Flutter UI

Key Points

Notes

1. **Setting and Configuring a Flutter Application**
   a. *pubspec.yaml*
      i. It defines the configuration of your Flutter application when it first start to load.

      ii. You can define many specific configurations such as define the assets such as images, audios, icons and files to be used by your Flutter app.

      iii. It can also define default font to be used for your Flutter application.

      iv. You can refer to your project *pubspec.yaml* file for details.

   b. *main.dart*
      i. Additional to *pubspec.yaml* file, your *main.dart* file contains entry point to your Flutter application which is the main function or method.

      ii. You can apply setting such as enable portrait mode before your Flutter application is loaded.

      iii. For example:

```
void main() {
  AppSettings settings = AppSettings();

  // Don't allow landscape mode
  SystemChrome.setPreferredOrientations([
          DeviceOrientation.portraitUp,
          DeviceOrientation.portraitDown,
       ])
      .then((_) => runApp(
        MyApp(settings: settings),
      ));
}
```

Summary

Key Points	Notes

c. *SystemChrome*
   i. It is a Flutter class that control how your Flutter application displays on the native platform. In fact, this is the only classes that you will use to manipulate the device itself.

   ii. Static methods that can be used to control the display of your Flutter application: https://api.flutter.dev/flutter/services/SystemChrome-class.html

   iii. Refer to previous example (*b.iii*) on *SystemChrome* class implementation.

2. Structural Widgets and Configuration

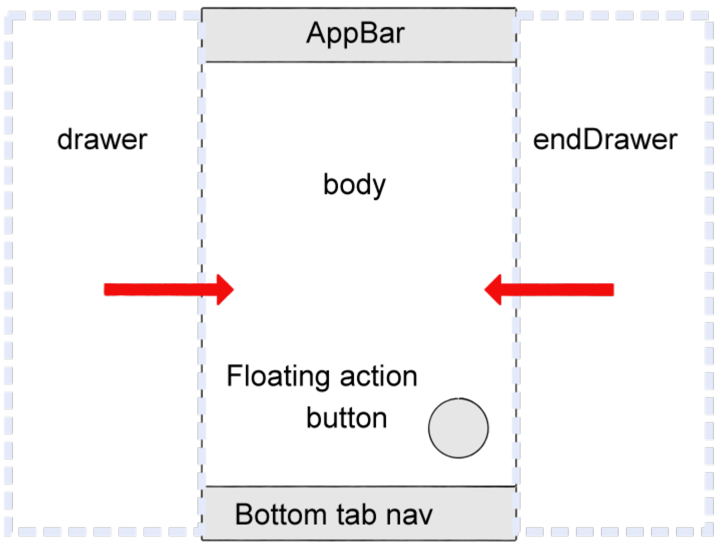
a. *MaterialApp widget*
   i. It is a generic top-level widget, which add Material design specific functionality and styling options to your Flutter application.

   ii. You can follow Material style guidelines https://material.io/design to get that standard Google look and feel. If you are working on iOS devices you can use *CupertinoApp* widget.

   iii. It is a standard in Flutter to define your structural widget first either using *MaterialApp* or *CupertinoApp* widget.

   iv. List of available properties for MaterialApp class https://api.flutter.dev/flutter/material/MaterialApp-class.html.

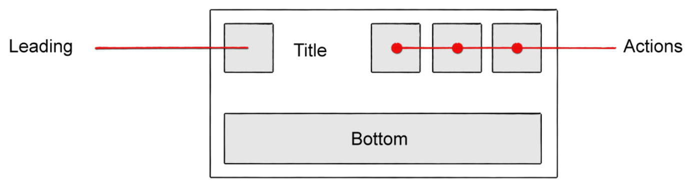
Summary

Key Points	Notes
	b. Scaffold widget i. It provides structure of your Flutter application or basic Material Design visual layout. ii. Flutter application structure:  AppBar drawer body endDrawer Floating action button Bottom tab nav *Scaffold Widget Properties (Windmill, 2020)* iii. List of available properties and static methods for Scaffold class https://api.flutter.dev/flutter/material/Scaffold-class.html *c. AppBar widget* i. It is commonly used with Scaffold property, which it is placed to the top of the screen with a certain height. It also includes navigation (back button and menu) automatically.
Summary	

Key Points

Notes

ii. AppBar structure:



AppBar Widget Properties (Windmill, 2020)

3. **Styling and Themes in Flutter**
    a. *Theme widget*
        i. It allows you to control color-related styles of theme used in your Flutter application.

        ii. To define or customize a theme for your Flutter application you can configure ThemeData class properties which is available at https://api.flutter.dev/flutter/material/ThemeData-class.html

    b. *MediaQuery and the of method*
        i. In Flutter the only measurement use to size any widget is based on the screen size in pixel. Thus, to customize the layout and spacing programmatically, a method called `MediaQuery` is commonly used with `of` method.

        ii. For example:

```
final width = MediaQuery.of(context).size.width *
0.8;
```

Summary

Key Points

Notes

4. Common Layout and UI Widgets

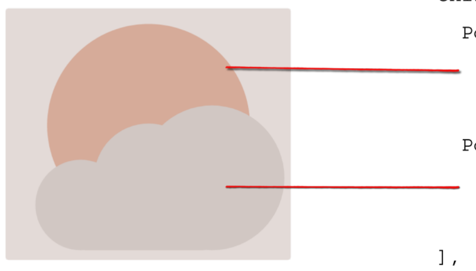
a. *Stack widget*

i. It is used to layer or stack widgets on top of each other. It treats non-positioned children similar to row and column.

ii. It treats positioned children relative to the stack's render box, using it's properties (top, left, right, bottom, width, height).

iii. Once it is laid out, Flutter will render with the first child being on the bottom of the stack.

iv. For example:



```
body: new Stack(
  children: <Widget>[
    Positioned(
      top: 100.0,
      child: Sun()),
    Positioned(
      top: 200.0,
      child: Clouds()),
  ],
),
```

Positioned Stack (Windmill, 2020)

v. Stack Class Methods and Properties
https://api.flutter.dev/flutter/widgets/Stack-class.html

b. *Table widget*

i. It is more strict than other layout widgets, which it will line out widgets in columns and rows. Each cell in the columns will have the same height and each cell in the row will have the same width.

ii. You need to define table widths and no table cell is empty.

Summary

Key Points

Notes

iii. For example:

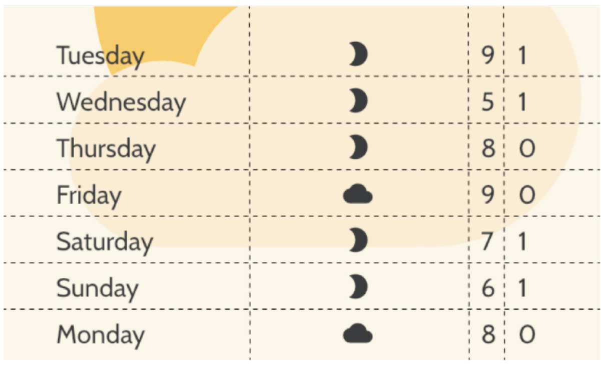
```
Table(
  columnWidths: {
    0: FixedColumnWidth(100.0),

    2: FixedColumnWidth(20.0),

    3: FixedColumnWidth(20.0),
  },
  defaultVerticalAlignment:
        TableCellVerticalAlignment.middle,

  children: <TableRow>[...],
);
```

iv. Table Layout:



Tuesday	☽	9	1
Wednesday	☽	5	1
Thursday	☽	8	0
Friday	☁	9	0
Saturday	☽	7	1
Sunday	☽	6	1
Monday	☁	8	0

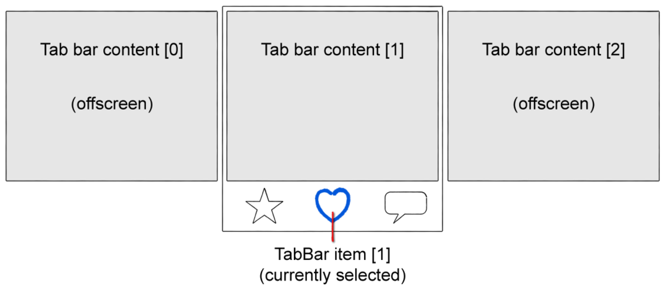
Table Diagram (Windmill, 2020)

v. Table class methods and properties
https://api.flutter.dev/flutter/widgets/Table-class.html

c. *TabBar widget*

i. Its children will be displayed in a scrollable, horizontal, view and make them tappable.

Summary

Key Points	Notes
	ii. For example:  Tab bar content [0] (offscreen) Tab bar content [1] Tab bar content [2] (offscreen) TabBar item [1] (currently selected) *Tab Related Widgets (Windmill, 2020)* iii. Tab uses a controller to manage event similar to `TextField`. In TabBar, you need to use `TabController` to select a tab and display its contents. iv. TabBar class methods and properties https://api.flutter.dev/flutter/material/TabBar-class.html **5. ListView and builders** a. It is similar to column or row, which it displays its children in a line. It is also scrollable and mostly used when the number of children is unknown or dynamic. b. By default `ListView` will build all its children once and then renders. The `ListView.builder` constructor takes a callback in the itemBuilder property and returns a widget. c. `ListView` will only render the items that are visible on the screen. For example Twitter with a long list of tweets. d. `ListView` class constructors and properties https://api.flutter.dev/flutter/widgets/ListView-class.html

Summary

Key Points	Notes
	6. References a. Windmill, E. (2020). Flutter in Action (1st Ed.). USA: Manning Publications. b. Flutter Official Documentation. Retrieved on 1 December 2020 from https://flutter.dev/docs

Summary