Flutter

BUILD APPS FOR ANY SCREEN

FROM: JONAS LIESKE, DANIEL WORBIS & MANUEL OSTERLOH

MODULE: ESD – ENTERPRISE SYSTEM DEVELOPMENT

AGENDA

- 1. Intro to Flutter and Dart
- 2. Flutter Basics
- 3. Advanced Techniques
- 4. Setup
- 5. Interactive Session
- 6. Q&A
- 7. Conclusion

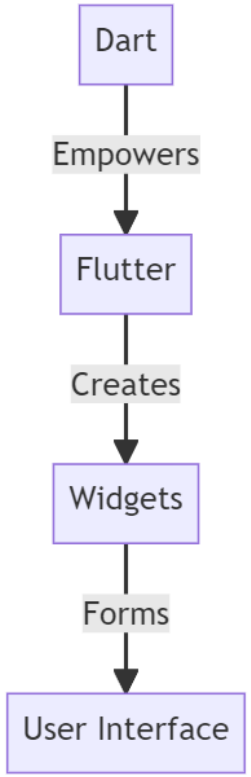
GitHub

tinyurl.com/esdflutter

INTRO TO FLUTTER AND DART

- What is Flutter and Dart?

- Flutter: Google's UI toolkit for natively compiled apps
- Mobile, Web & Desktop in a single codebase
- It offers
  - Cross platform development
  - Platform-specific code
  - Hot reload
  - Widgets
- Coded in Dart
  - Java + React Syntax
  - OOP
  - JIT- & AOT-compilation




Always has been
Flutter By Example
Wait it's all WIDGETS ?
imgflip.com

INTRO TO FLUTTER AND DART

- Why Flutter is NOT the Holy Grail

- Bloated Codebase
- Mobile != Web
  - iOS & Android ~95% the same
  - Web (desktop) has many different approaches
  - 1.2 – 1.5 * X still better than 3X
- Easy scaling and MVP → Native
- Flutter Web founders praise its app-centric services
- No hot reload for Flutter Web
- Chrome, Safari, Edge, Firefox support
- NO SEO!

INTRO TO FLUTTER AND DART

- Xamarin vs. Flutter vs. React Native

INTRO TO FLUTTER AND DART

- Xamarin vs. Flutter vs. React Native

		React Native	Xamarin
Developed by			
Programming Language	Dart	JS	
Performance	High (native ARM code)	Close to native	Native API leveraging
UI Design	- Material Design - Cupertino - Highly customizable	- Native-like experience - platform-specific components	- Access to full range of native features - High-quality UI
Third-Party Libraries	Growing ecosystem (flutter pub)	Extensive ecosystem (npm)	Shared code libraries across platforms
Community and Support	- Strong and active community - Extensive comprehendible documentation	- Large, active community - comprehensive documentation	- Smaller community - Part of .NET platform
Disadvantages	- Requires learning Dart - limited native modules	- Performance issues in complex tasks - Limited older device support	- .NET familiarity needed - Dependency on Visual Studio

INTRO TO FLUTTER AND DART

- Dart Special Features

Dart is **null-safe**

- Separates nullable from non-nullable types
- Compiler forces declaration of nullable types
- Reduces run-time errors

```
int? nullableInt; // Nullable type
int nonNullableInt; // Non-Nullable type

nullableInt! // Explicit non-nullable

nullableInt ?? 0; // int or 0

nullableString?.toLowerCase(); // string or null
```

INTRO TO FLUTTER AND DART

- Dart Special Features

Dart has extensions

- Allows adding new functionality to existing types without modifying the original class
- Useful for enhancing types from libraries where you don't have access to the source code

```
extension StringParsing on String {
   bool containsUppercase(String input){
      return input.contains(RegExp(r'[A-Z]'));
   }
}

void main() {
   String name = "Larry";
   print(name.containsUppercase(name));
}
```

INTRO TO FLUTTER AND DART

- Dart Special Features

Dart has a built-in builder pattern

- Use the same instance by using ".."
- No need to return instance on every function

```
var myButton = Button()
..color = Colors.blue
..text = "Click me"
..render();
```

INTRO TO FLUTTER AND DART

- Dart Special Features

Dart has simple access modifiers

- The default scope is public
- Private scope is being defined with an "_" at the beginning of the name
  => Forcing naming standard
- There are is no "protected" access scope

```
void _veryPrivateMethod() {
   print("Hello World");
}

void veryPublicMethod() {
   print("Hello World");
}
```

INTRO TO FLUTTER AND DART

- Dart Special Features

Dart has named constructors

- Less overloading of constructors
- Makes it possible to have various initialization scenarios

```
class Point {
   double x, y;

   // Normal constructor
   Point(this.x, this.y);

   // Named constructor
   Point.origin()
      : x = 0,
        y = 0;
}

Point(10,20); // x=10, y=20
Point.origin(); // x=0, y=0
```

INTRO TO FLUTTER AND DART

- Dart Special Features

Dart has optional + named parameters

- Named parameters can make code more readable
- Both optional and named parameters can have default values, used when an argument isn't provided.

```
class Person {
   String lastName;
   String occupation;

   Person(this.lastName,
      {this.occupation = 'unknown'}
   );
}

void main() {
   Person person1 = Person("Doe", 18);
   Person person2 = Person(
      "Doe", 18, occupation: "Project
Manager"
   );
}
```

FLUTTER BASICS

- What are Widgets?
- Widget Types

Flutter

Flutter Dev

FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Single-child layout widgets

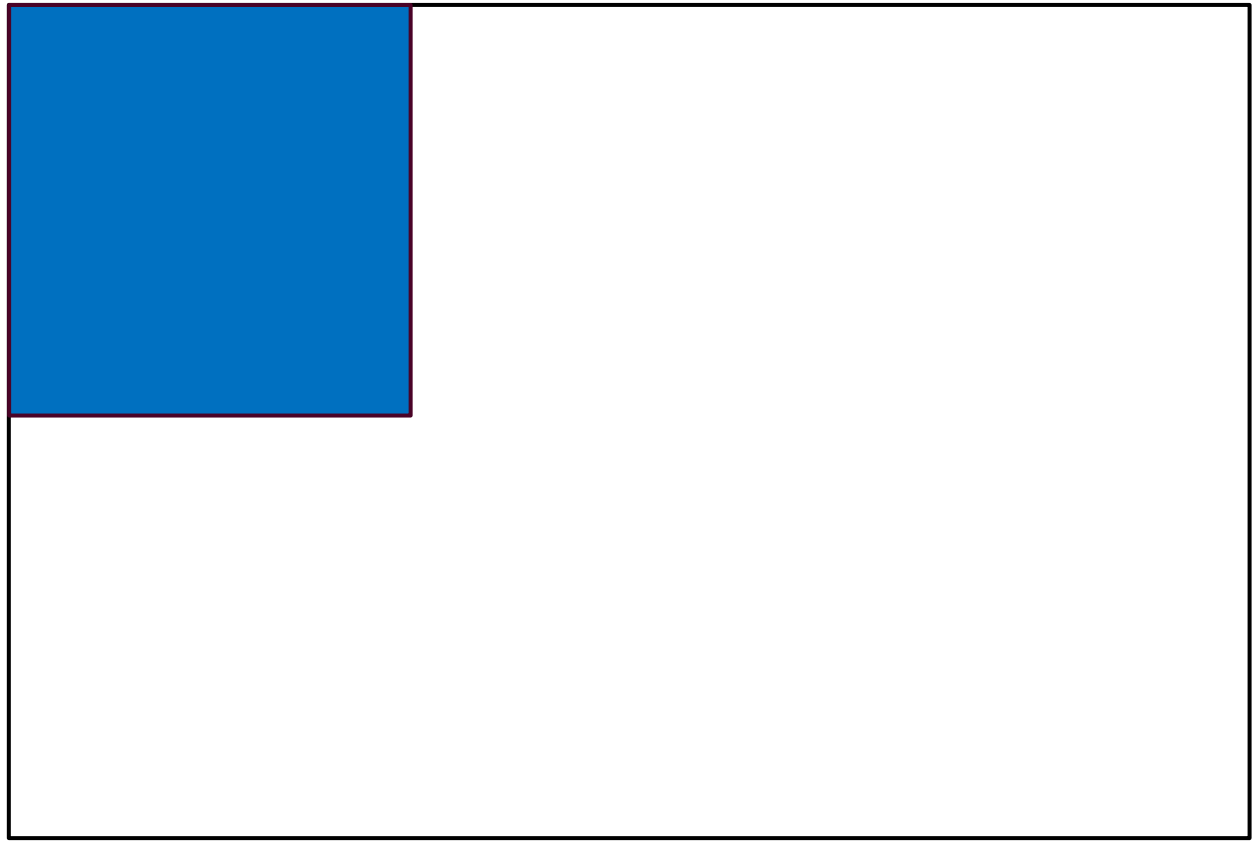
Multi-child layout widgets

FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Container

```
Container(
   height: 300,
   width: 300,
   color: Colors.blue
),
```

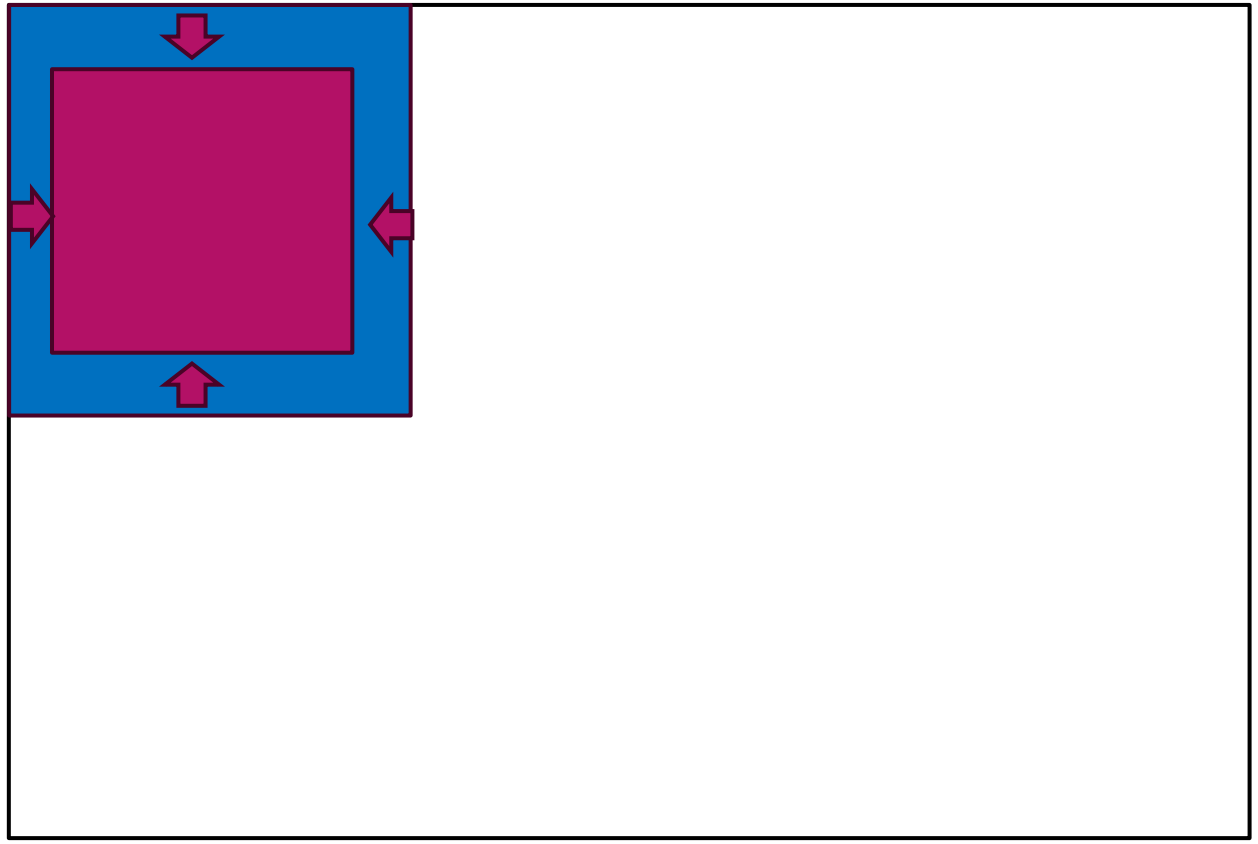


FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Container

```
Container(
   padding: EdgeInsets.all(10),
   height: 200,
   width: 200,
   color: Colors.blue
),
```

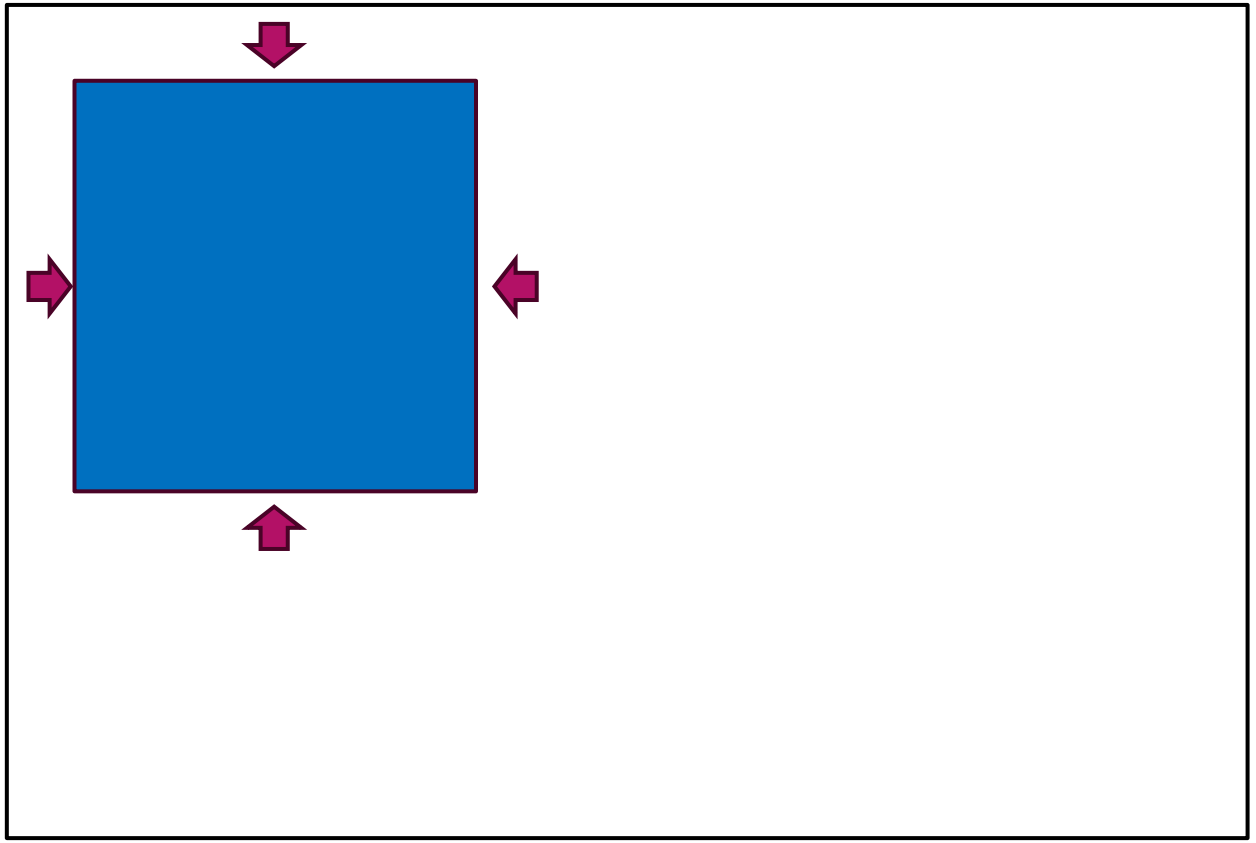


FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Container

```
Container(
   margin: EdgeInsets.all(10),
   height: 200,
   width: 200,
   color: Colors.blue
),
```

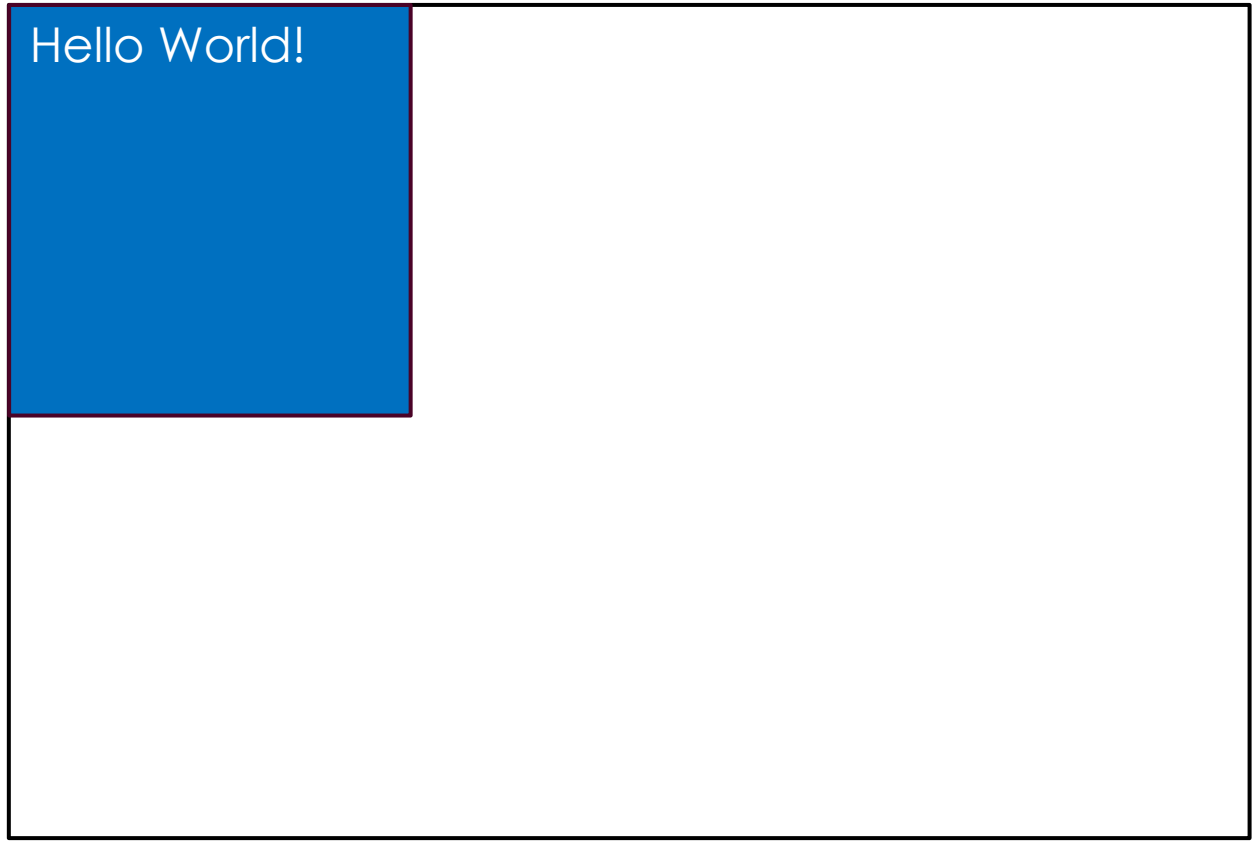


FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Container

```
Container(
   height: 200,
   width: 200,
   color: Colors.blue,
   child:
      const Text("Hello World!")
),
```

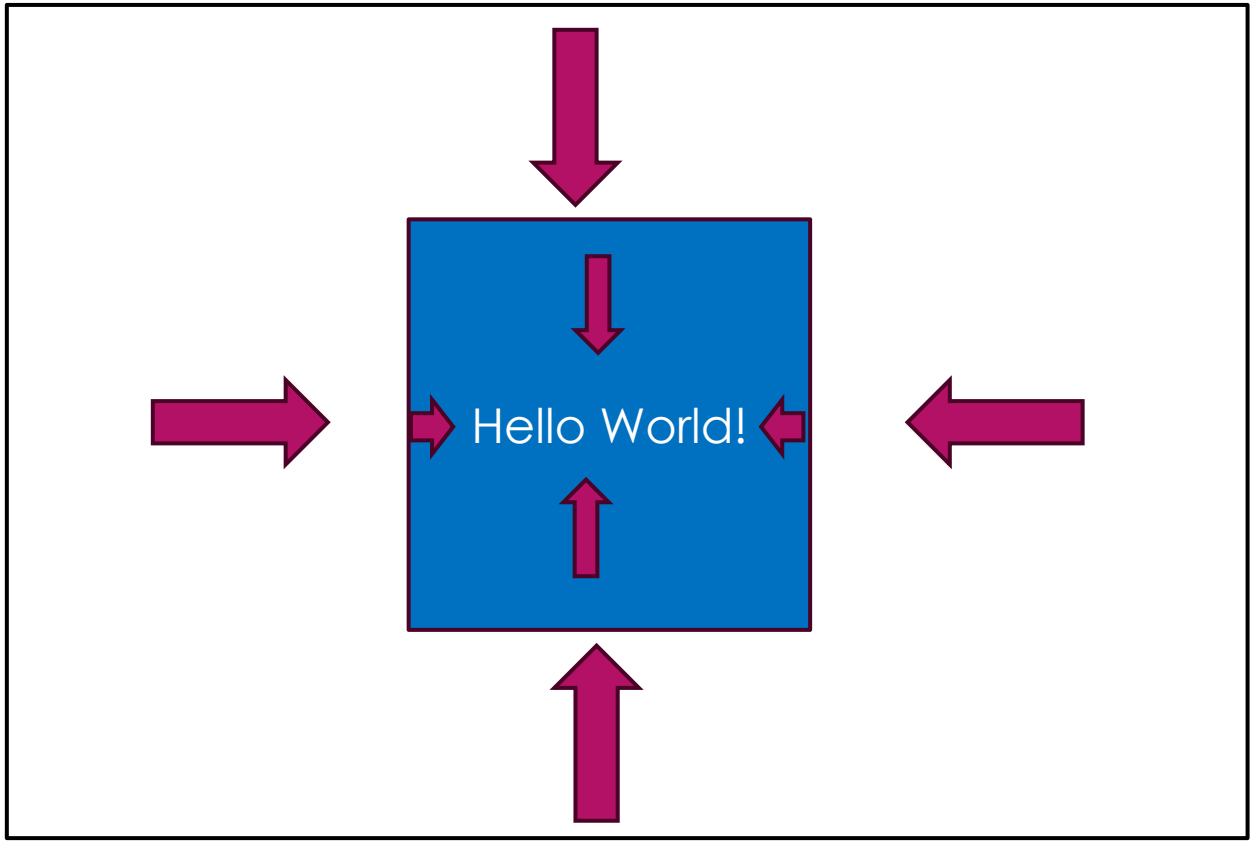


FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Center

```
Center(
   child: Container(
      height: 200,
      width: 200,
      color: Colors.blue,
      child:
         const Center(
            child: Text("...")
)))
```

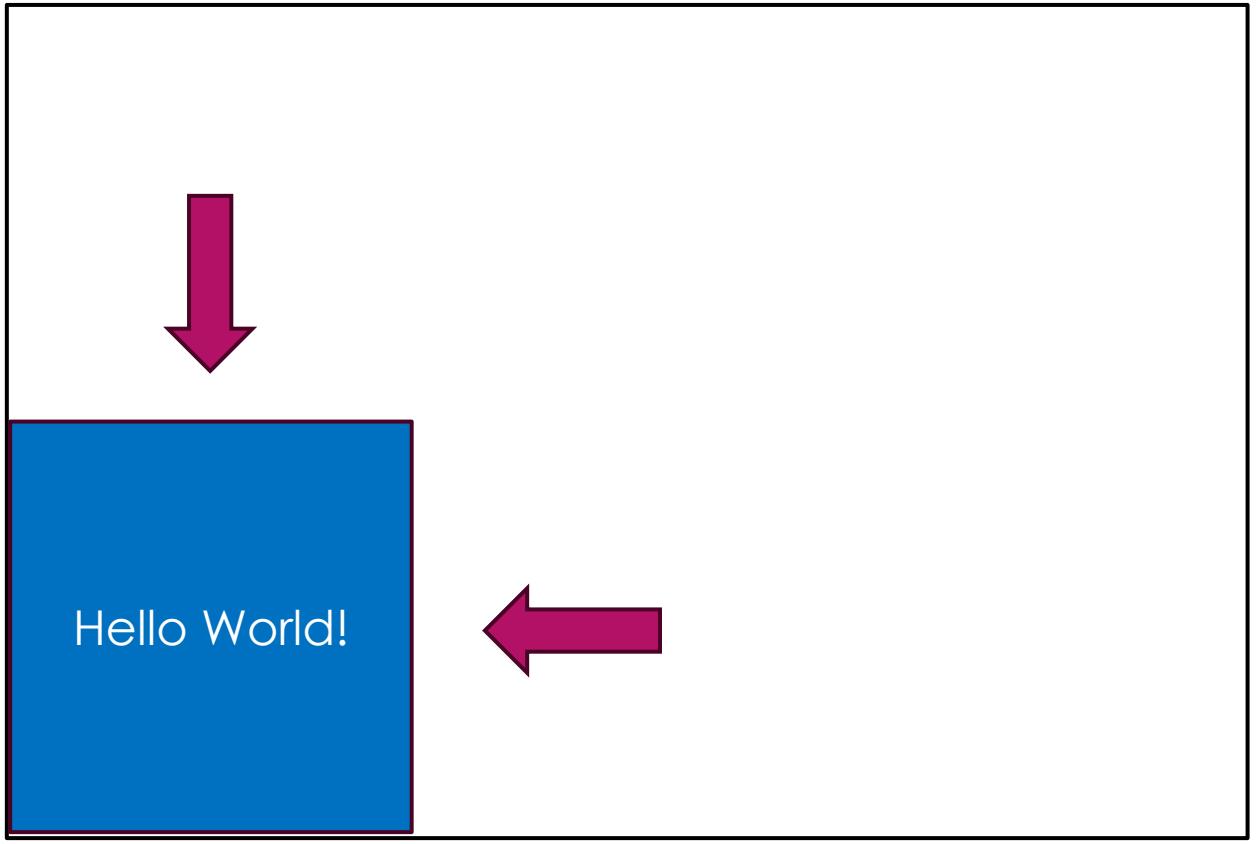


FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Align

```
Align(
   alignment: Alignment.bottomLeft,
   child: Container(
      height: 200,
      width: 200,
      color: Colors.blue,
      child:
         const Center(
            child: Text("...")
)))
```

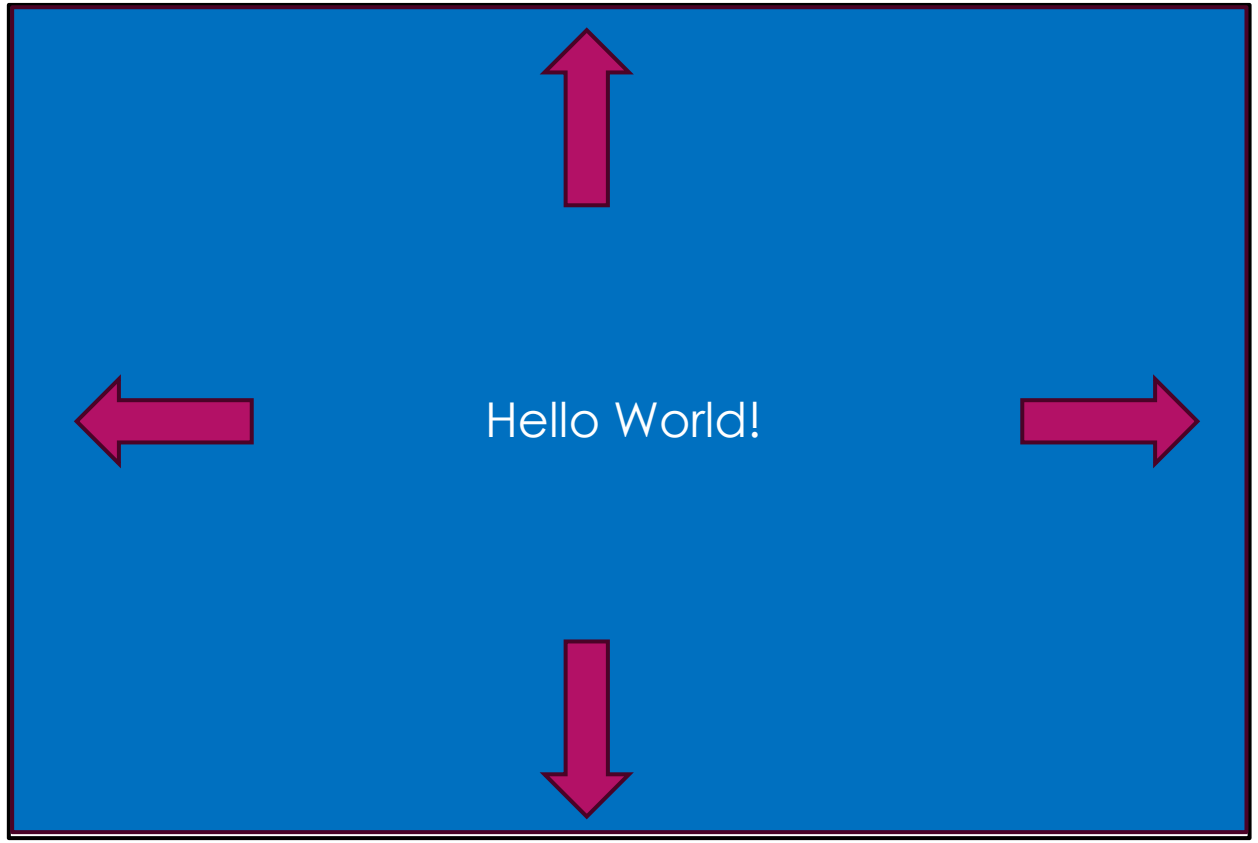


FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Expanded

```
Expanded(
   child: Container(
      color: Colors.blue,
      child: const Center(
         child:
            Text("Hello World!")
      )
))
```



FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Expanded

```
Expanded(
   child: Container(
      color: Colors.blue,
      child: const Center(
         child:
            Text("Hello World!")
      )
))
```

FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Expanded

```
Expanded(
   child: Container(
      color: Colors.blue,
      child: const Center(
         child:
            Text("Hello World!")
      )
))
```

The container doesn't know where to stop expanding.

Only use unconstrained sizes within:

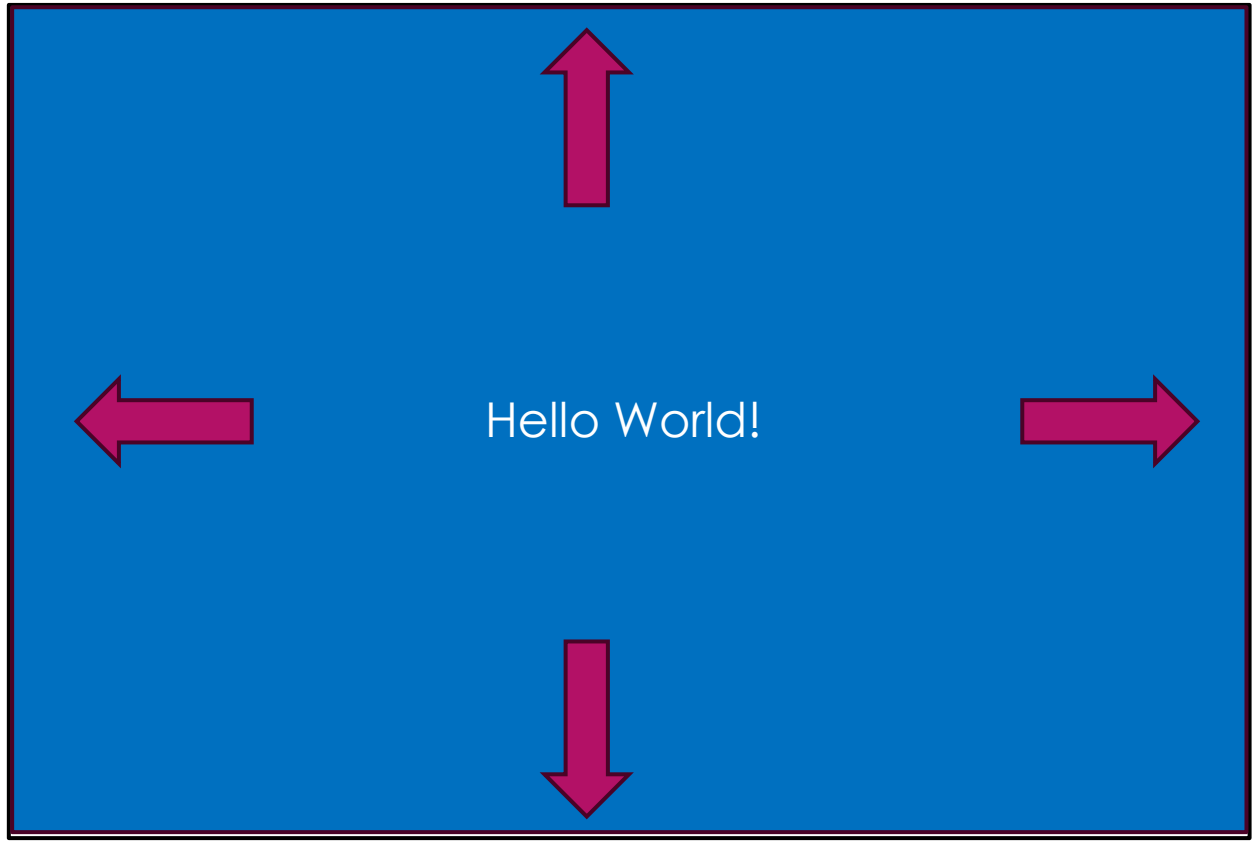
- LimitedBox
- Row
- Column

FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

LimitedBox

```
LimitedBox(
   child: Container(
      color: Colors.blue,
      child: const Center(
         child:
            Text("Hello World!")
      )
))
```



FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Single-child layout widgets

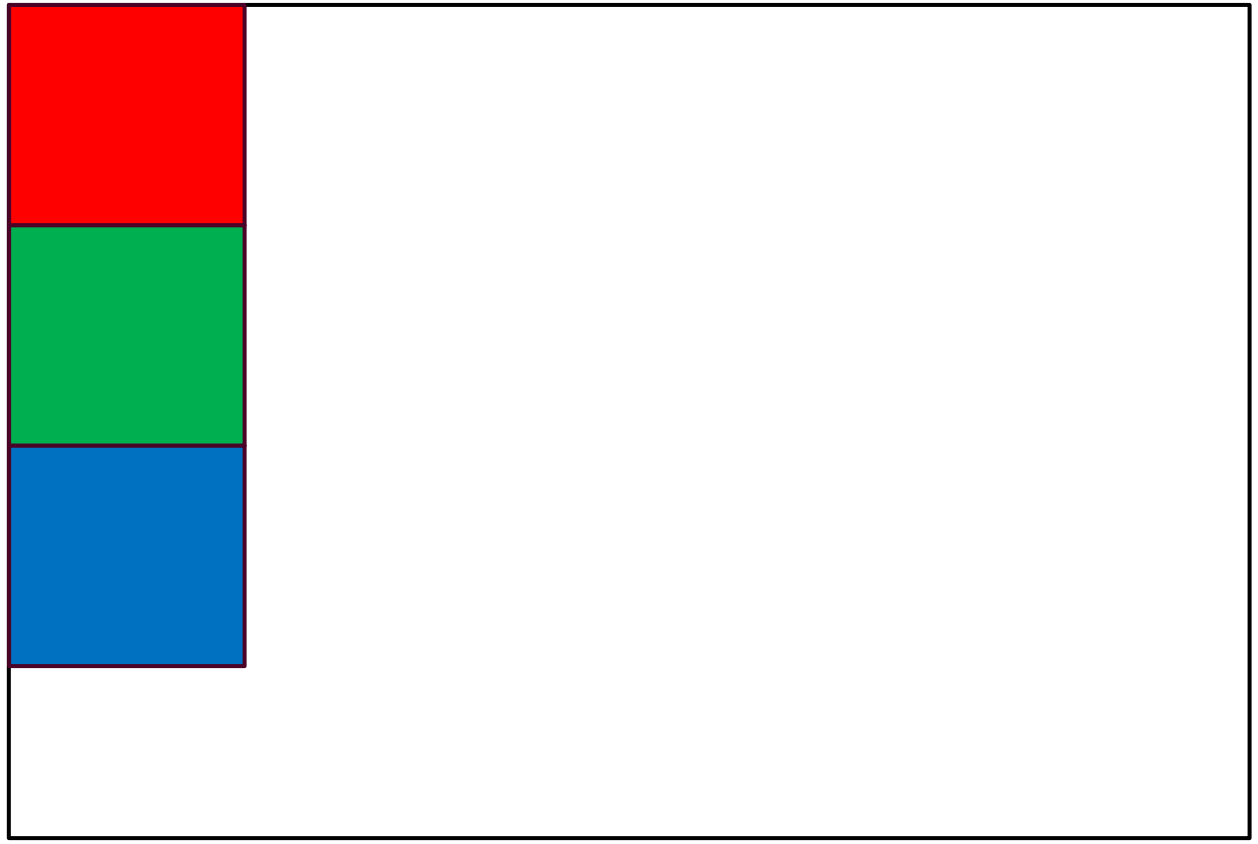
Multi-child layout widgets

FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Column

```
Column(
    children: [
        Container(...), // RED
        Container(...), // GREEN
        Container(...), // BLUE
    ]
)
```

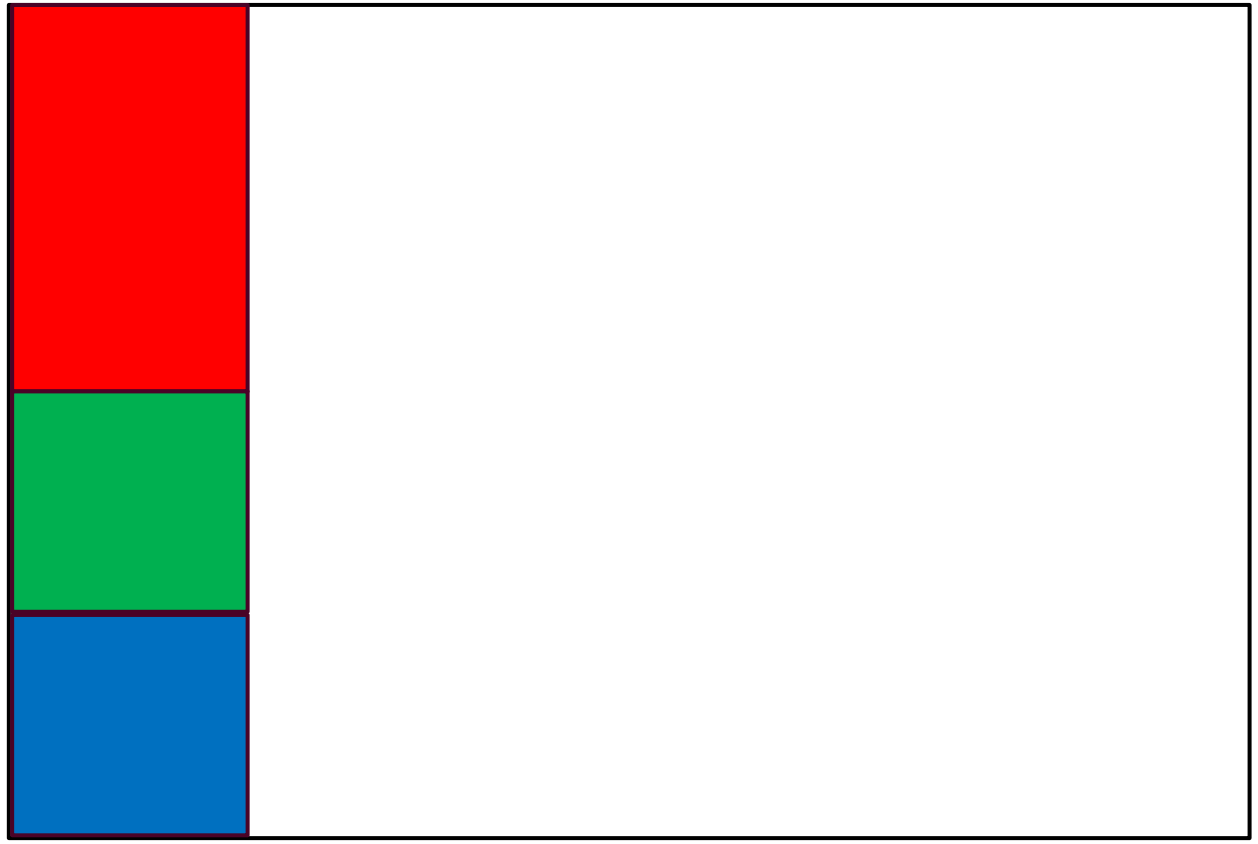


FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Column

```
Column(
   children: [
      Expanded(
       child: Container(...) // RED
      ),
      Container(...), // GREEN
      Container(...), // BLUE
   ]
)
```

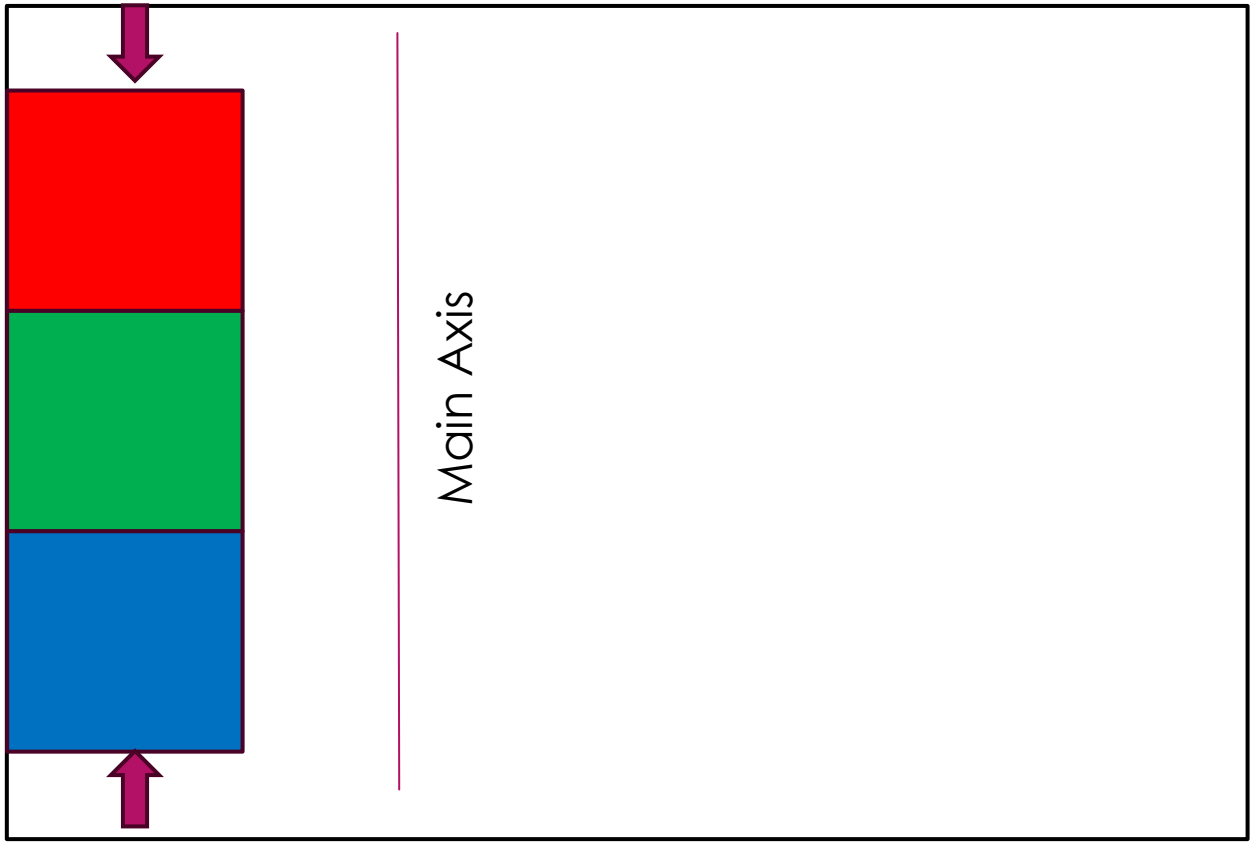


FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Column

```
Column(
   mainAxisAlignment:
      MainAxisAlignment.center,
   children: [
      Container(...), // RED
      Container(...), // GREEN
      Container(...), // BLUE
   ])
```

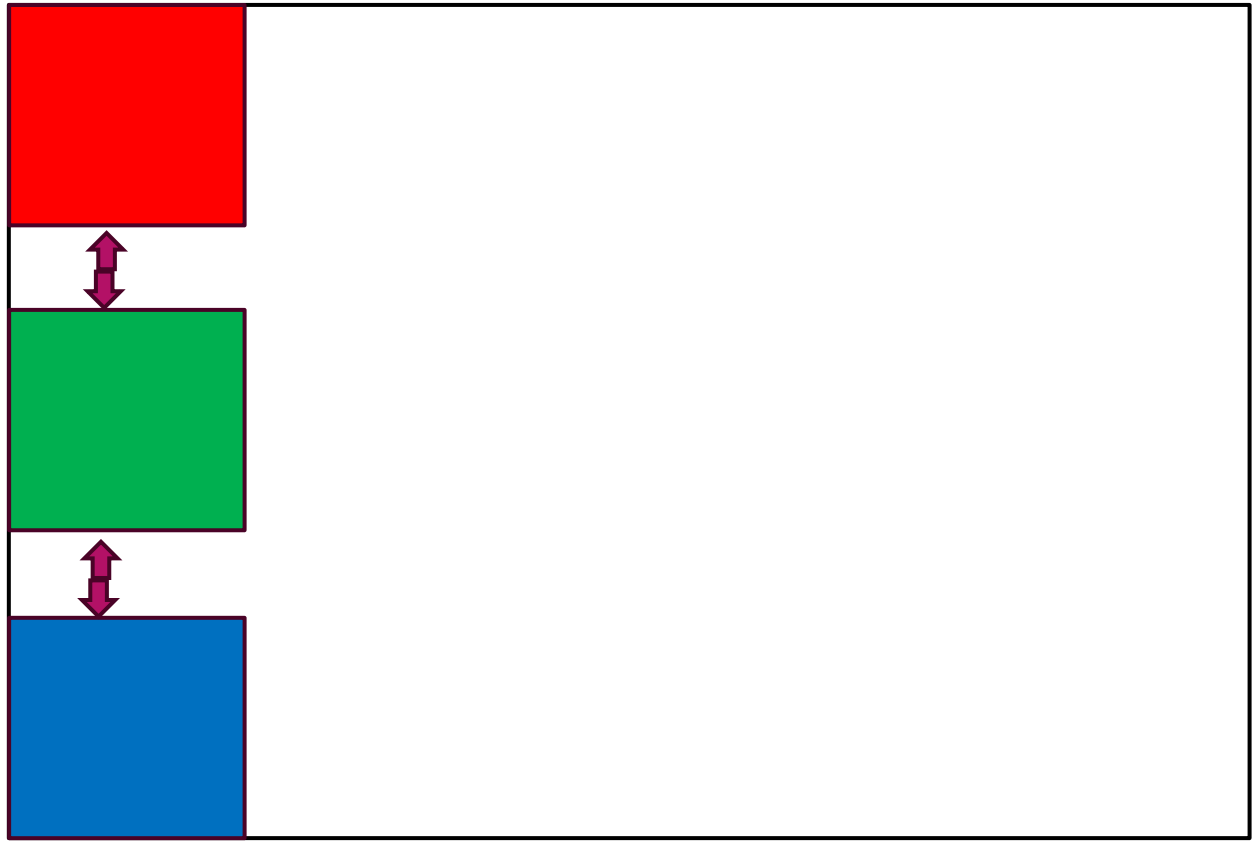


FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Column

```
Column(
   mainAxisAlignment:
    MainAxisAlignment.spaceBetween,
   children: [
      Container(...), // RED
      Container(...), // GREEN
      Container(...), // BLUE
   ])
```

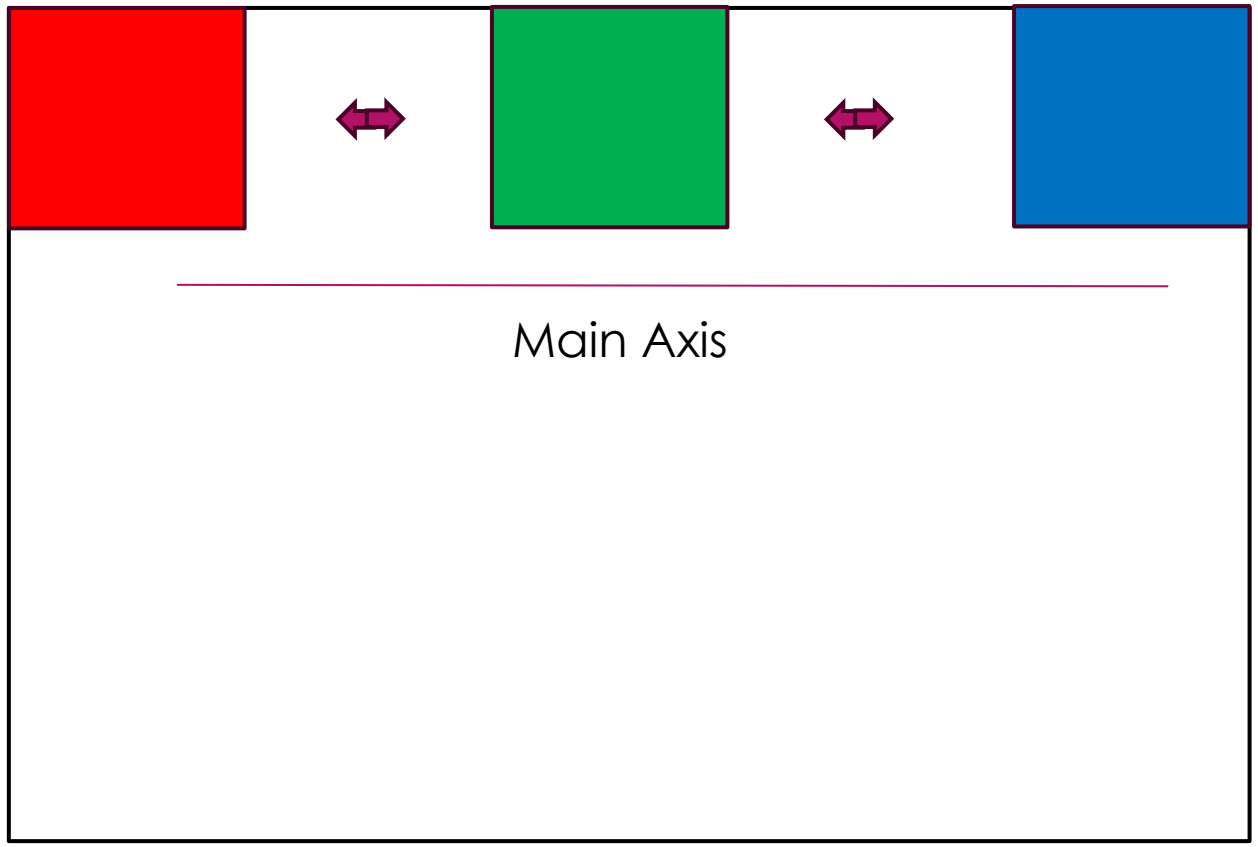


FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Row

```
Row(
   mainAxisAlignment:
    MainAxisAlignment.spaceBetween,
   children: [
      Container(...), // RED
      Container(...), // GREEN
      Container(...), // BLUE
   ])
```

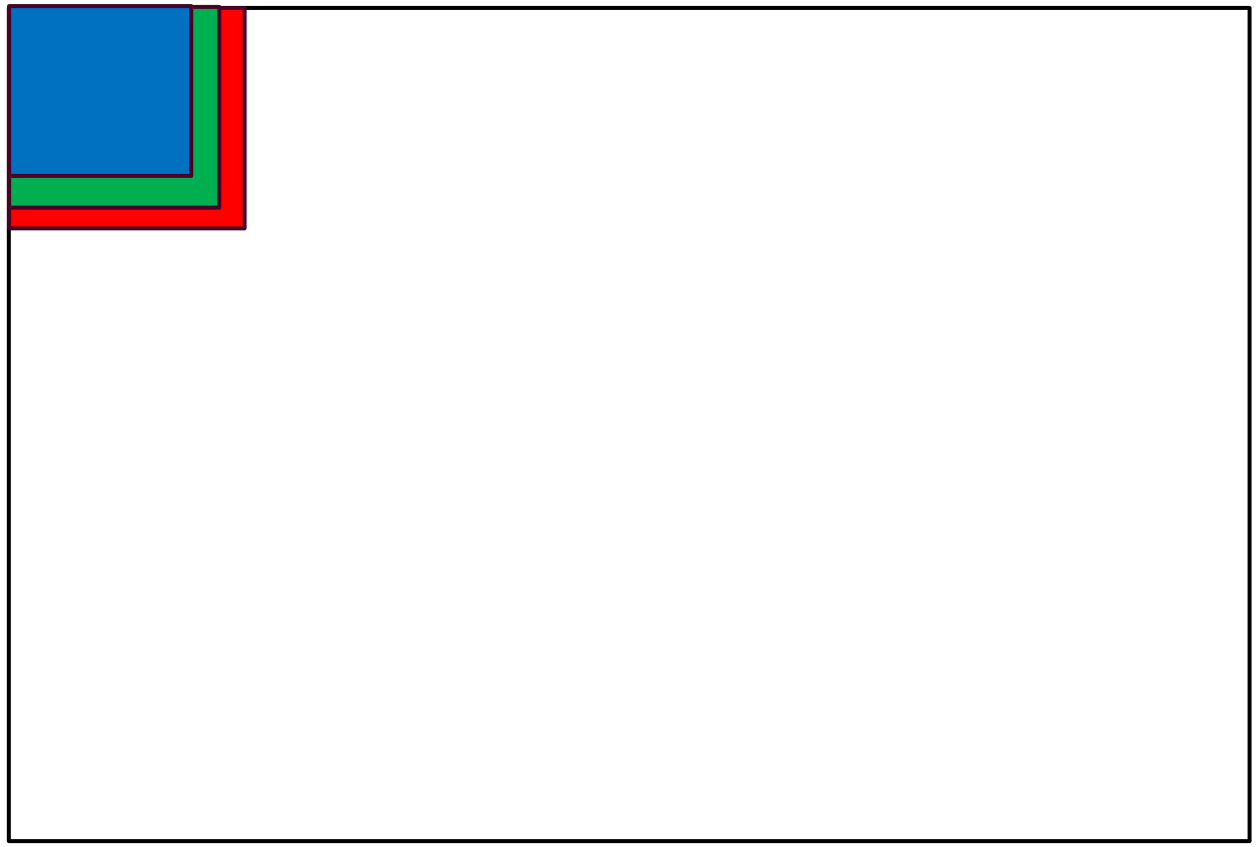


FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

Stack

```
Stack(
   children: [
    Container(...), // H/W 100 RED
    Container(...), // H/W 90 GREEN
    Container(...), // H/W 80 BLUE
   ])
```

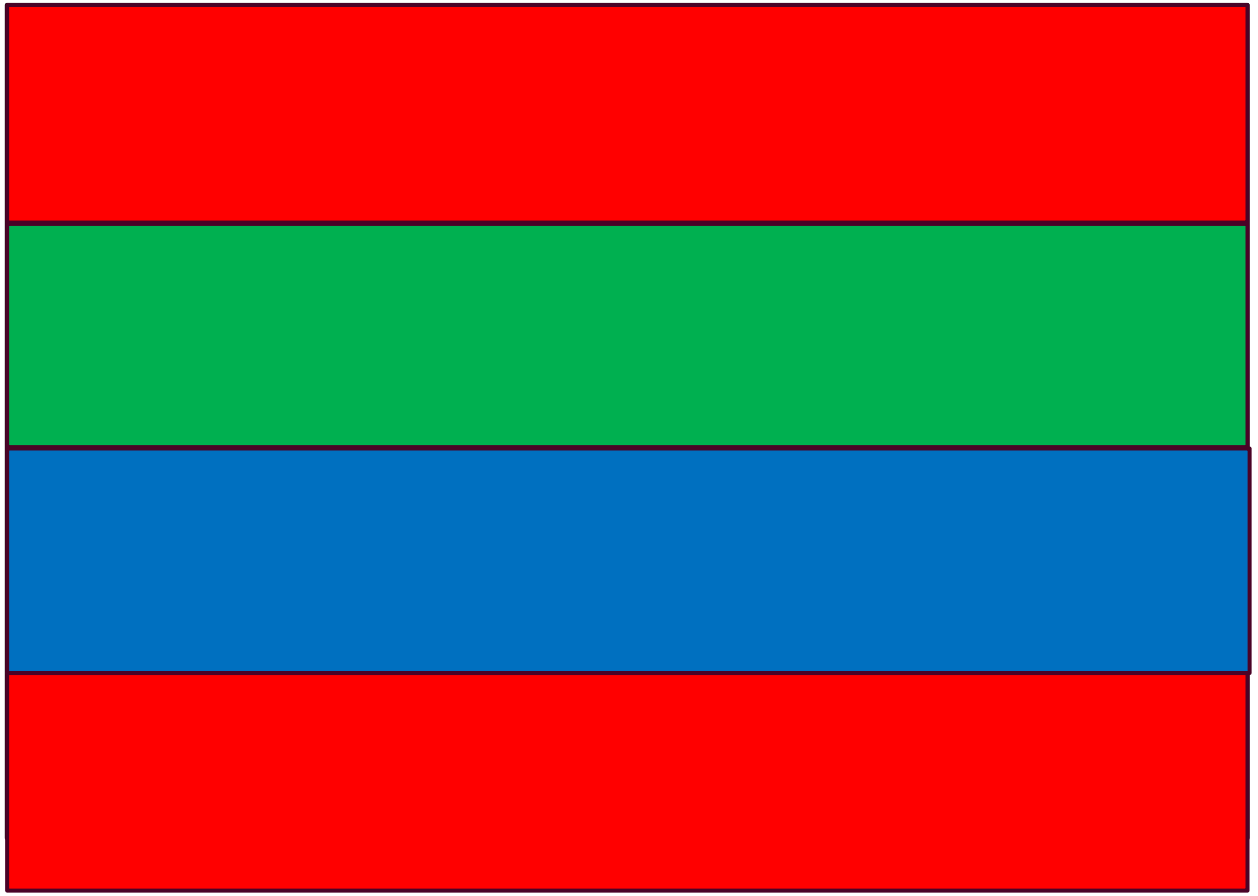


FLUTTER BASICS

- Layouts: Container, Row, Column, etc.

ListView

```
ListView(
   children: [
    Container(...), // RED
    Container(...), // GREEN
    Container(...), // BLUE
    … // More container
])
```

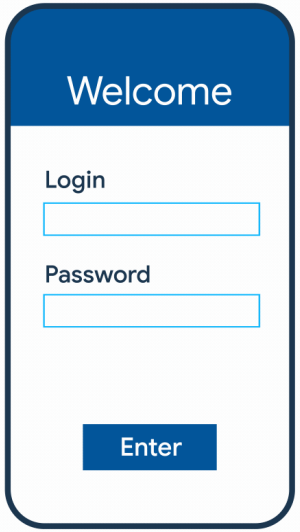


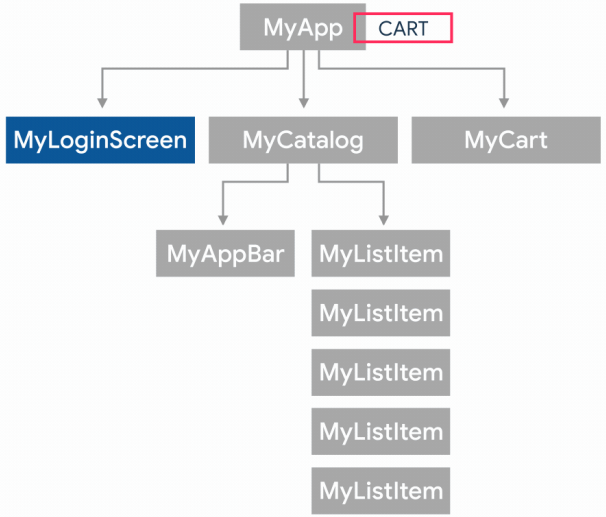
ADVANCED TECHNIQUES

- State Management

- Used to share applications states and data across an app
- Many approaches (setState, Provider, InheritedWidget, Redux,...)
- *Depends on your use cases and taste*

- Flutter is *declarative*

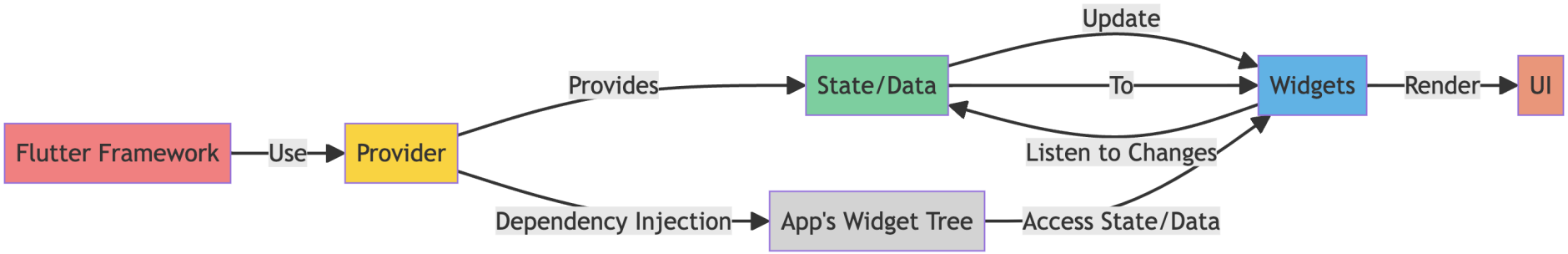




ADVANCED TECHNIQUES

- State Management

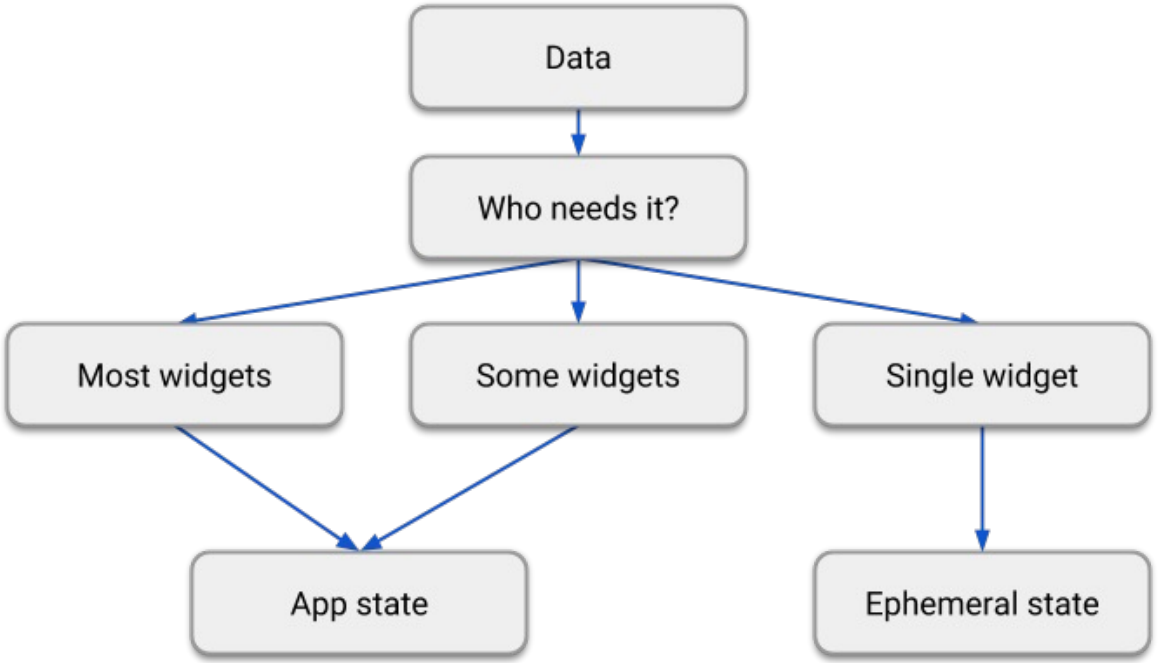
- Provider works similar to Observer Pattern
- With ChangeNotifier and Listeners
- On change, call notifyListeners() to trigger all builder methods of Consumer widgets



ADVANCED TECHNIQUES

- State Management

- *Two types of state:*
  - *Ephemeral (local): No other part of your app needs to access the state value*
  - *App (global): State you want to share across many app parts, e.g. user preferences, login info, shopping cart,…*



"The rule of thumb is: Do whatever is less awkward."

- Author of Redux, Dan Abramov

ADVANCED TECHNIQUES

- State Management

- *"setState()" is the simplest form of state management in Flutter*
- *It's built into the StatefulWidget.*
- *Suitable for small apps or withing single widgets, or closely related group of widgets*
- *Calling setState() notifies Flutter about a change*
- *Flutter then schedules a rebuild*

```
21  class MyHomePage extends StatefulWidget {
22    const MyHomePage({super.key});
23    // This widget is the home page of your application. It is stateful, meaning
24    // that it has a State object (defined below) that contains fields that affect
25    // how it looks.
26    // This class is the configuration for the state.
27    @override
28    _MyHomePageState createState() => _MyHomePageState();
29  }
30
31  class _MyHomePageState extends State<MyHomePage> {
32    int _counter = 0;
33
34    void _incrementCounter() {
35      setState(() {
36        // This call to setState tells the Flutter framework that something has
37        // changed in this State, which causes it to rerun the build method below
38        // so that the display can reflect the updated values. If we changed
39        // _counter without calling setState(), then the build method would not be
40        // called again, and so nothing would appear to happen.
41        _counter++;
42      });
43    }
```

ADVANCED TECHNIQUES

- State Management

- *"setState()" is the simplest form of state management in Flutter*
- *It's built into the StatefulWidget.*
- *Suitable for small apps or withing single widgets, or closely related group of widgets*
- *Calling setState() notifies Flutter about a change*
- *Flutter then schedules a rebuild*

```
@override
Widget build(BuildContext context) {
  // This method is rerun every time setState is called, for instance as done
  // by the _incrementCounter method above.
  //
  // The Flutter framework has been optimized to make rerunning build methods
  // fast, so that you can just rebuild anything that needs updating rather
  // than having to individually change instances of widgets.
  return Scaffold(
    appBar: AppBar(
      title: const Text('Flutter Demo Click Counter'),
    ), // AppBar
    body: Center(

      child: Column(

        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          const Text(
            'You have pushed the button this many times:',
          ), // Text
          Text(
            '$_counter',
            style: const TextStyle(fontSize: 25),
          ), // Text
        ], // <Widget>[]
      ) // Column
    ), // Center
    floatingActionButton: FloatingActionButton(
      onPressed: _incrementCounter,
      tooltip: 'Increment',
      child: const Icon(Icons.add),
    ), // FloatingActionButton
  ); // Scaffold
}
```

ADVANCED TECHNIQUES

- Plugins, Packages…

Packages: https://pub.dev/

- Adding a package dependency
  - `flutter pub add css_colors`
- Removing a package dependency
  - `flutter pub remove css_colors`

SETUP

- FlutLab.io

- Why FlutLab.io?
  - No local setup, access anywhere
  - Great for beginners, lots of templates, no IDE setup

- Usual IDEs:

Visual Studio Code

Android Studio

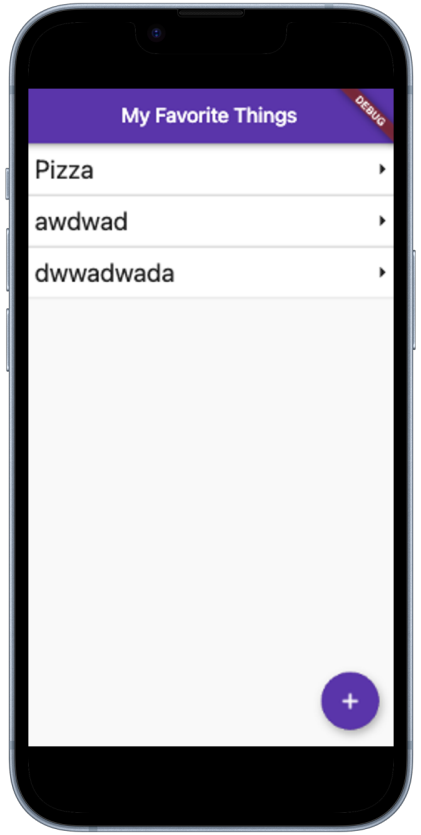
GitHub

tinyurl.com/esdtask

INTERACTIVE SESSION

- Let us build any app you want

- Work on any app idea you want for the next **30 minutes.**
- Start from scratch or build up onto our example project https://github.com/sebivenlo/ESD-2023-Flutter/tree/main/assignment
- Get use of the default Flutter widgets: https://docs.flutter.dev/ui/widgets
- Don't hesitate to **ask us** if you have any questions!



CONCLUSION

- Summary

- Flutter compared to other Languages
- Basics
  - Widgets
  - Layouts & UI
- Advanced
  - Plugins & Packages
  - State Management

CONCLUSION

- Resources. Where to learn more?

- Official Websites & Docs
- YouTube: Official Flutter Channel
- YouTube: Fireship Flutter Playlist (22 videos)

THANKS FOR PARTICIPATING

Sources

- Not ChatGPT.

1.Dart and Flutter Tutorial: marketsplash.com/tutorials/dart/dart-flutter
2.What's Flutter? - FreeCodeCamp: freecodecamp.org/news/https-medium-com-rahman-sameeha-whats-flutter
3.Flutter Layout Widgets: docs.flutter.dev/ui/widgets/layout
4.Frameworks for Cross-Platform Mobile App Development: apptunix.com/blog/frameworks-cross-platform-mobile-app-development
5.React Native Getting Started: reactnative.dev/docs/getting-started
6.Xamarin Documentation: learn.microsoft.com/en-us/xamarin
7.Using Packages in Flutter: docs.flutter.dev/packages-and-plugins/using-packages
8.Flutter Package Repository - pub.dev: pub.dev
9.Flutter for Web Development: miquido.com/blog/flutter-for-web-development
10.Flutter YouTube Tutorial: youtube.com/watch?v=3tm-R7ymwhc
11.State Management in Flutter: docs.flutter.dev/data-and-backend/state-mgmt/intro
12.Redux GitHub Discussion: github.com/reduxjs/redux/issues/1287#issuecomment-175351978Issue
13.Xamarin vs Flutter vs React Native - The Mobile Reality: themobilereality.com/blog/xamarin-vs-flutter-vs-react-native
14.Xamarin, Flutter, and React Native Comparison - Promatics: promatics.medium.com/heres-who-would-win-if-xamarin-flutter-and-react-native-fight-out-in-2021-8fa6e22fdfbb