Mobile Development

Cross-Platform Course 2023

Ver 3.10.2, Null-safety compliance

Caspian Consultancy Services

Modules

Fundamental 2022

- Introduction
- Setup & Softwares
- Flutter Project Structure
- What are Widgets?
- Useful Widgets
- State Management
- Creating Layout
- Creating Responsive Design
- Creating Buttons & DialogBox
- Creating Navigation & Passing Parameter
- Arrays, ListView & ListView Builder

Modules

Fundamental 2022

- FutureBuilder & StreamBuilder
- JSON, REST API & Flutter
- Persistent data with SQLite
- How to build your APK, AAB or IPA

Introduction

What is Flutter?

Flutter is an **open-source UI** software development kit created by Google.

It is used to develop cross platform applications for **Android**, **iOS**, **Linux**, **MacOS**, **Windows**, **Google Fuchsia**, and the **web** from a single codebase.

Setup & Softwares

Development Softwares

We need to install:

- Visual Studio Code
- Flutter|Dart Plugins
- Android Studio + AVD (Emulator)
- Xcode | Xcode CLi (If you use Mac)
- Chrome Browser (If you develop web)
- Visual Studio 2021 + UWP + Desktop Dev w/ C++ (If you develop Windows 11)

Setup & Softwares

Development Softwares

For Linux, we need to install:

- Clang,
- CMake,
- GTK,
- Ninja build
- Pkg-config

Or via Snapd

Setup & Softwares

We need these Softwares

Some Helpful Flutter Packages:

- Awesome Flutter Snippets
- Flutter Widget Wrap

Visual Studio Guide Preference:

- Settings > guide > Dart: Preview flutter UI Guides & Custom Tracking (On)

Setup & Softwares

We need these Softwares

Some Additional Softwares for Design & Prototyping:

- Adobe Photoshop or Illustrator
- Figma
- Adobe XD
- Lunacy

Online Tools:

- FlutterFlow.io
- Dhiwise Flutter
- Teta.so
- AppIcon.co

Create a New Project

Cli, VSC or Android Studio

This training will use Visual Studio Code:

1. Visual Studio Code

   > View > Command Palette... > Flutter: New Application Project

2. Terminal or CMD

   > flutter create yourProjectName

Flutter Project Structure

Understanding Files & Directory

Additional dart Tools

Android Project

Apk, AAB or IPA

iOS Project

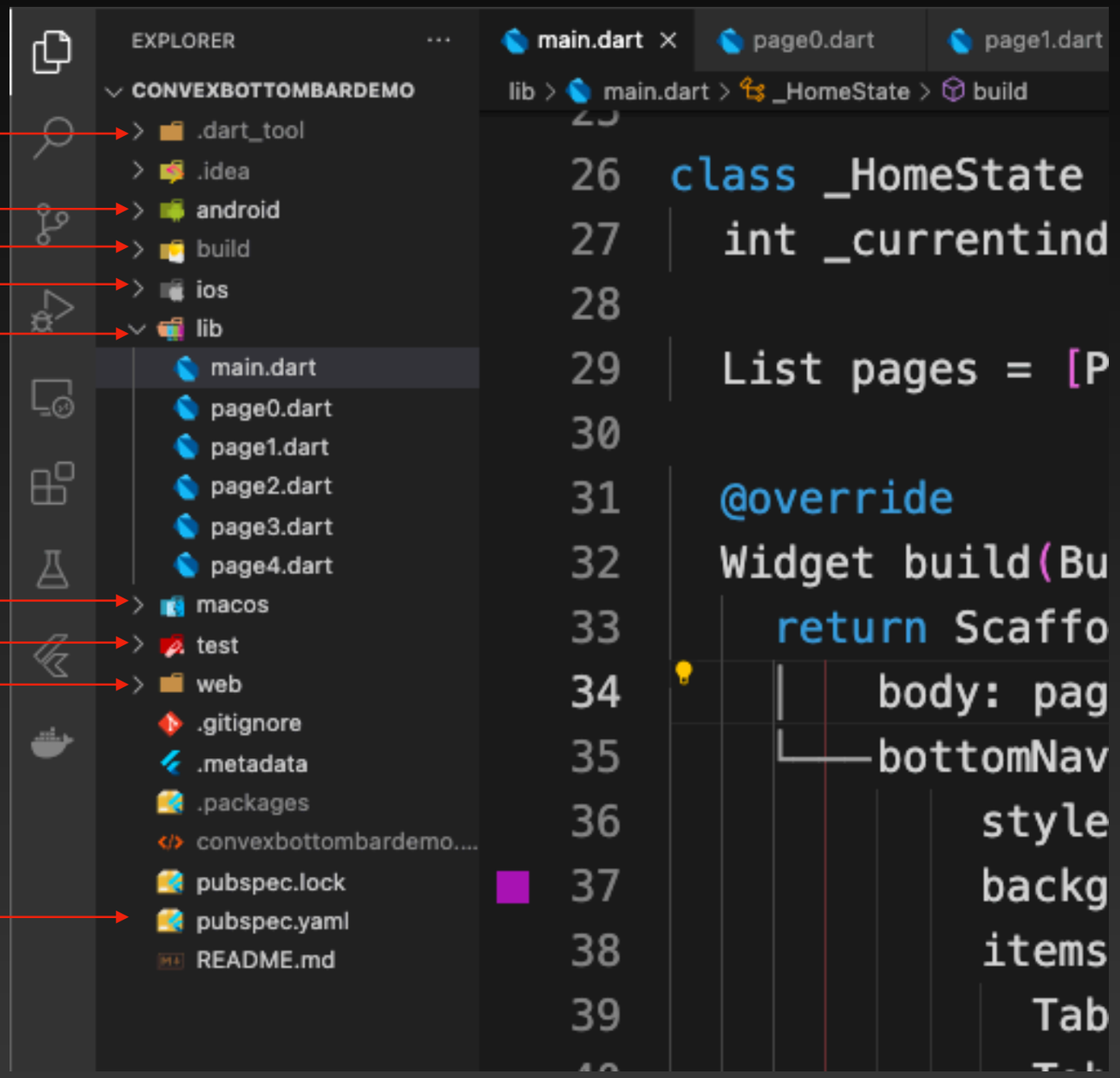
Our dart code Files

depends on your MacOS | Windows

Test Files (Optional)

Web

Dependencies Requirements



What are Widgets?

To understand Flutter, We need to understand Widgets

- Everything is a Widget
- OOP - Object Oriented Programming
- There are 2 types of Widgets:
  - Stateless Widgets
  - Stateful Widgets
  - Inherited Widgets (derived)

Widgets?

Stateless vs Stateful

- Stateless widgets are immutable, meaning that their properties can't change—all values are final.
- Stateful widgets maintain state that might change during the lifetime of the widget. Implementing a stateful widget requires at least two classes:
    i) a StatefulWidget class that creates an instance of
    ii) a State class. The StatefulWidget class is, itself, immutable, but the State class persists over the lifetime of the widget.
- Inherited Widget a special kind of widget that defines a context at the root of a sub-tree.

Stateless Widget

A Widget with no State!

Stateless Widget:

- Single Class
- No State

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

class ThirdScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Screen 3"),
      ), // AppBar
      body: Center(
        child: RaisedButton(
          child: Text('Go back Screen 1'),
          onPressed: () {
            //Guna ne utk balik ke screen 1
            Navigator.of(context).pushNamedAndRemoveUntil('/screen1',
            (Route<dynamic> route) => false);
            //----
          },
        ), // RaisedButton
      ), // Center
    ); // Scaffold
  }
}

```

Stateful Widget

A Widget with State!

The State class
Create State

Stateful Widget:

- 2 Classes: State & UI
- Has State

The UI class

A **stateful widget** is dynamic. The user can interact with a stateful widget (by typing into a form, or moving a slider, for example), or it changes over time (perhaps a data feed causes the UI to update).

Checkbox, Radio, Slider, InkWell, Form, and TextField are examples of stateful widgets, which subclass StatefulWidget.

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

class FirstScreen extends StatefulWidget {
  @override
  _FirstScreenState createState() => new _FirstScreenState();

}

class _FirstScreenState extends State<FirstScreen> {
  var _textController = new TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Screen 1'),
      ), // AppBar
      body: new ListView(
        children: <Widget>[
          new ListTile(title: new TextField(controller: _textController,),),
          new ListTile(title: new RaisedButton(
            child: new Text("Next"),
            onPressed: () {
              //do something when pressed

              //original kaedah just navigate
              //Navigator.of(context).push(route);

              // Kaedah 1
              //var route = new MaterialPageRoute(builder: (BuildContext context)

              //Kaedah 2
              Navigator.push(context, MaterialPageRoute(builder:(context) => new S
);
            }
          ),), // RaisedButton // ListTile
```

Useful or Common Widgets

A Widget with State!

Widget with single root, **child**:

- Container
- Center
- Buttons - ElevatedButton, OutlinedButton, TextButton
- Card
- SingleChildScrollView

Widgets with multiple, **children**:

- Column
- Row
- ListView
- Stack
- Wrap

State Management

For Managing State in Widgets

When do we need to use:

- Change of state, something change - value, color, shape
- Need to pass data from one widget to another
- Useful in Stateful widget & Inherited widget
- Inherited widget passing value

State Management

For Managing State in Stateful Widgets

Some State Management used in Flutter:

- SetState
- Provider
- GetX
- RiverPod
- Redux
- Fish-Redux
- BLoC/Rx
- MobX
- GetIt
- Binder
- States-rebuilder

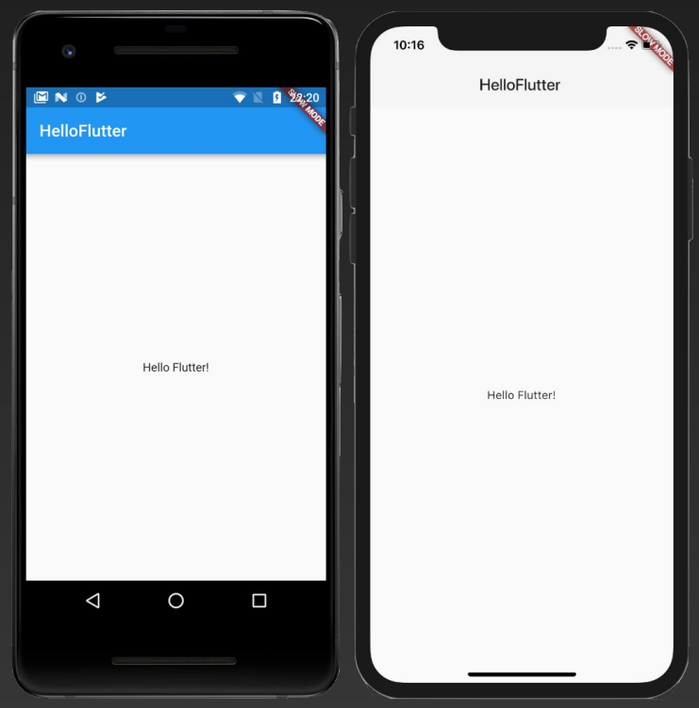
More info: https://flutter.dev/docs/development/data-and-backend/state-mgmt/options

Layout

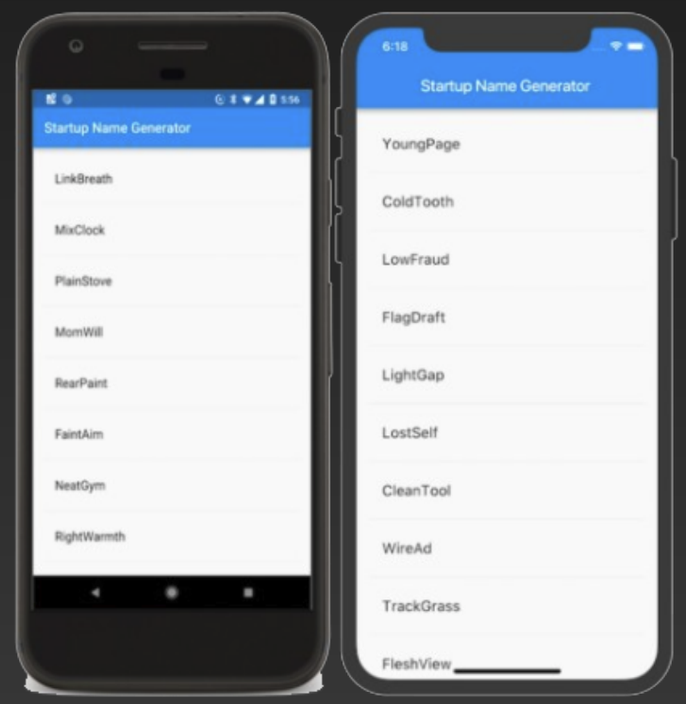
Creating Layout

Ui design in Flutter

- Flutter using Material Design in Android & Using ‘Cupertino‘ Design in iOS.
- Slight differences in rendering UIs.
- Not all designs can be same.



Material Cupertino



Material Cupertino

Flutter Layout & UI Elements

Basic Layout makes up of widgets

- Container
- Row
- Column
- Stack

} **Layout elements**

- Image
- Text
- Icon
- Buttons

} **UI elements**

REMEMBER:

- Widgets are used for both layout and UI elements.
- Compose simple widgets to build complex widgets.

Flutter Layout

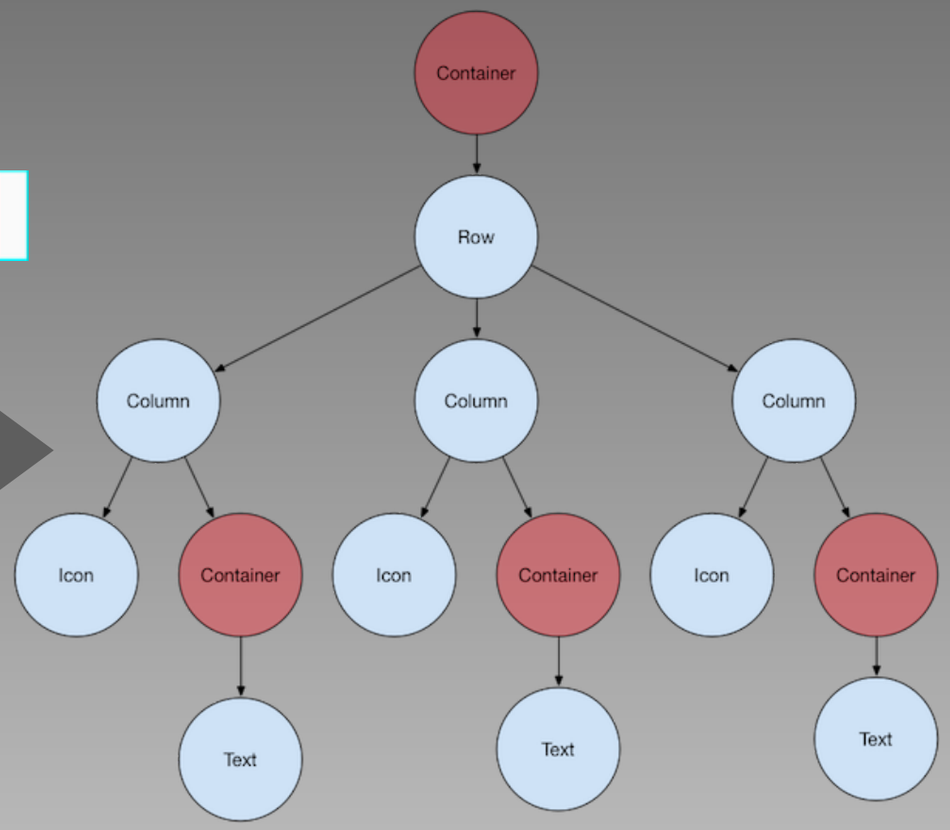
Step by Step

Final UI

Made up of Widgets:

Understand This in Container, Row, Column & Widgets

When you code it in Flutter

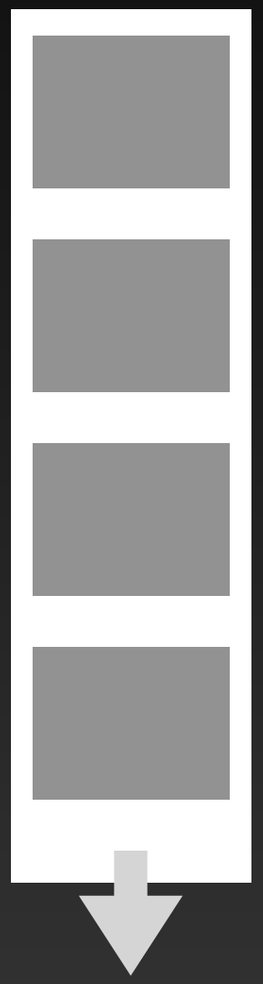


In Flutter, a common way of laying out widgets is composing them into groups of columns and row—this allows us to easily layout our components in a vertical / horizontal manner. When you wish to layout components in a **Horizontal** manner, then you will use a **Row** and for **Vertical** layout you will use a **Column**. There is also a Stack Widget for overlapping content

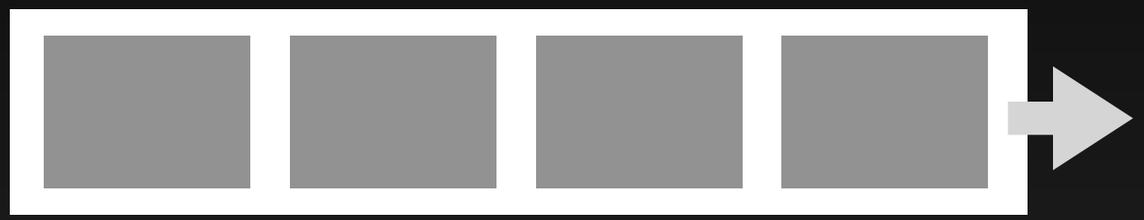
Layout Widgets

Column & Row | Stack

Column( )

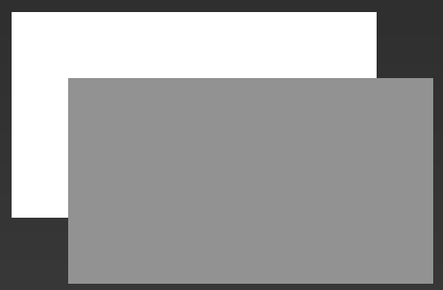


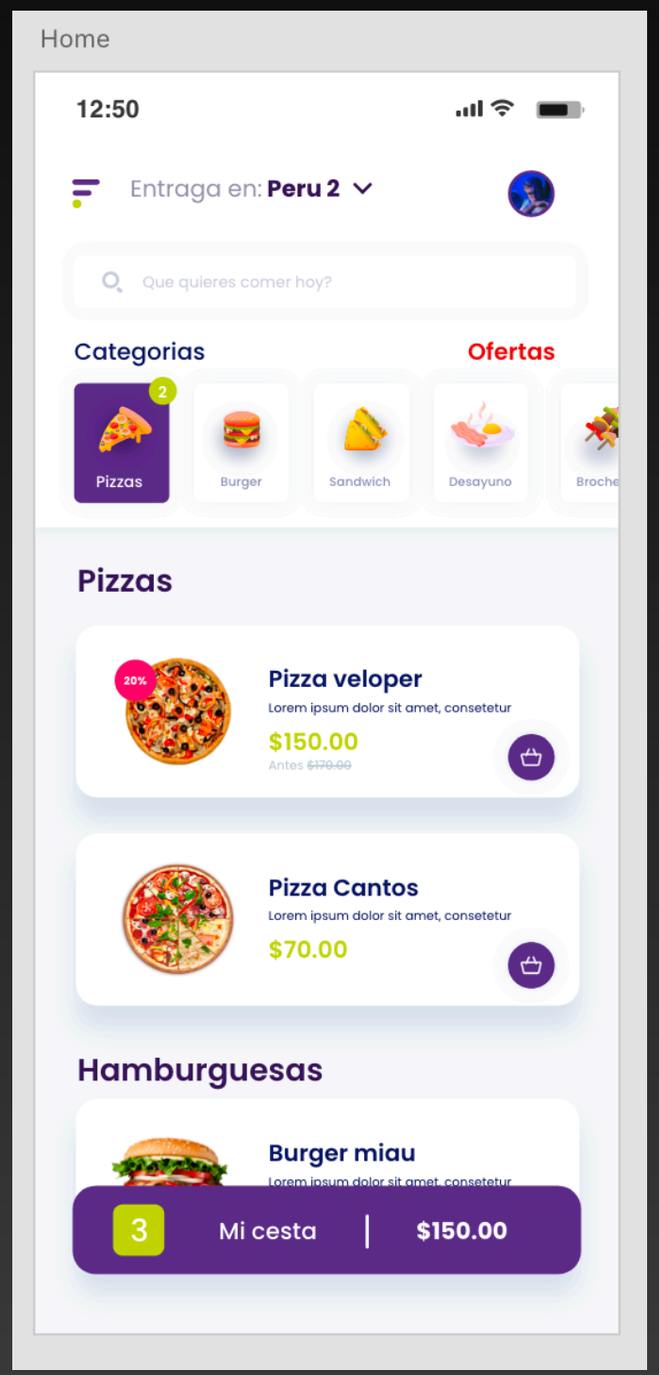
Row( )



Try Imagining your design
in Columns & Rows

Stack( ) For Overlapping

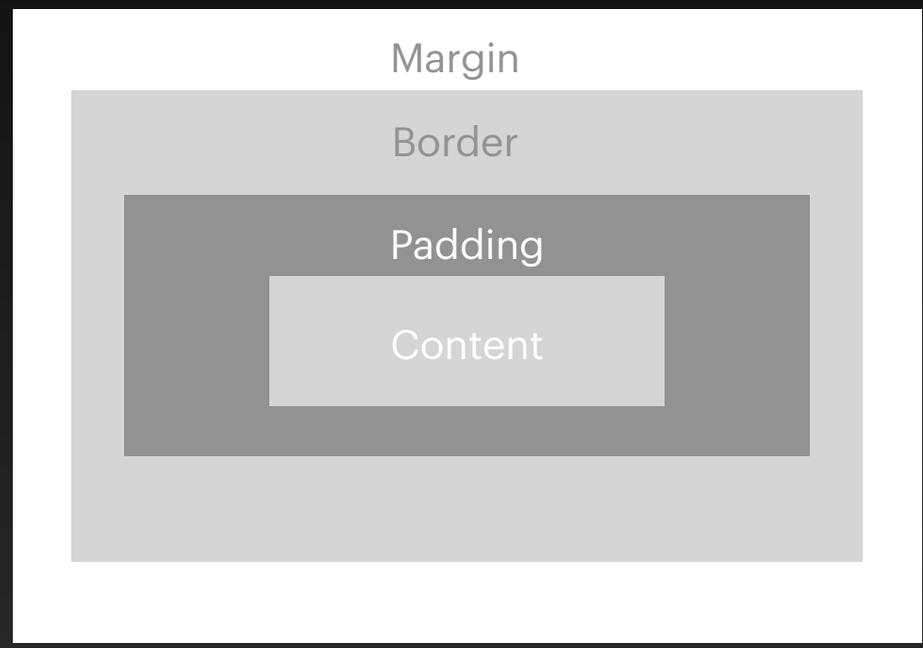




Container Widget

The Anatomy of Container Widget

- You can embed other Widgets in a Container to make them compartmentalised.
- Container is Optional
- Add padding, margins, borders
- Change background color or image
- Contains a single child widget, but that child can be a Row, Column, or even the root of a widget tree



Other Layout Widgets

Commonly used Widgets

Other widgets used in layout:

- **GridView**: Lays widgets out as a scrollable grid.
- **ListView**: Lays widgets out as a scrollable list.
- **Stack**: Overlaps a widget on top of another.
- **TabBar**:
- **NavigationBottomBar**:

Creating Layout

Creating basic Layout

Steps to create basic layout in Flutter:

1. Start with a design for your mobile app
2. Prototype your mobile app
3. Export all assets
4. Code in Dart/Flutter

Creating Layout

Beginning your Layout

You'll need a UI Design & Prototyping tool such as Adobe XD, Figma or SketchApp

This training will use Adobe XD, It's FREE 7-days Trial or you can use Lunacy.

Using XD to Create Layout

Visual Design to Create Layout

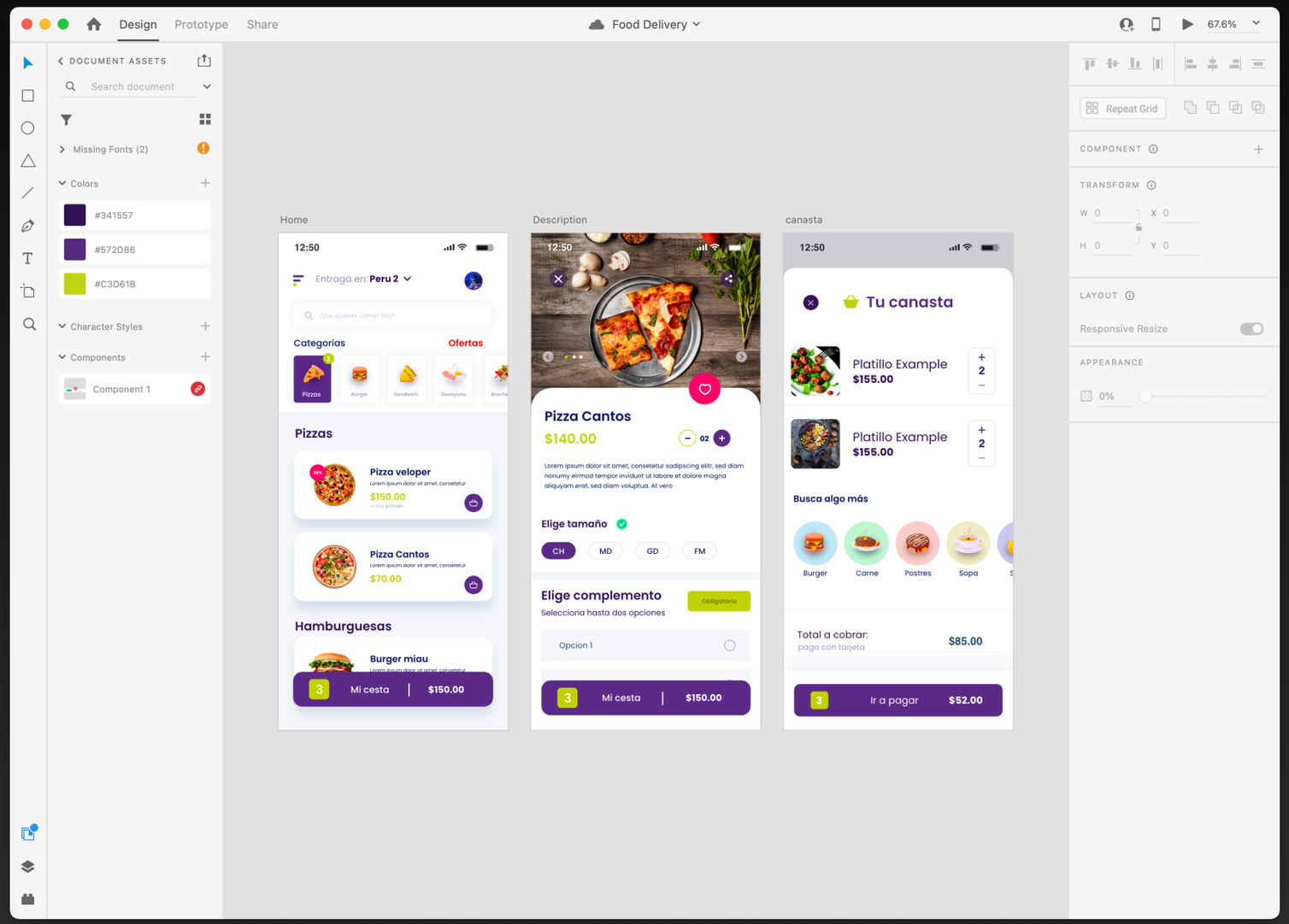
Adobe XD needs additional plugins to be effective

Adobe XD plugins that you'll need:

- Stock Images
- Flutter Export
- Lorem Ipsum
- XD to Flutter

Start the Design

Start with a design

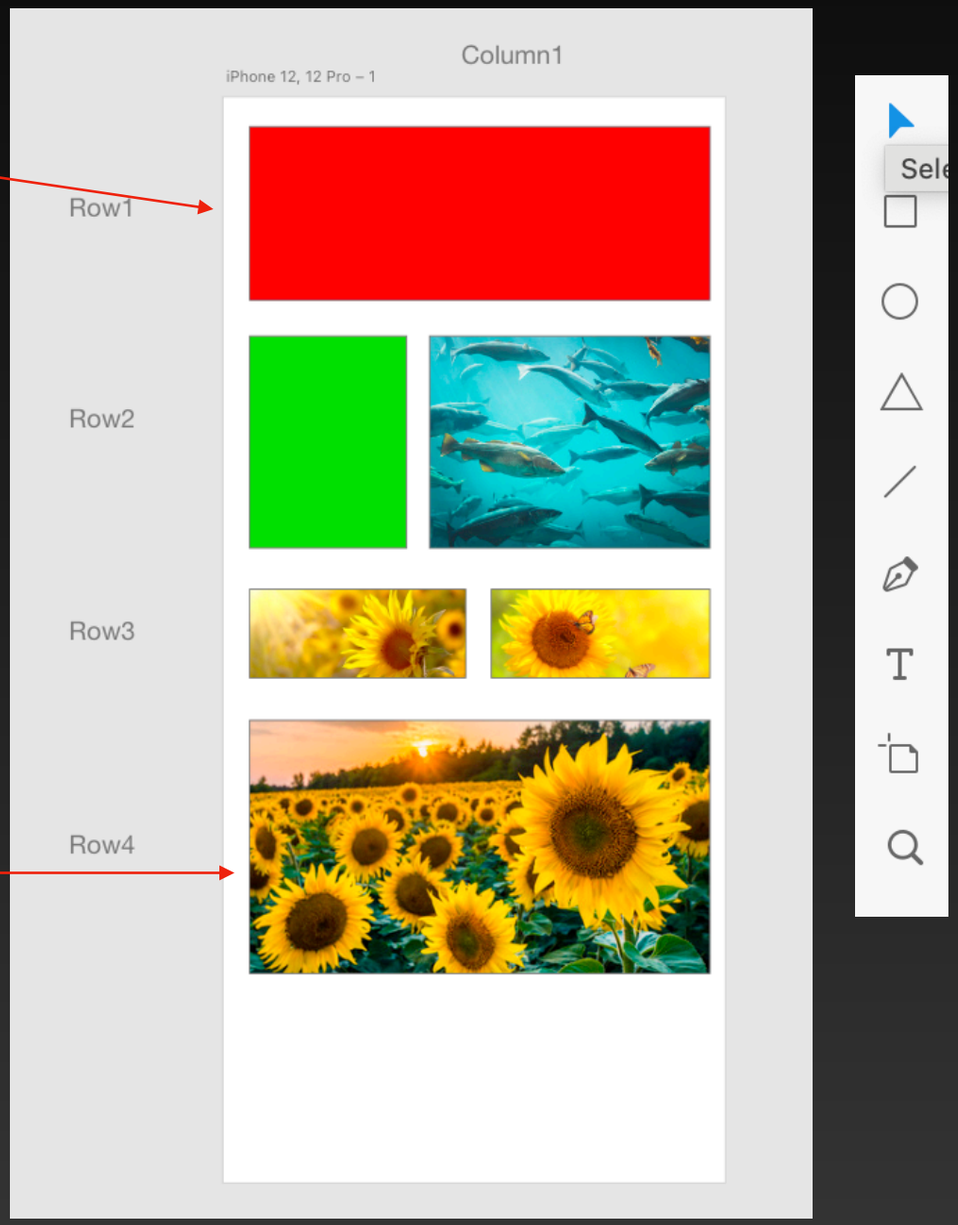


Simply Design using available Tools on XD

Creating Basic

In XD Create Primitives shapes

Primitives Shapes like
Rectangles with colors



Simply Design using
available Tools on XD

Replace Shapes with
Images later

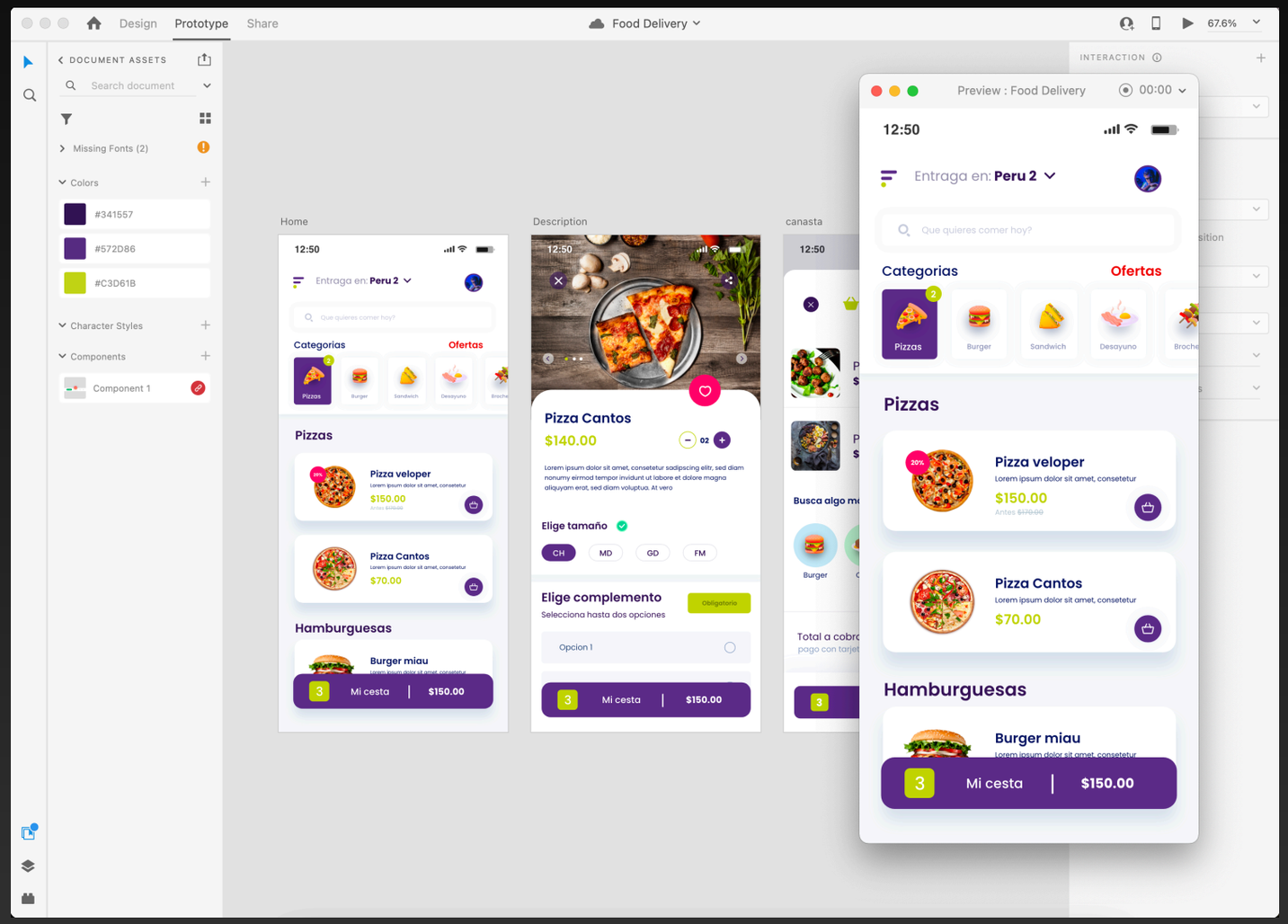
Using XD to Create Layout

Visual Design to Create Layout

After Creating Basic Shapes, Code in Visual Studio Code using Column, Row, Stack, Container & Other widgets.

Creating Layout

Prototype with working flow

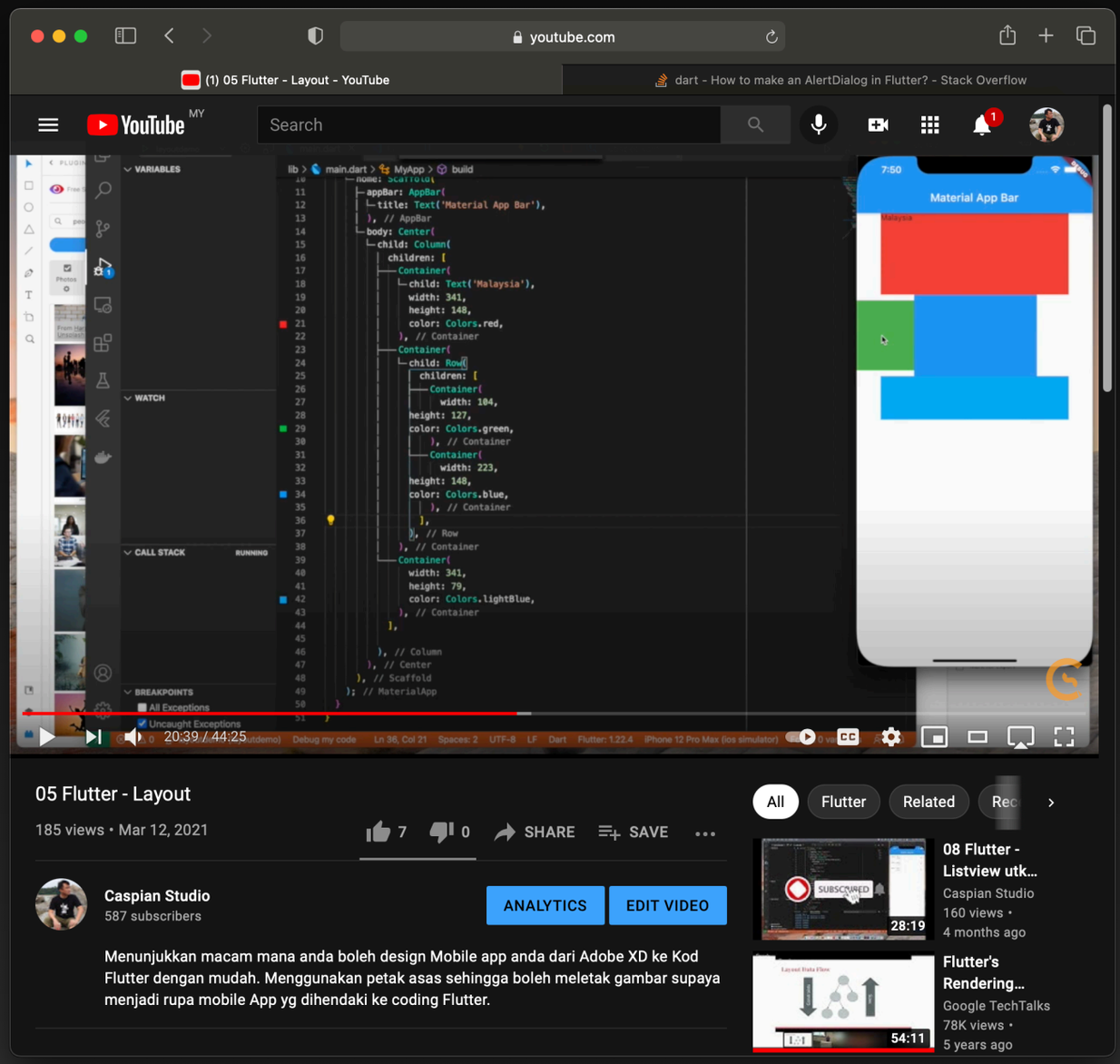


Can Preview on:

- Desktop
- Mobile App

Let's do Layout

Video how to do layout



Refer on: https://www.youtube.com/watch?v=pHqJcDev8nM?sub_confirmation=1

Responsive Layout

Creating Responsive Layout

Layout that responds to size of screen

- **LayoutBuilder**
  Builds a widget tree that can depend on the parent widget's size.
- **Sizer**
  A flutter package that automatically adapts UI to different screen sizes.
- **MediaQuery**
  Use the MediaQuery.of(context) method in your build functions

  Example:
  widthX = MediaQuery.of(context).size.width();

```
Widget build(BuildContext context) {
  return LayoutBuilder(
    builder: (context, constraints) {
      if (constraints.maxWidth < 600) {
        return MyOneColumnLayout();
      } else {
        return MyTwoColumnLayout();
      }
    },
  );
}
```

Creating Responsive Layout

Layout using Sizer

Same UI in any Device
Responsiveness made Simple

Creating Responsive Layout

Layout without Sizer

Creating Responsive Layout

How to use Sizer

1. Add sizer package to you Flutter app
2. Import ‘package:sizer/sizer.dart
3. Now wrap MaterialApp with ResponsiveSizer Widget.

```
ResponsiveSizer(
      builder: (context, orientation, deviceType) {
        return MaterialApp();
      }
  )
```

Parameters

.h - Returns a calculated height based on the device.

.w - Returns a calculated width based on the device

.sp - Returns a calculated sp based on the device

SizerUtil.orientation - for screen orientation portrait or landscape

SizerUtil.deviceType - for device type mobile or tablet

Creating Responsive Layout

Example to use Sizer

Widget Size

```
Container(
    width: 20.w,     //It will take a 20% of screen width
    height:30.h      //It will take a 30% of screen height
  )
```

Padding

```
Padding(
        padding: EdgeInsets.symmetric(vertical: 5.h, horizontal: 3.h),
        child: Container(),
);
```

Font Size

```
Text(
       'Sizer',style: TextStyle(fontSize: 15.sp),
     );
```

Creating Responsive Layout

Example to use Sizer

Orientation

```
Device.orientation == Orientation.portrait
  ? Container(    // Widget for Portrait
      width: 100.w,
      height: 20.5.h,
    )
: Container(    // Widget for Landscape
    width: 100.w,
    height: 12.5.h,
)
```

Device Type

```
SizerUtil.deviceType == DeviceType.mobile
  ? Container(    // Widget for Mobile
      width: 100.w,
      height: 20.5.h,
    )
  : Container(    // Widget for Tablet
      width: 100.w,
      height: 12.5.h,
    )
```

Interactivity

Creating Buttons

Making Interactivity

So many button options:

- ElevatedButton (Previously RaisedButton)
- OutlinedButton
- TextButton (Prev FlatButton)
- FloatingButton
- PopupMenuButton
- IconButton
- InkWell, can create your own buttons too

Basic Code Buttons

They are all the similar codes

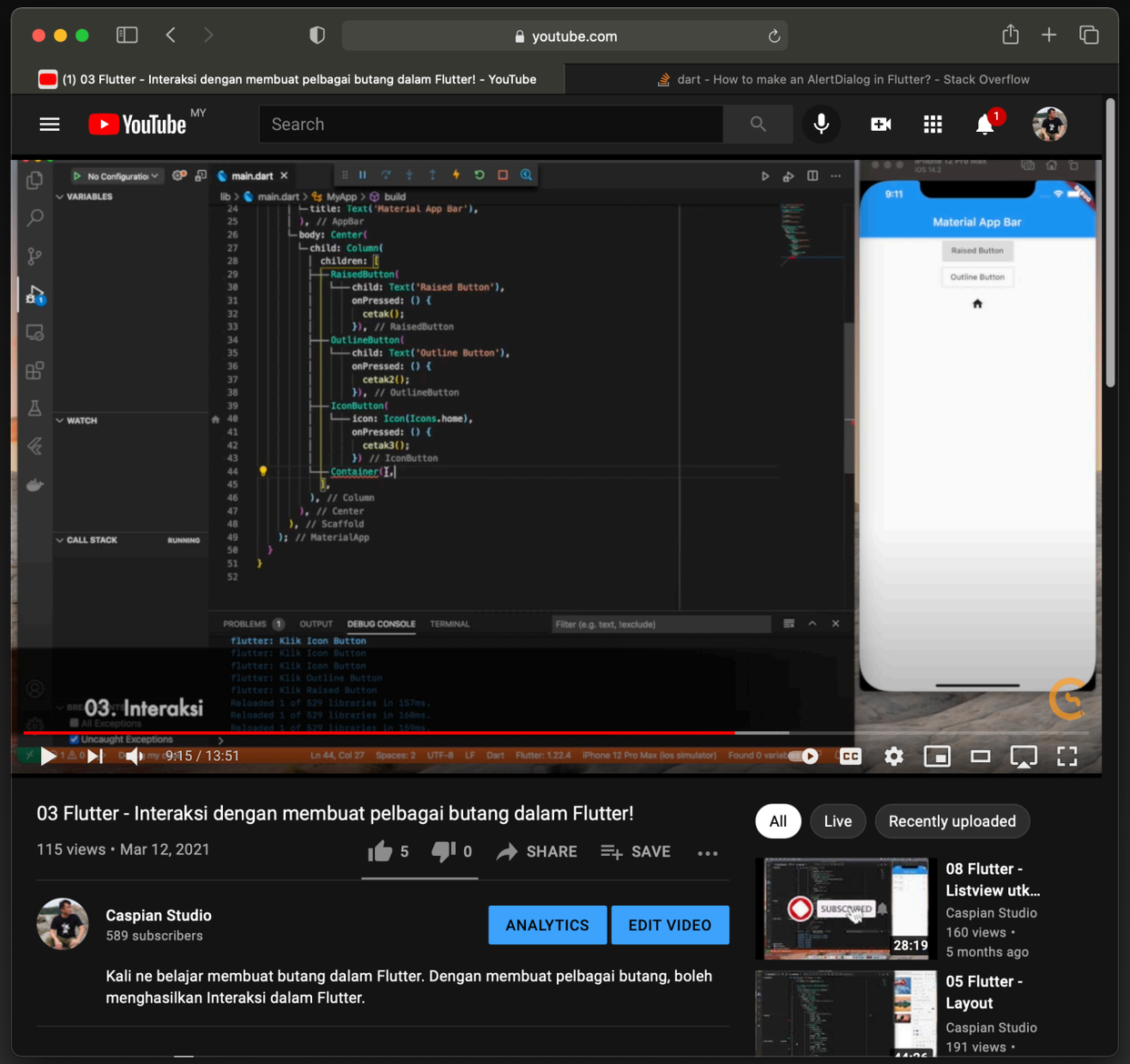
- A button has consisted of a text or an icon
- We can design a button UI using different shapes, colours, animations and behaviours.
- Button also can contain it ‘s child widgets for different usages.
- Buttons will have:

```
TextButton(
    onPressed: ( ) {
        //code inside
     },
    Child: Text(`My Button’),
    )
```

Can Refer to: https://medium.com/app-dev-community/flutter-button-types-with-examples-10ae487621a3

Let's Code

Video how to do Interactivity with Buttons



Refer on: https://www.youtube.com/watch?v=EkBdAiNvTT0?sub_confirmation=1

Creating DialogBoxes

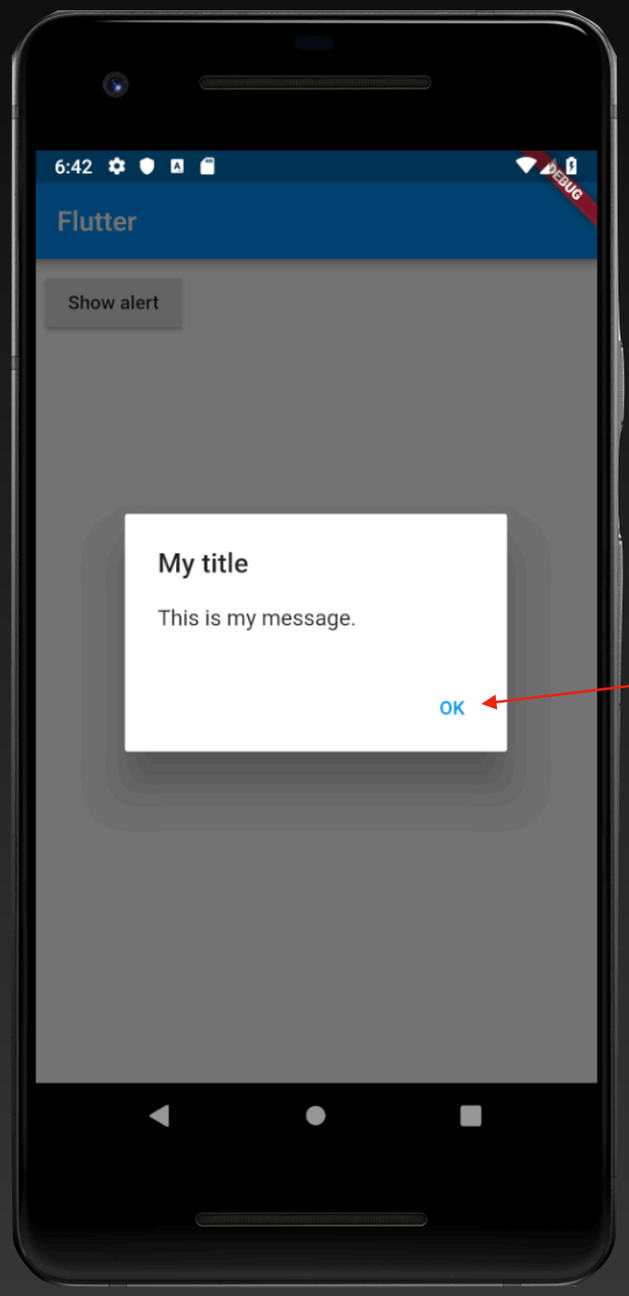
Those AlertDialog & Popup DialogBox

2 Types of AlertDialogs:

- Android
- Cupertino

Code for AlertDialog

Those AlertDialog & Popup DialogBox




```
showAlertDialog(BuildContext context) {

 // set up the button
 Widget okButton = TextButton(
  child: Text("OK"),
  onPressed: () { },
 );

 // set up the AlertDialog
 AlertDialog alert = AlertDialog(
  title: Text("My title"),
  content: Text("This is my message."),
  actions: [
   okButton,
  ],
 );

 // show the dialog
 showDialog(
  context: context,
  builder: (BuildContext context) {
   return alert;
  },
 );
}
```

Navigation

Creating Navigation

Navigate from one View to Another

It’s Easy to create Navigation:

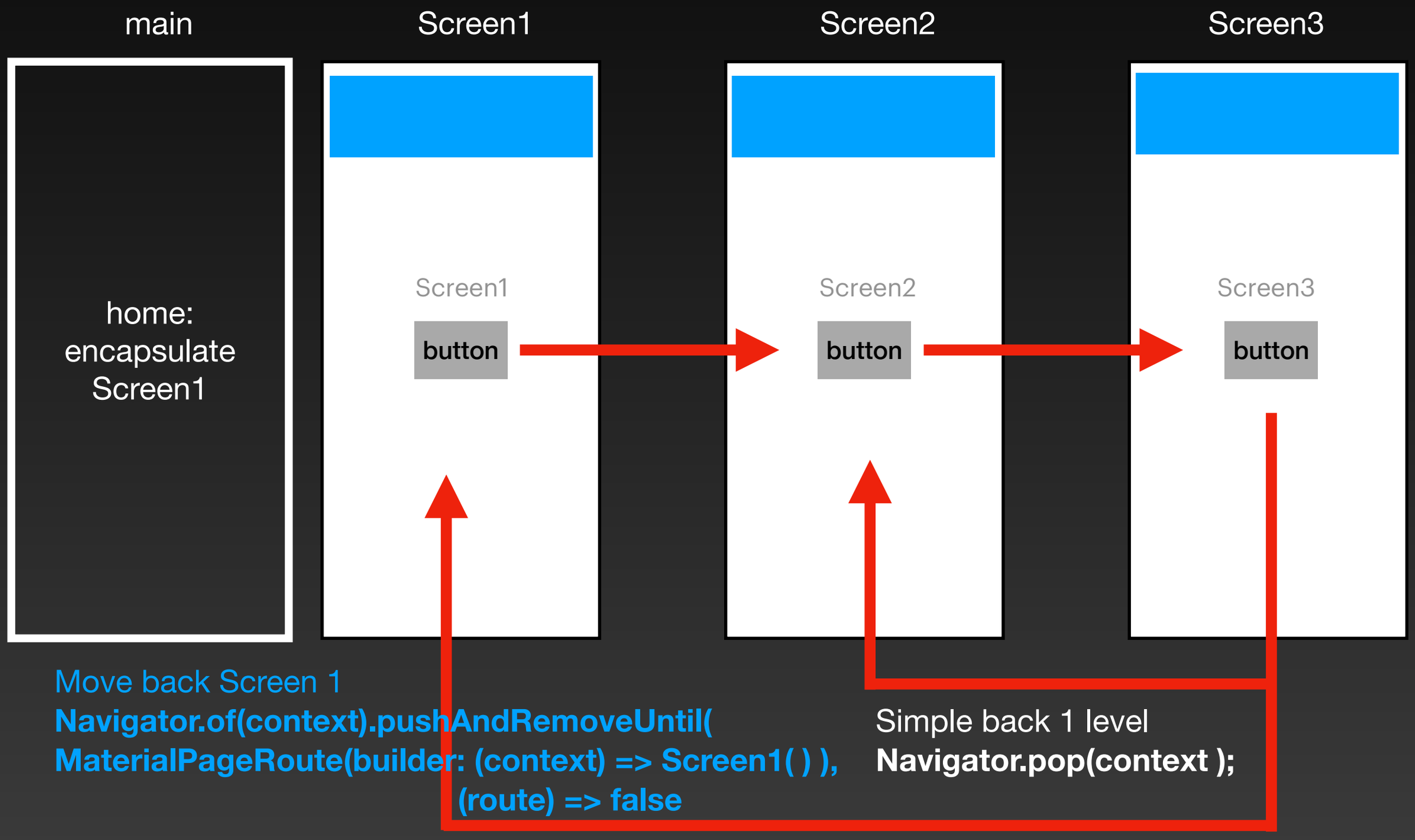
- To create a View, You create a new Class
- To create multiple Views, you create many Classes
- To navigate, you use Routes
- Use Navigator class

Creating Navigation

Navigate from one View to Another

Move to 1 level

Navigator.push(context, MaterialPageRoute(builder: (context) => Screen2( )));



Navigators

Navigate Code

To Move Forward use:

- Navigator.push(context, route) or
- Navigator.pushNamed(context, routename)

To Move Back use:

- Navigator.pop(context)

Note: There are many variations, choose which suits your requirement

Passing Parameters

Navigate & Passing Data

It’s Easy to pass data while Navigating:

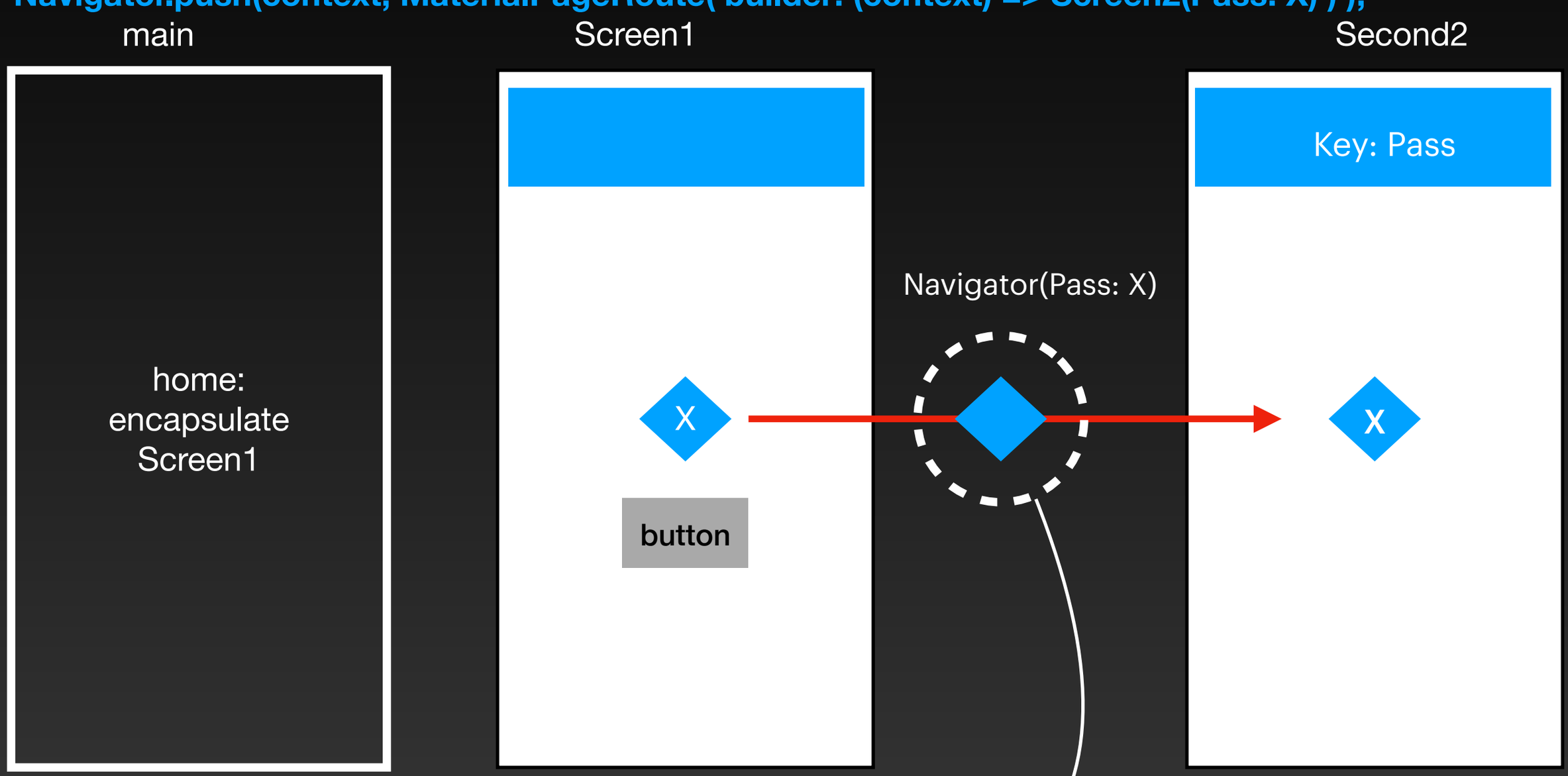
- Create a parameter or argument to pass in Navigator or Route
- Create a class Key to receive parameter passed
- Show the parameter

Passing Parameters

The Way Passing Data is done

Pass Obj X to next Screen

Navigator.push(context, MaterialPageRoute(builder: (context) => Screen2(Pass: X)));

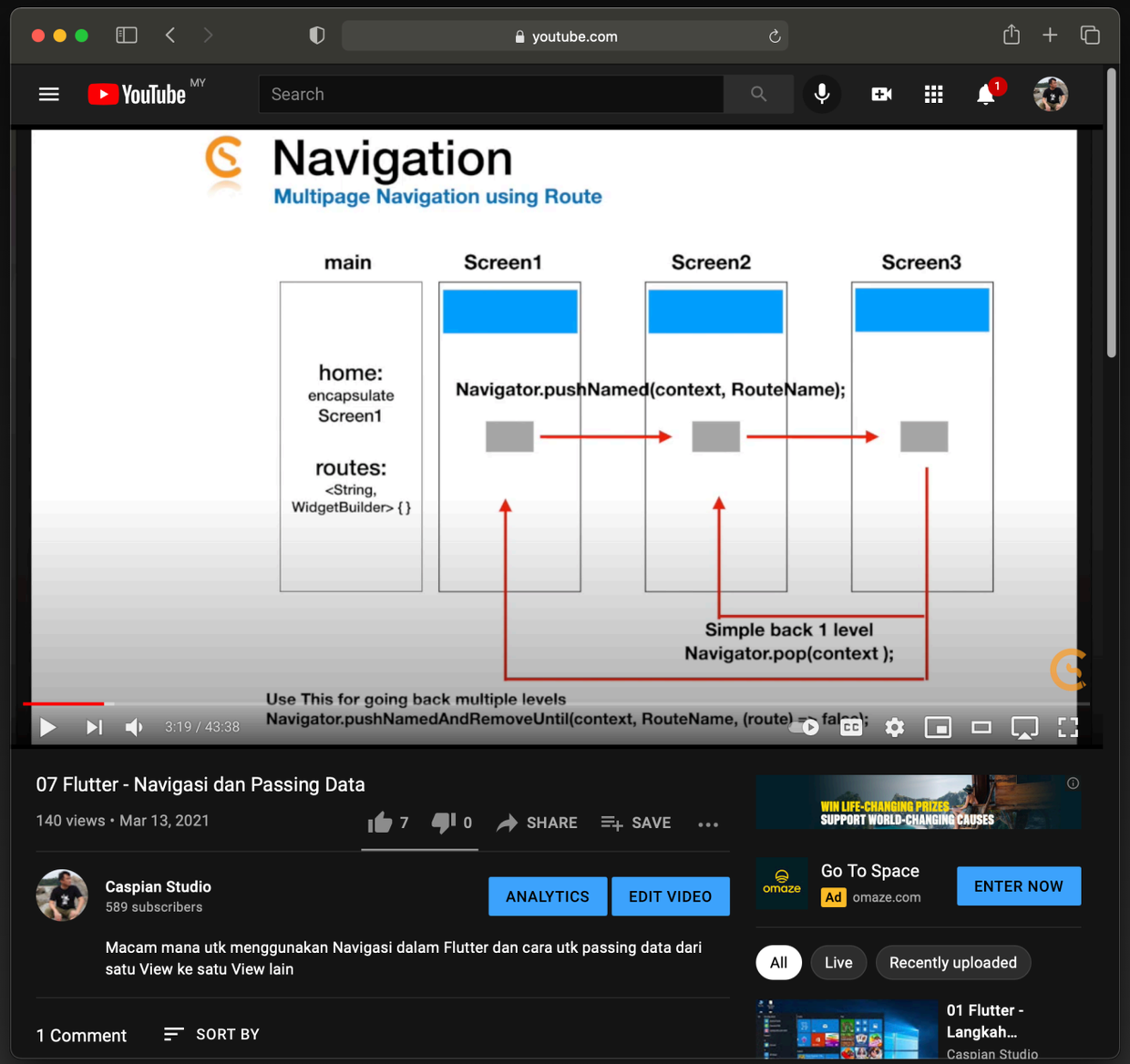


Passing single data, Can use Parameter or Argument: Pass

For passing multiple data, then create a Class and define the data as a Class Object

Let's Navigate & Pass Data

Video how to do Navigation & Passing Parameter



Refer on: https://www.youtube.com/watch?v=8EZkpyoufeU?sub_confirmation=1

If we have many Data

Array

When you have many Data

Use List as an array:

- List<type> arrayname = [item, item, item...];
- arrayname.Add(item)
- arrayname.RemoveAt(index)

- Can use Dismissible widget if you want animated list when deleting (do ListView first).

ListView

Creating a ListView manually

You create manually:

```
ListView(
  children: <Widget>[
    ListTile(
      leading: Icon(Icons.map),
      title: Text('Map'),
    ),
    ListTile(
      leading: Icon(Icons.photo_album),
      title: Text('Album'),
    ),
    ListTile(
      leading: Icon(Icons.phone),
      title: Text('Phone'),
    ),
  ],
);
```

Refer to https://flutter.dev/docs/cookbook#lists

ListView Builder

Creating a ListView Dynamically

You create using a Builder:

```
ListView.builder(
  itemCount: items.length,
  itemBuilder: (context, index) {
    return ListTile(
      title: Text(items[index]),
    );
  },
)
```

Refer to https://flutter.dev/docs/cookbook#lists

FutureBuilder & StreamBuilder

Understanding these 2 Builders

- FutureBuilder for single event
- StreamBuilder for multiple events
- Async & Await for Asynchronous operations

Future Builder

Understanding Future

```
FutureBuilder(
  future: http.get('http://awesome.data')
    builder: (context, snapshot) {
    if (snapshot.connectionState ==
        ConnectionState.done) {
      return AwesomeData(snapshot.data);
    } else {
      return CircularProgressIndicator();
  }
}
```

Future Builder

Understanding Future

FutureBuilder is a Widget that will help you to execute some asynchronous function and based on that function's result your UI will update.

Why you will use FutureBuilder?

- If you want to render widget after async task then use it.
- Handle loading process by simply using ConnectionState.waiting
- Don't need any custom error controller. Can handle error simply dataSnapshot.error != null

Future Builder

Understanding Future

When using FutureBuilder widget we need to check for future state i.e future is resolved or not and so on. There are various State as follows:

- **ConnectionState.none**: It means that the future is null and initialData is used as defaultValue.
- **ConnectionState.active**: It means the future is not null but it is not resolved yet.
- **ConnectionState.waiting**: It means the future is being resolved, and we will get the result soon enough.
- **ConnectionState.done**: It means that the future has been resolved.

Stream Builder

Understanding Stream

The StreamBuilder that can listen to exposed streams (continuous flow of data) and return widgets and capture snapshots of received stream data. We will deal with Stream in Firebase training.

The stream builder takes 3 arguments:

- initial:
- stream:
- builder:
  which can convert elements of the stream into widgets

JSON, REST API & Flutter

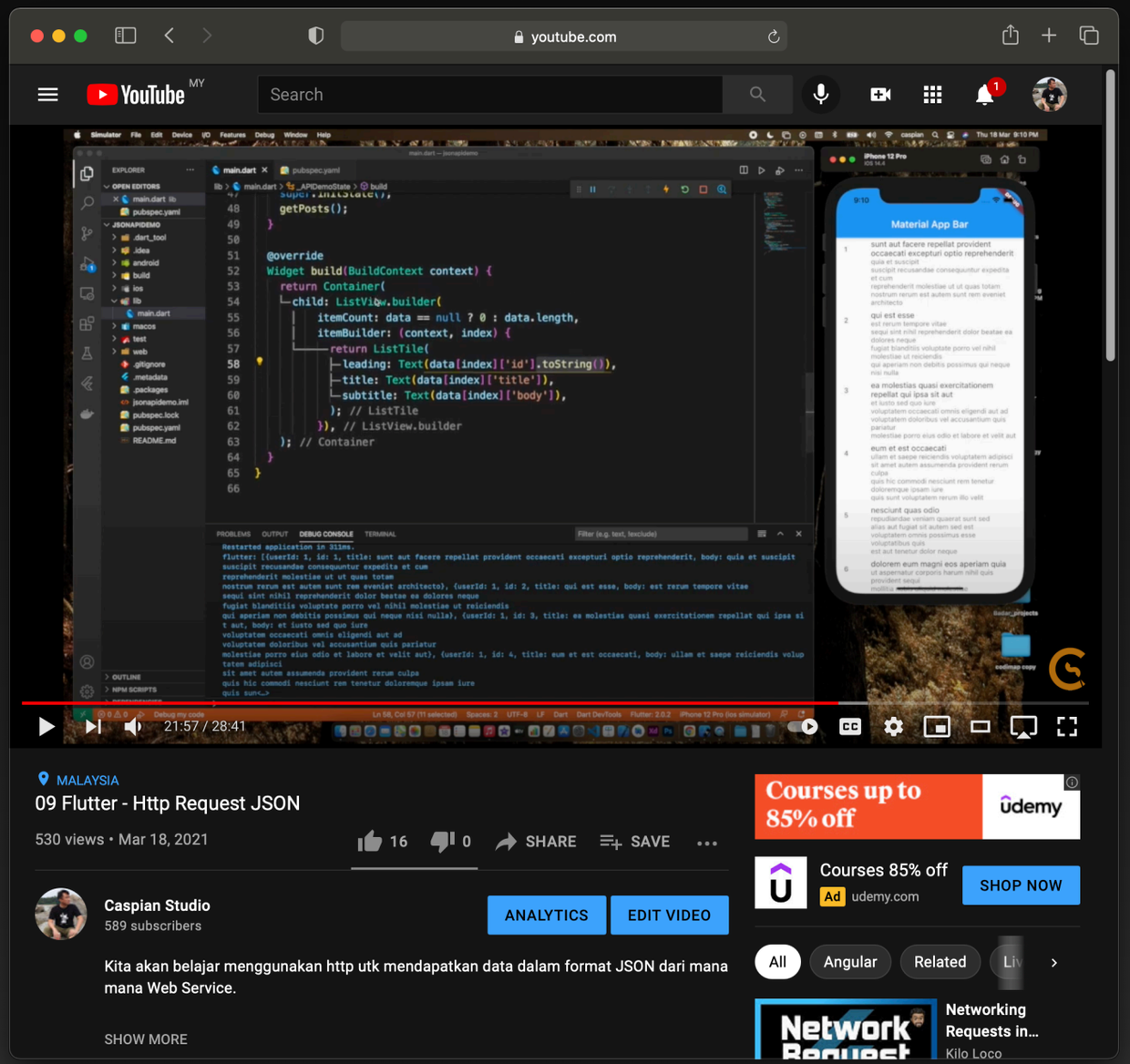
Getting Data from the Internet via API

Steps:

- Install **http** package in **pub.dev**
- Get JSON as the data source
- Do a conversion
- Use **FutureBuilder** or **StreamBuilder**

JSON, REST API & Flutter

Video how it's done



Refer to https://www.youtube.com/watch?v=jH8l-VvLffk?sub_confirmation=1

WebView

WebView & Flutter

Web in App

Steps:

- Install **WebView_flutter** or f**lutter_inappwebview** packages in **pub.dev**
- **Webview Cookie Manager** for Cookies issues with WebView
- Get URL
- Code
- Ensure AndroidManifest and InfoPlist permission
- Min Android 19 & iOS 11

WebView & Flutter

Code Sample for WebView

Code:

```
//declare
InAppWebViewController? controller;

//code
InAppWebView(
  initialUrlRequest: URLRequest(url: Uri.parse('https://www.caspian.my')),
  initialOptions: InAppWebViewGroupOptions(
     crossPlatform: InAppWebViewOptions(
      javaScriptEnabled: true,
      useOnDownloadStart: true,
      ),
     ),
   onWebViewCreated: (controller) {
   this.controller = controller;
},
```

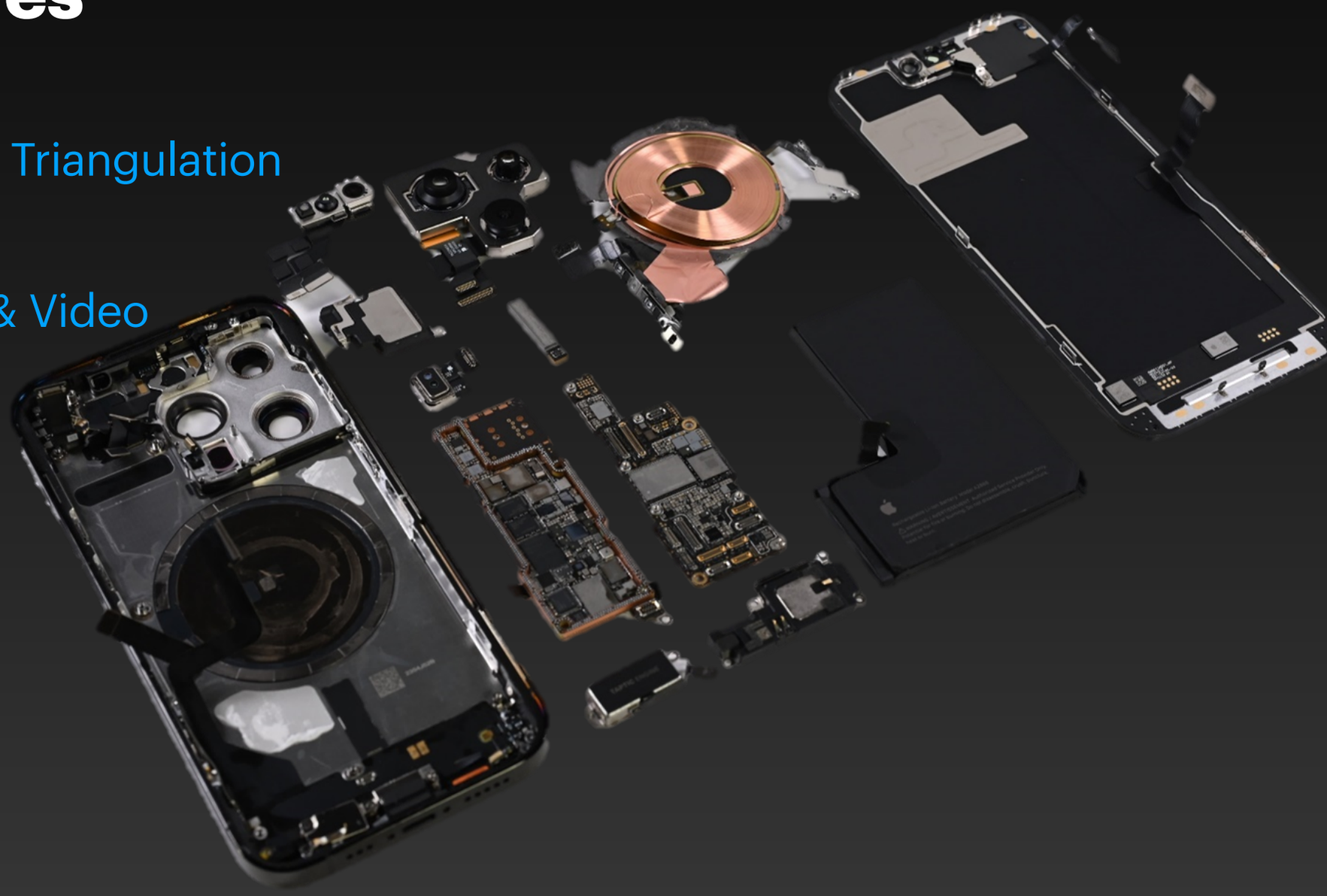
Device Features

Device Features

Smart Features in Mobile Apps

Normal Features

- Location
  - GPS, Wifi & Tower Triangulation
- Camera
  - Capturing Photo & Video
- Audio
  - Mp3

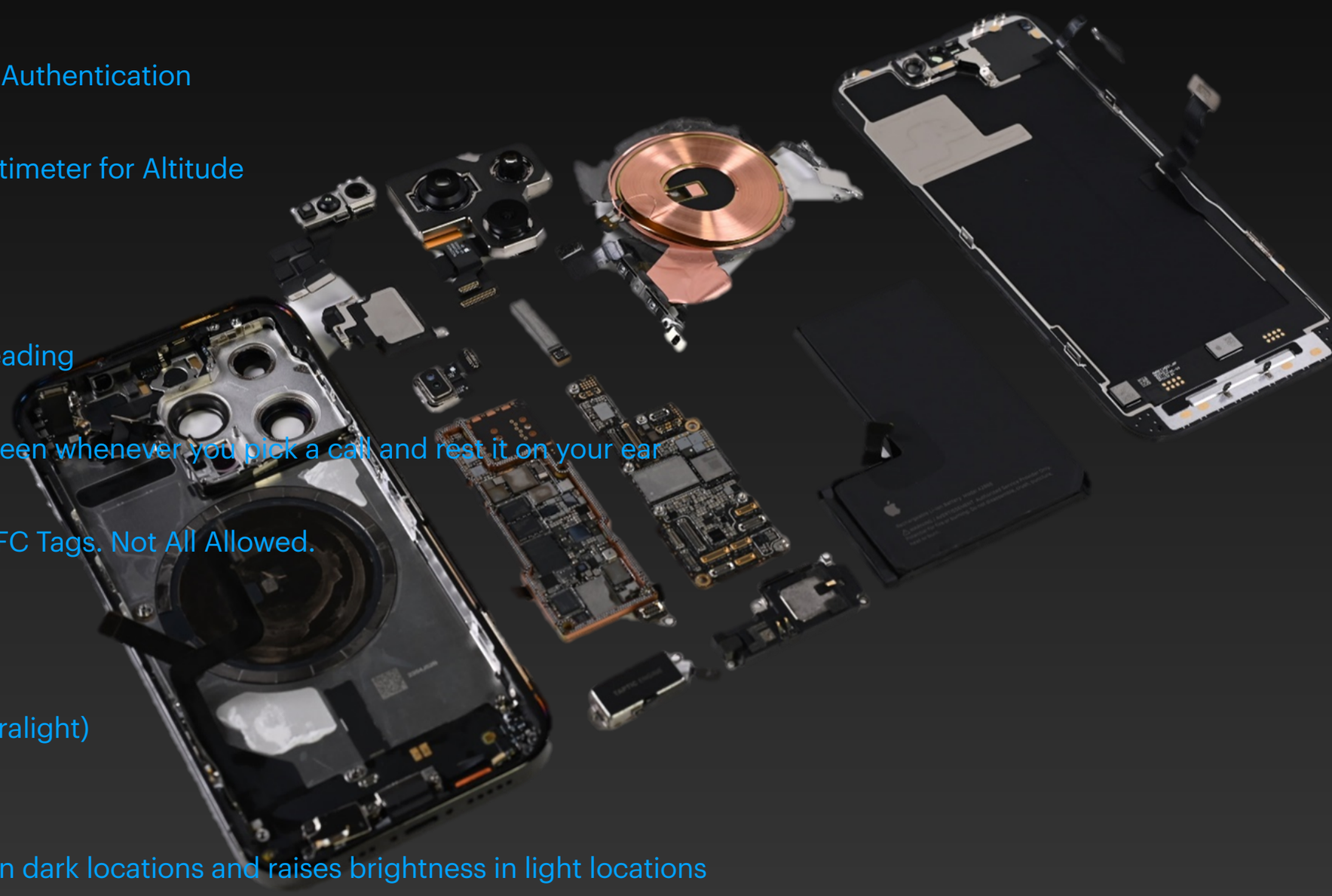


Device Features

Smart Features in Mobile Apps

Smart Sensors

- Face ID
  - For Security Features, Login & Authentication
- Barometer
  - Information: air Pressure, as Altimeter for Altitude
- Three-axis gyro (gyroscope)
  - Information: X,Y,Z Vibration
- Accelerometer
  - Information : Acceleration, Heading
- Proximity sensor
  - To assists in turning off the screen whenever you pick a call and rest it on your ear
- NFC
  - Since 2014 (iPhone 6), Read NFC Tags. Not All Allowed.
    - NTAG®
    - ICODE®
    - FeliCa™
    - MIFARE® (Desfire, Plus, Ultralight)
- LIDAR camera
- Ambient light sensor
  - The sensor lowers brightness in dark locations and raises brightness in light locations



Flutter Map

Creating a Simple Map

Steps To Create Google Maps:

- Use Package “GoogleMap for Flutter”
- Get an API key at https://cloud.google.com/maps-platform/
- Add API Keys in AndroidManisfest & InfoPlist
- Code in Flutter:
  - Create GoogleMapController
  - Add GoogleMap() and it’s attributes

Flutter Map

Flutter Simple Map Code

```
//declaration
late GoogleMapController mapController;
static const LatLng _center = LatLng(2.91667, 101.7);

//put in body

Center(
    Child: GoogleMap(
       onMapCreated: (controller) {
          mapController = controller;
        },
       initialCameraPosition: const CameraPosition(
       target: _center,
       zoom: 12.0,
      ),
    ),

),
```

Flutter GPS Location

Flutter Simple Map Code

//Import 3rd Package Library

```
import 'package:geolocator/geolocator.dart';
```

//declaration

```
LatLng? _currentLocation;
String _locationText = ‘’;
```

//create a function

```
 void getCurrentLocation() async {
    try {
      Position position = await Geolocator.getCurrentPosition(
        desiredAccuracy: LocationAccuracy.high,
      );
      setState(() {
        _currentLocation = LatLng(position.latitude, position.longitude);
        _locationText =
            'Latitude: ${_currentLocation!.latitude.toStringAsFixed(6)}, Longitude: $
{_currentLocation!.longitude.toStringAsFixed(6)}';
      });
         } catch (e) {
      print('Failed to get current location: $e');
    }
  }
```

//put in Text

```
Text(_locationText)
```

Flutter Camera

Flutter Camera Using ImagePicker

Steps To Camera:

- Use Package “Image Picker”
- Create a Button
- Create Function to User Image Picker
- Make sure Modify AndroidManifest & InfoPlist to allow Permission

Flutter Camera

Flutter Camera Using ImagePicker Code

```dart
//Import 3rd Party Library
import 'package:image_picker/image_picker.dart';

//declare
File? imageFile;
final ImagePicker picker = ImagePicker();

//create a  generalise function
Future<void> pickImage(ImageSource source) async {
    final pickedFile = await picker.pickImage(
      source: source,
      imageQuality: 25,
    );
    if (pickedFile != null) {
      setState(() {
        imageFile = File(pickedFile.path);
      });
    }
  }
```

Flutter Camera

Flutter Camera Using ImagePicker Code

```
//Use in a Button
ElevatedButton(
   onPressed: () => pickImage(ImageSource.camera),
   child: const Text(' Take Photo '),
),
```

```
ElevatedButton(
   onPressed: () => pickImage(ImageSource.gallery),
   child: const Text(' Photo Album '),
),
```

Animations

Flutter Animation

Basic Transitions & Animating Widgets

Transitions:

- SlideTransition, ScaleTransition, RotationTransition, SizeTransition, FadeTransition, PositionedTransition, RelativePositionedTransition, DecoratedBoxTransition, AlignTransition, DefaultTextStyleTransition
- Just use **Page_Transition** package

Animated Widgets:

- AnimatedCrossFade, AnimatedContainer, AnimatedPadding, AnimatedAlign, AnimatedPositioned, AnimatedPositionedDirectional, AnimatedOpacity, AnimatedDefaultTextStyle, AnimatedPhysicalModel
- Just use **TweenAnimationBuilder** and **AnimatedBuilder**

Flutter Animation

Transitions with page_transition

```
Navigator.push(context, PageTransition(type: PageTransitionType.fade, child: DetailScreen()));
Navigator.push(context, PageTransition(type: PageTransitionType.leftToRight, child: DetailScreen()));
Navigator.push(context, PageTransition(type: PageTransitionType.rightToLeft, child: DetailScreen()));
Navigator.push(context, PageTransition(type: PageTransitionType.rightToLeft, child: DetailScreen(), isIos: true));
Navigator.push(context, PageTransition(type: PageTransitionType.topToBottom, child: DetailScreen()));
Navigator.push(context, PageTransition(type: PageTransitionType.bottomToTop, child: DetailScreen()));
Navigator.push(context, PageTransition(type: PageTransitionType.scale, alignment: Alignment.bottomCenter, child: DetailScreen()));
Navigator.push(context, PageTransition(type: PageTransitionType.size, alignment: Alignment.bottomCenter, child: DetailScreen()));
Navigator.push(context, PageTransition(type: PageTransitionType.rotate, duration: Duration(second: 1), child: DetailScreen()));
Navigator.push(context, PageTransition(type: PageTransitionType.rightToLeftWithFade, child: DetailScreen()));
Navigator.push(context, PageTransition(type: PageTransitionType.leftToRightWithFade, child: DetailScreen()));
Navigator.push(context, PageTransition(type: PageTransitionType.leftToRightJoined, child: DetailScreen(), childCurrent: this));
Navigator.push(context, PageTransition(type: PageTransitionType.rightToLeftJoined, child: DetailScreen(), childCurrent: this));
Navigator.push(context, PageTransition(type: PageTransitionType.topToBottomJoined, child: DetailScreen(), childCurrent: this));
Navigator.push(context, PageTransition(type: PageTransitionType.bottomToTopJoined, child: DetailScreen(), childCurrent: this));
Navigator.push(context, PageTransition(type: PageTransitionType.leftToRightPop, child: DetailScreen(), childCurrent: this));
Navigator.push(context, PageTransition(type: PageTransitionType.rightToLeftPop, child: DetailScreen(), childCurrent: this));
Navigator.push(context, PageTransition(type: PageTransitionType.topToBottomPop, child: DetailScreen(), childCurrent: this));
Navigator.push(context, PageTransition(type: PageTransitionType.bottomToTopPop, child: DetailScreen(), childCurrent: this));
```

Flutter Animation

Transitions with page_transition

Page 1

```
class Page1 extends StatelessWidget {
  const Page1({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.of(context).push(_createRoute());
          },
          child: const Text('Go!'),
        ),
      ),
    );
  }
}

Route _createRoute() {
  return PageRouteBuilder(
    pageBuilder: (context, animation, secondaryAnimation) => const Page2(),
    transitionsBuilder: (context, animation, secondaryAnimation, child) {
      return child;
    },
  );
}
```

Flutter Animation

Transitions with page_transition

Page 2

```
class Page2 extends StatelessWidget {
  const Page2({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: const Center(
        child: Text('Page 2'),
      ),
    );
  }
}
```

Flutter Animation

TweenAnimationBuilder and AnimatedBuilder

Here's one example that uses TweenAnimationBuilder to animate a Container's color from red to green

```
TweenAnimationBuilder<Color>(
  tween: Tween<Color>(begin: Colors.red, end: Colors.green),
  duration: const Duration(seconds: 1),
  // child is *optional* so we can pass null or omit it
  child: null,
  // builder is *required*
  // note that the third argument is an (optional) child
  builder: (BuildContext context, Color color, Widget? child) {
    return Container(color: color);
  },
)
```

Flutter Animation

TweenAnimationBuilder and AnimatedBuilder

Here's one example that uses AnimatedBuilder to animate a Container's rotating:

```
AnimatedBuilder(
  animation: someAnimation,
  // pass a child widget
  child: Container(color: Colors.red),
  // get the child back as a (nullable) Widget object
  builder: (BuildContext context, Widget? child) {
    // this rebuilds on every animation tick
    return Transform(
      alignment: Alignment.center,
      transform: Matrix4.rotationZ(2 * pi * someAnimation.value),
      // this is only built once
      child: child,
    );
  },
)
```

Local Data Sqlite

SQLite

How to use Sqlite in Flutter

Flutter apps can make use of the SQLite databases via the **sqflite** package available on pub.dev.

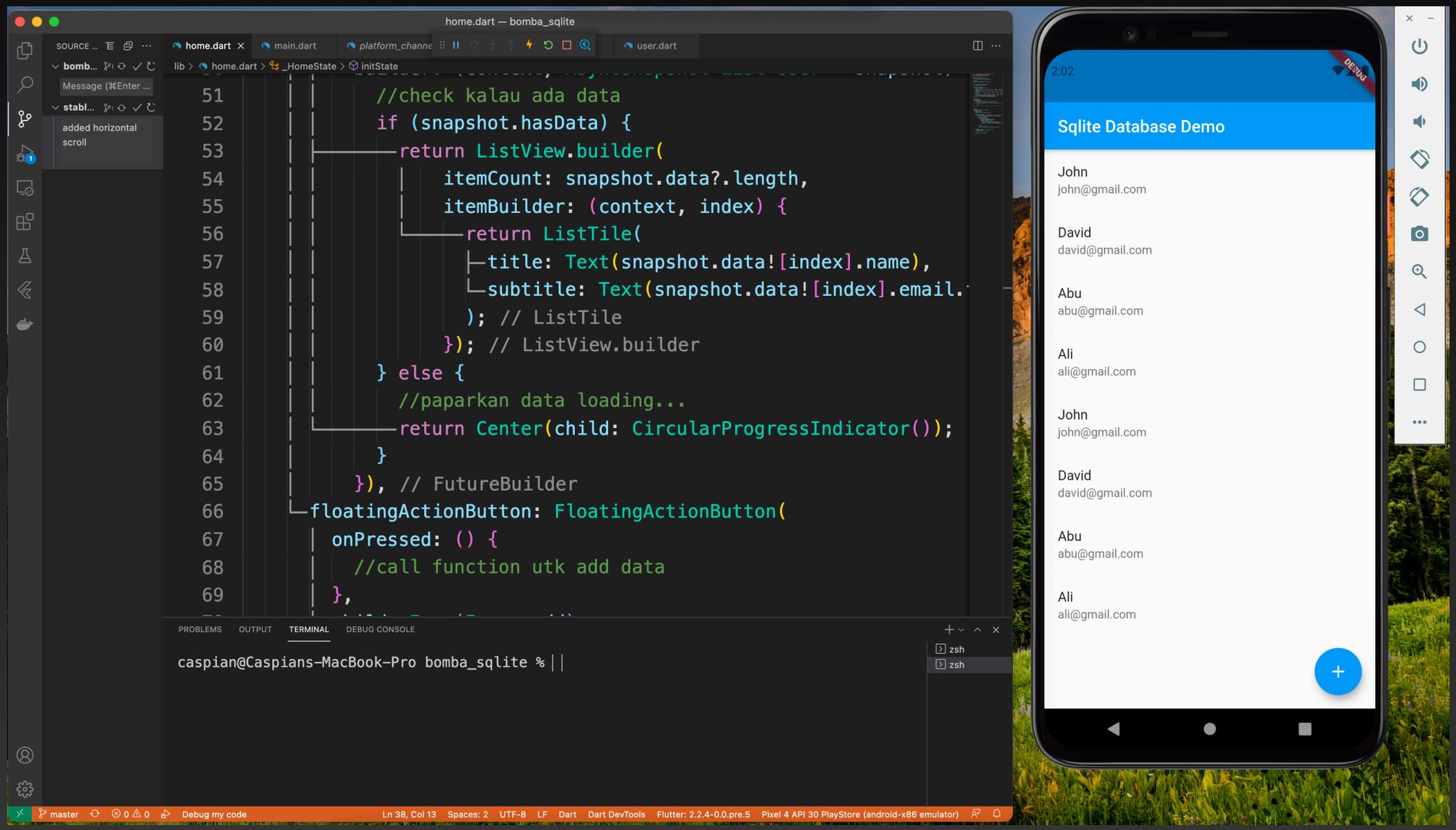
If you are new to SQLite and SQL statements, review the SQLite Tutorial to learn the basics first.

Steps:

- Add the **sqflite** dependencies in pubspec.yaml
- Define the data model.
- Open the database.
- Create the model table.
- Do CRUD - Create, Read, Update, Delete

SQLite

Our end Product



Best Practices

Best Practices

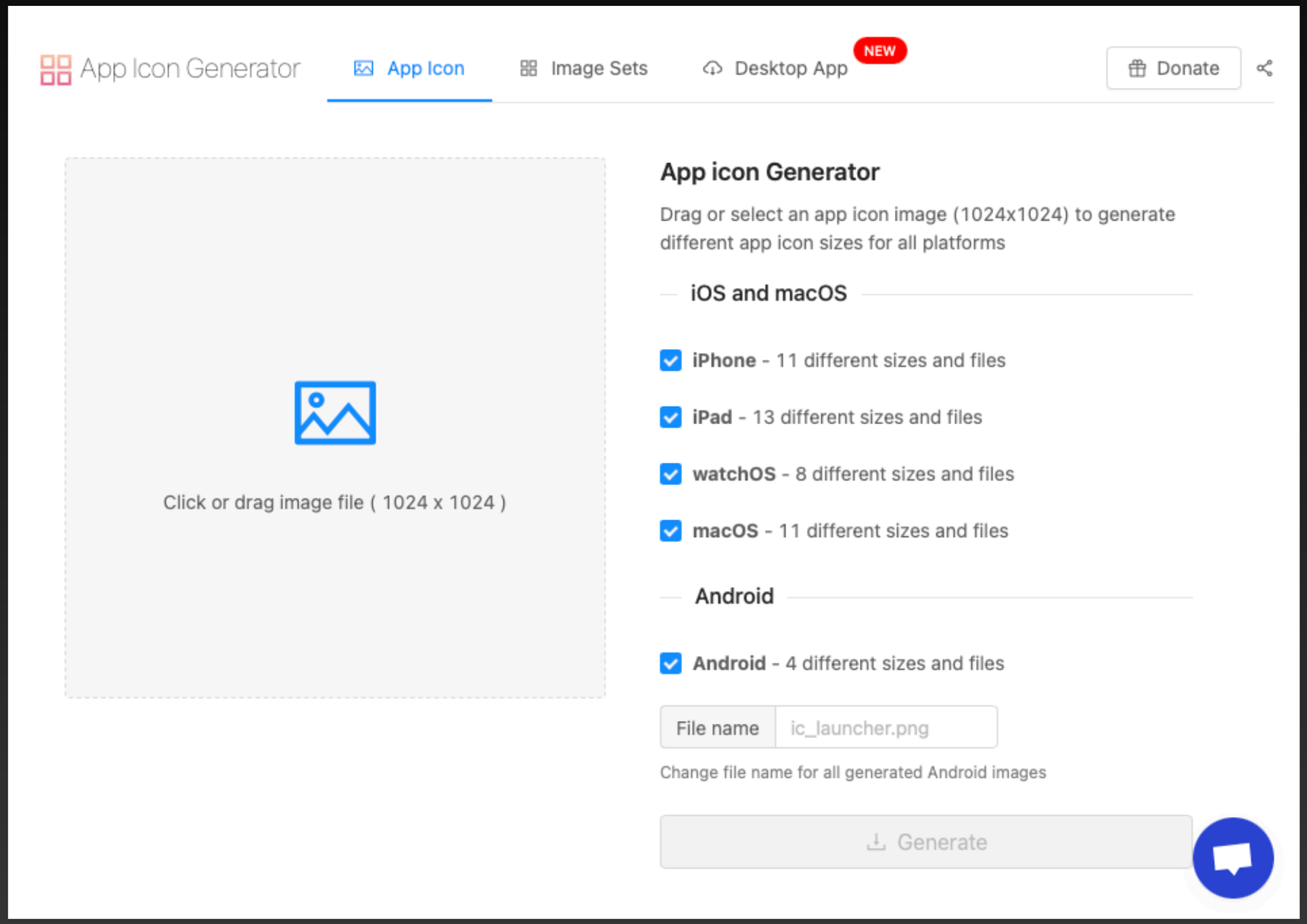
Tips on Flutter Codes

- Use the const keyword whenever possible
- Define constant with leading “k”
- Use commas to intend your code
- Follow naming convention
- Use the right widget
- Use private variable/method whenever possible
- Proper use of setState()
- Avoid deep trees instead create a separate widget
- Avoid method implementation within widget instead create a separate method
- Avoid using nullable unless it is nullable
- Use cascade (..)
- Use spread operator (...)
- define theme/route in a separate file
- avoid using hardcoded strings (internationalised your app)
- avoid using hardcoded style, decoration, etc.
- Use relative import over package import within your app
- Use a stateless widget whenever it is possible
- Use flutter lint

Build App

How to Create Your Own Icons

For Beginners to create Android & iOS Icons



Goto: https://appicon.co

How to build APK, AAB or IPA

Using Flutter Cli on your Terminal or Console

Just type in the Terminal:

flutter build apk or
flutter build appbundle or
Flutter build ipa or
Flutter build web or
Flutter build windows

You can also compile in:
Android Studio for Android apps
Xcode for iOS Apps

Submission Criteria

Checklists

Android:

- Paid Developer ID
- Valid keystore for your App
- All Icons & Graphics
- Version & App ID
- Signed apk or Appbundle

iOS:

- Paid Apple Developer ID
- All Icons & Graphics
- Version & App ID

Android Deployment

Steps to Deployment

Goto: https://docs.flutter.dev/deployment/android

iOS Deployment

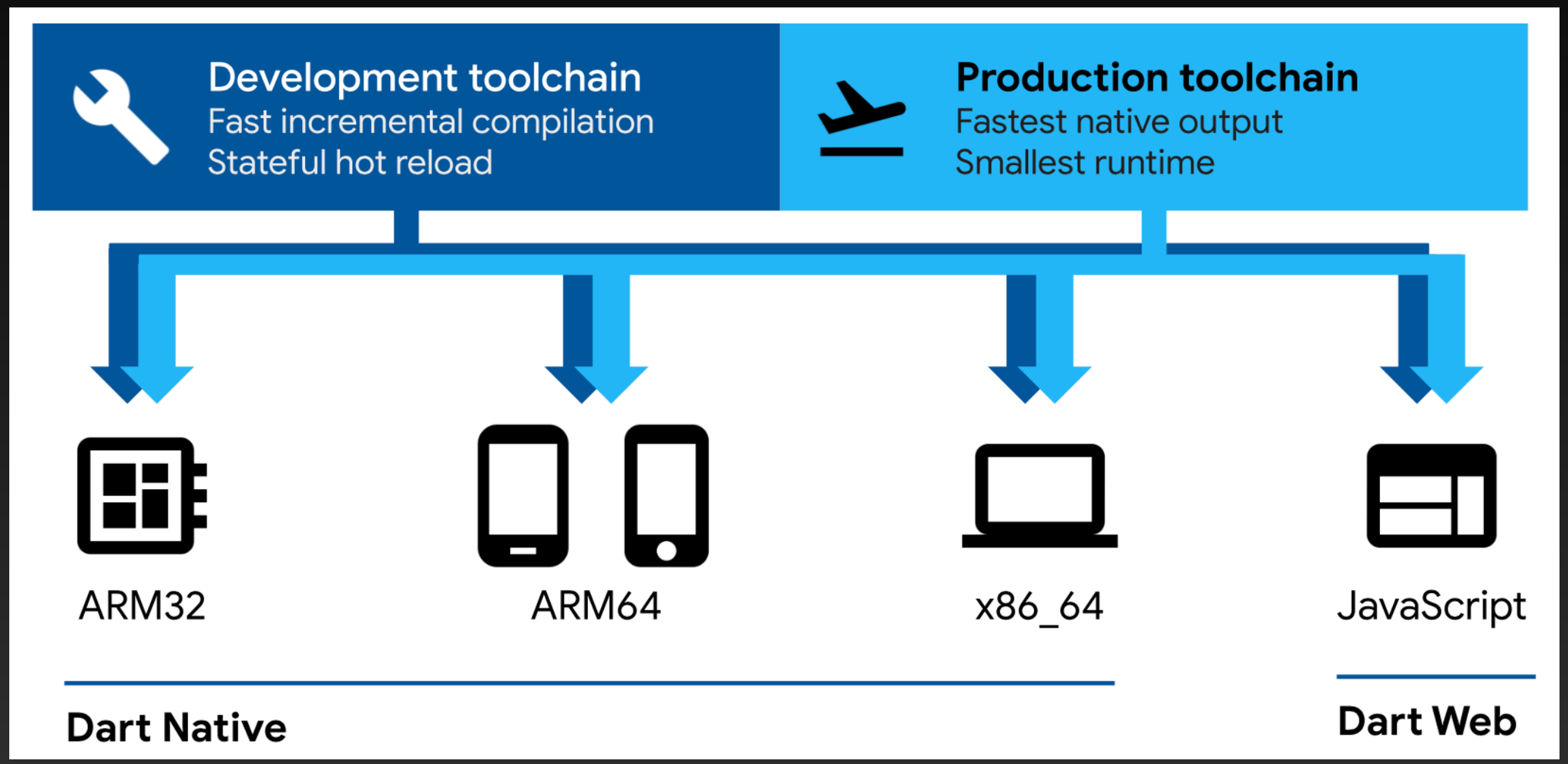
Steps to Deployment

Goto: https://docs.flutter.dev/deployment/ios

Dart Language

What is Dart Language?

Platforms: Dart Native & Dart Web



Dart Native & Dart Web

Under the hood

Native platform:
For apps targeting mobile and desktop devices, Dart includes both a Dart VM with just-in-time (JIT) compilation and an ahead-of-time (AOT) compiler for producing machine code.

Web platform:
For apps targeting the web, Dart includes both a development time compiler (dartdevc) and a production time compiler (dart2js). Both compilers translate Dart into JavaScript.

Important Concepts

You need to know

- Everything you can place in a variable is an object, and every object is an instance of a class.
- Dart is strongly typed, type annotations are optional.
- If you enable null safety, variables can’t contain null unless you say they can. You can make a variable nullable by putting a question mark (?) at the end of its type. If you know that an expression never evaluates to null but Dart disagrees, you can add ! to assert that it isn’t null (and to throw an exception if it is).
- Dart supports generic types, like List<int> (a list of integers) or List<Object> (a list of objects of any type).
- Dart supports top-level functions (such as main()), as well as functions tied to a class or object (static and instance methods, respectively). You can also create functions within functions (nested or local functions).
- Similarly, Dart supports top-level variables, as well as variables tied to a class or object (static and instance variables). Instance variables are sometimes known as fields or properties.
- Dart doesn’t have the keywords public, protected, and private. If an identifier starts with an underscore (_), it’s private to its library.
- Dart has both expressions (which have runtime values) and statements (which don’t).
  For example, the conditional expression condition ? expr1 : expr2 has a value of expr1 or expr2.
  Compare that to an if-else statement, which has no value.
- Dart tools can report two kinds of problems: warnings and errors.

Dart Libraries

Dart has a rich set of core libraries

- **dart:core**
  Built-in types, collections, and other core functionality. This library is automatically imported into every Dart program.
- **dart:collection**
  Richer collection types such as queues, linked lists, hashmaps, and binary trees
- **dart:convert**
  Encoders and decoders for converting between different data representations, including JSON and UTF-8.
- **dart:io**
  File, socket, HTTP, and other I/O support for non-web applications
- **dart:async**
  Support for asynchronous programming, with classes such as Future and Stream
- **dart:typed_data**
  Lists that efficiently handle fixed-sized data (for example, unsigned 8-byte integers) and SIMD numeric types
- **dart:math**
  Mathematical constants and functions, plus a random number generator
- **dart:isolate**
  Concurrent programming using isolates — independent workers that are similar to threads but don’t share memory, communicating only through messages
- **dart:HTML**
  HTML elements and other resources for web-based applications that need to interact with the browser and the Document Object Model (DOM)
- **Supplementary packages** such as characters, intl, http, crypto, markdown, XML, Windows Integration, SQLite, compression

Sound null safety

types in your code are non-nullable by default

- The Dart language now supports sound null safety!
- Dart Version 2.12 and above
- Meaning that variables can’t contain null unless you say they can
- You can make a variable nullable by putting a question mark (?) at the end of its type. If you know that an expression never evaluates to null but Dart disagrees, you can add (!) to assert that it isn’t null

Learn Dart

Learning Dart is easy

- Similar to C, Javascript or Java
- Single Inheritance. Instead use Mixins
- Goto dart.dev website

Hello Dart

Learning Dart is easy

Hello World
Every app must have a main() function. To display text on the console, you can use print() function:

```
void main() {
  print('Hello, World!');
}
```

Variables
Even in type-safe Dart code, most variables don’t need explicit types, thanks to type inference:

```
var name = 'Voyager I';
var year = 1977;
var antennaDiameter = 3.7;
var flybyObjects = ['Jupiter', 'Saturn', 'Uranus', 'Neptune'];
var image = {
  'tags': ['saturn'],
  'url': '//path/to/saturn.jpg'
};
```

Dart Types

Learning Dart is easy

Build-In Types

The Dart language has special support for the following types:

- numbers
- strings
- booleans
- lists (also known as arrays)
- sets
- maps
- runes (for expressing Unicode characters in a string)
- symbols

Numbers

The Dart language has 2 flavors:

- int
- double

Control flow Statements

Dart supports the usual flow statements

```dart
if (year >= 2001) {
  print('21st century');
} else if (year >= 1901) {
  print('20th century');
}

for (var object in flybyObjects) {
  print(object);
}

for (int month = 1; month <= 12; month++) {
  print(month);
}

while (year < 2016) {
  year += 1;
}
```

Also in a widget, the conditional expression condition ? expr1 : expr2 has a value of expr1 or expr2. Compare that to an if-else statement

Example:

itemCount: data == null ? 0 : data.length,

Functions

Dart functions is similar to C/C++

```dart
int fibonacci(int n) {
  if (n == 0 || n == 1) return n;
  return fibonacci(n - 1) + fibonacci(n - 2);
}

var result = fibonacci(20);
```

A shorthand => (arrow) syntax is handy for functions that contain a single statement. This syntax is especially useful when passing anonymous functions as arguments:

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

Normally

```dart
void main() => runApp(new MyApp());
```

shorthand

Async Await Future

Dart functions is similar to C/C++

Async

async and await help make asynchronous code easy to read. Use Future< > in front of a function that uses async

```
Future<void> createDescriptions(Iterable<String> objects) async {
  for (var object in objects) {
    try {
      var file = File('$object.txt');
      if (await file.exists()) {
        var modified = await file.lastModified();
        print(
            'File for $object already exists. It was modified on $modified.');
        continue;
      }
      await file.create();
      await file.writeAsString('Start describing $object in this file.');
    } on IOException catch (e) {
      print('Cannot create description for $object: $e');
    }
  }
}
```

Async Await Future

Dart functions is similar to C/C++

Async*

Use async* to give a nice, readable way to build streams. Stream means data is continuous streaming. Use the word Stream< > in front of a function that does streaming.

```dart
Stream<String> report(Spacecraft craft, Iterable<String> objects) async* {
  for (var object in objects) {
    await Future.delayed(oneSecond);
    yield '${craft.name} flies by $object';
  }
}
```

Static, Final, Const

Similar but no quite the same

- "static", "final", and "const" mean entirely distinct things in Dart:
- "static" means a member is available on the class itself instead of on instances of the class. That's all it means, and it isn't used for anything else.
- "final" means single-assignment: a final variable or field *must* have an initializer. Once assigned a value, a final variable's value cannot be changed. final modifies *variables*.
- “const" means that the object's entire deep state can be determined entirely at compile time and that the object will be frozen and completely immutable.

Late

Sometime you don’t need it yet

late
Late variables this means that we can have non-nullable instance fields that are initialized later: Accessing value before it is initialized will throw a runtime error in null-safety

In other word, late runs “lazily”, which means it is not run at all until it is referenced the first time. That’s fine if you don’t specify an initial value

Drag & Drop

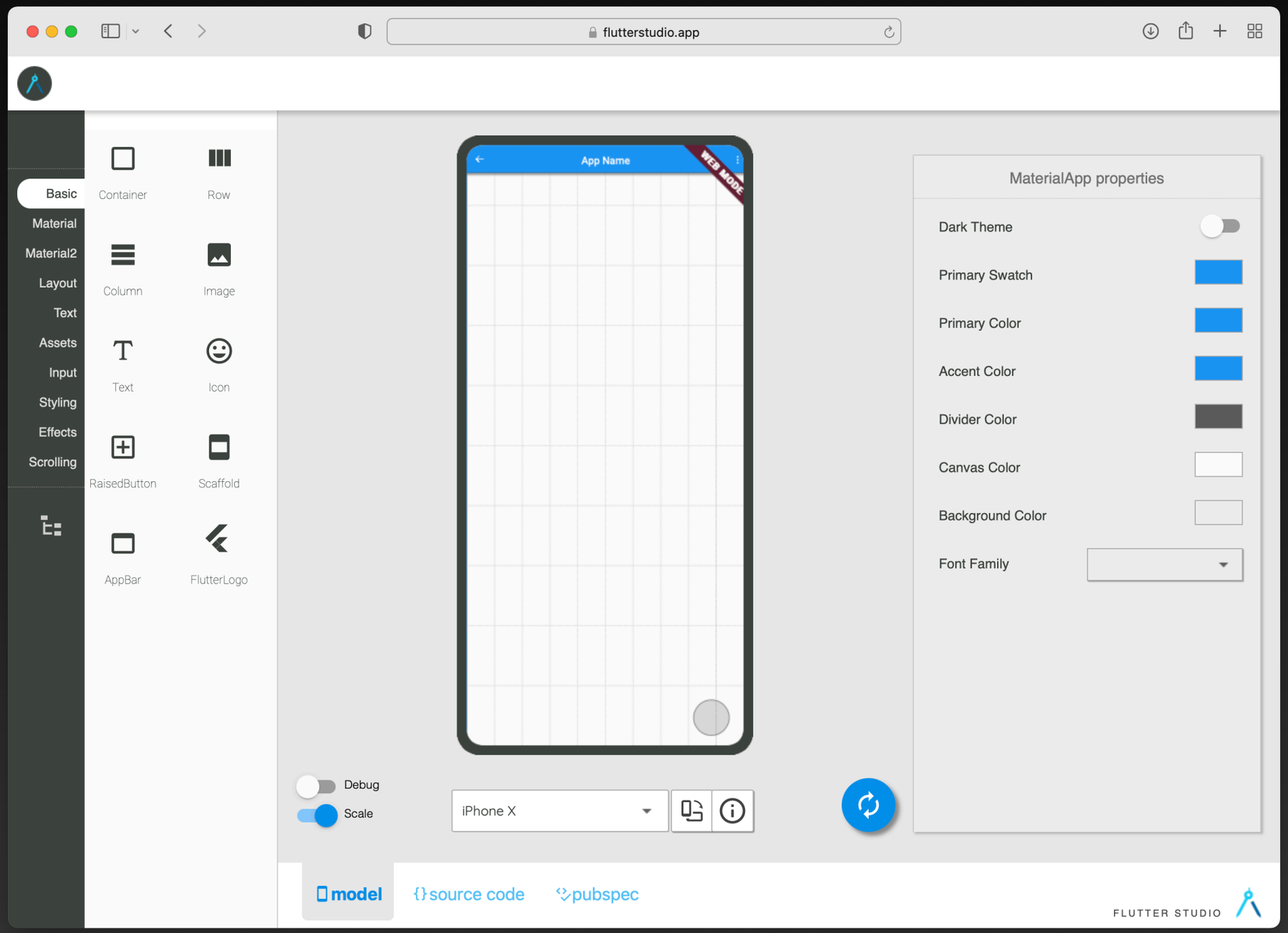
Tools for Flutter

Sometime you need help

There are a few good tools to help you in Flutter mobile apps development. These are a few tools…

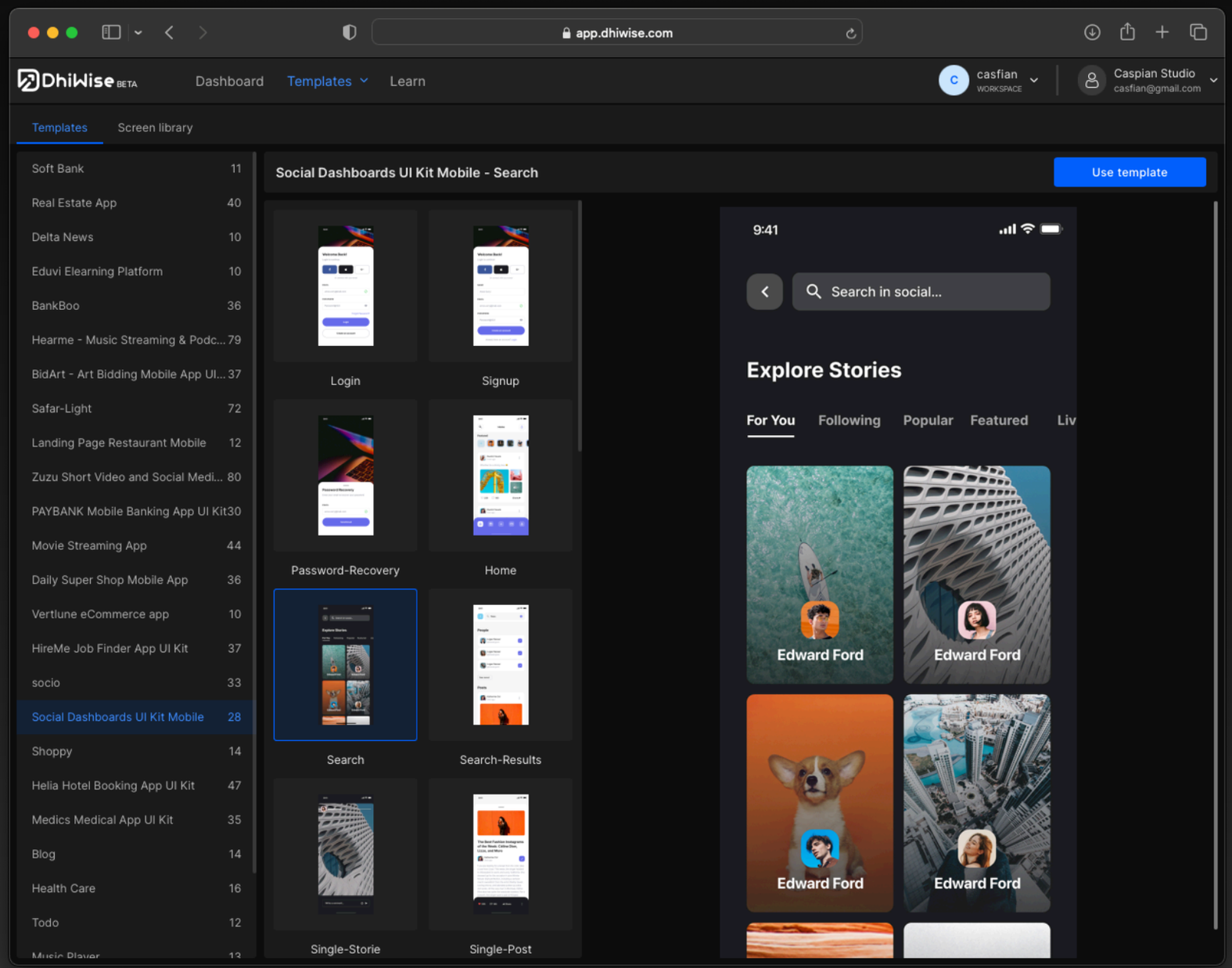
FlutterStudio

No-Code Tool



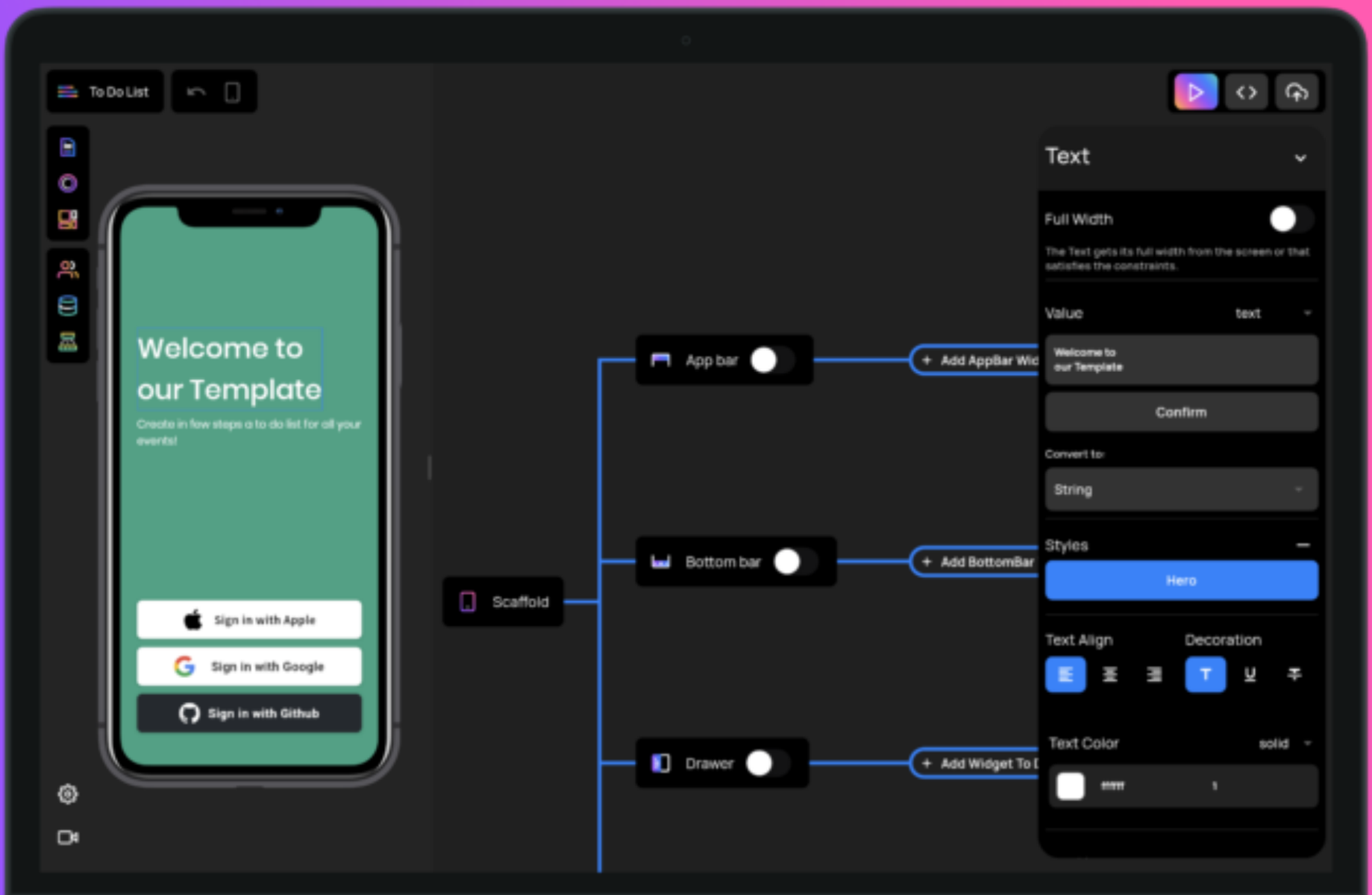
DhiWise Flutter

No-Code Tool derived template Figma/Sketch/XD



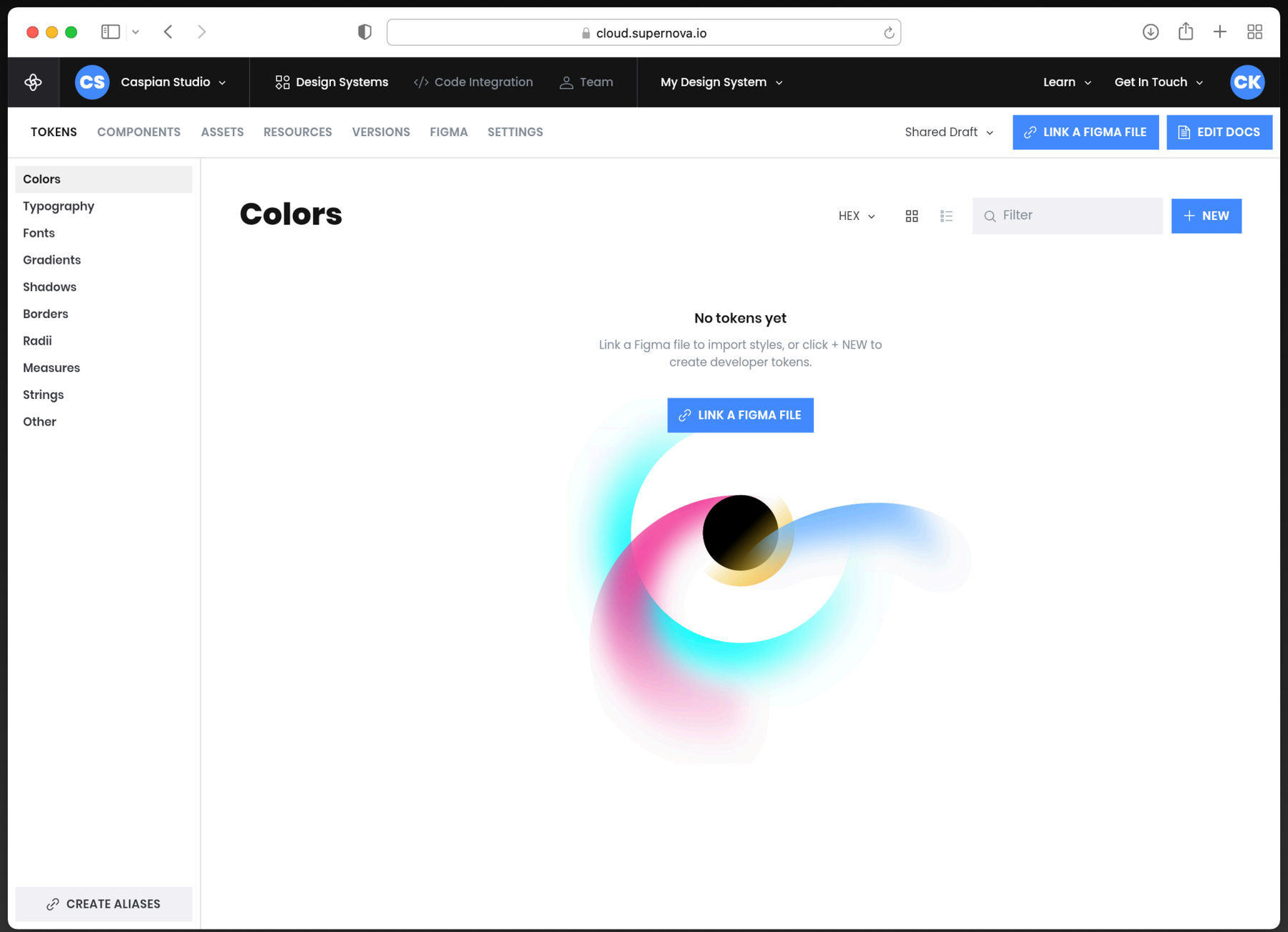
Teta.so

No-Code Tool



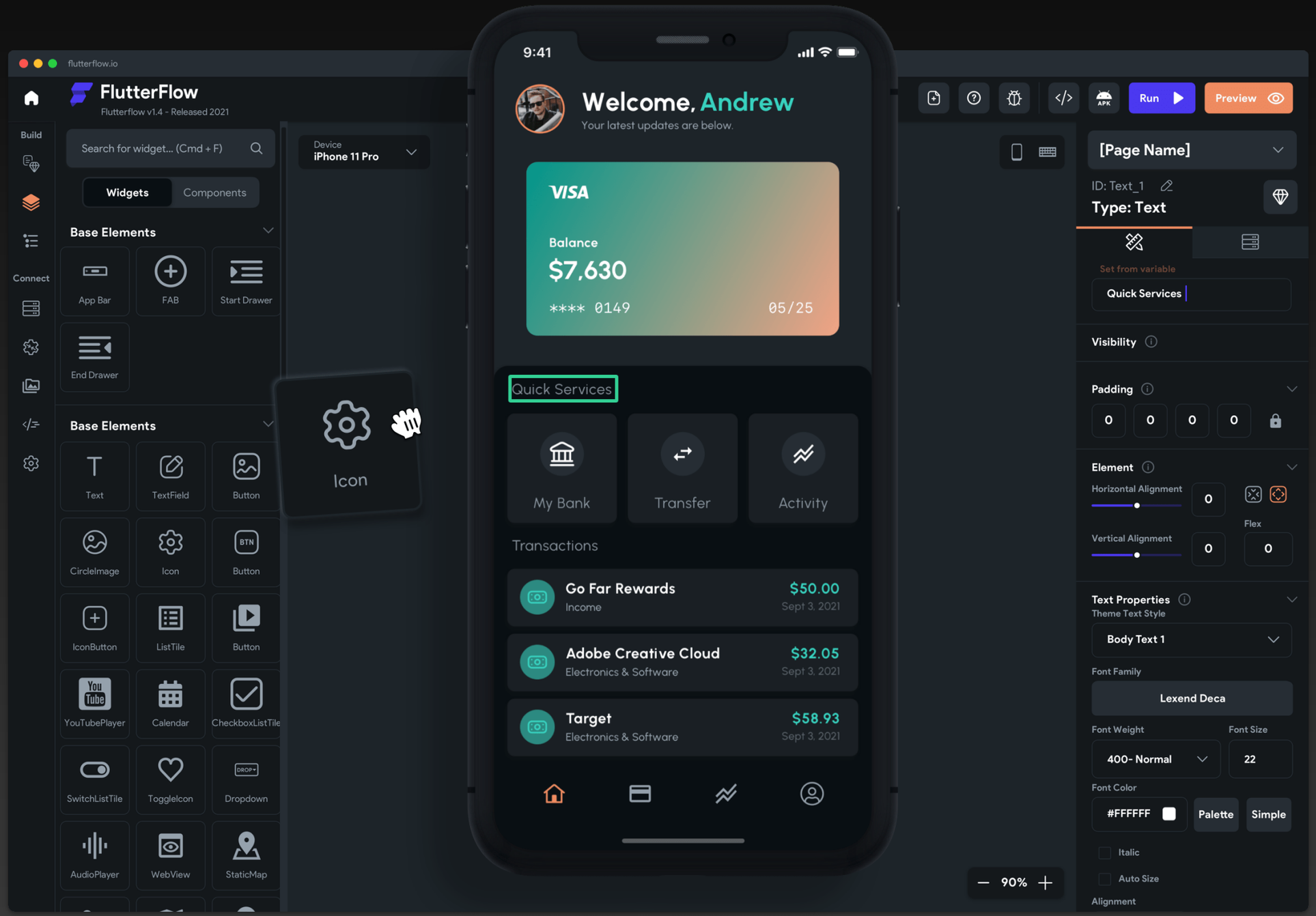
Supernova

From Prototyping to Code



FlutterFlow

No-Code Tool



FlutterFlow

No-Code Tool

We have Training on FlutterFlow

Contact: Caspian Consultancy Services