

# Mobile Applications

## 9. Introduction to cross-platform apps development

Boni García

[boni.garcia@uc3m.es](mailto:boni.garcia@uc3m.es)

Telematic Engineering Department  
School of Engineering

2024/2025

**uc3m** | Universidad **Carlos III** de Madrid



# Table of contents

1. Introduction
2. React
3. React Native
4. Flutter
5. Takeaways

# 1. Introduction

- As we have learned, there are different ways for developing mobile apps:
  1. **Native** development involves creating apps specifically for a given platform (Android or iOS) using platform-specific programming languages and tools
  2. **Hybrid** development combines web technologies (HTML, CSS, JavaScript) with a *WebView* native container to create apps that work across multiple platforms
  3. **Cross-platform** development uses frameworks that allow developers to write code once and deploy it on multiple platforms
    - Unlike hybrid apps, cross-platform apps are compiled into native code
  4. **Progressive Web Apps (PWAs)** are websites that behave like apps
    - They run in a browser but can be installed on a device and used offline
  5. **Low-Code/No-Code Platforms** that allow non-developers to create apps using drag-and-drop interfaces and pre-built templates

# 1. Introduction

App type	Pros	Cons
Native	+ High performance and responsiveness + Best user experience and design consistency	- Requires separate codebases for each platform (more development time and cost)
Hybrid	+ Easy for developers familiar with web development + Single codebase for multiple platforms	- Slower performance compared to native apps - Limited access to advanced device features
Cross-platform	+ Saves development time and cost with a single codebase + Good performance for most apps	- Performance may not match fully native apps - Limited access to certain platform-specific features (depending on the framework)
PWAs	+ No app store submission required + Can work on any device with a browser + Cost-effective and fast to develop	- Limited access to device hardware and native features - Can't match the performance of native apps
Low-Code/ No-Code	+ Speeds up development for simple apps + Requires little to no programming knowledge	- Limited flexibility and scalability for complex apps

This unit we study the basics of **React Native** and **Flutter**

# Table of contents

1. Introduction

2. React

- Sandbox
- Local setup
- Frameworks
- TypeScript

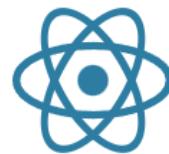
3. React Native

4. Flutter

5. Takeaways

## 2. React

- React is an open source front-end **JavaScript library** for building user interfaces (UIs) based on components created by Meta (formerly Facebook)
  - It is typically used to build Single-Page Application (**SPA**)
  - SPAs are web applications that interacts with the user by dynamically rewriting the current web page with new data from the web server
    - Some popular SPAs are the web versions of Instagram, Facebook, Netflix, or Airbnb



### React

The library for web and native user interfaces

<https://react.dev/>

## 2. React

- The key features of React are the following:
  - Component-based
    - React apps are built using reusable components (like Lego blocks)
    - Each component manages its own state and logic
  - JSX (JavaScript XML)
    - JSX allows us to write HTML elements in JavaScript and place them in the DOM (Document Object Model)
    - It is not mandatory, but recommended for easier development
  - Virtual DOM (lightweight, in-memory representation of the real DOM)
    - React uses a Virtual DOM to optimize updates, making rendering faster
    - Instead of updating the real DOM directly, React compares changes in memory first
  - Rich ecosystem
    - Works well with libraries and frameworks like [Redux](#) (state management), [React Router](#) (navigation), or [Next.js](#) (server-side rendering)

## 2. React

- There are different ways to create React apps:

### 1. Using a sandbox

- A sandbox refers is an isolated, browser-based environment where we can write, run, and test React code without setting up a local project
  - It is a convenient way to quickly prototype without setting anything up locally
  - A popular sandbox is [CodeSandbox](#)

### 2. Setting up a local project

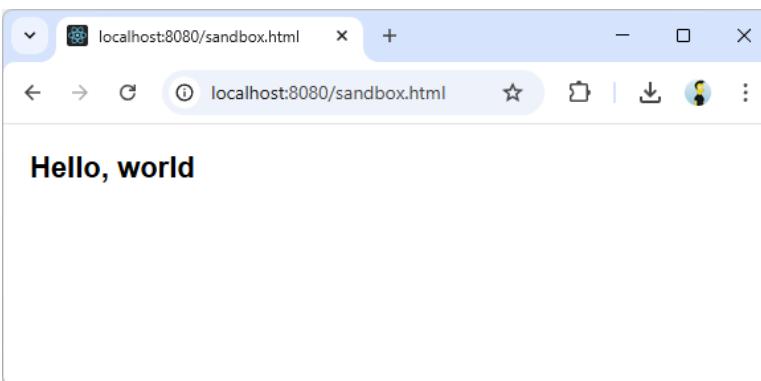
- Using a build tool (convenient for basic projects)
  - The classical tool is called [Create React App](#), but it is deprecated nowadays
  - Modern options are: [Vite](#), [Parcel](#), or [Rsbuild](#)
- Using a framework (convenient for complex projects)
  - Provide enhanced features (e.g., SEO, routing, optimizations, and others)
  - For example: [Next.js](#), [Remix](#), [Gatsby](#), or [Expo](#)

## 2. React - Sandbox

- There are basic sandboxes examples in the React doc using JSX, e.g.:

```
function ({ name }) {
  return <h1>Hello, {name}</h1>;
}

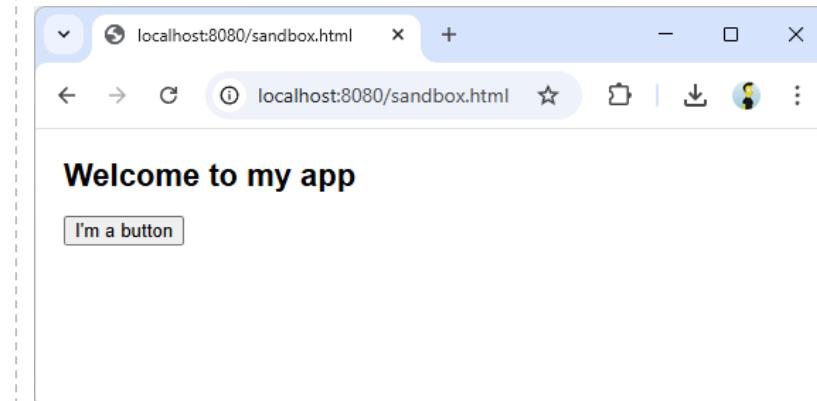
let App = function App() {
  return <Greeting name="world" />
}
```



```
function MyButton() {
  return (
    <button>
      I'm a button
    </button>
  );
}

let App = function MyApp() {
  return (
    <div>
      <h1>Welcome to my app</h1>
      <MyButton />
    </div>
  );
}
```

React **components** are JavaScript functions that return markup (**Greeting** and **MyButton** in these examples)

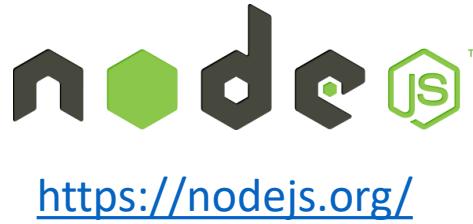


## 2. React - Local setup

- To set up a local React project, the requirements are:

### 1. Node.js

- Node.js is an open source, cross-platform JavaScript runtime environment that enables the execution of JavaScript code outside a web browser
- Node.js comes with **NPM**, its default package manager



```
> node --version  
v22.15.0  
  
> npm --version  
10.6.0
```

After install Node.js, we can execute node and npm as command-line tools

### 2. Code editor (not mandatory, but highly recommended)

- One of the most popular IDEs for React is Visual Studio Code
- Other alternatives are: WebStorm, Sublime Text, or Atom



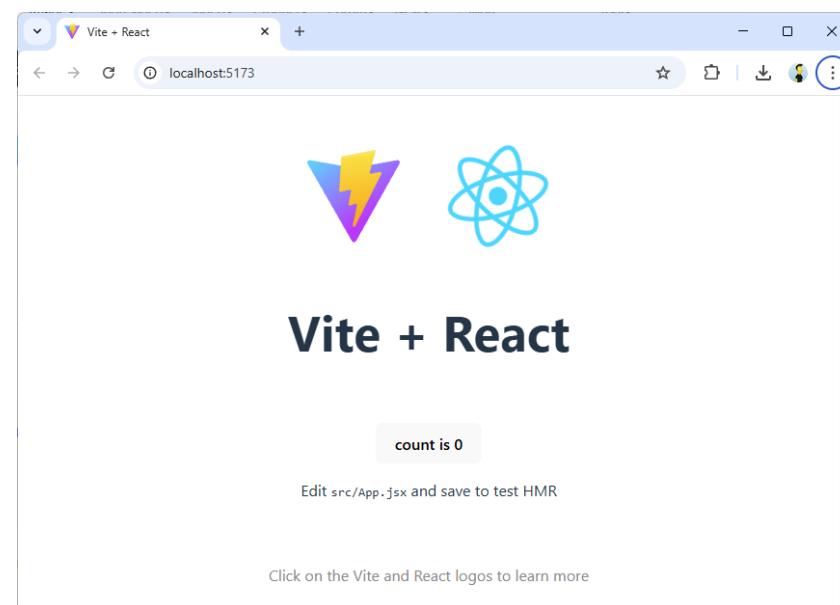
Visual Studio Code

<https://code.visualstudio.com/>

## 2. React - Local setup

- For instance, we can use Vite to set up a local React project:

```
> npm create vite@latest hello-world-vite -- --template react  
  
> cd hello-world-vite  
  
> npm install  
  
> npm run dev
```



<https://vite.dev/>

<https://react.dev/learn/build-a-react-app-from-scratch>

## 2. React - Frameworks

- React frameworks support all the features required to deploy and scale an app in production
  - For example, Next.js:

```
> npx create-next-app@latest

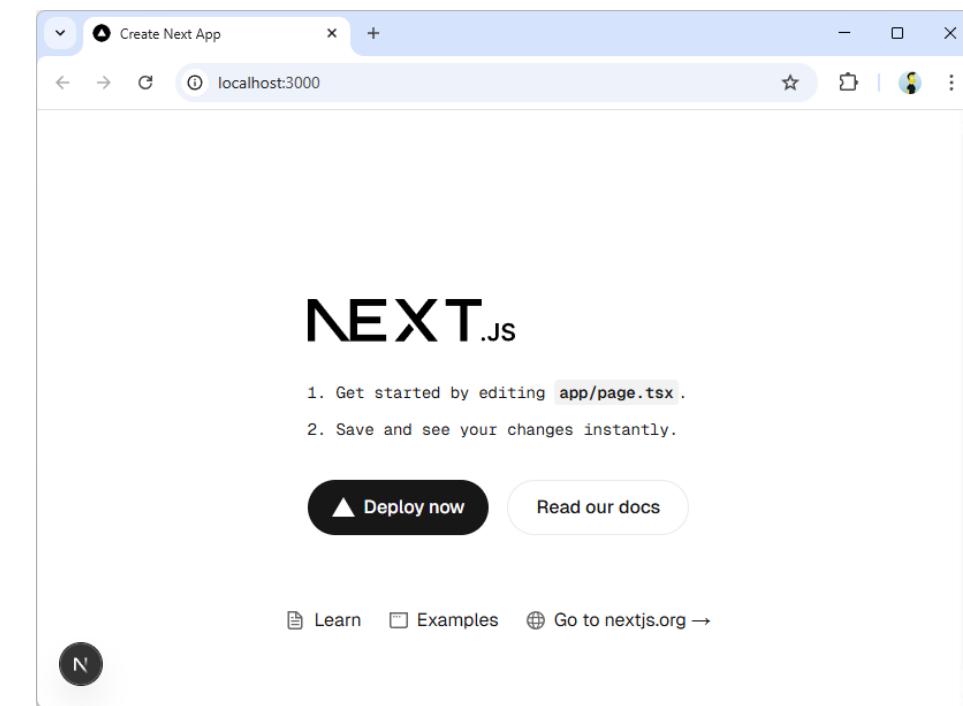
? What is your project named? » my-app
? Would you like to use TypeScript? » No / Yes
? Would you like to use ESLint? » No / Yes
? Would you like to use Tailwind CSS? » No / Yes
? Would you like your code inside a `src/` directory? » No / Yes
? Would you like to use App Router? (recommended) » No / Yes
? Would you like to use Turbopack for `next dev`? » No / Yes
? Would you like to customize the import alias (`@/*` by default)? » No / Yes

Creating a new Next.js app in C:\Users\boni\dev\react\my-app.

> cd my-app
> npm run dev
```

npx is a CLI tool that comes with npm used to execute Node.js packages without installing them globally or locally

**NEXT.JS**  
<https://nextjs.org/>



## 2. React - TypeScript

- **JavaScript (JS)** is a high-level, often just-in-time compiled programming language
  - JavaScript was first released on Netscape 2 (a web browser) in 1996
  - It is most well-known as the scripting language for web pages, being one of the core technologies of the Web, alongside HTML and CSS
  - It has dynamic typing, i.e., type of a variable is known at runtime. Type checking occurs also at runtime
- **TypeScript (TS)** is an open source programming language developed by Microsoft, first released in 2012
  - TypeScript is often referred to as a superset or extension of JavaScript
  - TypeScript extends JavaScript by adding static typing, i.e., the type of a variable is known at compile-time. This feature help developers build large-scale, robust applications more effectively
  - Unlike JavaScript, TypeScript can't be executed directly in a browser – it must be *transpiled* into JavaScript first



## 2. React - TypeScript

- The following table summarizes the advantages and disadvantages of using JavaScript and TypeScript in React development:

	<b>JavaScript</b>	<b>TypeScript</b>
Pros	<ul style="list-style-type: none"><li>Faster development time: JavaScript is a dynamic and flexible language that allows developers to build applications quickly</li><li>Easier to learn: Shallow learning curve</li></ul>	<ul style="list-style-type: none"><li>Code quality: TypeScript adds static typing to JavaScript, which helps catch bugs at compile-time</li><li>Developer experience: Features such as auto-completion and error checking</li></ul>
Cons	<ul style="list-style-type: none"><li>No static typing: JavaScript is a dynamically typed, which can lead to bugs at runtime</li><li>Code become complex: Without the benefit of static typing, it can be challenging to maintain code quality and readability as applications grow</li></ul>	<ul style="list-style-type: none"><li>Steep learning curve: TypeScript is more difficult to learn than JavaScript</li><li>Longer development time: TypeScript's additional syntax and static typing can slow down development time, especially for small projects</li></ul>

# Table of contents

1. Introduction

2. React

3. React Native

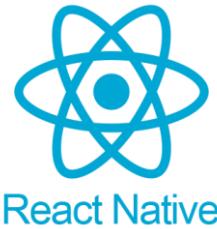
- Expo
- Hello World
- Views
- Core components

4. Flutter

5. Takeaways

# 3. React Native

- **React Native** is an open source cross-platform app framework for building mobile apps (Android and iOS)
  - Internally it uses the React library to manage the components. For that reason, we can use JavaScript or TypeScript to develop React Native apps
  - Like React, React Native is maintained by Meta (formerly Facebook) and a community of individual developers and companies



<https://reactnative.dev/>

# 3. React Native - Expo

- The recommended way to set up a local React Native project is using a Framework (i.e., a toolbox with all the necessary features to build production ready apps) like Expo
- **Expo** is a framework and platform for building and deploying universal React Native apps
- **Expo Go** is an open source mobile app for testing React Native apps on any Android or iOS device
  - It is available on both the Android Play Store and iOS App Store
    - [Android Play Store](#) (Android Lollipop and greater)
    - [iOS App Store](#) (iOS 13 and greater)



<https://expo.dev/>



<https://expo.dev/client>

# 3. React Native - Hello world

- We can create a basic React Native app with Expo using npx as follows:

```
> npx create-expo-app hello-world
```

```
✓ Downloaded and extracted project files.  
✓ Installed JavaScript dependencies.
```

```
 Your project is ready!
```

To run your project, navigate to the directory and run one of the following npm commands.

- cd hello-world
- npm run android
- npm run ios # you need to use macOS to build the iOS project - use the Expo app if you need to do iOS development without a Mac
- npm run web

# 3. React Native - Hello world

```
> npm run android  
  
> hello-world@1.0.0 android  
> expo start --android  
  
Starting project at C:\Users\boni\Documents\dev\react-native-examples\hello-world  
Starting Metro Bundler  
> Opening exp://10.118.107.205:8081 on Pixel_4_API_30
```



```
> Metro waiting on exp://10.118.107.205:8081  
> Scan the QR code above with Expo Go (Android) or the Camera app (iOS)
```

```
> Using Expo Go  
> Press s | switch to development build  
  
> Press a | open Android  
> Press w | open web  
  
> Press j | open debugger  
> Press r | reload app  
> Press m | toggle menu  
> shift+m | more tools  
> Press o | open project code in your editor  
  
> Press ? | show all commands
```

Logs for your project will appear below. Press Ctrl+C to exit.  
Android Bundled 834ms index.js (644 modules)

To run our app locally,  
we need a running AVD  
(e.g., executed with  
Android Studio)



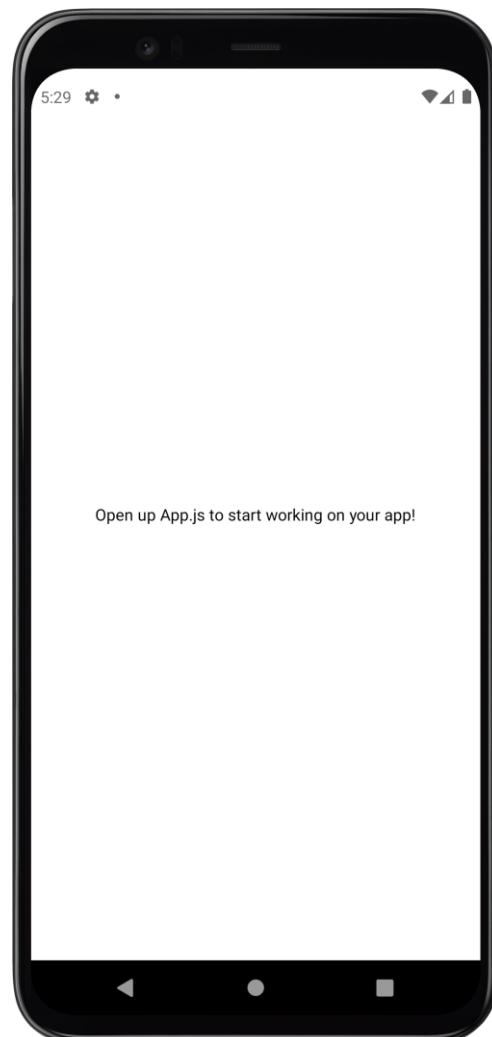
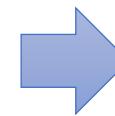
# 3. React Native - Hello world

Fork me on GitHub

```
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View } from 'react-native';

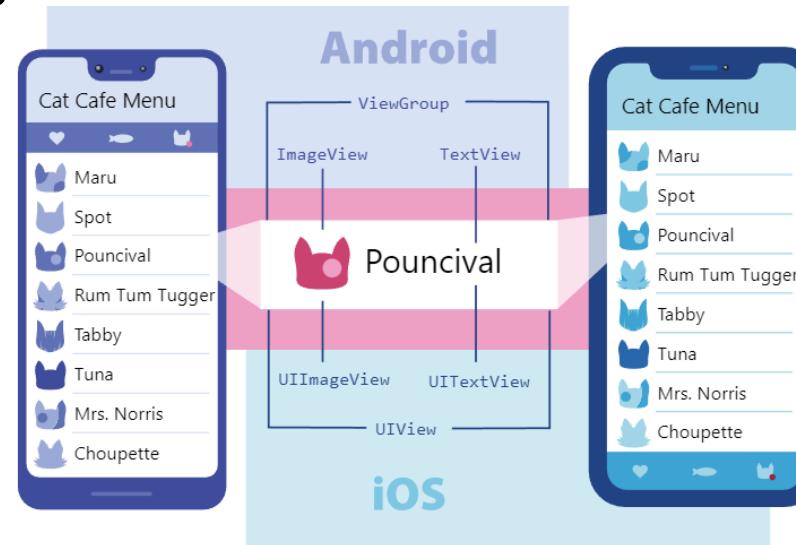
export default function App() {
  return (
    <View style={styles.container}>
      <Text>Open up App.js to start working on your app!</Text>
      <StatusBar style="auto" />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```



# 3. React Native - Views

- A **view** is the basic building block of UI both in Android and iOS
  - A view is a rectangular portion of the screen which can be used to display text, images, etc.
- With React Native, we invoke these views with JavaScript using React components
- At runtime, React Native creates the corresponding Android and iOS views for those components



<https://reactnative.dev/docs/intro-react-native-components>

# 3. React Native - Core Components

- React Native comes with a set of essential, ready-to-use native components to building ours app. These components are called **React Native's Core Components**

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
<code>&lt;View&gt;</code>	<code>&lt;ViewGroup&gt;</code>	<code>&lt;UIView&gt;</code>	A non-scrolling <code>&lt;div&gt;</code>	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
<code>&lt;Text&gt;</code>	<code>&lt;TextView&gt;</code>	<code>&lt;UITextView&gt;</code>	<code>&lt;p&gt;</code>	Displays, styles, and nests strings of text and even handles touch events
<code>&lt;Image&gt;</code>	<code>&lt;ImageView&gt;</code>	<code>&lt;UIImageView&gt;</code>	<code>&lt;img&gt;</code>	Displays different types of images
<code>&lt;ScrollView&gt;</code>	<code>&lt;ScrollView&gt;</code>	<code>&lt;UIScrollView&gt;</code>	<code>&lt;div&gt;</code>	A generic scrolling container that can contain multiple components and views
<code>&lt;TextInput&gt;</code>	<code>&lt;EditText&gt;</code>	<code>&lt;UITextField&gt;</code>	<code>&lt;input type="text"&gt;</code>	Allows the user to enter text

<https://reactnative.dev/docs/intro-react-native-components>

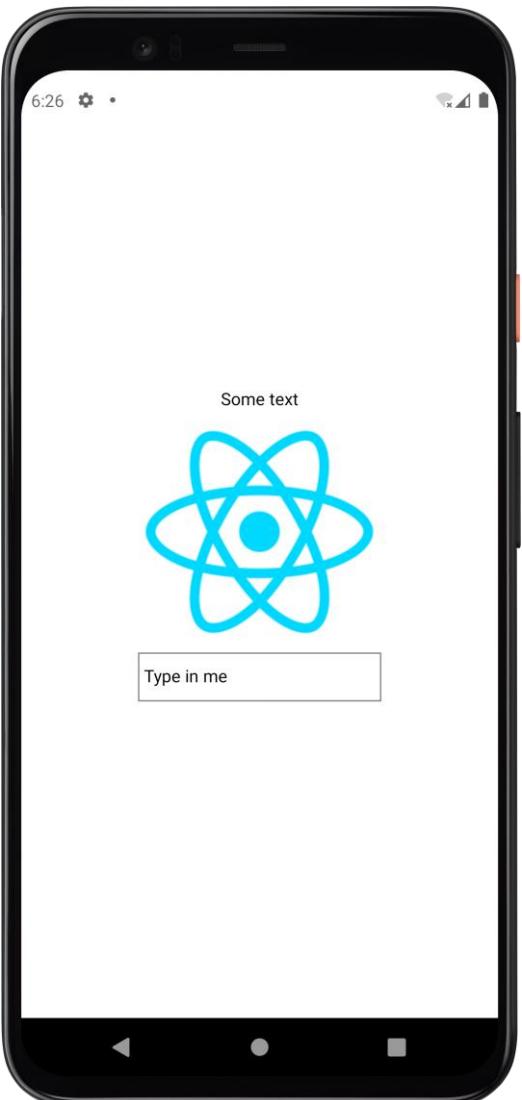
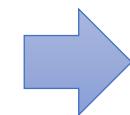
# 3. React Native - Core Components

```
import React from 'react';
import { StyleSheet, View, Text, Image, TextInput } from 'react-native';

const App = () => {
  return (
    <View style={styles.container}>
      <Text>Some text</Text>
      <Image source={require('./assets/react.png')} style={{width: 200, height: 200}} />
      <TextInput
        style={{
          width: 200,
          height: 40,
          borderColor: 'gray',
          borderWidth: 1,
        }}
        defaultValue="Type in me"
      />
    </View>
  );
};

export default App;

const styles = StyleSheet.create({
  container :{
    justifyContent: 'center', //Centered horizontally
    alignItems: 'center', //Centered vertically
    flex:1
  }
});
```



Fork me on GitHub

# Table of contents

1. Introduction

2. React

3. React Native

4. Flutter

- Setup
- Dart
- Widgets
- Hello World
- Examples

5. Takeaways

# 4. Flutter

- **Flutter** is an open source cross-platform framework created by Google used to develop applications for mobile (Android, iOS), web, and desktop from a single codebase
- Flutter applications are written in **Dart** language
  - Dart programming language developed by Google since 2011
  - Dart aims to help developers build UIs effectively
  - Dart is open source, object-oriented, and statically typed



<https://flutter.dev/>



<https://dart.dev/>

## 4. Flutter - Setup

- To install Flutter and Dart in our machine, we use the following tutorial:
  - <https://docs.flutter.dev/get-started/install>
- Once it is installed, we can execute the command-line tools flutter and dart:

```
> flutter --version
Flutter 3.29.3 • channel stable • https://github.com/flutter/flutter.git
Framework • revision ea121f8859 (2 weeks ago) • 2025-04-11 19:10:07 +0000
Engine • revision cf56914b32
Tools • Dart 3.7.2 • DevTools 2.42.3

> dart --version
Dart SDK version: 3.7.2 (stable) (Tue Mar 11 04:27:50 2025 -0700) on "windows_x64"
```

- We can use the following command to verify our installation:

```
> flutter doctor -v
```

# 4. Flutter - Dart

- The key features of Dart are the following:
  - Compiled language
    - Can be compiled to native machine code (for mobile/desktop apps) or JavaScript (for web apps)
    - Uses Just-In-Time (JIT) compilation during development for hot reload
    - Uses Ahead-Of-Time (AOT) compilation for production for optimized performance
  - Object-oriented and supports for functional programming
    - Classes, inheritance, interfaces, mixins
    - Supports lambdas (anonymous functions), higher-order functions, and collections (like map, filter, reduce)
  - Static typing (types are checked at compile time)
  - Null safety (to prevent null reference exceptions)
  - Asynchronous programming with `async/await`
    - Built-in support for futures and streams for handling async operations

# 4. Flutter - Dart

- Hello world:

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

```
> dart hello_world.dart  
Hello, Dart World!
```

- Variables and data types:

```
void main() {  
    // Variables (type inference with 'var')  
    var name = "Alice"; // String  
  
    int age = 25; // Explicit type  
    double height = 5.9;  
    bool isStudent = true;  
  
    // Dynamic type (can change at runtime)  
    dynamic dynamicVar = "Hello";  
    dynamicVar = 42; // Now an int  
  
    // Constants (compile-time)  
    const PI = 3.14;  
    final currentTime = DateTime.now(); // Runtime constant  
  
    print("$name is $age years old"); // String interpolation  
}
```

```
> dart variables_data_types.dart  
Alice is 25 years old
```

# 4. Flutter - Dart

- Control flow and loops:

```
void main() {  
    int score = 85;  
  
    // If-else  
    if (score >= 90) {  
        print("A");  
    } else if (score >= 80) {  
        print("B"); // Output: B  
    } else {  
        print("C");  
    }  
  
    // For loop  
    for (var i = 0; i < 3; i++) {  
        print(i); // 0, 1, 2  
    }  
  
    // While loop  
    int count = 0;  
    while (count < 2) {  
        print("Count: $count"); // Count: 0, Count: 1  
        count++;  
    }  
  
    // Switch-case  
    String grade = "B";  
    switch (grade) {  
        case "A":  
            print("Excellent!");  
            break;  
        case "B":  
            print("Good!"); // Output: Good!  
            break;  
        default:  
            print("Unknown");  
    }  
}
```

```
> dart control_flow.dart  
B  
0  
1  
2  
Count: 0  
Count: 1  
Good!
```

Fork me on GitHub

# 4. Flutter - Dart

- Functions:

```
// Basic function
void greet(String name) {
    print("Hello, $name!");
}

// Optional positional parameters
void sayHello(String name, [String? title]) {
    print("Hello, ${title ?? ''} ${name}");
}

// Named parameters (with defaults)
void describe({String name = "User", int age = 0}) {
    print("$name is $age years old");
}

// Main function
void main() {
    greet("Alice"); // Hello, Alice!

    describe(name: "Bob", age: 30); // Bob is 30 years old.

    sayHello("Alice"); // Hello, Alice
    sayHello("Alice", "Dr."); // Hello, Dr. Alice

    // Arrow function (short syntax)
    int add(int a, int b) => a + b;
    print(add(2, 3)); // 5
}
```

```
> dart functions.dart
Hello, Alice!
Bob is 30 years old.
Hello, Alice
Hello, Dr. Alice
5
```

Fork me on GitHub

# 4. Flutter - Dart

- Collections (list, sets, maps):

```
void main() {
    // List (ordered, mutable)
    List<String> fruits = ["Apple", "Banana"];
    fruits.add("Cherry");
    print(fruits[1]); // Banana

    // Set (unique items)
    Set<int> numbers = {1, 2, 2, 3}; // {1, 2, 3}
    print(numbers.contains(2)); // true

    // Map (key-value pairs)
    Map<String, int> ages = {
        "Alice": 25,
        "Bob": 30,
    };
    print(ages["Bob"]); // 30
}
```

```
> dart collections.dart
Banana
true
30
```

Fork me on GitHub

# 4. Flutter - Dart

- Classes and objects:

```
// Class with constructor
class Person {
    String? name;
    int age;

    // Constructor (short syntax)
    Person(this.name, this.age);

    // Named constructor (multiple constructor with
    // descriptive name for different initialization scenarios)
    Person.guest() : name = "Guest", age = 0;

    // Method
    void introduce() {
        print("I'm $name, $age years old");
    }
}

void main() {
    var alice = Person("Alice", 25);
    alice.introduce();

    var guest = Person.guest();
    guest.introduce();
}
```

```
> dart oop.dart
I'm Alice, 25 years old
I'm Guest, 0 years old
```

Fork me on GitHub

# 4. Flutter - Dart

- Mixins:

```
// Mixins are a way of defining code that can
// be reused in multiple class hierarchies
mixin Swimming {
    void swim() => print("Swimming!");
}

mixin Flying {
    void fly() => print("Flying!");
}

// Apply mixins to a class
class Duck with Swimming, Flying {
    void quack() => print("Quack!");
}

void main() {
    var duck = Duck();
    duck.swim(); // Output: "Swimming!"
    duck.fly(); // Output: "Flying!"
    duck.quack(); // Output: "Quack!"
}
```

```
> dart mixins.dart
Swimming!
Flying!
Quack!
```

Fork me on GitHub

# 4. Flutter - Dart

- Async programming:

```
// Future represents a value (String) that will be available later
// async: Marks a function as asynchronous
Future<String> fetchUser() async {
    // await: Pauses execution until the Future completes (without blocking other code)
    await Future.delayed(Duration(seconds: 2)); // Simulate network request
    return "Alice";
}

// Using async/await
void getUser() async {
    String user = await fetchUser(); // Waits for fetchUser() to complete
    print("User: $user"); // Prints after 2 seconds
}

void main() {
    getUser(); // Starts the async operation
    print("Loading..."); // Runs immediately (non-blocking)
}
```

```
> dart async.dart
Loading...
User: Alice
```

Fork me on GitHub

# 4. Flutter - Dart

- Error handling:

```
void main() {  
  try {  
    // The operator ~/ divides two numbers and  
    // returns the result as an integer while the  
    // operator / returns the result as a double  
    var result = 100 ~/ 0;  
    print(result);  
  } catch (e) {  
    print("Error: $e");  
  } finally {  
    print("Done");  
  }  
}
```

```
> dart error_handling.dart  
Error: IntegerDivisionByZeroException  
Done
```

Fork me on GitHub

# 4. Flutter - Widgets

- Flutter provides a rich set of core **widgets** to build responsive Uis, such as:
- Basic (UI structure and layout)
  - Text, Container, Row & Column, Stack, Padding, Center, SizedBox, Expanded & Flexible
- Interactive
  - Buttons, TextField, Checkbox, Radio, Switch, Slider, GestureDetector, InkWell, ...
- Platform-specific
  - Material (Android): MaterialApp, Scaffold, AppBar, FloatingActionButton, Card, ...
  - Cupertino (iOS): CupertinoApp, CupertinoNavigationBar, CupertinoButton, CupertinoPicker, ...
- Navigation and routing
  - Navigator, PageRoute, BottomNavigationBar, TabBar & TabBarView, ...
- State management
  - StatefulWidget, InheritedWidget, Provider, ValueNotifier, ChangeNotifier
- List and grids
  - ListView, GridView, ListTile, ListView.builder, ...

<https://docs.flutter.dev/ui/widgets>

# 4. Flutter - Hello World

- To implement a “Hello World” app in Flutter, we can do the following:

1. Create a new Flutter project

```
> flutter create hello_world
```

2. We can check and edit our app (using Dart)

The official doc recommend to use Visual Studio Code for coding (although other IDEs can be used)



Visual Studio Code

<https://code.visualstudio.com/>

3. Run the app

```
> cd hello_world  
> flutter run
```

# 4. Flutter - Hello World

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

void main() {
  runApp(const MyApp());
}

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

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key, required this.title});

  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}
```

```
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            const Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '_counter',
              style: Theme.of(context).textTheme.headlineMedium,
            ),
          ],
        ),
        floatingActionButton: FloatingActionButton(
          onPressed: _incrementCounter,
          tooltip: 'Increment',
          child: const Icon(Icons.add),
        ),
      );
  }
}
```

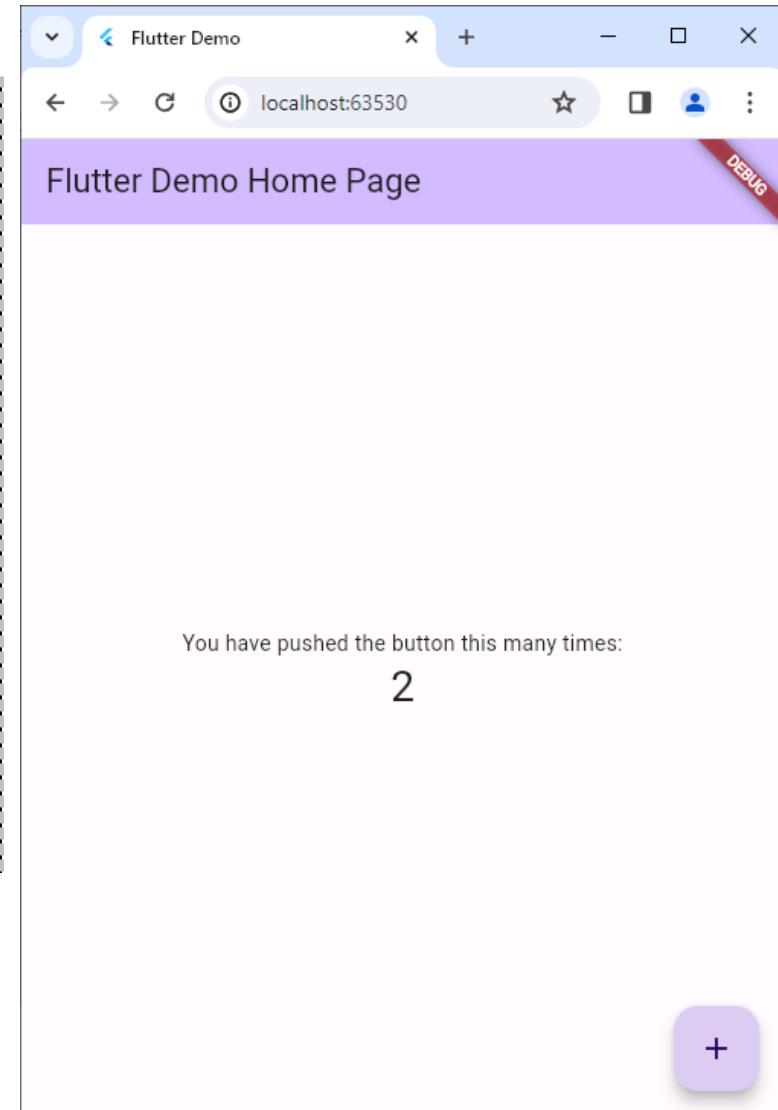
# 4. Flutter - Hello World

```
> flutter run
Connected devices:
Windows (desktop) • windows • windows-x64      • Microsoft Windows [Version 10.0.19044.3086]
Chrome (web)        • chrome   • web-javascript • Google Chrome 123.0.6312.86
Edge (web)          • edge     • web-javascript • Microsoft Edge 123.0.2420.53
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (or "q" to quit): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...           13.5s
This app is linked to the debug service: ws://127.0.0.1:63553/Q8vuDIwFoCo=/ws
Debug service listening on ws://127.0.0.1:63553/Q8vuDIwFoCo=/ws

To hot restart changes while running, press "r" or "R".
For a more detailed help message, press "h". To quit, press "q".
```

A Dart VM Service on Chrome is available at: <http://127.0.0.1:63553/Q8vuDIwFoCo=>  
The Flutter DevTools debugger and profiler on Chrome is available at:  
<http://127.0.0.1:9100?uri=http://127.0.0.1:63553/Q8vuDIwFoCo=>

The same codebase can  
be deployed as a desktop  
or web app

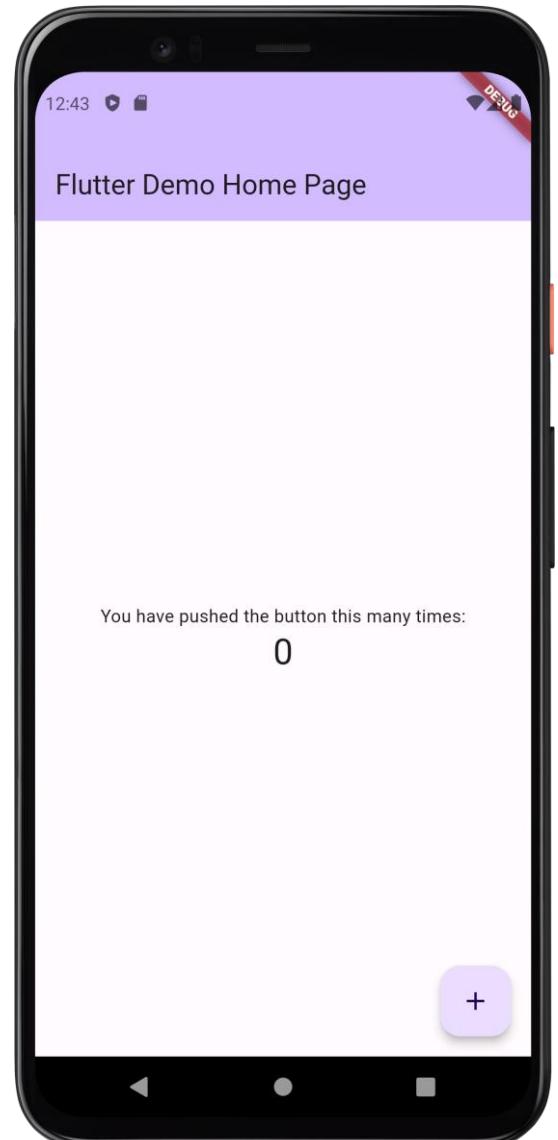


# 4. Flutter - Hello World

```
> flutter run
Launching lib\main.dart on Android SDK built for x86 in debug mode...
Running Gradle task 'assembleDebug'...                                2,477ms
✓ Built build\app\outputs\flutter-apk\app-debug.apk.
Installing build\app\outputs\flutter-apk\app-debug.apk...             915ms
Syncing files to device Android SDK built for x86...                  48ms

Flutter run key commands.
r Hot reload.
R Hot restart.
h List all available interactive commands.
d Detach (terminate "flutter run" but leave application running).
c Clear the screen
q Quit (terminate the application on the device).

A Dart VM Service on Android SDK built for x86 is available at:
http://127.0.0.1:64007/fgVC3u2TxHg=/
The Flutter DevTools debugger and profiler on Android SDK built for x86 is available at:
http://127.0.0.1:9100?uri=http://127.0.0.1:64007/fgVC3u2TxHg=/
D/eglCodecCommon( 6814): setVertexArrayObject: set vao to 0 (0) 1 0
D/EGL_emulation( 6814): eglMakeCurrent: 0xe064b780: ver 2 0 (tinfo 0xc76ef3c0)
D/eglCodecCommon( 6814): setVertexArrayObject: set vao to 0 (0) 1 0
```



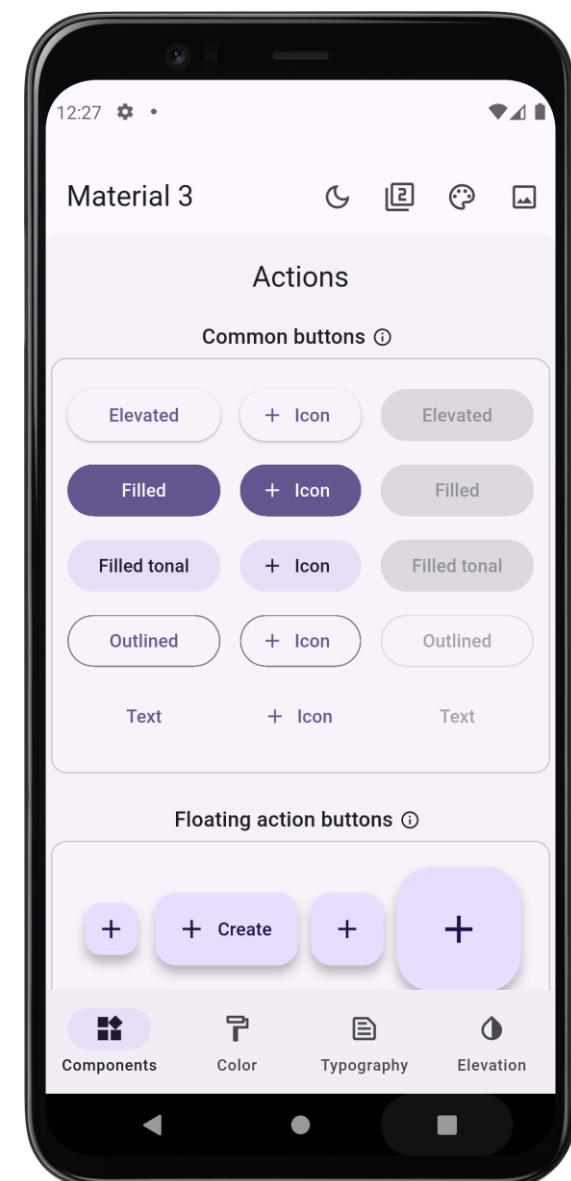
If we have a running AVD, the same app will be deployed as an Android APK

# 4. Flutter - Examples

- There are plenty of sample apps maintained by the Flutter team:

<https://github.com/flutter/samples>

For example, this app contains a comprehensive set of Material 3 components



Fork me on GitHub

# Table of contents

1. Introduction
2. React
3. React Native
4. Flutter
5. Takeaways

# 5. Takeaways

- React Native is an open source cross-platform framework for building mobile apps (Android, iOS) from a single codebase
  - React Native is based in React, and therefore, it allows to create interactive UIs based on JavaScript (or TypeScript) components
  - To ease the testing in real devices, we can use Expo Go to execute the app we are developing
- Flutter is an open source cross-platform framework created by Google that allows to develop applications for mobile (Android, iOS), web, and desktop from a single codebase
  - Flutter applications are written in Dart language
  - Dart is programming language developed by Google aimed to help developers build UIs effectively