Cross-Platform Mobile Development with Flutter
Outline

• Cross-Platform Development Tools
  • Different approaches

• Flutter Overview
  • Dart

• Hands-on with Flutter
  • Reactful programming

• Discussion on Native vs Multiplatform development
The Mobile Framework Market

• Developing multiple versions of the same application is costly!
• Smartphone platform market has recently been stable, but *smart devices* market is expanding:
  • IoT, TV, Wearables, Vehicles, ..

<table>
<thead>
<tr>
<th>Platform</th>
<th>Market Capture*</th>
<th>Language</th>
<th>IDE</th>
<th>Market Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>88%</td>
<td>Java, Kotlin</td>
<td>Android Studio</td>
<td>Google Play Store</td>
</tr>
<tr>
<td>iOS</td>
<td>11.9%</td>
<td>Objective-C, Swift</td>
<td>Xcode</td>
<td>iTunes App Store</td>
</tr>
<tr>
<td>Others (Windows, Blackberry, ..)</td>
<td>&lt; 0.1%</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

*2018 Q1 according to Statista*
Native Development

• Android
  • Android SDK + Android Studio
  • Kotlin/Java

• iOS
  • Xcode
  • Objective-C / Swift

• Maybe even a desktop app?
Cross-Platform Development

- Alternative principle to native app development
- Cross-Platform Tools (CPT), sometimes Multi-platform tools
- Share a code base between platforms
  - As large as possible
    “The ultimate goal of cross-platform mobile app development is to achieve Native app performance and run on as many platforms as possible”[1]
- Decrease dev. and maintenance cost of applications

Cross-Platform Development

- Alternative principle to native app development
- Cross-Platform Tools (CPT), sometimes Multi-platform tools
- Share a code base between platforms as large as possible
- Decrease dev. and maintenance cost of applications

The ultimate goal of cross-platform mobile app development is to achieve Native app performance and run on as many platforms as possible”[1]

Can you name any multiplatform technologies you are already using, daily?

Web-based CPT

Historically the most common CPT approach.

- App UI with HTML + CSS
- Functionality with JavaScript

Strategies:

1. “Normal webpage”
   - Load the app. from a remote webserver

2. Hybrid Application
   - Webpage contained locally on the device, using a web-to-native wrapper
   - Uses platform’s built-in browser engine
     - Bridging between the WebView and Wrapper Application
   - Can be distributed on the platforms application market
More approaches and examples

• Hybrid
  • Apache Cordova, PhoneGap, Ionic

• Interpreted Approach
  • A dedicated interpreter on the device executes non-native code
  • e.g. Titanium, React Native (Javascript)

• Cross compiled
  • Non-native code is compiled to native during build
  • e.g. Xamarin (C#, .NET) , Flutter (Dart)

• Others
Interpreted approach

• Don’t rely on a WebView
• But can still write programs in non-native language
• For example, Javascript with an interpreter:
  • JavaScriptCore - the default interpreter on iOS devices
  • On Android devices, V8 is a widely used engine
• Also uses a bridging layer like Hybrid approach
Cross-compile approach

• Code is compiled to native byte code
• Thus, no bridging layer
• Instead, the CPT Framework SDK includes a cross-compiler which maps functionality to the underlying platforms’ SDK
  • E.g., Flutter
Adapted from Biørn-Hansen et al.

Runtime Abstraction Layer: WebView/Interpreter
or
Mappings between CPT framework SDK and native SDK

Native SDK API-s
• Location
• Camera
• Storage
• Phone
• …
Flutter is an app SDK for building high-performance, high-fidelity apps for iOS, Android, and web (tech preview) from a single codebase.

The goal is to enable developers to deliver high-performance apps that feel natural on different platforms. We embrace differences in scrolling behaviors, typography, icons, and more.

• Unveiled 2015, @ Dart Developer Summit under the name Sky
  • Stable v1.0 released December 2018
• Developed by Google, open-source

https://flutter.dev/docs/resources/technical-overview
Flutter – a set of tools

• React-style framework
• Dart – modern object-oriented language
• Skia 2D rendering engine
  • Works with different hardware and softw. platforms
    • Google Chrome, Android, Firefox, Sublime Text 3
    • OpenGL ES2-accelerated
• Set of widgets – uniform design language
• Development tools
• Can target
  • Android
  • iOS (still need Apple hardware to develop)
  • (Web applications)
• Created by Google in 2011, OO language
• Syntax similar to Java, Javascript
• Statically typed, has type inference

// Define a function.
printInteger(int aNumber) {
  print('The number is $aNumber.'); // Print to console.
}

// This is where the app starts executing.
main() {
  var number = 42; // Declare and initialize a variable.
  printInteger(number); // Call a function.
}

Web-based dart console: https://dartpad.dev/
https://dart.dev/samples
Some differences to Java, Kotlin

• Dart doesn’t have keywords public, protected, private.
  • If an identifier starts with an underscore (_), it’s private to its library.
    ```dart
    var _gameScore = 0;
    ```

• Unlike Kotlin, you have to handle null-safety

• *Spread* operator for lists
  ```dart
  var list = [1, 2, 3];
  var list2 = [0, ...list]; // list2 is now [0,1,2,3]
  ```

• Asynchronous calls built-in
  • async, await keywords
    [https://dart.dev/codelabs/async-await](https://dart.dev/codelabs/async-await)

[https://dart.dev/guides/language/language-tour](https://dart.dev/guides/language/language-tour)
Dart Functions

- => shorthand

- Both positional and named parameters supported

- Positional:
  ```dart
  void sendMessage(String from, String msg) { }
  void sendMessage(String from, String msg, [int priority]) { }
  void sendMessage({String from, String msg, int priority}) { }
  sendMessage(from: "Joe", msg:"Hi", priority:1 );
  ```

- Named arguments (all optional).
  - Define:
    ```dart
    void sendMessage({String from, String msg, int priority}) { }
    ```
  
  - Invoke:
    ```dart
    void sendMessage({String from, @required String msg, int priority}) { }
    ```

- Specify required:

More info: https://dart.dev/guides/language/language-tour#functions
Dart Constructors

• No data class like Kotlin.
• However, there is a short-hand for initializing variables

```dart
class Person {
  String name;
  int age;

  Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
}
```

// Same can be written as
```dart
class Person {
  String name;
  int age;

  Person (this.name, this.age);
}
```
Flutter Widgets

• Flutter’s UIs are made up of Widgets
  • widget is an **immutable declaration** of part of the user interface

• “Everything’s a widget”
  • Even layouts, fonts, colour themes

• Roughly equivalent to *View*-s in Android
  • Can be Visible (Button, Text) or Invisible (Container)

• Flutter Widgets are **immutable**!
  • Flutter creates a new tree of widget instances. In comparison, an Android view is drawn once and does not redraw until invalidate is called.

https://flutter.dev/docs/resources/technical-overview#everythings-a-widget
Widget hierarchy

• A widget’s main job is to provide a `build()` method
  • describes how to display the widget in terms of other, lower level widgets.

• Widgets are composed into a tree.
  • The root widget acts as the “application” object.
  • Child objects inherit parents’ properties.

• You can also control the layout of a widget by composing it with other widgets.
  • For example, to center a widget, you wrap it in a `Center` widget. There are widgets for padding, alignment, row, columns, and grids, ...

https://flutter.dev/docs/development/ui/widgets-intro
Image: https://flutter.dev/docs/development/ui/layout
Hello World in Flutter

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

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(home: Text("Hello World!"));
  }
}
```
Material Design Library

• Material is a visual design language that is standard on mobile and the web.
  • Flutter offers a rich set of Material widgets.

• MaterialApp – root Widget of the library

• Scaffold Widget – provides standard material components
  • a default app bar with title
  • body property - holds the widget tree for the home screen. The widget subtree can be quite complex.

• Similarly, a Cupertino library exists

https://api.flutter.dev/flutter/material/MaterialApp-class.html
Material Design

Cupertino

Images from: https://pub.dev/packages/flutter_platform_widgets
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text("Lecture App"),
        ), // AppBar
        body: Center(
          child: Column(
            children: <Widget>[Text("Hello"), Text("World"), RaisedButton(
              child: Text("Greet!"), onPressed: () => print("Hi"), color: Colors.amberAccent,
            ) // RaisedButton
          ]), // <Widget>[]
        )
      )
    )
  }
}
Understanding the widget tree

• Define the unique characteristics of a widget by implementing a `build()` function
  • `Build()` returns a tree (or hierarchy) of widgets.

• With this nesting (composition), you form a widget tree

• Flutter efficiently renders the UI based on the widget descriptions
Flutter Declarative UI

• Flutter uses a declarative approach to program the UI; including updating the UI

• In Native Android, you declare the XML layout, but then programmatically directly modify elements of the inflated layout

• Flutter, by contrast, lets the developer describe the current UI state and leaves the transitioning to the framework.

• Widgets are immutable, lightweight blueprints, not the actual UI rendered objects
Declarative vs Imperative UI example

• Assume the user entered an invalid input and we want to show this in UI with color and text:
  • Imperative:
    ```java
    // Referring to existing b,c objects
    b.setBackgroundColor("#FF0000");
    c.setText("Invalid input!");
    ```
  • Flutter:
    ```dart
    return ViewB(
        color: red,
        child: Text("Invalid input!"),
    );
    ```

https://flutter.dev/docs/get-started/flutter-for/declarative
State changes in UI

• Instead of updating the UI when state changes, you only update the state, and that triggers a redraw of the user interface.

• UI is automatically updated when state changes
Stateful vs stateless widgets

• Stateless widgets - immutable
  • their properties can’t change—all values are final.
  • Usually Icon, Button, Text

• Stateful widgets
  • maintain state that might change during the lifetime of the widget.
  • for example, it can change its appearance in response to events triggered by user interactions or when it receives data.
  • E.g. Checkbox, Radio, Slider, InkWell, Form, and TextField

https://flutter.dev/docs/get-started/codelab#step-3-add-a-stateful-widget
Stateful widgets

• Implement two classes:
  1) a StatefulWidget
     • Immutable, re-created
  2) a State class
     • Persists over lifetime of widget

• When the widget’s state changes, the state object calls setState(), telling the framework to redraw the widget.

https://flutter.dev/docs/get-started/codelab#step-3-add-a-stateful-widget
Creating a StatefulWidget

```dart
class MyStatefulWidget extends StatefulWidget {
  @override
  State<MyStatefulWidget> createState() {
    return MyState();
  }
}
```

```dart
class MyState extends State {
  var _gameScore = 0;

  @override
  Widget build(BuildContext context) {
    return Text("Score is ${_gameScore}");
  }
}
```

- Widget has to create the State object
- State implements the build() method
- You set the new state values inside the setState() method
Code Example
Managing State

• There are several approaches to who should manage the state of a widget.
  • Parent widget manages state, forwards state values to child widget in build
  • Widget manages its own state

• Basic rule of thumb:
  • If the state is just to do with aesthetics (e.g. icon changes, manage it within the widget itself)
  • If it involves user data, manage it in the parent

• If the tree is large, forwarding state can be cumbersome
  • InheritedWidget, ScopedModel for these cases

https://flutter.dev/docs/development/ui/interactive#managing-state
Stateful Widget A

```dart
var myState;
void updateAState() { .. }

Button
onPressed: updateAState()

text: myState.toString()
```

Stateful Widget B

```dart
var myState;
void updateBState() { .. }

Button
onPressed: updateBState()

text: myState.toString()
```

Pass down pointers to `myState`, `updateBState`

Stateless Widget C

```dart
var myState;

Button
onPressed: updateBState()

text: myState.toString()
```

**Widget manages own state**

**Parent manages state**

https://flutter.dev/docs/development/ui/interactive#managing-state
Flutter System Overview

Image: https://flutter.dev/docs/resources/technical-overview
Rendering in Flutter

There’s more than one tree

```
App
|-- ContainerWidget
    |-- TextWidget
        Color:red
        Text: „Foo Bar“

Element

AppRender
|-- ContainerWidgetRender
    |-- TextWidgetRender
```

Foo Bar
Rendering Pipeline

https://www.youtube.com/watch?v=UUfXWzp0-DU
Discovering Widgets

03/12/2019

Mobile & Cloud Lab. Institute of Computer Science, University Of Tartu

Widget catalog

Create beautiful apps faster with Flutter's collection of visual, structural, platform, and interactive widgets. In addition to browsing widgets by category, you can also see all the widgets in the widget index.

Accessiblity
Make your app accessible.

Animation and Motion
Bring animations to your app.

Assets, Images, and Icons
Manage assets, display images, and show icons.

Async
Async patterns to your Flutter application.

Basics
Widgets you absolutely need to know before building your first Flutter app.

Cupertino (iOS-style widgets)
Beautiful and high-fidelity widgets for current iOS design language.

https://flutter.dev/docs/development/ui/widgets
Flutter Project Structure

• Projects are generated by the Flutter command-line tool (which IDE uses)
• Each project has a lib folder, which contains .dart files, e.g. main.dart
• Platform-specific sub-folders
  • android, ios
  • Source and compiled code
• pubspec.yaml
  • Defines project dependencies, versions
Adding dependencies, assets

• Flutter uses Dart’s own build system, and the Pub package manager.
  • The tools delegate the building of the native Android and iOS wrapper apps to the respective build systems.

• External dependencies to use in Flutter are generally defined in pubspec.yaml
  • Find Flutter packages in Pub: https://pub.dev/flutter

• Assets are also declared in pubspec.yaml
pubspec.yaml

```yaml
name: flutter_app
description: A new Flutter application.

version: 1.0.0+1

environment:
  sdk: ">=2.1.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
cupertino_icons: ^0.1.2

dev_dependencies:
  flutter_test:
    sdk: flutter

flutter:
  uses-material-design: true
```

[https://dart.dev/tools/pub/pubspe](https://dart.dev/tools/pub/pubspe)
Example with assets

Specifying in .yaml:

```yaml
# The following section is specific to Flutter.
flutter:
  uses-material-design: true

assets:
  - images/test.png
```

Creating a Widget:

```dart
Image.asset("images/test.png",
  width: 600,
  height: 240,
  fit: BoxFit.cover,
)
```
Accessing mobile features

• How do I use the Camera, Maps, SMS, ... of the smartphone from Flutter?

• Will it work exactly the same on iOS and Android?

• This functionality is split into plugins

• Repository of packages is available at:
  https://pub.dev/flutter

• E.g., geolocator for GPS
  • https://pub.dev/packages/geolocator

• If a functionality is missing, you can try and implement it yourself

  https://flutter.dev/docs/get-started/flutter-for/android-devs#flutter-plugins
https://flutter.dev/docs/development/platform-integration/platform-channels
Discussion

So is there any point in learning native anymore?

Multi-platform w/ Flutter
- Shared code-base, 2(+1) target platforms
- Dart – fairly new language
- Rapid development
- Some overhead
- Same UI engine on older devices
- Still have to track plugins features on all target platforms

Native
- Android TV, Android Wear
- Access to latest APIs immediately
- Kotlin/Java – larger developer pool
- Although Flutter is fast, theoretically fastest result will be native
- Relying on backwardsCompat libraries
- Ecosystem is larger
References & Materials:


