image.asset, for obtaining an image from an AssetBundle using a key.
- new Image.network, for obtaining an image from a URL.
- new Image.file, for obtaining an image from a File.
- new Image.memory, for obtaining an image from a Uint8List.

```
Image (
        image:    NetworkImage("https:// ...."),    or
                  AssetImage("Path\Imagefilename")
)
```

Note: For free photos/Images/Icon

https://unsplash.com/

https://icons8.com/

https://www.vecteezy.com/

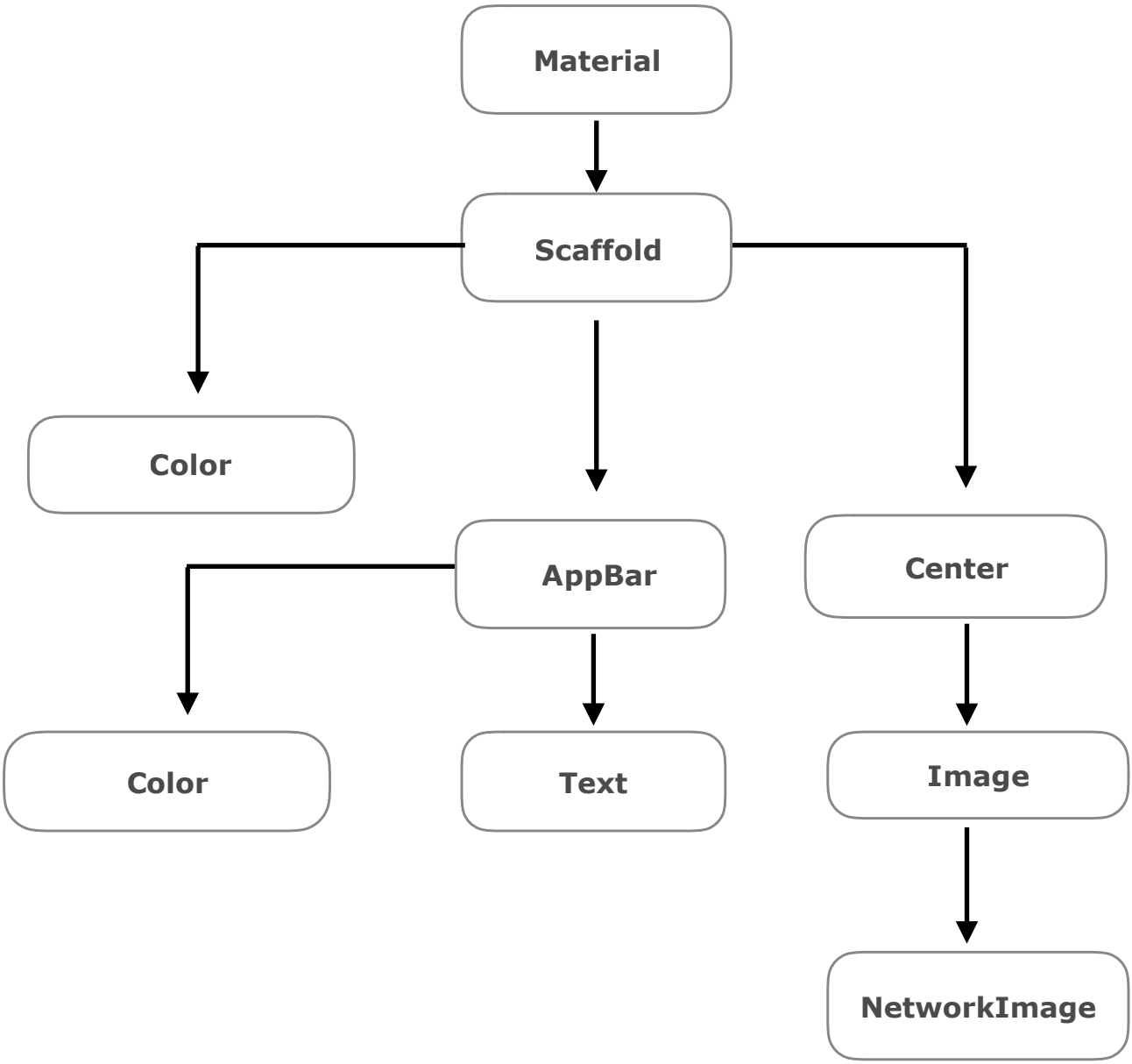
5. Visualization Widget Tree

Consider following App and if you draw simple widget trees structure,

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

void main() {
  runApp(
        MaterialApp(
         title: "App Demo",
         home: Scaffold(
                    appBar: AppBar(
                    title: Text("Widget Tree"),
                    ),
                    body: Center(
                              child: Image(
                              image: NetworkImage("https:// …."),
                              ),
                        ),
                    ),
              )
        );
}
```

The result will come out in fig 1.



6. Container Widget

A convenience widget that combines common painting, positioning, and sizing widgets.

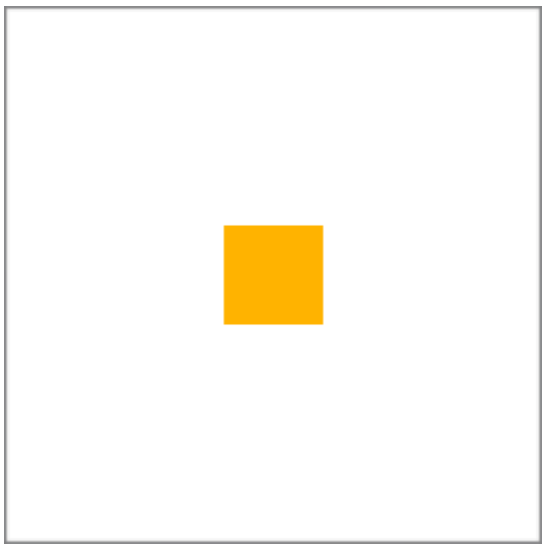
https://api.flutter.dev/flutter/widgets/Container-class.html

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

void main() {
  runApp(
        MaterialApp(
         title: "App Demo",
         home: Scaffold(
                        appBar: AppBar(
                        title: Text("Widget Tree"),
                        ),
                        body: Center(
                                        child: Container(
                                        margin: EdgeInsets.all(10.0),
                                        color: Colors.amber[600],
                                        width: 48.0,
                                        height: 48.0,
                                        ),
)                                   ),
                        )
        );
}
```

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

void main() {
  runApp(
        MaterialApp(
         title: "App Demo",
         home: Scaffold(
                        appBar: AppBar(
                        title: Text("Widget Tree"),
                        ),
                        body: Container(
                                        constraints: BoxConstraints.expand(
                                        height: Theme.of(context).textTheme.display1.fontSize *
                                        1.1 + 200.0,
                                         ),
                                padding: EdgeInsets.all(8.0),
                                color: Colors.blue[600],
                                 alignment: Alignment.center,
                                child: Text('Hello World',
                                        style: Theme.of(context)
                                                .textTheme
                                                .display1
                                                .copyWith(color: Colors.white)),
                                        transform: Matrix4.rotationZ(0.1),
                                        )
                        )
                );
}
```




```
Column(
  children: <Widget>[
    Text('Deliver features faster'),
    Text('Craft beautiful UIs'),
    Expanded(
      child: FittedBox(
        fit: BoxFit.contain, // otherwise the logo will be tiny
        child: const FlutterLogo(),
      ),
    ),
  ],
)
```

7. Column Widget

A widget that displays its children in a vertical array.

- To cause a child to expand to fill the available vertical space, wrap the child in an Expanded widget.
- The Column widget does not scroll
- If you have a line of widgets and want them to be able to scroll if there is insufficient room, consider using a ListView.

```
Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  mainAxisSize: MainAxisSize.min,
  children: <Widget>[
              Text('We move under cover and we move as one'),
              Text('Through the night, we have one shot to live another
                    day'),
              Text('We cannot let a stray gunshot give us away'),
              Text('We will fight up close, seize the moment and stay in it'),
              Text('It's either that or meet the business end of a bayonet'),
              Text('The code word is 'Rochambeau,' dig me?'),
              Text('Rochambeau!',
              style:
DefaultTextStyle.of(context).style.apply(fontSizeFactor: 2.0)),
  ],
)
```

Troubleshooting

When the incoming vertical constraints are unbounded

When a Column has one or more Expanded or Flexible children, and is placed in another Column, or in a ListView, or in some other context that does not provide a maximum height constraint for the Column, you will get an exception at runtime saying that there are children with non-zero flex but the vertical constraints are unbounded.

The key to solving this problem is usually to determine why the Column is receiving unbounded vertical constraints.

One common reason for this to happen is that the Column has been placed in another Column (without using Expanded or Flexible around the inner nested Column). When a Column lays out its non-flex children (those that have neither Expanded or Flexible around them), it gives them unbounded constraints so that they can determine their own dimensions (passing unbounded constraints usually signals to the child that it should shrink-wrap its contents). The solution in this case is typically to just wrap the inner column in an Expanded to indicate that it should take the remaining space of the outer column, rather than being allowed to take any amount of room it desires.
Another reason for this message to be displayed is nesting a Column inside a ListView or other vertical scrollable. In that scenario, there really is infinite vertical space (the whole point of a vertical scrolling list is to allow infinite space vertically). In such scenarios, it is usually worth examining why the inner Column should have an Expanded or Flexible child: what size should

the inner children really be? The solution in this case is typically to remove the Expanded or Flexible widgets from around the inner children.