What the Flutter?

Jakob Hoyer
14.11.2022
Android Team Lead
@Mooncascade
Outline

Cross-Platform Development
Flutter
Dart
Architecture
Cross-Platform Dev
Pros & cons of cross-platform

+ Cheaper
+ More consistent on different platforms
+ Easier to develop
+ Faster?

- Performance issues
- Support and integrations
- more layers - more issues
- Look & feel?
<table>
<thead>
<tr>
<th>Technology</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cordova/Phonegap/Ionic</td>
<td>2009</td>
</tr>
<tr>
<td>Xamarin</td>
<td>2013</td>
</tr>
<tr>
<td>React Native</td>
<td>2015</td>
</tr>
<tr>
<td>Flutter</td>
<td>2018</td>
</tr>
<tr>
<td>Kotlin Multiplatform</td>
<td>2020</td>
</tr>
</tbody>
</table>
Native applications

https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514
Cordova, Phonegap, Ionic

- App written with web technologies
  - JS, HTML, CSS
  - React, Vue, Angular
- Wrapped in a native app
- Runs in a webview - browser without toolbars
Cordova, Phonegap, Ionic

https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514
Cordova, Phonegap, Ionic

https://www.geeksforgeeks.org/what-is-apache-cordova/
React Native

- App written with web technology
  - React, Redux
- but the UI is native
- Bridge between JS and native realms
  - can become a bottleneck
React Native

https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514
Xamarin

Traditional

Xamarin Forms

https://www.xamarinhelp.com/xamarin-forms-making-traditional-xamarin-obsolete/
Kotlin Multiplatform

- App logic, repositories, data sources are written in Kotlin
- UI written natively - Kotlin, Swift, JS
- Kotlin module is imported to native projects
- Compiled to machine code or JavaScript
Kotlin Multiplatform

Swift / Obj-C
UIViewController

Kotlin / Java
Activity

JS
HTML Document

Presentation
Model
Network
Storage

https://speakerdeck.com/ecgreb/developpement-android-et-ios-avec-kotlin-multiplatform
Flutter
Flutter

- Open source framework by Google, released 2018
- Build apps from a single codebase for
  - Android, iOS
  - Web
  - Windows, Linux, macOS
  - Embedded devices (TVs, smart screens, cars etc)
- Powered by Dart language
Flutter - core concepts

- Everything is a Widget
- Declarative UI
- Uses its own rendering engine - Skia
- Compiles to machine code or JavaScript
- Uses plugins to interface with the platform
Ambient Computing

Ambient computing is the concept of blending computing power into our everyday lives in a way that is embedded into our surroundings - invisible but useful.


“Focus starts to move away from any individual device towards an environment where your services are available wherever you need them.”

Fuchsia OS

- New open source operating system by Google
- Os for a diverse ecosystem of hardware and software
- Runs Flutter apps
- https://fuchsia.dev/
Everything is a Widget
Declarative UI

- Developer describes the current UI state
- Flutter calls the function whenever the state changes
- Takes the burden from developers of manually constructing and manipulating the UI
// Imperative style
b.setColor(red)
b.clearChildren()
ViewC c3 = new ViewC(...)
b.add(c3)

// Declarative style
return ViewB(
    color: red,
    child: const ViewC(),
);
Declarative UI

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

  void _incrementCounter() {
    setState(() {
      _counter++;  // state is changed (btn click)
    });
  }

  @override
  Widget build(BuildContext context) {
    return Text(
      'Counter value is $_counter',
    );
  }
}
```

← initial state

← state is changed (btn click)

Widget tree is rebuilt with new state
Declarative UI

● It’s a trend
  ○ web - React
  ○ Android - Jetpack Compose
  ○ iOS - SwiftUI
Hot reload

- edit the code & save
- changes are injected to Dart Virtual Machine
- UI is rebuilt
- app state is preserved

Performing hot reload...
Reloaded 1 of 1127 libraries in 525ms.
Performing hot reload...
Reloaded 1 of 1127 libraries in 462ms.
Performing hot reload...
Reloaded 1 of 1127 libraries in 439ms.
Performing hot reload...
Reloaded 1 of 1127 libraries in 447ms.
How it’s possible to have high quality, pixel perfect UI on every device?

- Flutter app is bundled with custom renderer
- Full control over the look and feel
- Looks same on every platform
- Skia is used in Chrome, Mozilla, Android etc

https://skia.org/
Plugins

- Flutter CLI to create, validate and publish plugins
- Federated plugins
- Awesome plugin ecosystem
- [https://pub.dev/packages?sort=popularity](https://pub.dev/packages?sort=popularity)
COME TO THE DART SIDE
Dart

- “Client-optimized language for fast apps on any platform”
- Open source, released 2013 by Google
- Object-oriented, class-based, garbage-collected, C-style syntax
- Compiles to either native code (arm, x64) or JavaScript
- **Ahead-of-Time & Just-in-Time compilation**
  - AOT to release the app to production - fast
  - JIT for development - hot reload, debugging
- **async/await**
- **sound type system**
- **sound null safety**
Widgets
Basic building blocks

- Text, Buttons, Image
- Container, Stack, Column, Row
- Padding
- Card, ListTile etc
- Awesome docs
  - Examples [https://docs.flutter.dev/development/ui/widgets](https://docs.flutter.dev/development/ui/widgets)
  - Widget of the Week videos - [https://youtu.be/XawP1i314WM](https://youtu.be/XawP1i314WM)
@override
Widget build(BuildContext context) {
  final textTheme = Theme.of(context).textTheme;
  return Card(
    margin: const EdgeInsets.only(left: 16, right: 16, top: 16),
    clipBehavior: Clip.antiAlias,
    shape: const RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(8),
    ), // RoundedRectangleBorder
    elevation: 2,
    child: Column(
      children: [
        Image.network(artwork.imageUrl),
        Padding(
          padding: const EdgeInsets.only(left: 16, right: 16, top: 32),
          child: Container(
            color: Colors.amber,
            child: Center(
              child: Text(
                artwork.title,
                style: textTheme.titleLarge,
                textAlign: TextAlign.center,
              ), // Text
            ), // Center
          ), // Padding
        ), // Padding
      ],
    ), // Card
  );
Create custom widgets to modularise and reduce nesting

1. Use `StatelessWidget` when you don’t need mutable state.
2. Implement `build()` method
Values can be provided through the constructor
class Rotation extends StatelessWidget {
  const Rotation({super.key, required this.rad, required this.child});

  final double rad;
  final Widget child;

  @override
  Widget build(BuildContext context) {
    return Container(
      transform: Matrix4.rotationX(rad),
      child: child,
    ); // Container
  }
}
- **Use** `StatefulWidget` **when you need to hold a state that might change during the lifetime of the widget.**

- **Use** `setState()` **to change the state and trigger rebuild.**
How to share state?

- Lift the state up in the widget tree
- For that we can use:
  - InheritedWidget
  - BLoC pattern
How to share state?

https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html
InheritedWidget

https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html
- Business Logic Components
- Separates presentation from business logic
- Predictable state
- Reusability
- Simple & Lightweight
- Highly Testable
- A Cubit is a simple state holder widget
- It exposes functions which can be invoked to trigger state changes
- UI components subscribe to state changes and redraw themselves

```dart
class CounterCubit extends Cubit<int> {
  CounterCubit() : super(0);

  void increment() => emit(state + 1);
}
```

https://bloclibrary.dev/#/coreconcepts?id=cubit
- A bloc is a more advanced state holder widget
- It relies on events to trigger state changes rather than functions
- It converts the incoming events into outgoing states
● Predictability - knowing the sequence of state changes as well as exactly what triggered those changes.
Let’s put it together!

- **CounterBloc** - contains the business logic
- **CounterView** - builds the UI
- **CounterPage** - binds them together
- **BlocBuilder** subscribes to state changes
- `context.read<CounterBloc>()` to access the bloc for sending events
Thank You!