Week 8 to 10

14 Dec 2025

MOBILE APPLICATIONS IT 319

Modern User Interface Concepts in Flutter

Week 8–10 | IT 319 | Practical UI & Interaction

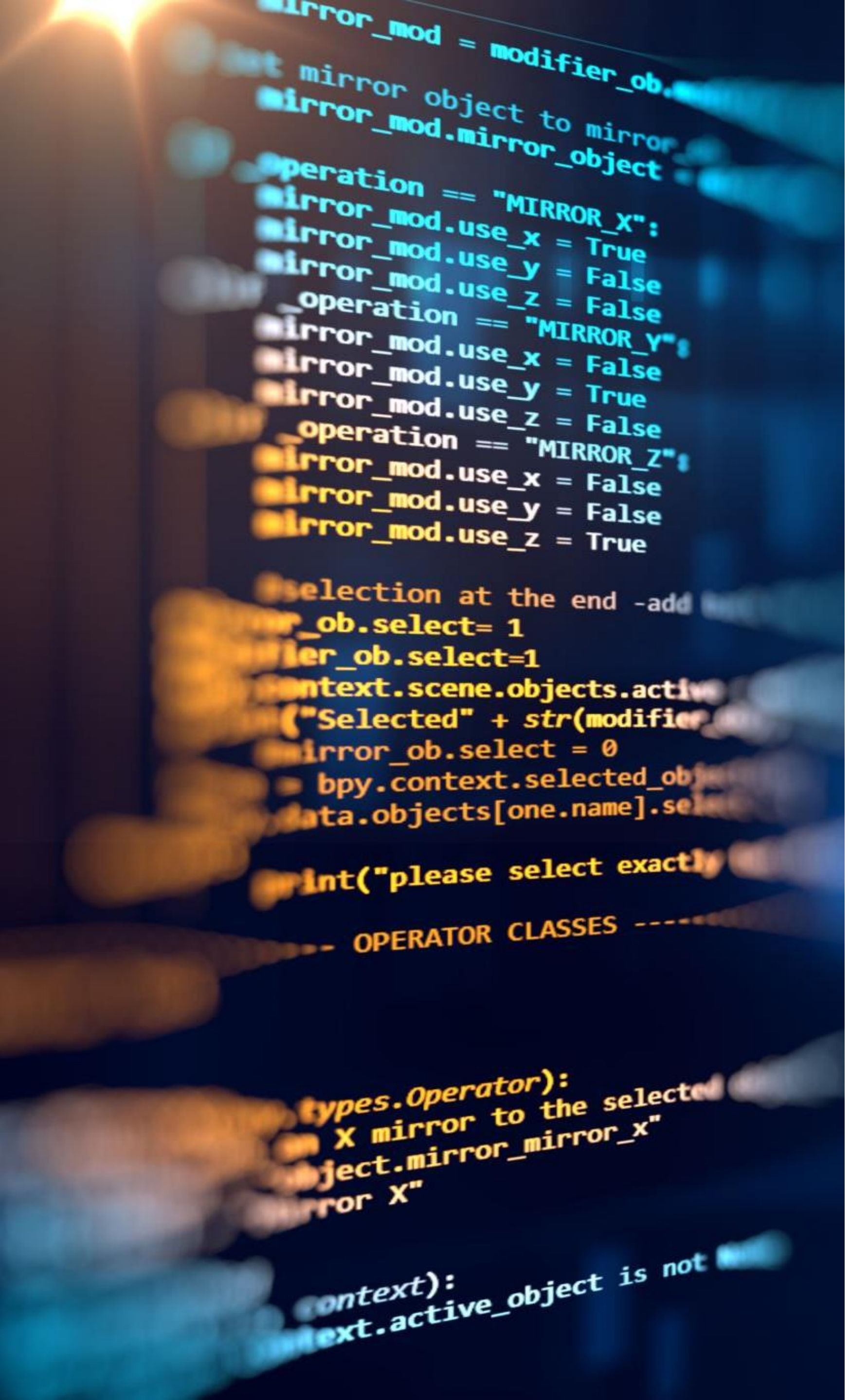
IT DEPT.

TIU

3RD GRADE

Lect. Mohammad Salim

Thses slide notes are based on many different online resources such as Flutter.dev, Flutter Apprentice Book and The complete Fluter bootcamp course

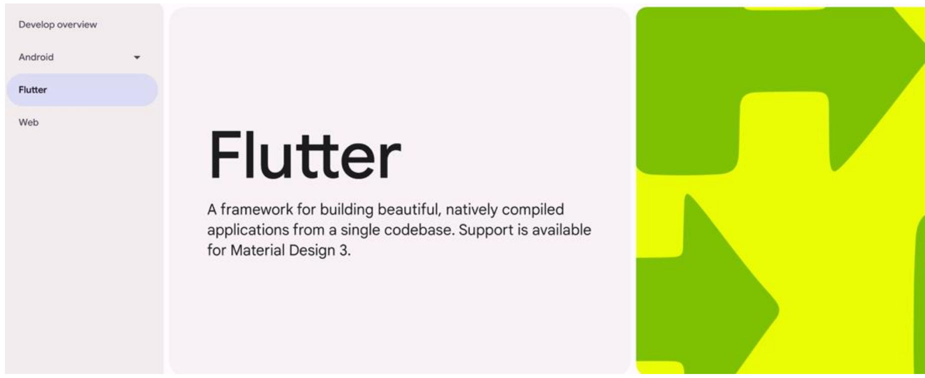


LECTURE OBJECTIVES

- Explain what Material Design and Material 3 are
- Describe how Flutter handles user input
- Distinguish between gestures and form input
- Explain responsive UI concepts in Flutter
- Understand the role of packages and pubspec.yaml

MATERIAL DESIGN

material.io/develop/flutter



Flutter

A framework for building beautiful, natively compiled applications from a single codebase. Support is available for Material Design 3.

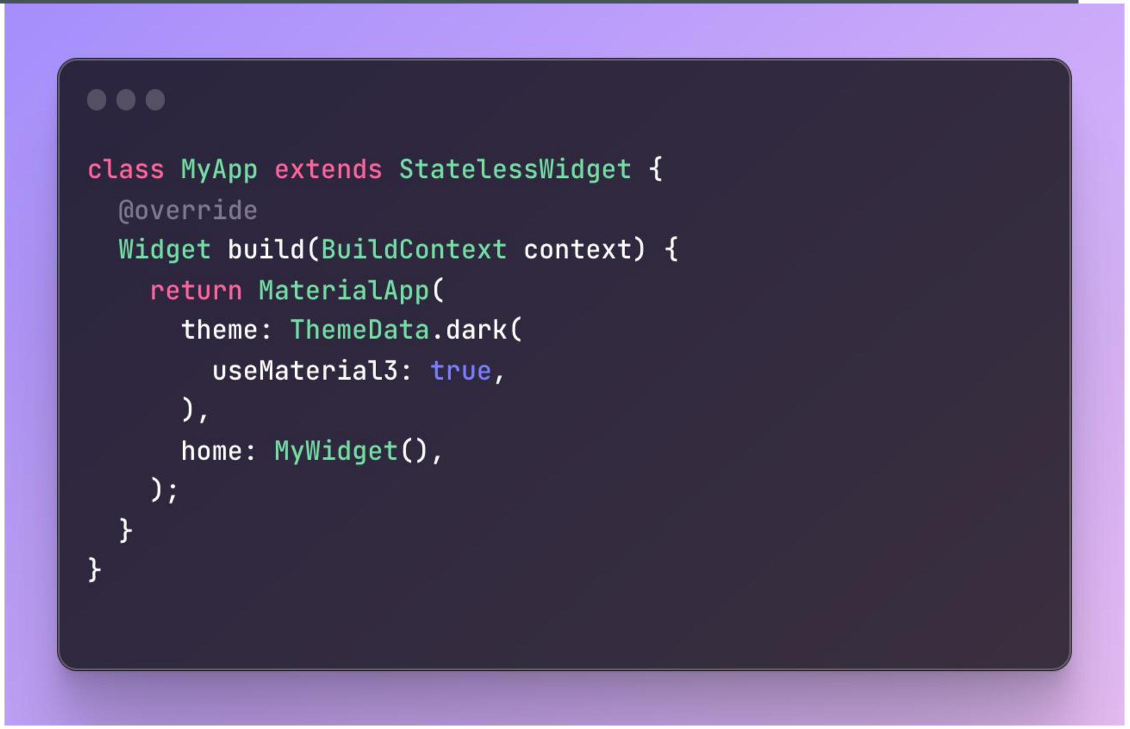
- **Material Design**
  Material Design is a design system developed by Google that provides **standard rules for colors, typography, shapes, and components**.
  It helps developers build applications that are:
- Visually consistent
- Easy to use
- Familiar across platforms
- Material Design focuses on **clarity, consistency, and usability**.

MATERIAL FLUTTER

Material Design in Flutter
Flutter provides built-in support for Material Design through:

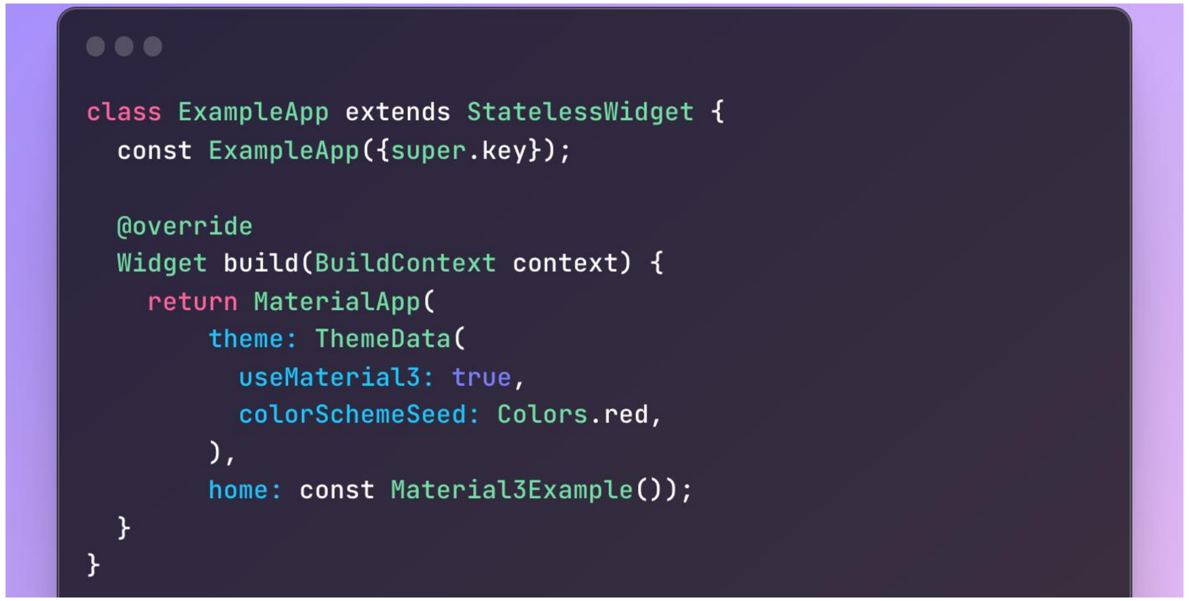
- Predefined widgets
- Default themes
- Platform-adaptive behavior

This allows developers to focus on **functionality**, while Flutter handles **visual consistency**.

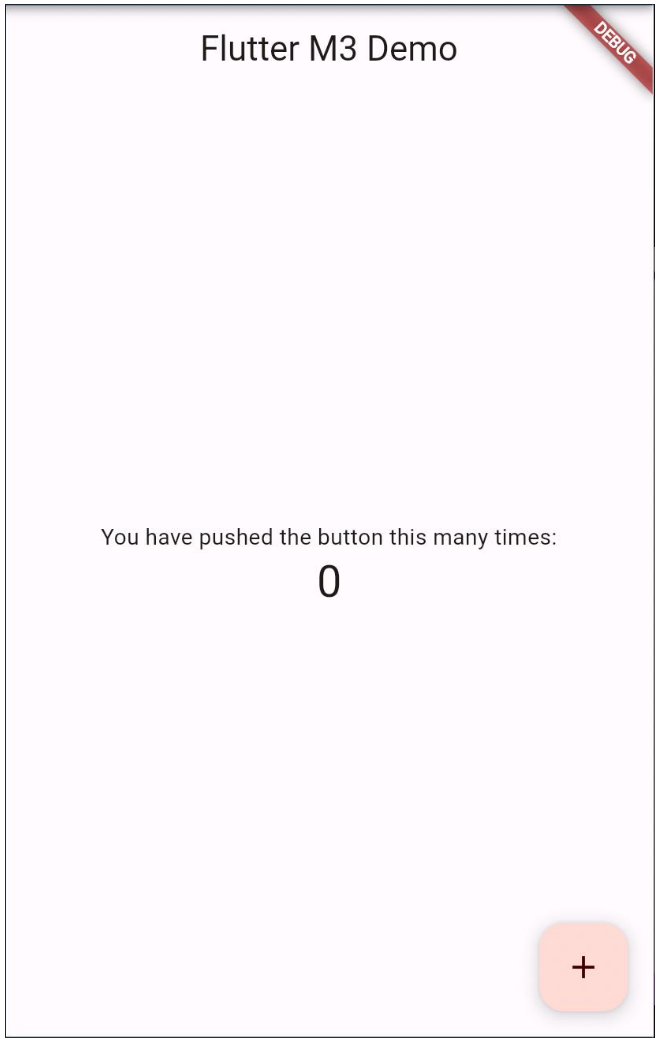


```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData.dark(
        useMaterial3: true,
      ),
      home: MyWidget(),
    );
  }
}
```

MATERIAL COMPONENTS (FLUTTER)



```
class ExampleApp extends StatelessWidget {
  const ExampleApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        theme: ThemeData(
          useMaterial3: true,
          colorSchemeSeed: Colors.red,
        ),
        home: const Material3Example());
  }
}
```



Flutter M3 Demo

You have pushed the button this many times:

0

Material 3 aims to make applications look **more modern and adaptive** while reducing manual styling.

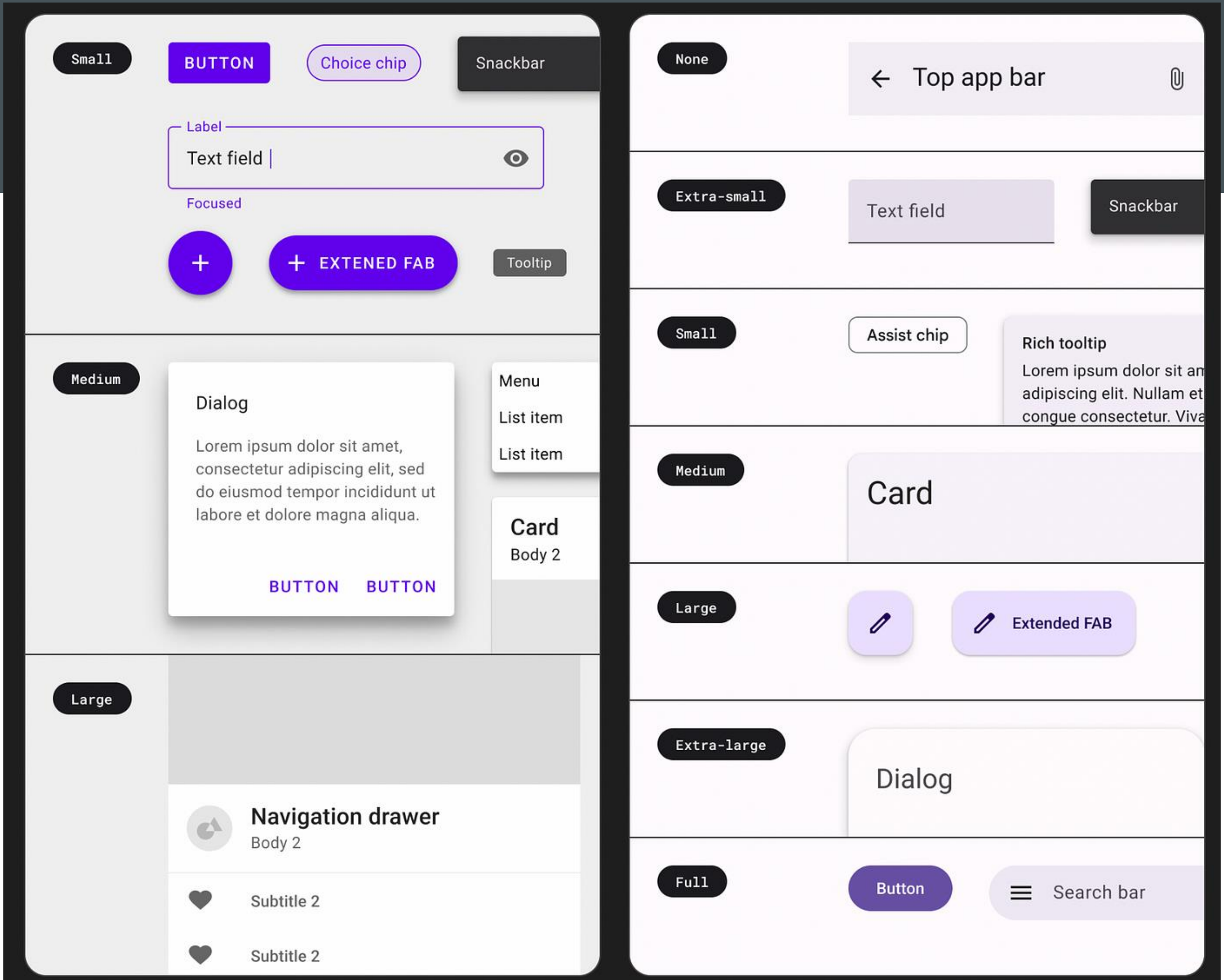
Material 3 is enabled by default in Flutter (Flutter 3.16+).
You can still set `useMaterial3: true` explicitly for clarity, or `useMaterial3: false` to temporarily use Material 2.

MATERIAL DESIGN 3(FLUTTER)

Material Design 3 is the latest version of Material Design.

It introduces:

- **More rounded shapes**
- **Updated typography**
- **Improved color handling**
- **Modernized UI components**



M2: Three-level shape scale based on the size of the component container.

M3: Seven-level shape scale based on the roundedness of shape corners.

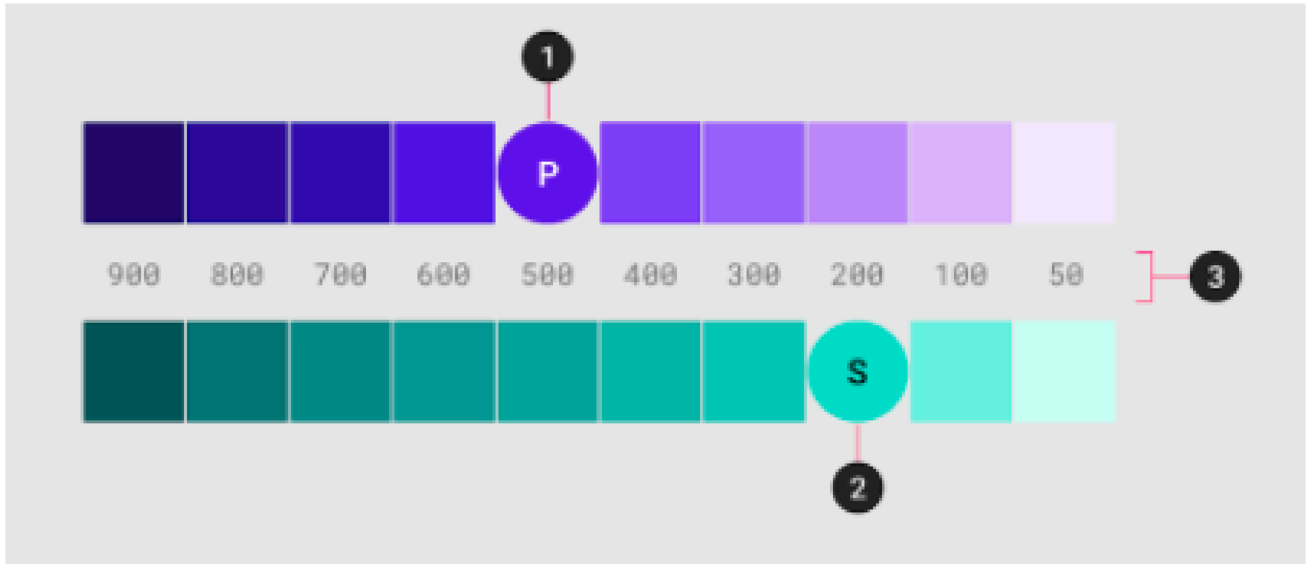
THEMES AND COLORS IN FLUTTER

A theme defines the **overall visual appearance** of an application.
It controls:

- Primary colors
- Secondary colors
- Background colors
- Text and icon colors

Using themes ensures:

- Consistency across screens
- Better readability
- Easier maintenance

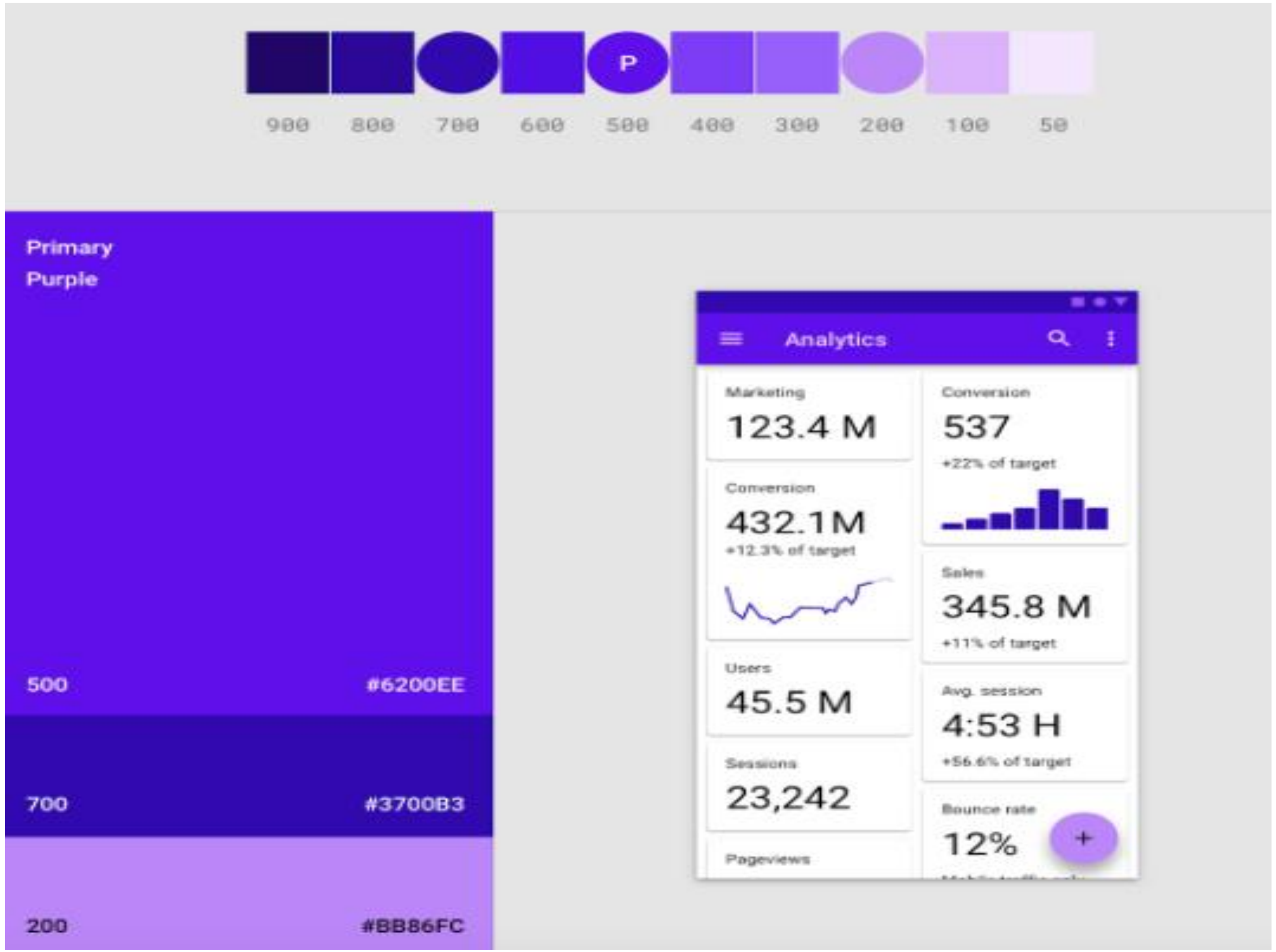


A sample primary and secondary palette

1. Primary color
2. Secondary color
3. Light and dark variants

PRIMARY COLOR

- A top app bar uses light and dark primary color variants to distinguish it from a system bar.
- This UI uses a primary color and two primary variants.

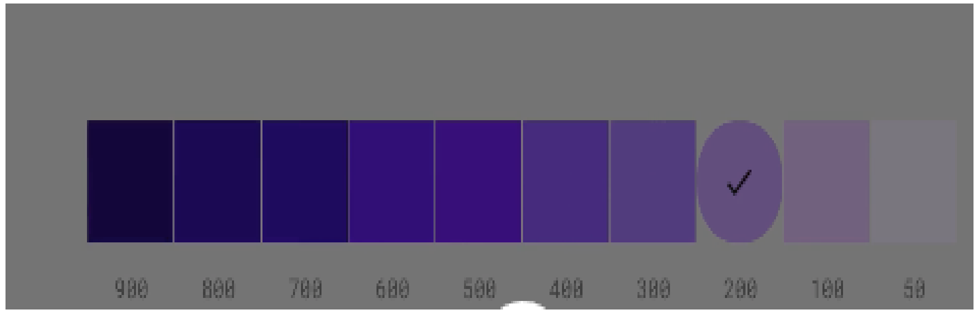


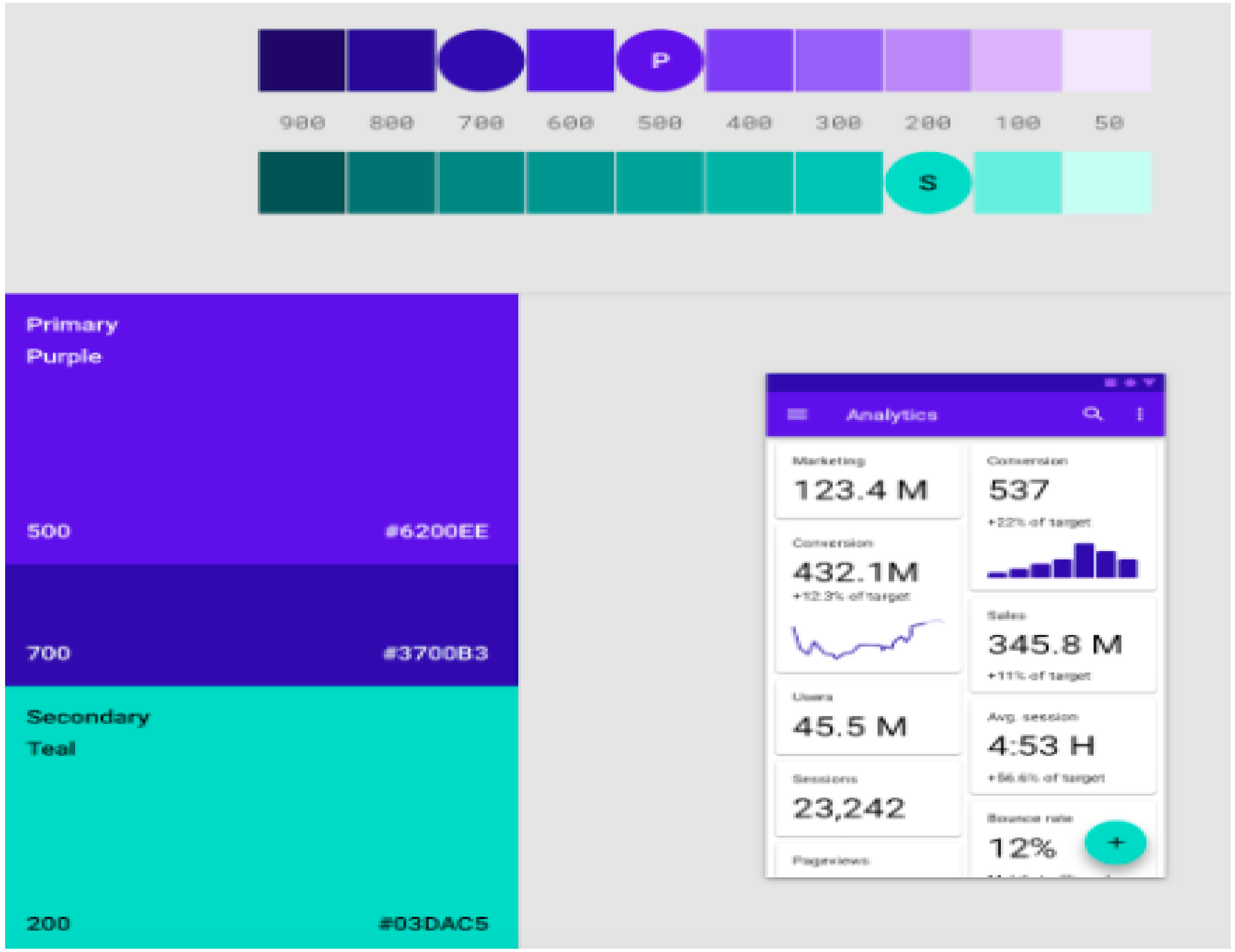
SECONDARY COLOR

- Dark and light variants of primary and secondary colors
- This UI uses a color theme with a primary color, a primary variant, and a secondary color.

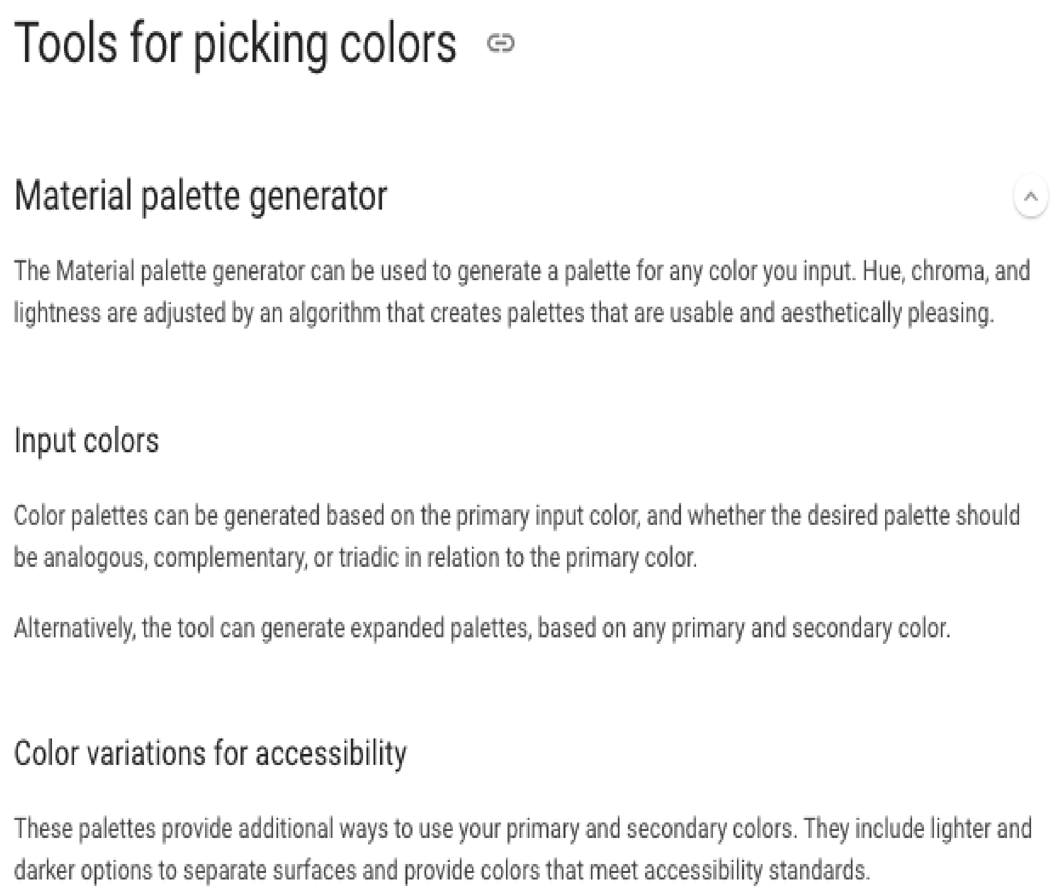
Color swatches

A **swatch** is a sample of a color chosen from a range of similar colors.





TOOLS FOR PICKING COLORS



Tools for picking colors

Material palette generator

The Material palette generator can be used to generate a palette for any color you input. Hue, chroma, and lightness are adjusted by an algorithm that creates palettes that are usable and aesthetically pleasing.

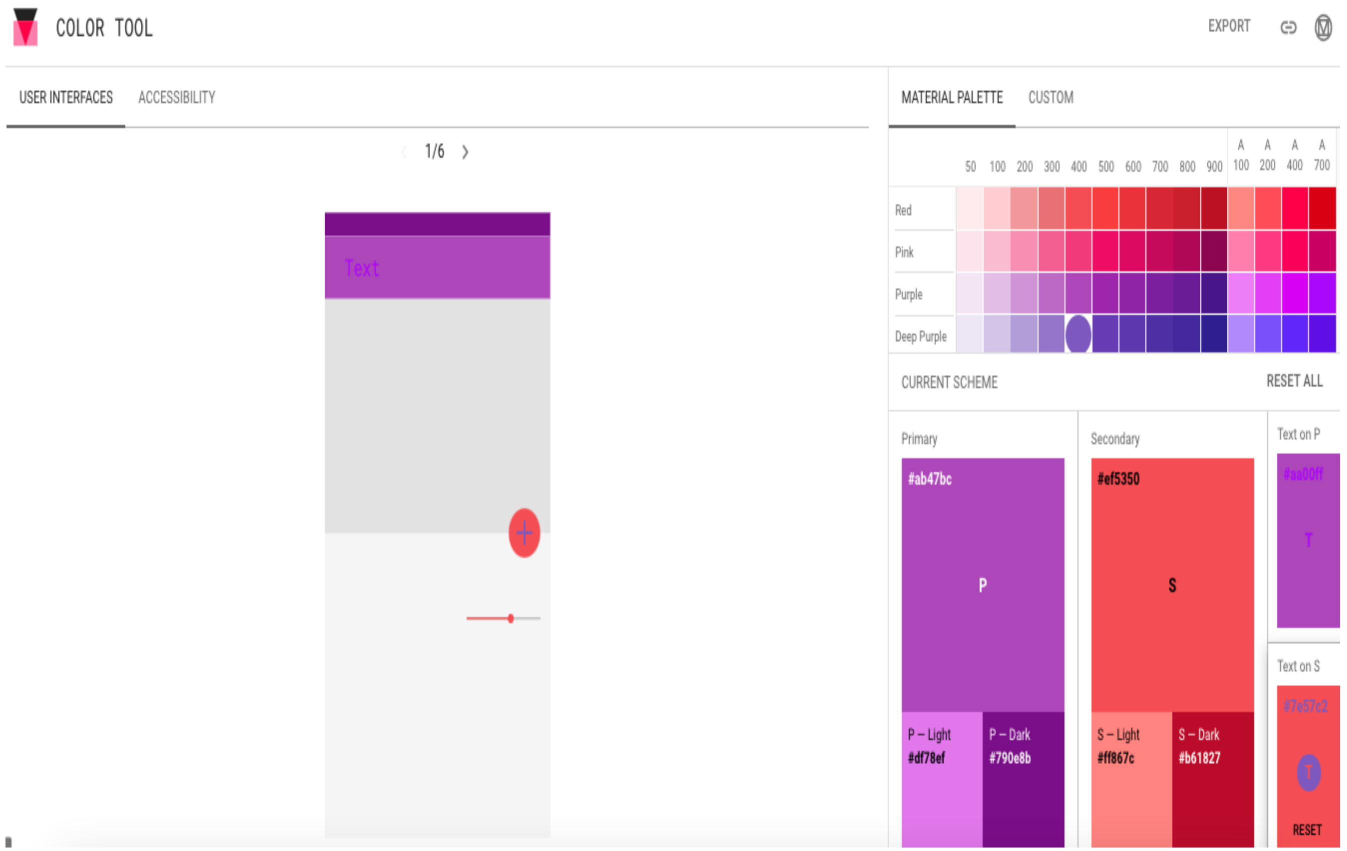
Input colors

Color palettes can be generated based on the primary input color, and whether the desired palette should be analogous, complementary, or triadic in relation to the primary color.

Alternatively, the tool can generate expanded palettes, based on any primary and secondary color.

Color variations for accessibility

These palettes provide additional ways to use your primary and secondary colors. They include lighter and darker options to separate surfaces and provide colors that meet accessibility standards.



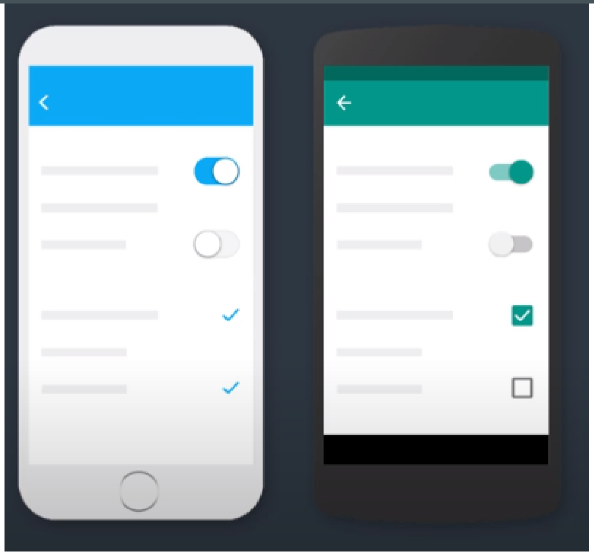
Primary #ab47bc P
P – Light #df78ef
P – Dark #790e8b
Secondary #ef5350 S
S – Light #ff867c
S – Dark #b61827
Text on P #aa00ff
Text on S #7e57c2

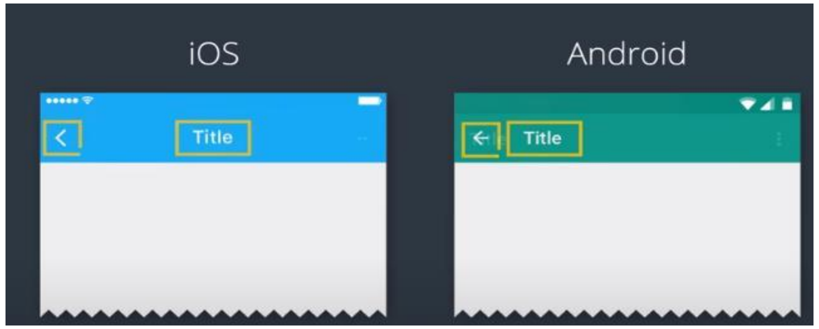
https://material.io/resources/color/

BUILDING PLATFORM SPECIFIC UI (IOS & ANDROID

- With Flutter, we are able to design apps that look native to both **iOS** and **Android** using a single code base.
- For example, an **appBar** will render differently on iOS and Android. The title text, position and font are appropriate to the platform as is the back navigation button.
- In Flutter, you can import **dart:io** and use **Platform property** to look up which platform you are currently running on. The API is quite nice:

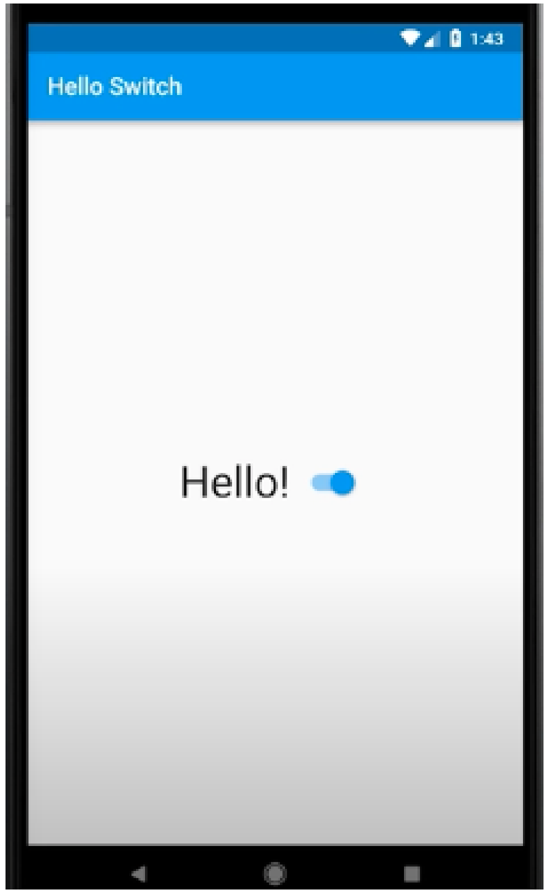
```
import 'dart:io';


Platform.isIOS // Returns true on iOS devices
Platform.isAndroid // Returns true on Android devices
```

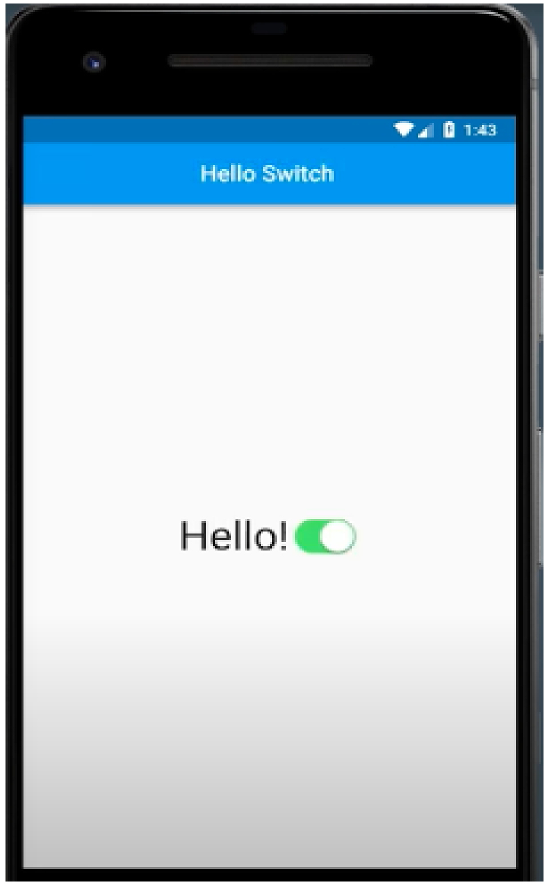




BUILDING PLATFORM SPECIFIC UI (IOS & ANDROID

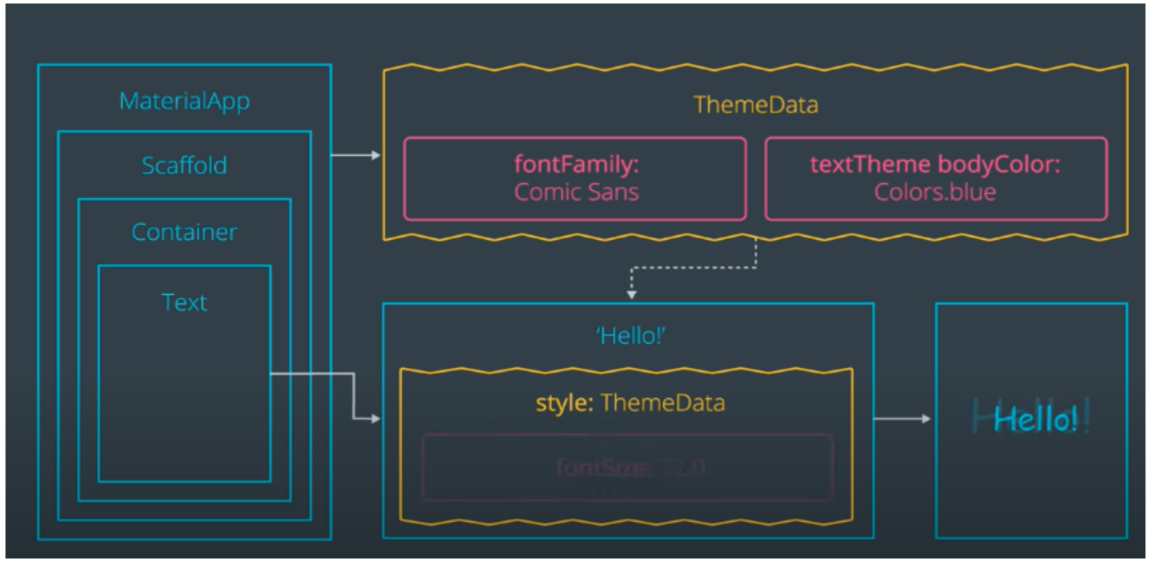
- To incorporate specific native widgets, we can use the themes platform property to determine the platform and build a relevant widgets accordingly in either the **Material** or **Cupertino** style.
- Here is a **Material switch** when we toggle between device platforms via the Flutter inspector, the switch style does not change.
- To use **Cupertino** widgets, we import the **Cupertino.dart** package.
- This allows us to use the **Cupertino switch**.
- We will check if the platform is **iOS**, and if it is, we show the **CupertinoSwitch**. Now, when we toggle between devices, the iOS version shows the default **iOS** toggle.

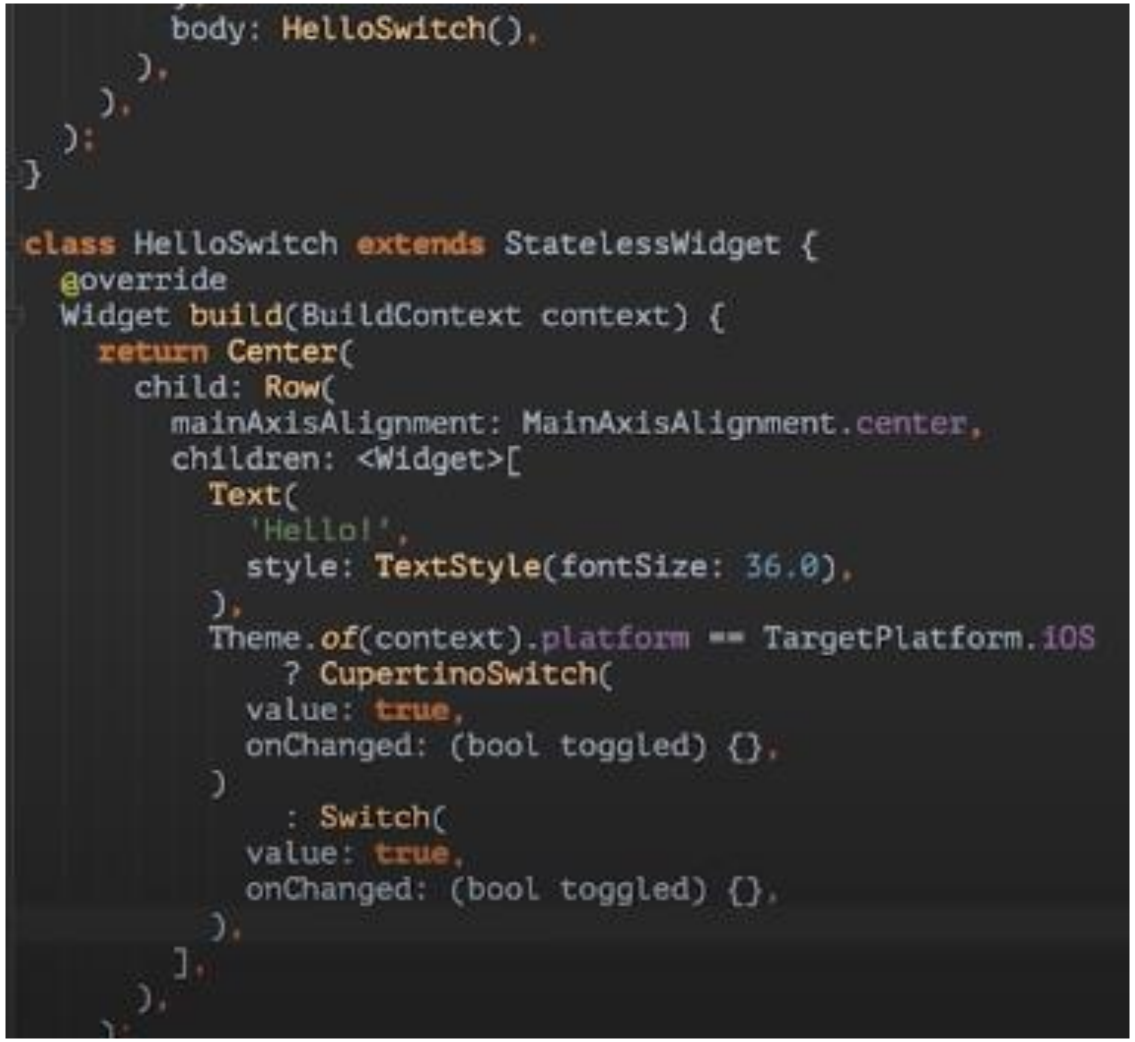




BUILDING PLATFORM SPECIFIC UI (IOS & ANDROID

- You can customize the master theme and its properties.
- This master theme is propagated down the widget tree. Child widgets are able to inherit the master theme's styling.
- The child widget can also override the theme and customize their styling.





```
        body: HelloSwitch(),
      ),
    ),
  );
}

class HelloSwitch extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Text(
            'Hello!',
            style: TextStyle(fontSize: 36.0),
          ),
          Theme.of(context).platform == TargetPlatform.iOS
              ? CupertinoSwitch(
            value: true,
            onChanged: (bool toggled) {},
          )
              : Switch(
            value: true,
            onChanged: (bool toggled) {},
          ),
        ],
      ),
```

USER INPUT IN MOBILE APPLICATIONS

User input allows users to interact with an application.
Common types of input include:

- Text input (forms, search fields)
- Button interactions
- Touch gestures
- Handling user input correctly is essential for **interactive and responsive applications**.

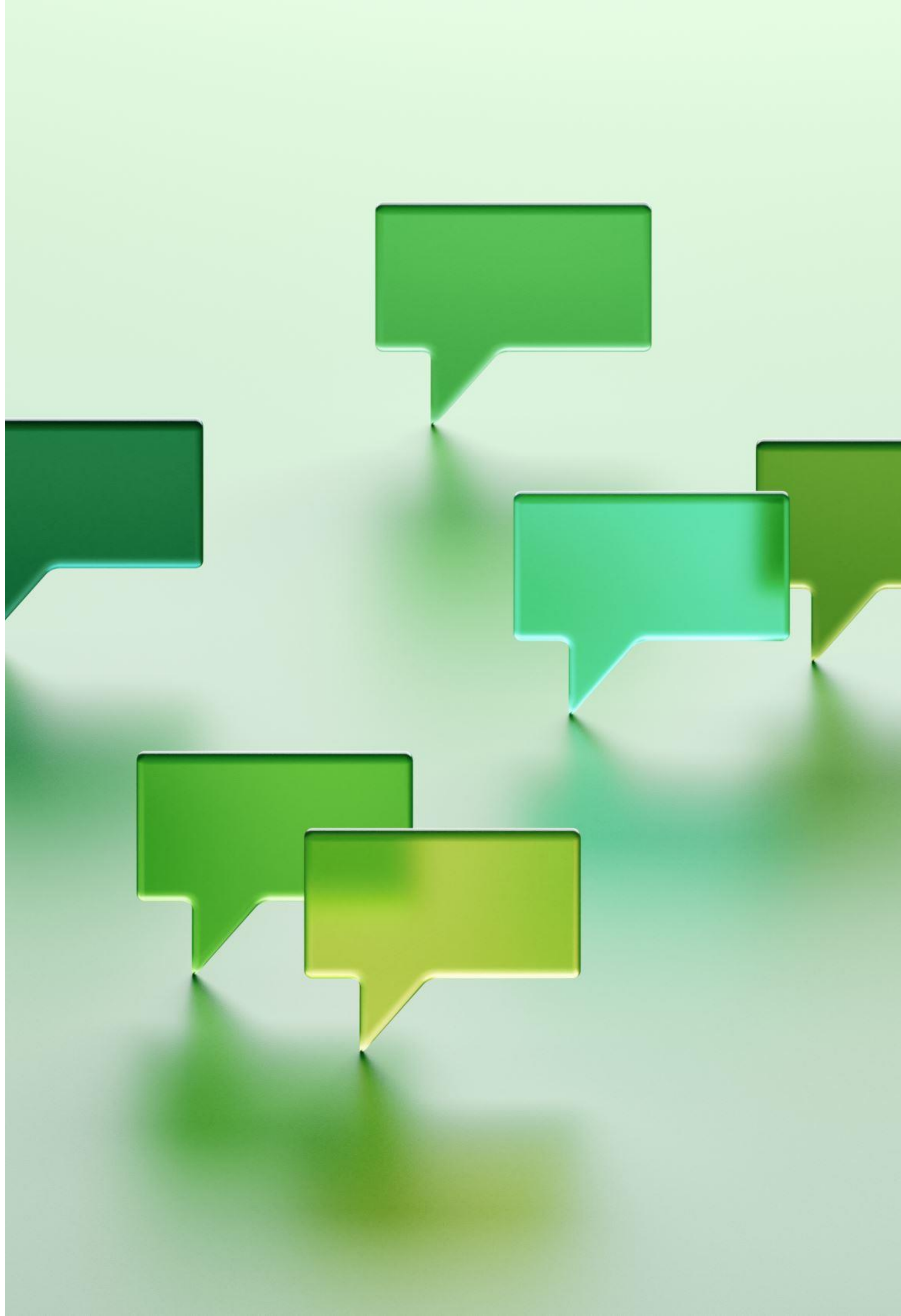
TEXT INPUT CONCEPTS

Text Input in Flutter
Flutter provides widgets for text input that allow users to:

- Enter text
- Submit forms
- Interact with application data

Text input is commonly used in:

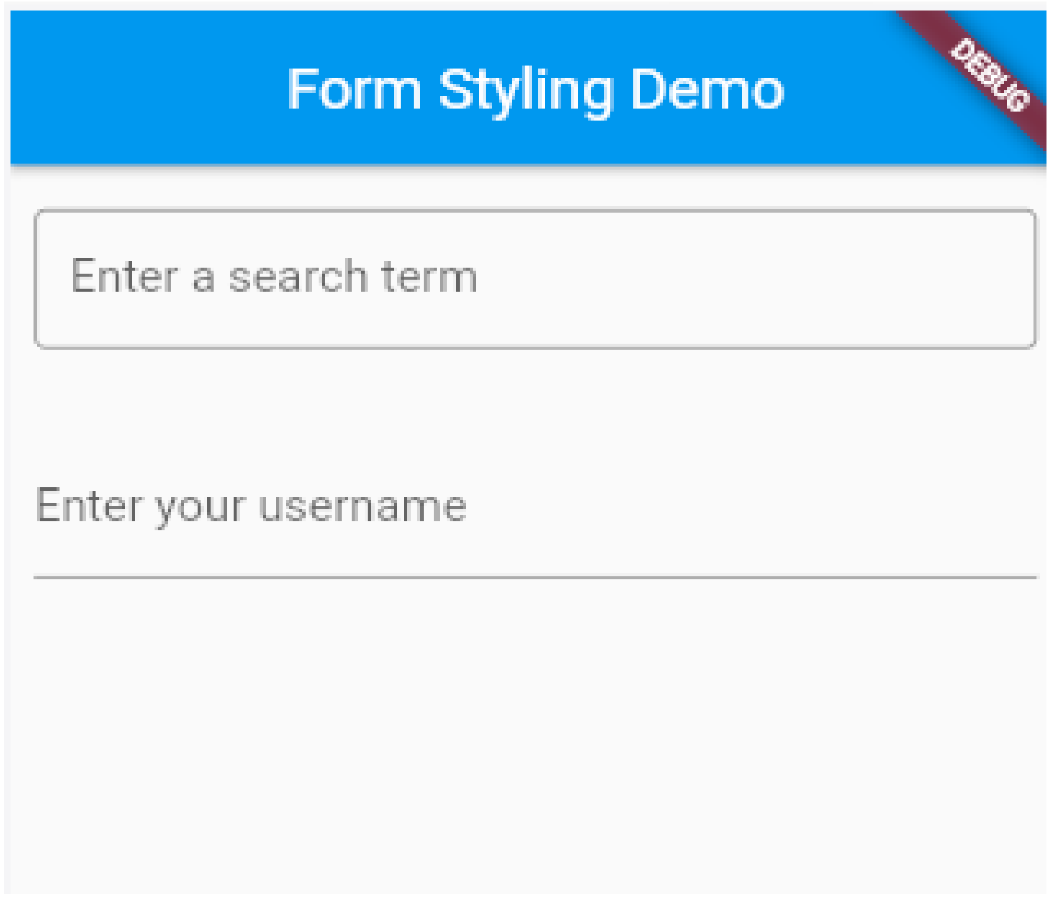
- Login screens
- Registration forms
- Search functionality



TEXTFIELD VS TEXTFORMFIELD

Key Idea:
TextFormField supports **structured input and validation**, making it suitable for forms.

Widget	Description
TextField	Used for simple text input
TextFormField	Used inside forms with validation



HANDLING USER ACTIONS

When users interact with an app, Flutter detects:

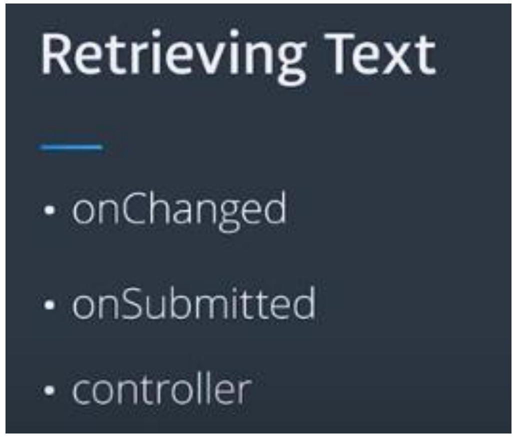
- Button presses
- Text changes
- Form submissions

These actions allow the application to:

- Update data
- Show feedback
- Navigate between screens

How do you run a callback function every time the text changes? With Flutter, you have two options:

1. Supply an **onSubmitted or onChanged**() callback to a **TextField** or a **TextFormField**.
2. Use a **TextEditingController**.



Retrieving Text

- onChanged
- onSubmitted
- controller

HANDLE CHANGES TO A TEXT FIELD

2. Use a `TextEditingController`

A more powerful, but more elaborate approach, is to supply a `TextEditingController` as the `controller` property of the `TextField` or a `TextFormField`.

To be notified when the text changes, listen to the controller using the `addListener()` method using the following steps:

1. Create a `TextEditingController`.
2. Connect the `TextEditingController` to a text field.
3. Create a function to print the latest value.
4. Listen to the controller for changes.

```
class _MyCustomFormState extends State<MyCustomForm> {
  // Create a text controller. Later, use it to retrieve the
  // current value of the TextField.
  final myController = TextEditingController();
```

Connect the `TextEditingController` to a text field

Supply the `TextEditingController` to either a `TextField` or a `TextFormField`. Once you wire these two classes together, you can begin listening for changes to the text field.

```
TextField(
  controller: myController,
),
```

```
@override
void initState() {
  super.initState();

  // Start listening to changes.
  myController.addListener(_printLatestValue);
}
```

Create a function to print the latest value

You need a function to run every time the text changes. Create a method in the `_MyCustomFormState` class that prints out the current value of the text field.

```
void _printLatestValue() {
  print('Second text field: ${myController.text}');
}
```

GESTURES

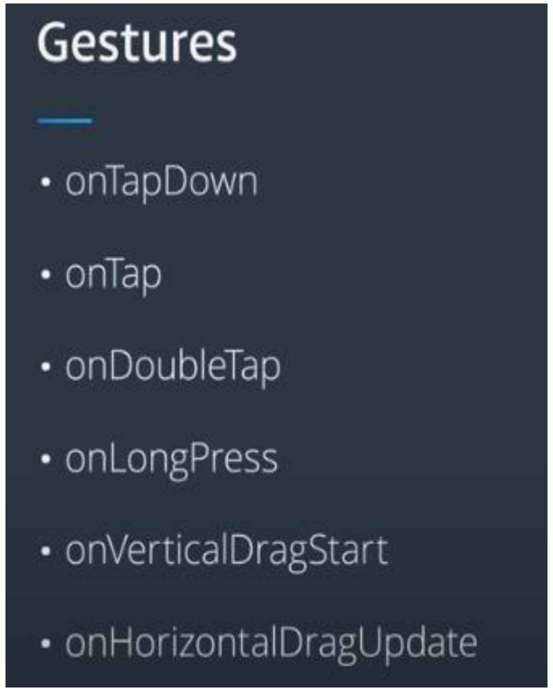
Gestures are touch-based interactions such as:

- Tapping
- Pressing
- Touching UI elements

Flutter supports gestures to make applications:

- More interactive
- More intuitive
- More responsive

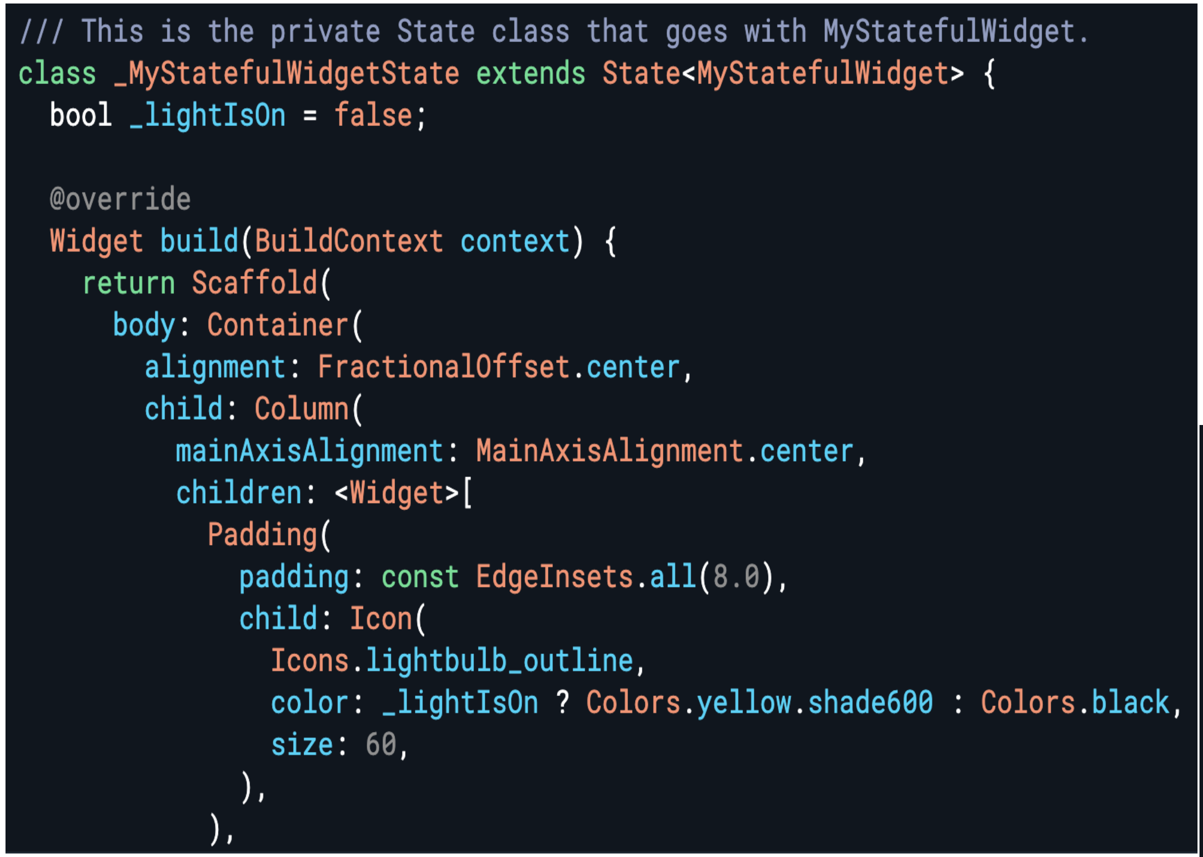
Use Material 3's **InkRipple** and **InkResponse** for modern touch feedback effects.



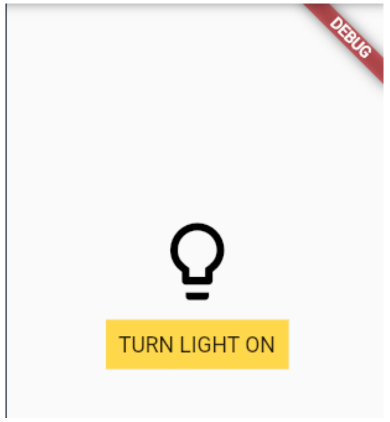
Gestures

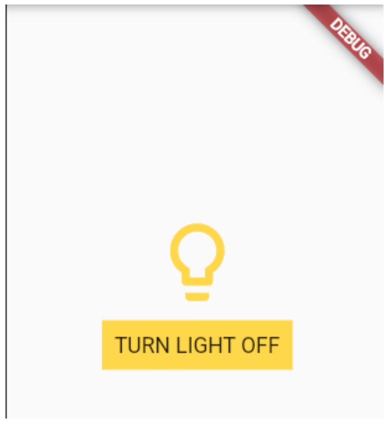
- onTapDown
- onTap
- onDoubleTap
- onLongPress
- onVerticalDragStart
- onHorizontalDragUpdate

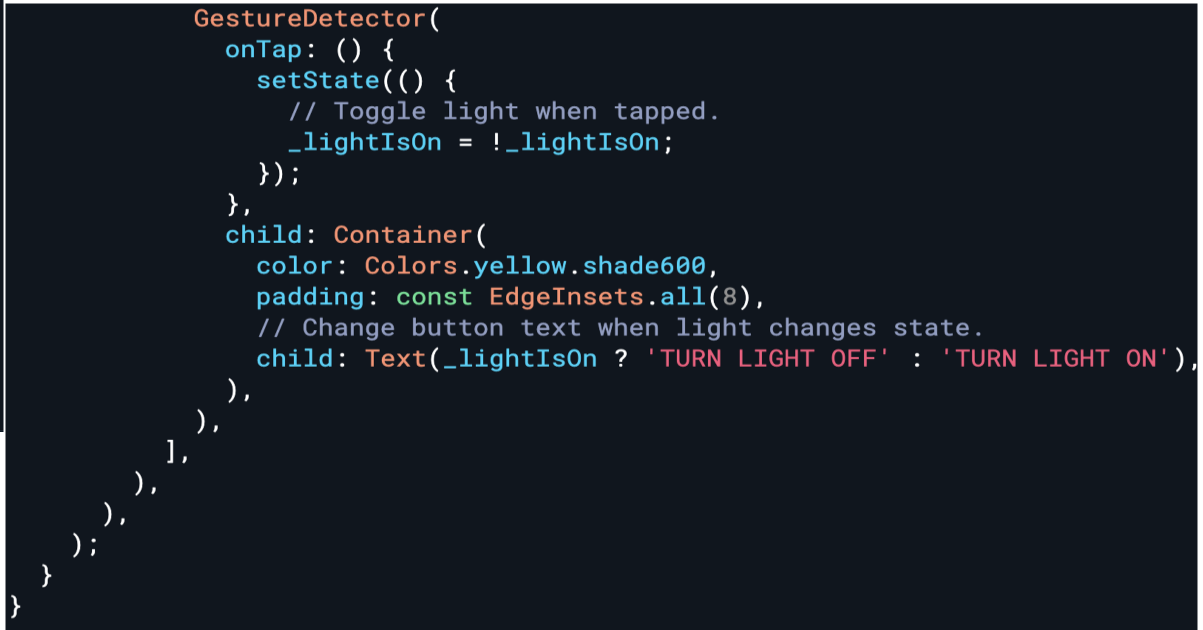
EXAMPLE



```
/// This is the private State class that goes with MyStatefulWidget.
class _MyStatefulWidgetState extends State<MyStatefulWidget> {
  bool _lightIsOn = false;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        alignment: FractionalOffset.center,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Padding(
              padding: const EdgeInsets.all(8.0),
              child: Icon(
                Icons.lightbulb_outline,
                color: _lightIsOn ? Colors.yellow.shade600 : Colors.black,
                size: 60,
              ),
            ),
```







```
            GestureDetector(
              onTap: () {
                setState(() {
                  // Toggle light when tapped.
                  _lightIsOn = !_lightIsOn;
                });
              },
              child: Container(
                color: Colors.yellow.shade600,
                padding: const EdgeInsets.all(8),
                // Change button text when light changes state.
                child: Text(_lightIsOn ? 'TURN LIGHT OFF' : 'TURN LIGHT ON'),
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

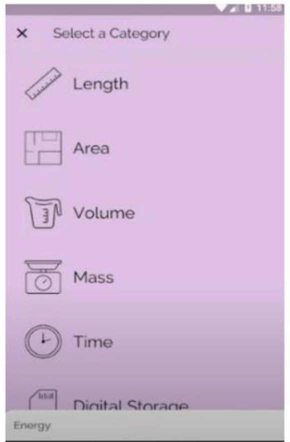
RESPONSIVE DESIGN

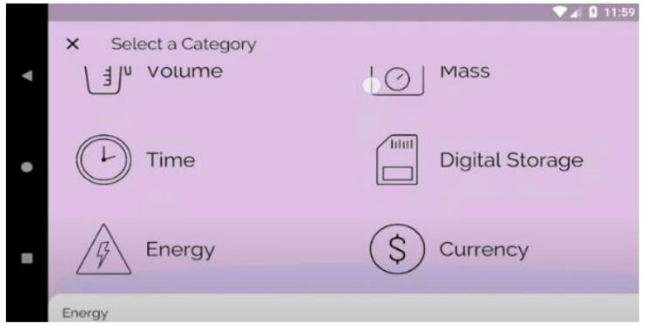
Responsive design means that an application **adapts its layout** based on:

- Screen size
- Screen orientation

Responsive design ensures that applications work well on:

- Phones
- Tablets
- Different screen resolutions





RESPONSIVE UI IN FLUTTER

Flutter supports responsive design by providing:

- Screen size information
- Layout flexibility
- Dynamic UI rebuilding

This allows developers to create **one application** that works across **multiple devices**.

CREATING A RESPONSIVE FLUTTER APP

- Flutter allows you to create apps that self-adapt to the device's screen size and orientation.

There are two basic approaches to creating Flutter apps with responsive design:

- **Use the LayoutBuilder class**

From its builder property, you get a BoxConstraints object. Examine the constraint's properties to decide what to display. For example, if your maxWidth is greater than your **width** breakpoint, return a **Scaffold** object with a row that has a list on the left. If it's narrower, return a Scaffold object with a drawer containing that list.

- **Use the MediaQuery.of() method in your build functions**

This gives you the size, orientation, etc, of your current app. This is more useful if you want to make decisions based on the complete context rather than on just the size of your particular widget. Again, if you use this, then your build function automatically runs if the user somehow changes the app's size.

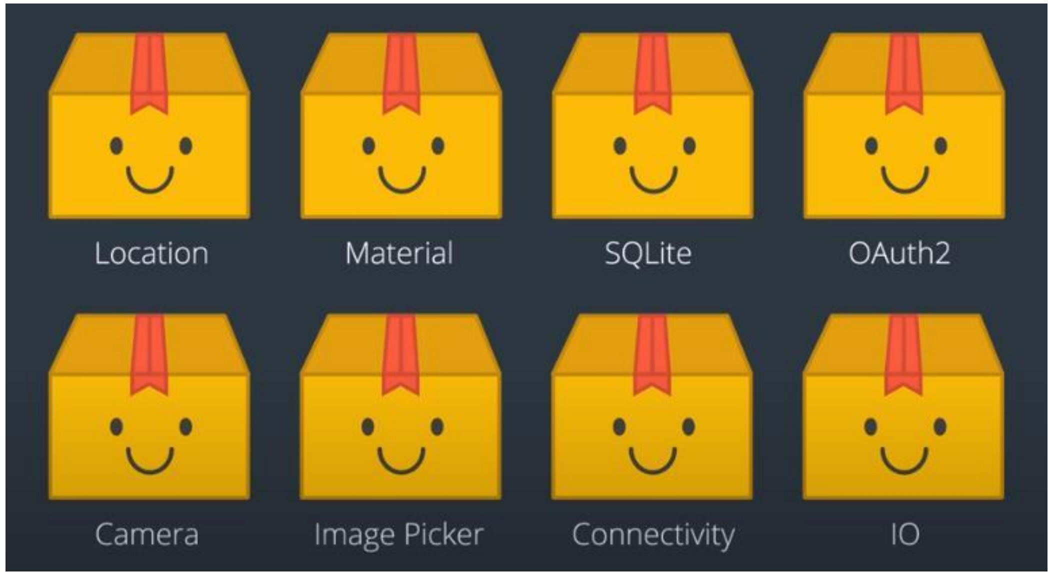
Other useful widgets and classes for creating a responsive UI:

AspectRatio, CustomSingleChildLayout, CustomMultiChildLayout, FittedBoxFractionallySizedBox, LayoutBuilder, MediaQuery, MediaQueryData, OrientationBuilder.

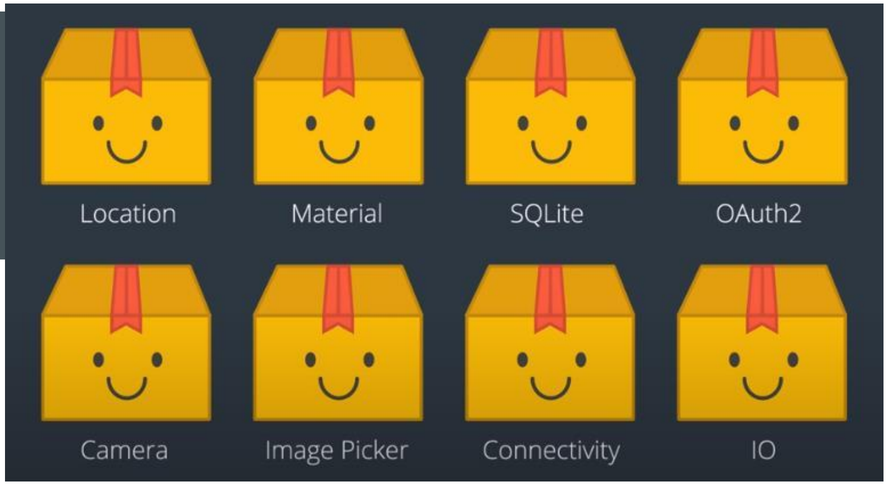
PACKAGES AND DEPENDENCIES

Packages are reusable libraries that add functionality to applications.
They help developers:

- Save time
- Avoid rewriting code
- Use tested solutions

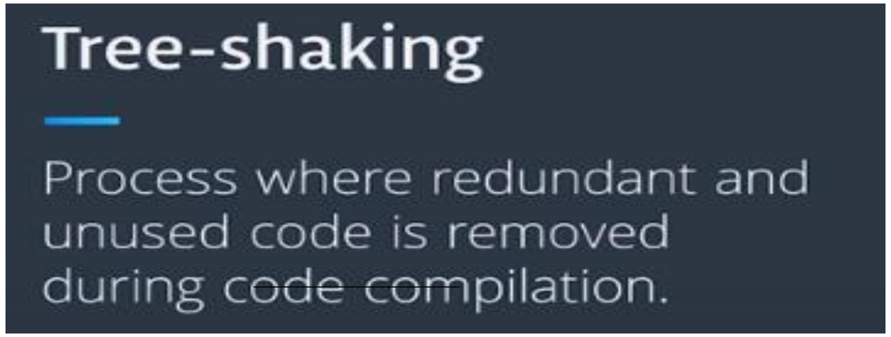


PACKAGES, PLUGINS, AND PUBSPEC.YAML



pubspec.yaml File
The pubspec.yaml file contains:

- Application metadata
- Dependencies (packages)
- Assets and fonts
- Even though you import a large **package**, only the functions you use end up being compiled down to code in **release mode.** This is because Flutter uses **tree shaking** to **remove redundant and unused code** in the **compilation process** for *the binary used in* ***production***.
- You can search for **packages** on the **Dart packages site**. Using **packages** and **plugins** can make your development that much more efficient.

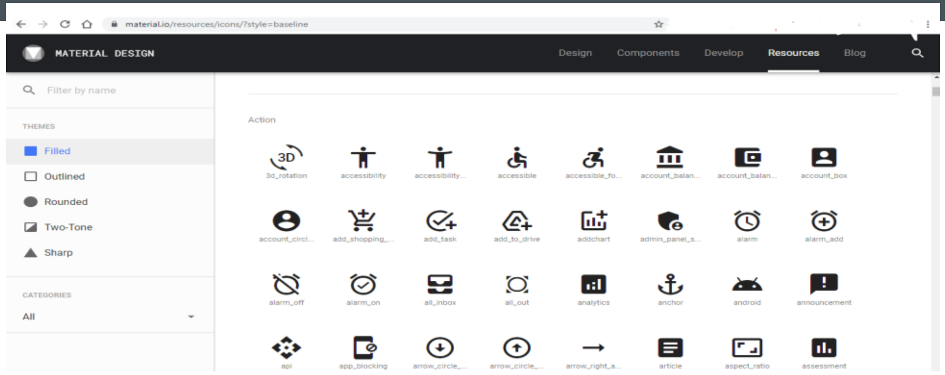


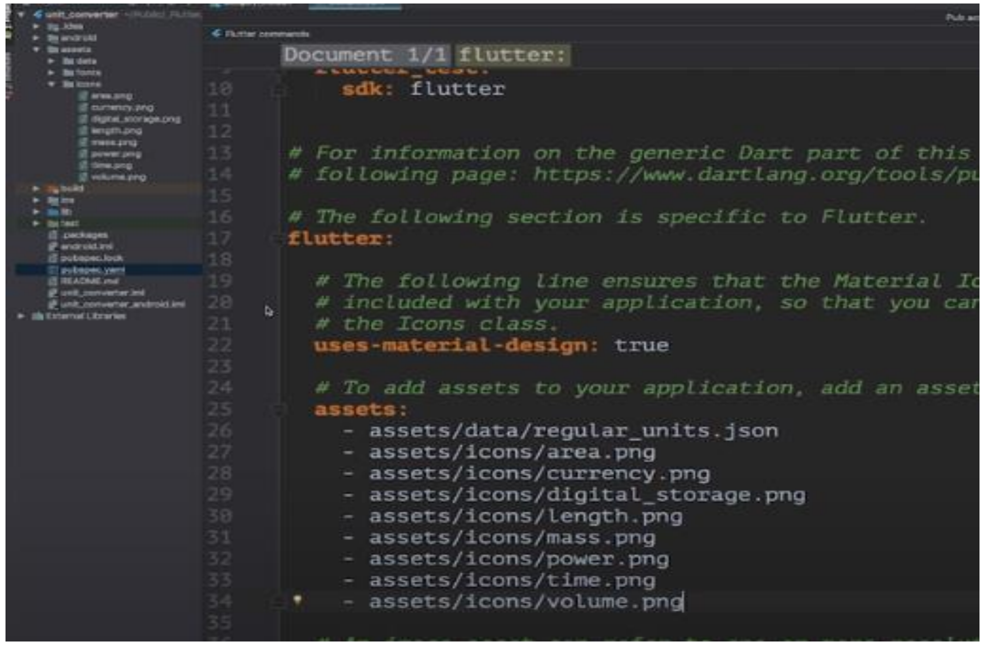
Tree-shaking

Process where redundant and unused code is removed during code compilation.

IMAGE AND ICON ASSETS

- **Icons** and **images** are also saved and managed within **the assets directory** and **pubspec.yaml file**.
- **Material design** provides over **built-in icons**, such as **play, refresh, alarm, pets, insert photo, and more.** The material components also incorporate these icons.
- These icons can be used in icon buttons which let you specify a function to run when the icon is tapped.
- The flutter **image widget** has separate **constructors** based on whether your **path points to an asset, local file, or from the Web.**

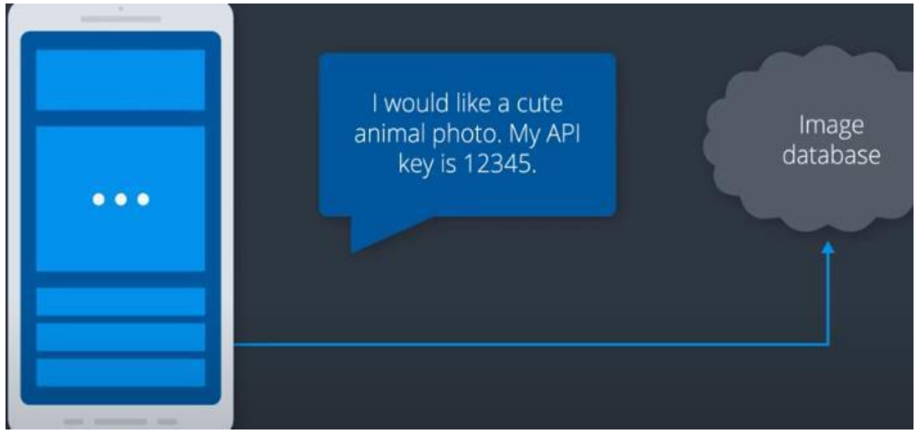


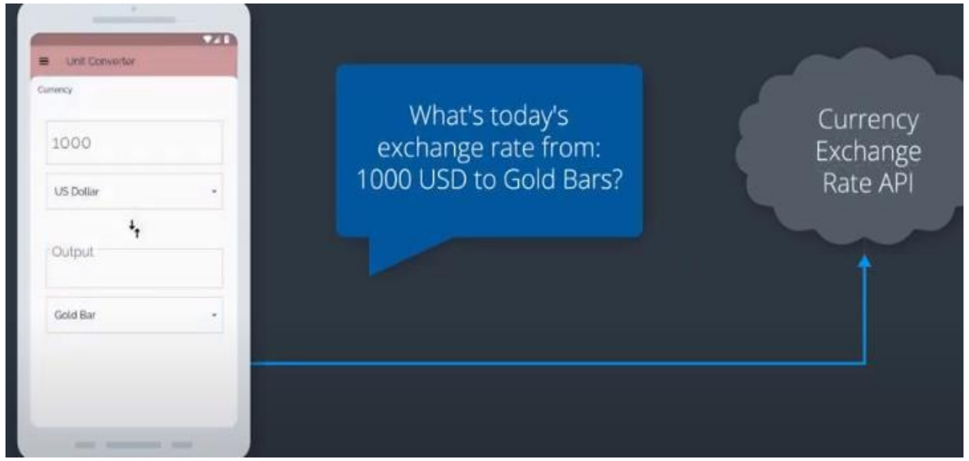


```yaml
    sdk: flutter


# For information on the generic Dart part of this
# following page: https://www.dartlang.org/tools/pu

# The following section is specific to Flutter.
flutter:

  # The following line ensures that the Material Ic
  # included with your application, so that you car
  # the Icons class.
  uses-material-design: true

  # To add assets to your application, add an asset
  assets:
    - assets/data/regular_units.json
    - assets/icons/area.png
    - assets/icons/currency.png
    - assets/icons/digital_storage.png
    - assets/icons/length.png
    - assets/icons/mass.png
    - assets/icons/power.png
    - assets/icons/time.png
    - assets/icons/volume.png
```

NO ASSETS? USE AN API!

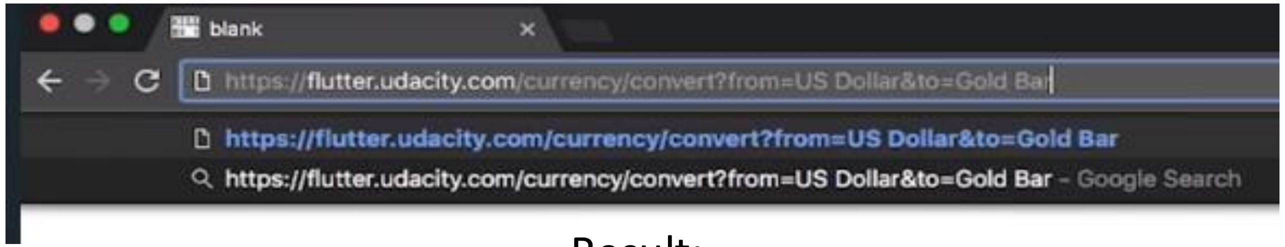
- Sometimes, the data you want to save changes. So, you can't save it in an **asset**.
- For example, you might want to randomly select a **cute animal picture** from an **API** *each day or for this app*, you might want to know this moment's **exchange rate for currencies.**
- For this, we will call an **API** which **retrieves real-time data**. Connecting to **APIs** can be done with the **HTTP client in dart:io**.
- We create a **HTTP client** that points to our **endpoint**. Our current app does not require **authentication** or an **API key**.
- So we just hit the **endpoint** with our to from an **amount query parameters** to get the **unit conversion** that we want.




Unit Converter
Currency
1000
US Dollar
Output
Gold Bar
What's today's exchange rate from: 1000 USD to Gold Bars?
Currency Exchange Rate API

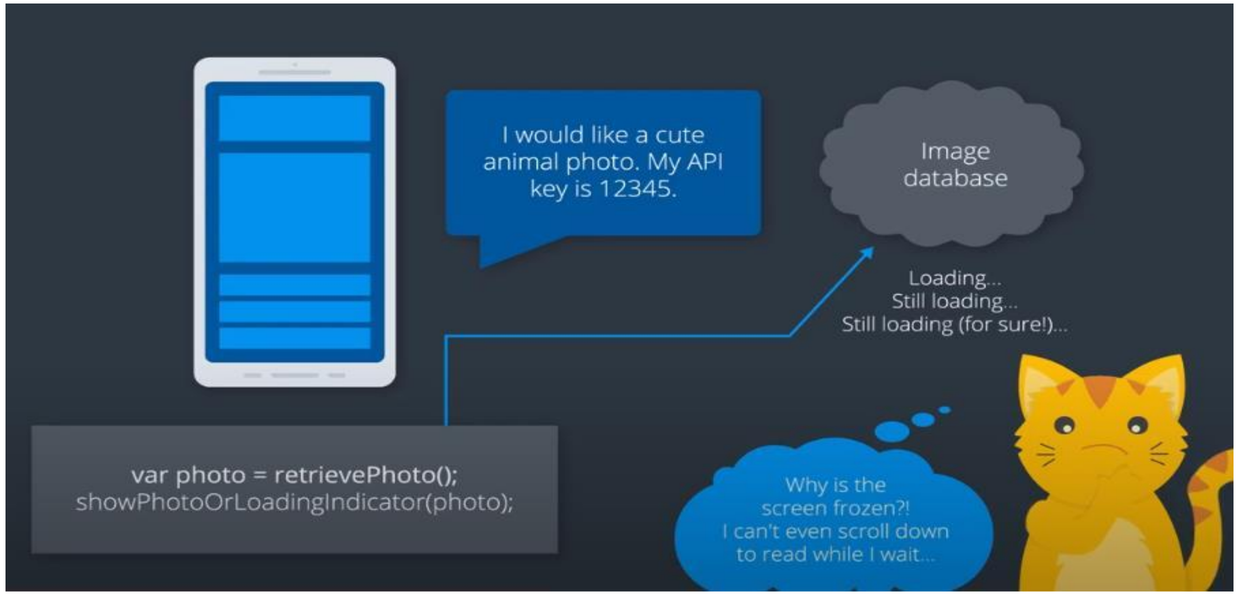
NO ASSETS? USE AN API!

- You can try out some **queries** by going to **flutter.udacity.com/currency** in your ***browser***.
- An API doesn't immediately return your data the way a local asset would because Dart runs in a single thread.
- If we solely wait for the **API call** to **return**, we would see **a frozen screen** and we wouldn't be able to interact with the app.
- The **API call** may take some time to return based on the **server's speed, your Internet connection, and other factors**.


blank
https://flutter.udacity.com/currency/convert?from=US Dollar&to=Gold Bar
https://flutter.udacity.com/currency/convert?from=US Dollar&to=Gold Bar
https://flutter.udacity.com/currency/convert?from=US Dollar&to=Gold Bar - Google Search

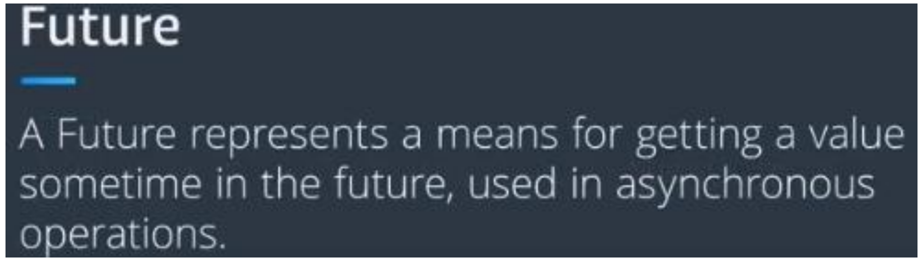
Result:

```
{"status":"ok","conversion":0.40439439630352586}
```


I would like a cute animal photo. My API key is 12345.
Image database
Loading...
Still loading...
Still loading (for sure!)...
var photo = retrievePhoto();
showPhotoOrLoadingIndicator(photo);
Why is the screen frozen?!
I can't even scroll down to read while I wait...

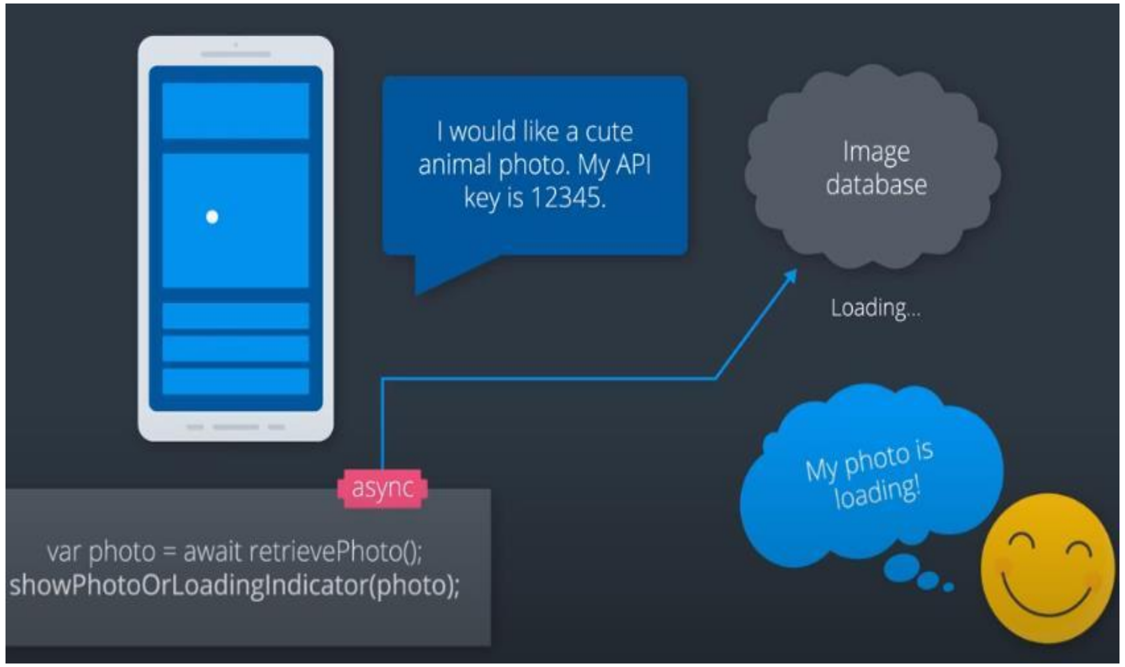
NO ASSETS? USE AN API!

- Rather than wait for it, we wrap the API in an asynchronous operation.
- This lets your app continue to run without getting blocked.
- **Dart** *uses* ***future objects*** *to represent* ***asynchronous operations***.
- When a **function** that returns **a future is invoked**, two things happen.
- *First, the function cues up work to be done and returns an incomplete future object.*
- *Later, when a value is available, the future object completes with that value or with an error*. We'll discuss **errors** later.



Future

A Future represents a means for getting a value sometime in the future, used in asynchronous operations.



```
var photo = await retrievePhoto();
showPhotoOrLoadingIndicator(photo);
```

REVISION QUIZ

1. **Material Design in Flutter provides predefined widgets and themes to ensure visual consistency, while Material Design 3 modernizes the UI through updated typography, colors, and component shapes.**
   ☐ True ☐ False
2. **In Flutter, themes control an application's overall visual appearance, and platform-specific UI allows apps to render native-looking components for both Android and iOS using a single code base.**
   ☐ True ☐ False
3. **User input in mobile applications includes text input, button interactions, and gestures, and TextFormField is preferred over TextField when structured input and validation are required.**
   ☐ True ☐ False
4. **Gestures enhance application interactivity, while responsive design ensures that a Flutter app adapts its layout based on screen size and orientation across different devices.**
   ☐ True ☐ False
5. **Flutter applications can be extended using packages defined in the pubspec.yaml file, and when assets are not suitable for dynamic data, APIs can be used with asynchronous operations to avoid blocking the user interface.**
   ☐ True ☐ False

- Material Design provides UI consistency
- Material 3 modernizes application appearance
- User input enables interaction
- Gestures enhance usability
- Responsive design supports multiple devices
- Packages extend application functionality
- Use Assets if your images, videos are ready, otherwise API!