MOBILE APPLICATION

INTRODUCTION

CHAPTER ONE

OUTLINE

- Mobile technology
- Mobile Application
- Mobile Application Developments
- types of mobile applications
- What is flutter
- Features of Flutter
- Flutter Widgets
- Widget Build Visualization

MOBILE TECHNOLOGIES ?

is a technology that is mostly used in cellular communication and other related aspects.

- ✓ GPS navigation
- ✓ Internet browsing
- ✓ Gaming
- ✓ instant messaging tool
- ✓ etc.

MOBILE APPLICATION

- A mobile application, most commonly referred to as an app, is a type of application software designed to run on a mobile device, such as a smartphone , tablet or computer.
- A mobile application also may be known as an app, web app, online app, iPhone app or smartphone app

TYPES OF MOBILE APPS:

TYPES OF MOBILE APPS BY TECHNOLOGY

- There are three basic types of mobile apps if we categorize them by the technology used to code them:

1. **Native apps** are created for one specific platform or operating system.
2. **Web apps** are responsive versions of websites that can work on any mobile device or OS because they're delivered using a mobile browser.
3. **Hybrid apps** are combinations of both native and web apps, but wrapped within a native app, giving it the ability to have its own icon or be downloaded from an app store.

1. NATIVE APPS

- Native apps are built specifically for a mobile device's operating system (OS). Thus, you can have native Android mobile apps or native iOS apps.
- Technology Used: Native apps are coded using a variety of programming languages. Some examples include: Java, Kotlin, Python, Swift, Objective-C, C++, and React.

NATIVE APPS

TECHNOLOGY USED

Java, Kotlin, Python, Swift, Objective C, etc.

PROS

1. Faster, better performance
2. Native UI
3. Can access device features

CONS

1. Higher cost to maintain
2. Takes up space in the device
3. Updates must be downloaded

NATIVE APPS

- **Pros**: Because of their singular focus, native apps have the advantage of being faster and more reliable in terms of performance. They're generally more efficient with the device's resources than other types of mobile apps. Native apps utilize the native device UI.
- And because native apps connect with the device's hardware directly, they have access to a broad choice of device features like Bluetooth, phonebook contacts, camera roll, and more.

NATIVE APPS

- **Cons**: However, the problem with native apps lies in the fact that if you start developing them, you have to duplicate efforts for each of the different platforms. The code you create for one platform cannot be reused on another. This drives up costs. Not to mention the effort needed to maintain and update the codebase for each version.
- And then, every time there's an update to the app, the user has to download the new file and reinstall it. This also means that native apps do take up precious space in the device's storage.

2. WEB APPS

- Web apps behave similarly to native apps but are accessed via a web browser on your mobile device. They're not standalone apps in the sense of having to download and install code into your device.
- **Technology Used**: Web apps are designed using HTML5, CSS, JavaScript.

WEB APPS

TECHNOLOGY USED

HTML5, CSS, JavaScript, Ruby, etc.

PROS

1. Web-based so performs on all devices
2. Easier to maintain
3. Users don't run out of storage

CONS

1. Dependent on a browser
2. Needs an internet connection
3. May not always integrate with device hardware

WEB APPS

- **Pros**: Because it's web-based, there is no need to customize to a platform or OS. This cuts down on development costs.
- Plus, there's nothing to download. They won't take up space on your device memory like a native app, making maintenance easier – just push the update live over the web. Users don't need to download the update at the app store

WEB APPS

- **Cons**: But this is also pertinent: web apps are entirely dependent on the browser used on the device. There will be functionalities available within one browser and not available on another, possibly giving users varying experiences.
- And because they're shells for websites, they won't completely work offline. Even if they have an offline mode, the device will still need an internet connection to back up the data on your device, offer up any new data, or refresh what's on screen.

3. HYBRID APPS

- And then there are the hybrid apps. These are web apps that look and feel like native apps. They might have a home screen app icon, responsive design, fast performance, even be able to function offline, but they're really web apps made to look native.
- **Technology Used**: Hybrid apps use a mixture of web technologies and native APIs. They're developed using: Ionic, Objective C, Swift, HTML5, and others.

HYBRID APPS

TECHNOLOGY USED

Ionic, Objective C, Swift, HTML5, etc.

PROS

1. Quicker/cheaper to build
2. Load quickly
3. Less code to maintain

CONS

1. Lacks power of native apps
2. Slower since it has to download each element
3. Certain features might not be usable on devices

HYBRID APPS

- **Pros**: Building a hybrid app is much quicker and more economical than a native app. As such, a hybrid app can be the minimum viable product – a way to prove the viability of building a native app. They also load rapidly, are ideal for usage in countries with slower internet connections, and give users a consistent user experience. Finally, because they use a single code base, there is much less code to maintain.

HYBRID APPS

Cons

- Hybrid apps might lack in power and speed, which are hallmarks of native apps.
- Slower since it has to download each element
- Certain features might not be usable on devices

MOBILE APPLICATION DEVELOPMENT

- Mobile application development is the complete process of creating all of the designs, assets, and code required to implement an software application that runs on a mobile device and all of its supporting services that are accessed from the mobile application.

DEVELOPING MOBILE APPS

- The **cross-platform** development framework has the ability to write one code and can deploy on the various platform (Android, iOS, and Desktop). It saves a lot of time and development efforts of developers. There are several tools available for cross-platform development, a new framework has introduced in the cross-platform development family named **Flutter** developed from Google.

CREATE MOBILE APPLICATIONS

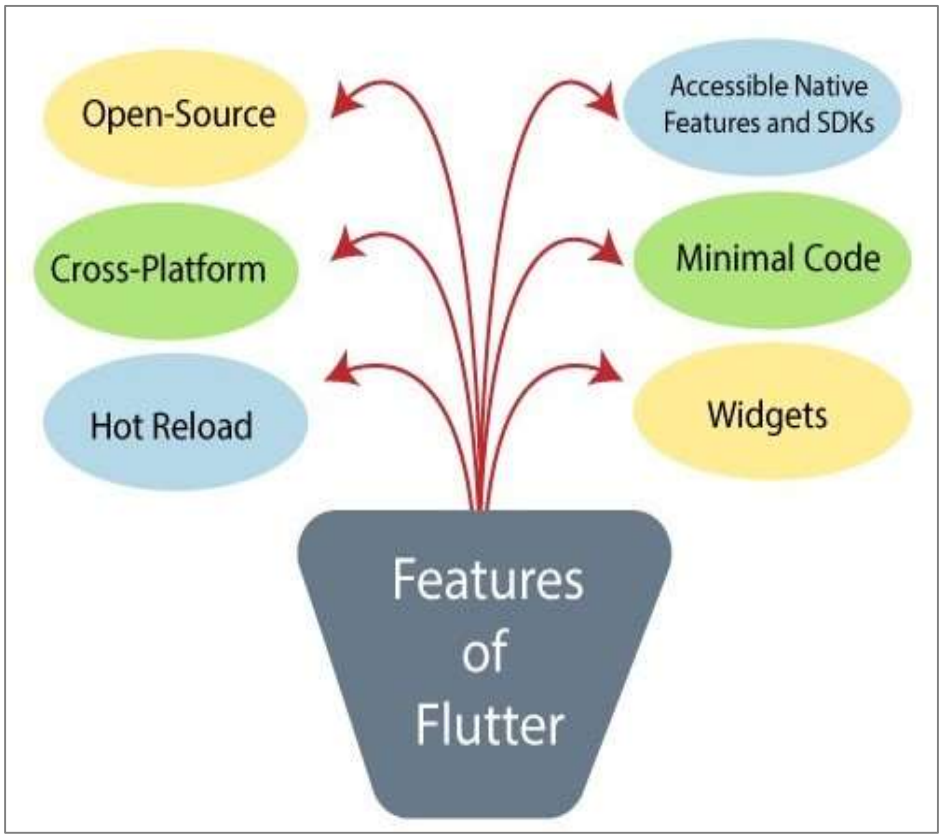
USE FLUTTER

FLUTTER

- Flutter is Google's portable UI framework for building modern, native, and reactive applications for iOS and Android.
- It is an **open-source** UI Software Development Kit (SDK) created by Google
- Flutter uses **Dart**, a modern object-oriented language
- Flutter also offers many ready to use **widgets** (UI) to create a modern application. These widgets are optimized for mobile environment and designing the application using widgets is as simple as designing HTML.

FEATURES OF FLUTTER

Flutter gives easy and simple methods to start building beautiful mobile and Desktop apps with a rich set of material design and widgets. Here, we are going to discuss its main features for developing the mobile framework.



FEATURES OF FLUTTER

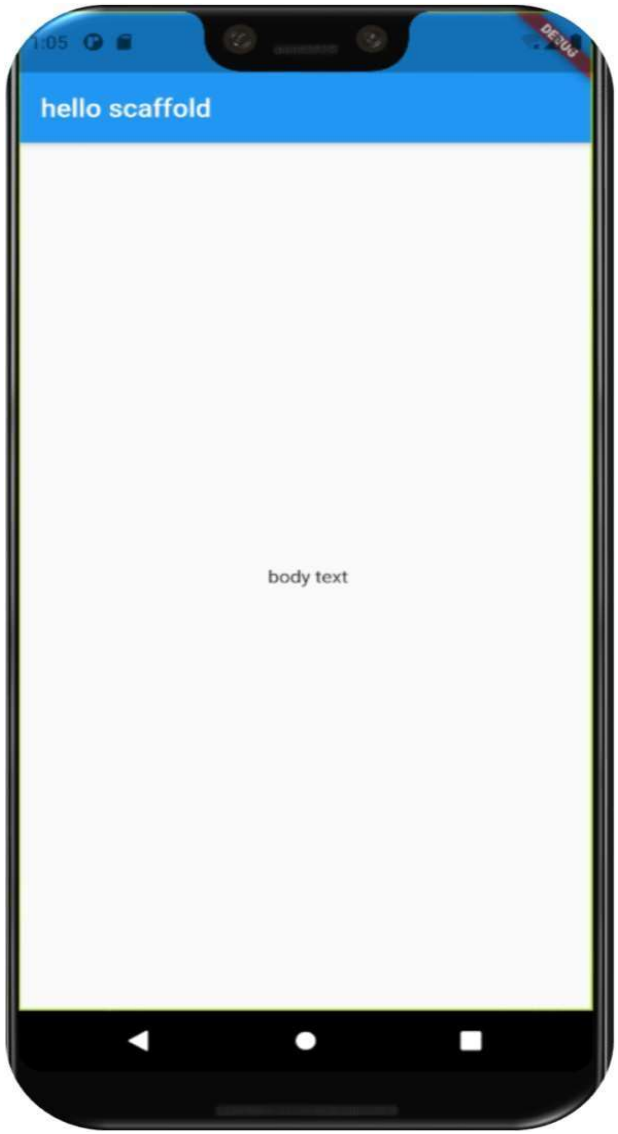
- **Open-Source:** Flutter is a free and open-source framework for developing mobile applications.
- **Cross-platform:** This feature allows Flutter to write the code once, maintain, and can run on different .
- **Hot Reload:** Whenever the developer makes changes in the code, then these changes can be seen immediately with Hot Reload.
- **Accessible Native Features and SDKs:** This feature allows can easily access the SDKs on both platforms.
- **Minimal code:** Flutter app is developed by Dart programming language, which uses JIT and AOT compilation to improve the overall start-up time.
- **Widgets:** The Flutter framework offers widgets, which are capable of developing customizable specific designs. Most importantly, Flutter has two sets of widgets: **Material** Design and **Cupertino** widgets

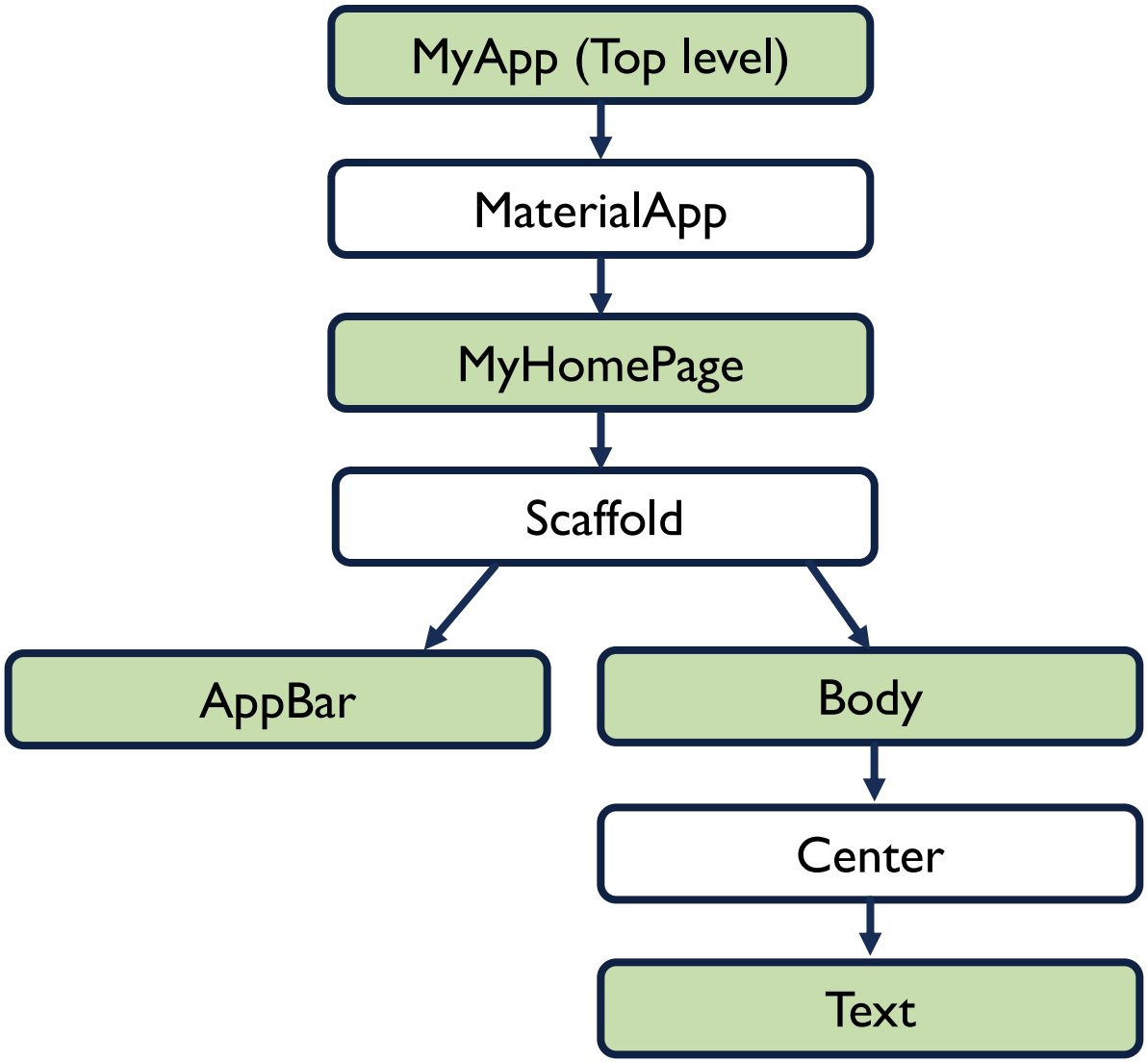
FLUTTER WIDGETS

- **In Flutter, Everything is a widget**. Flutter Widget is basically a user interface component that is used to create interface of the app. It represents an immutable part of the user interface that can be any graphics, text, shapes, and animations are created using widgets.

- In Flutter, the application is itself a widget that contains many sub widgets. It means the app is the top-level widget, and its UI is build using one or more children widgets, which again includes sub child widgets. This feature helps you to create a complex user interface very easily.

FLUTTER WIDGET



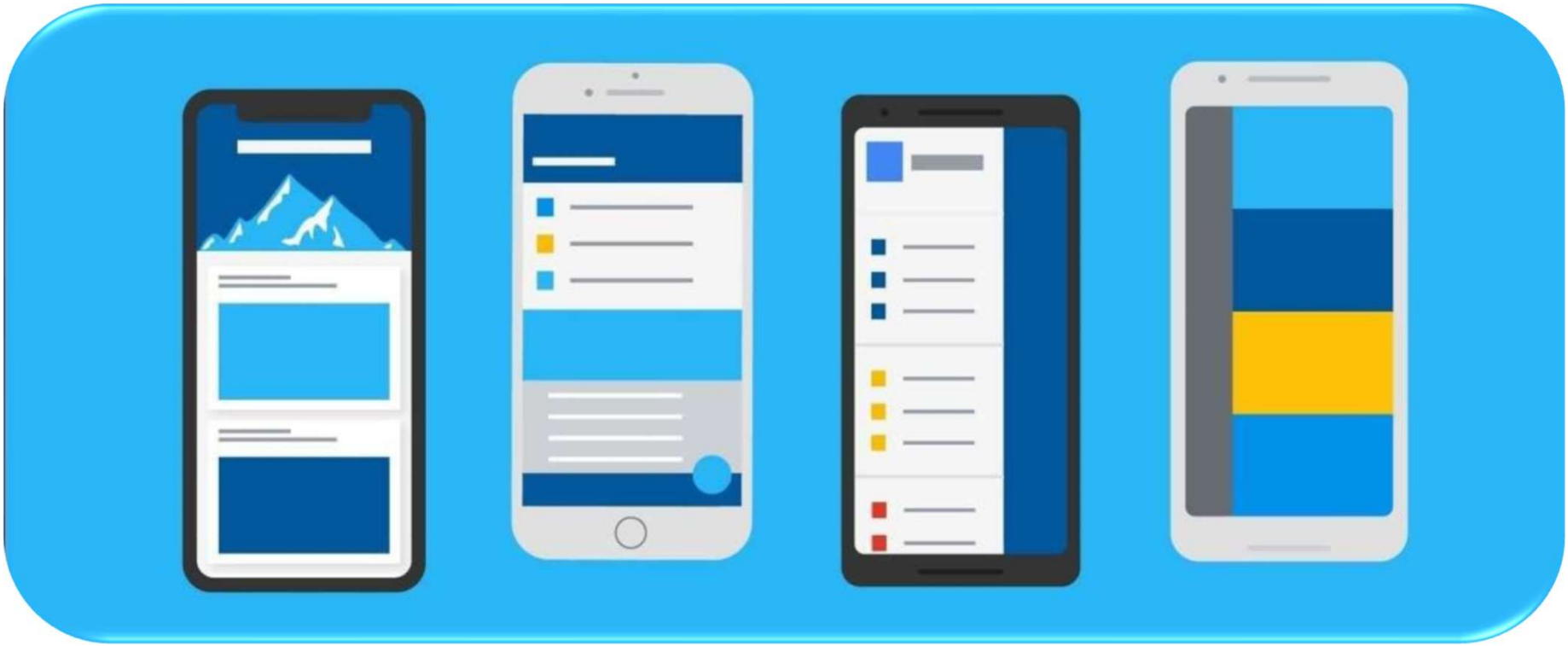


FLUTTER WIDGET

- *MyApp* is the user created widget and it is build using the Flutter native widget, *MaterialApp*.
- *MaterialApp* has a home property to specify the user interface of the home page, which is again a user created widget, *MyHomePage*.
- *MyHomePage* is build using another flutter native widget, *Scaffold*
- *Scaffold* has two properties – *body* and *appBar*
- *body* is used to specify its main user interface and *appBar* is used to specify its header user interface
- *Header UI* is build using flutter native widget, *AppBar* and *Body UI* is build using *Center* widget.
- The *Center* widget has a property, *Child*, which refers the actual content and it is build using *Text* widget

FLUTTER WIDGET

- To be specific, Flutter application is itself a widget. Flutter widgets also supports animations and gestures.



FLUTTER WIDGET

- ✓ Widgets with structuring elements such as a list, grid, text, and button
- ✓ Widgets with input elements such as a form, form fields, and keyboard listeners
- ✓ Widgets with styling elements such as font type, size, weight, color, border, and shadow
- ✓ Widgets to lay out the UI such as row, column, stack, centering, and padding
- ✓ Widgets with interactive elements that respond to touch, gestures, dragging, and dismissible
- ✓ Widgets with animation and motion elements such as hero animation, animated container, animated crossfade, fade transition, rotation, scale, size, slide, and opacity
- ✓ Widgets with elements like assets, images, and icons
- ✓ Widgets that can be nested together to create the UI needed

WIDGET BUILD VISUALIZATION

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

1) Platform specific widgets
2) Layout widgets
3) State maintenance widgets
4) Platform independent / basic widgets

1) 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.**

1) PLATFORM SPECIFIC WIDGETS

material widgets

BottomNavigationBar

TabBar

TabBarView

ListTile

RaisedButton

FloatingActionButton

FlatButton IconButton

DropdownButton

***Cupertino* widgets**

CupertinoButton

CupertinoDatePicker

CupertinoTimerPicker

CupertinoNavigationBar

CupertinoTabBar

CupertinoTabScaffold

CupertinoTabView

CupertinoTextField

2) 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.

2) LAYOUT WIDGETS

- **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.

3) STATE MAINTENANCE WIDGETS

- To build the UI, you use two main types of widgets, ***StatelessWidget*** and ***StatefulWidget***. A stateless widget is used when the values (state) do not change, and the stateful widget is used when values (state) change.

4) 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.

✓ **Text**

✓ **Image**

✓ **Icon**

✓ **Button**

✓ **Etc..**

ANY QUESTION ?