Fast Prototypes with Flutter + Kotlin/Native

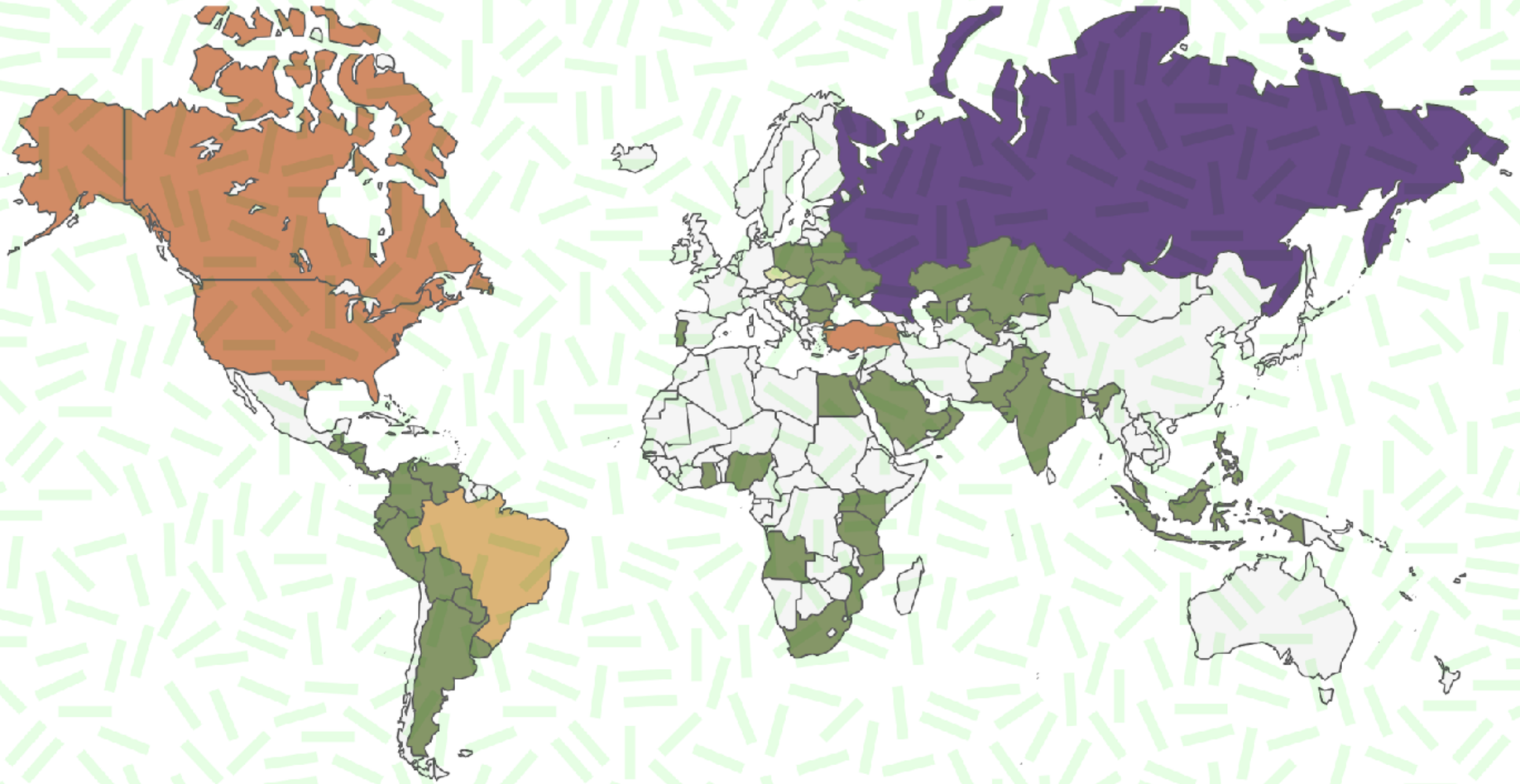
JB Lorenzo | @bennegeek

APRIL 08+09
DROIDCON BOS 19

The Story

OLX P&Tech Conference App, Berlin

The Team




Auto
Trader
.co.za


Avito


AUTOVIT.RO


Domofond


dubizzle


fixly


imovirtual


letgo


OLX


otodom.


OTOMOTO


property24

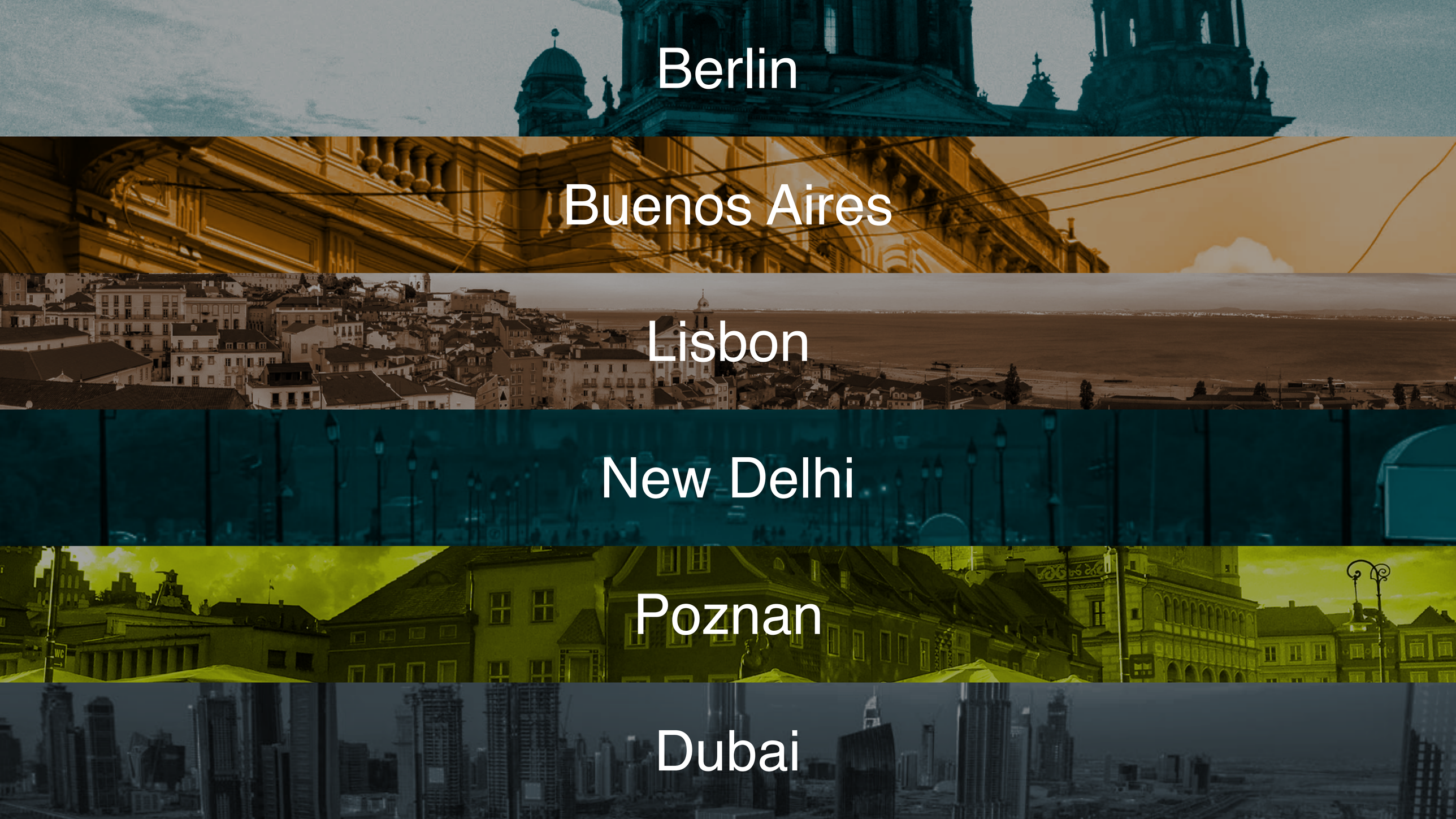

shedd


STANDVIRTUAL


STRADIA


storia

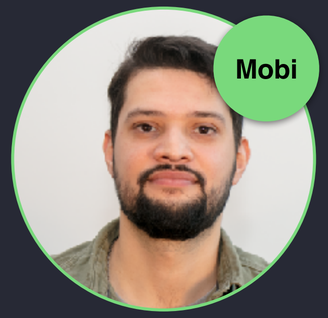

TRADUS


Berlin
Buenos Aires
Lisbon
New Delhi
Poznan
Dubai

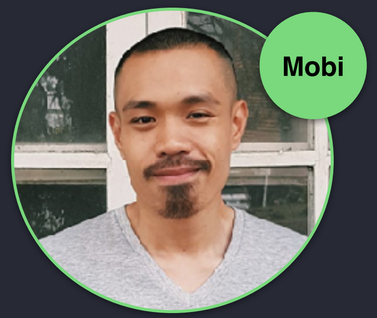
The Story

OLX P&Tech Conference App, Berlin

The Team



Cristiano Madeira



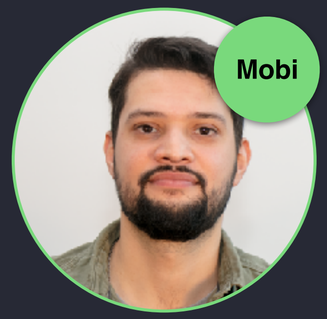
Rem Cruz

JB Lorenzo

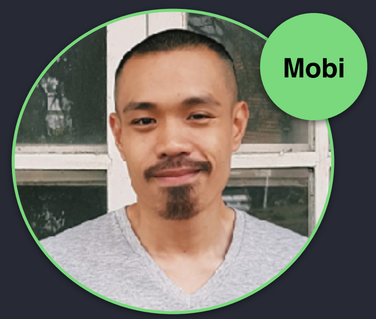
Ayelen Chavez

Karen Banzon

Caio Moritz



Cristiano Madeira



Rem Cruz

JB Lorenzo

Ayelen Chavez

Karen Banzon

Caio Moritz

* 3 devs
* 2 managers
* 1 ux/prod
* /w real work

* ~1 month deadline

Mobi = Android + iOS

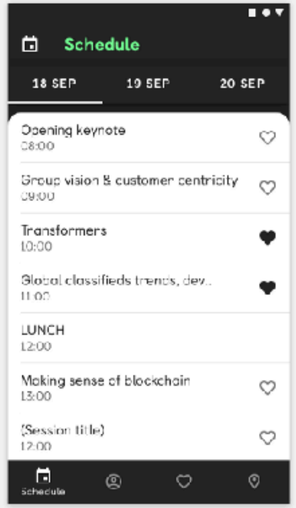
The Story

Loading Screen

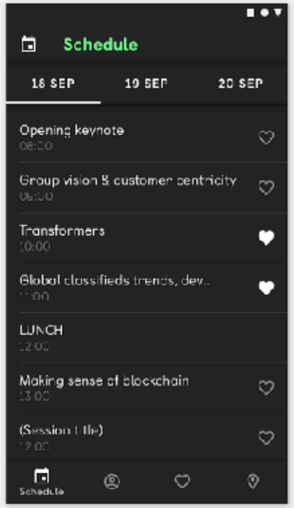
Login Page

Schedule - A

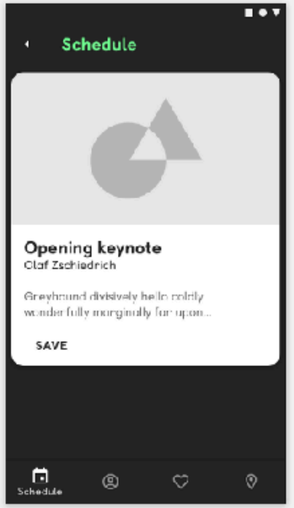
Schedule - B



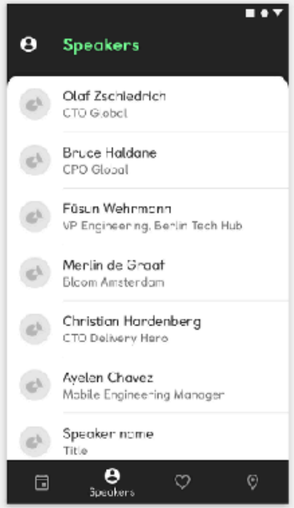
Schedule - C



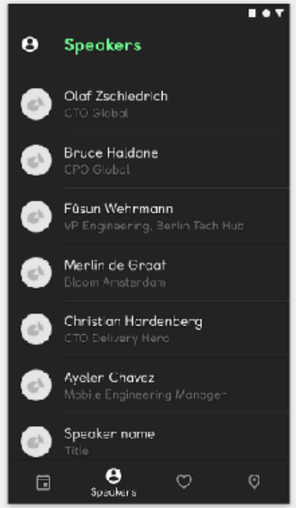
Schedule - Details



Speakers - A

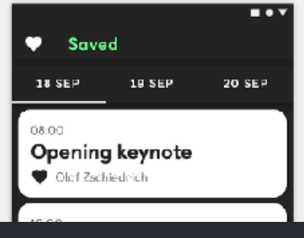


Speakers - B



Speakers - Details

Saved



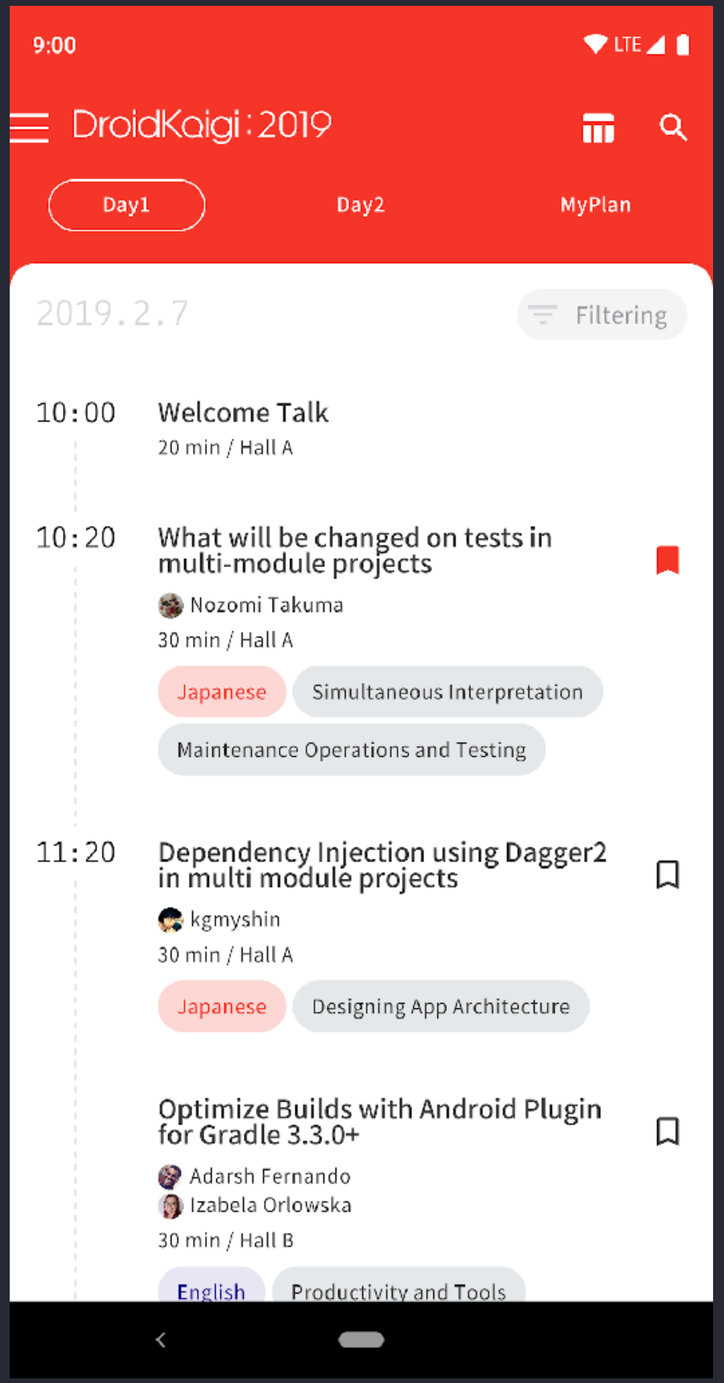
@bennegeek

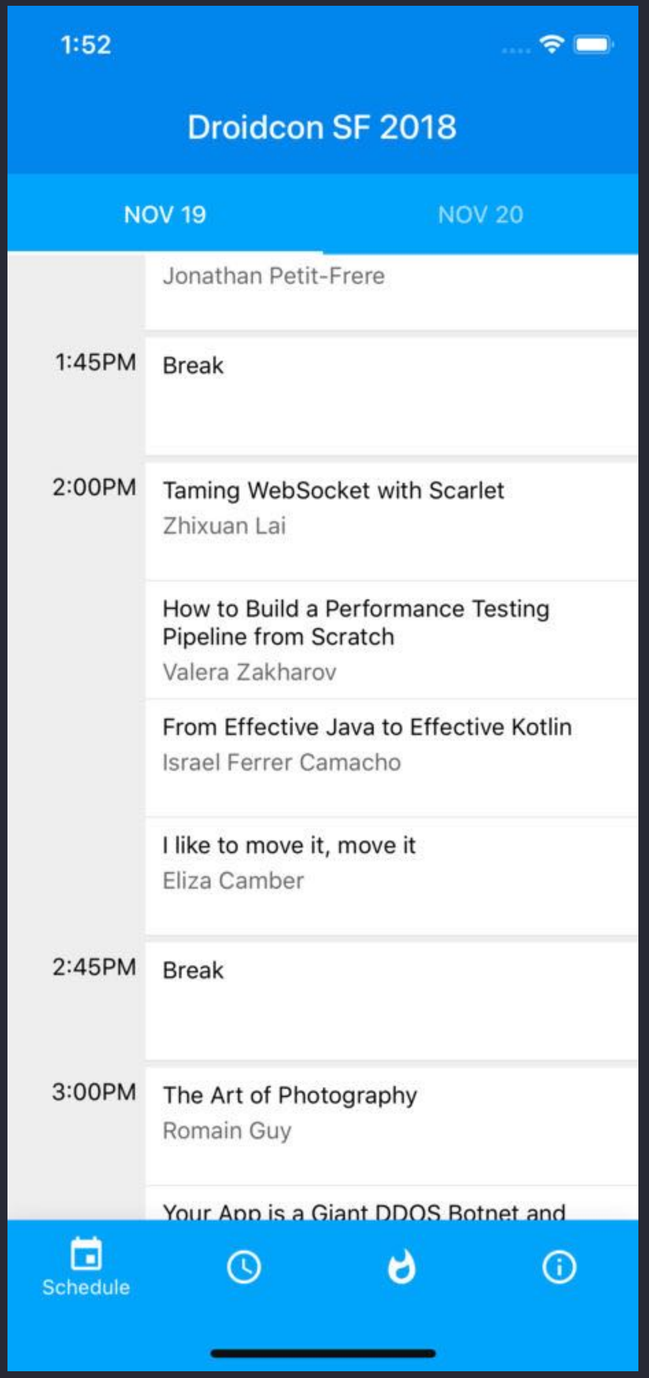
Kotlin/Native

Firebase

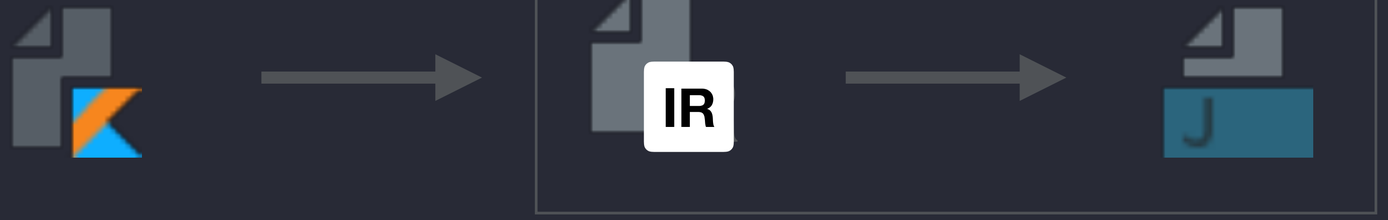
Material Components

Kotlin/Native



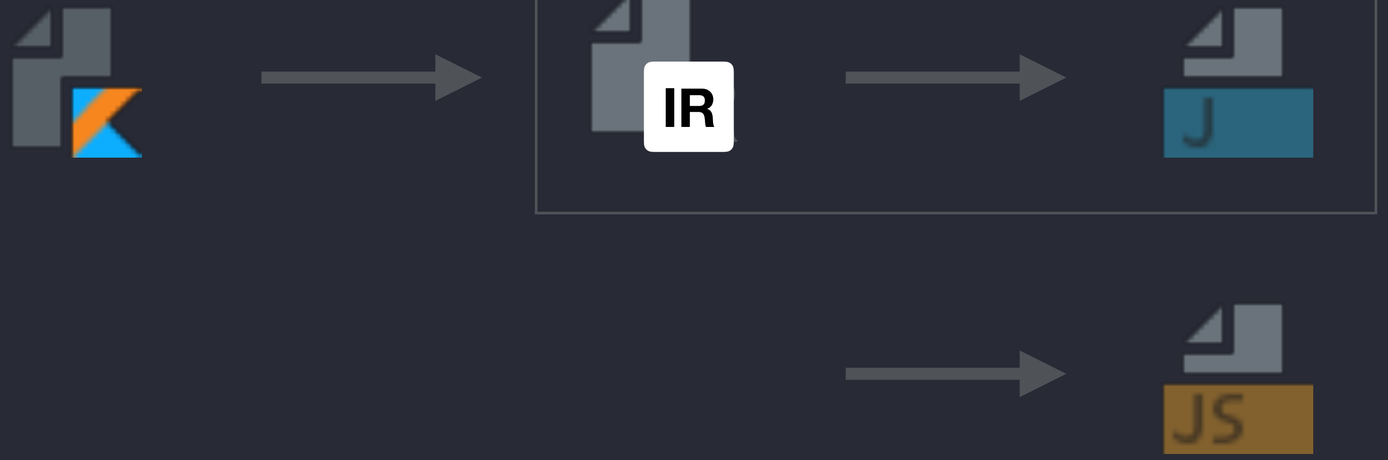


Kotlin/Native



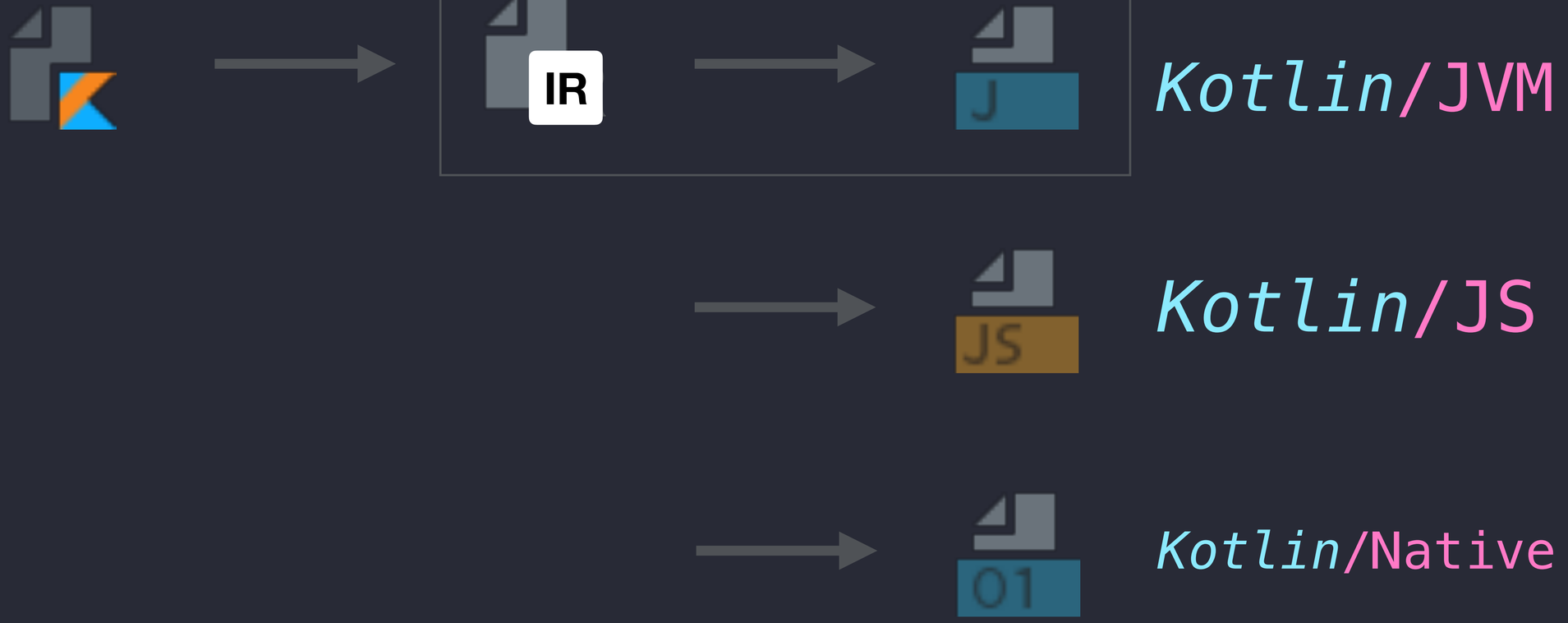
https://github.com/dkandalov/kotlin-native-snake

Kotlin/Native



https://github.com/dkandalov/kotlin-native-snake

Kotlin
Multiplatform

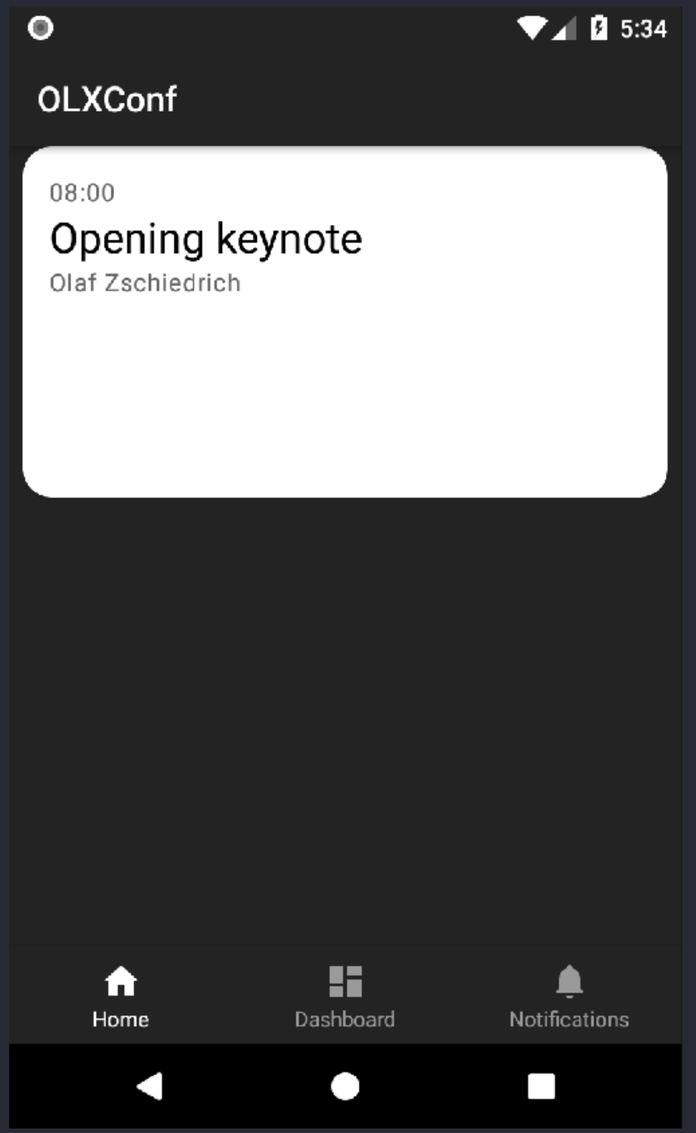


https://github.com/dkandalov/kotlin-native-snake

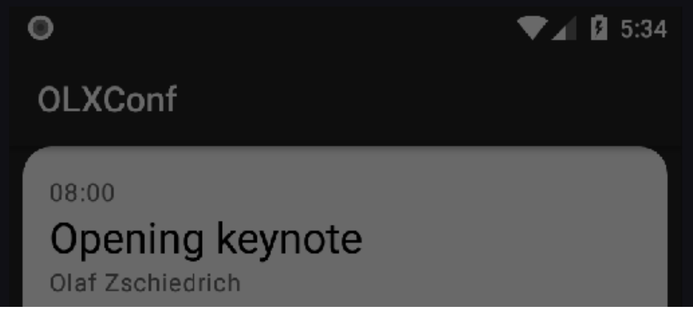
Kotlin/Native

Firebase

Material Components



Kotlin/Native



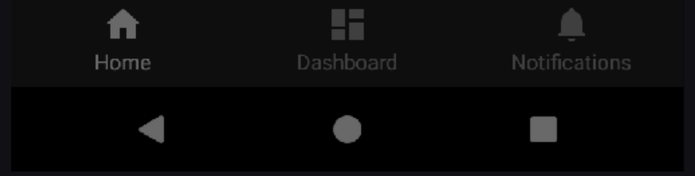
August 16th, 2018

Remerico Cruz 🏡 11:19 AM

`Note: You should not use the com.android.support and com.google.android.material dependencies in your app at the same time.`

`If you don't want to switch over to the new androidx and com.google.android.material packages yet, you can use Material Components via the com.android.support:design:28.0.0-alpha3 dependency.`

Material Components



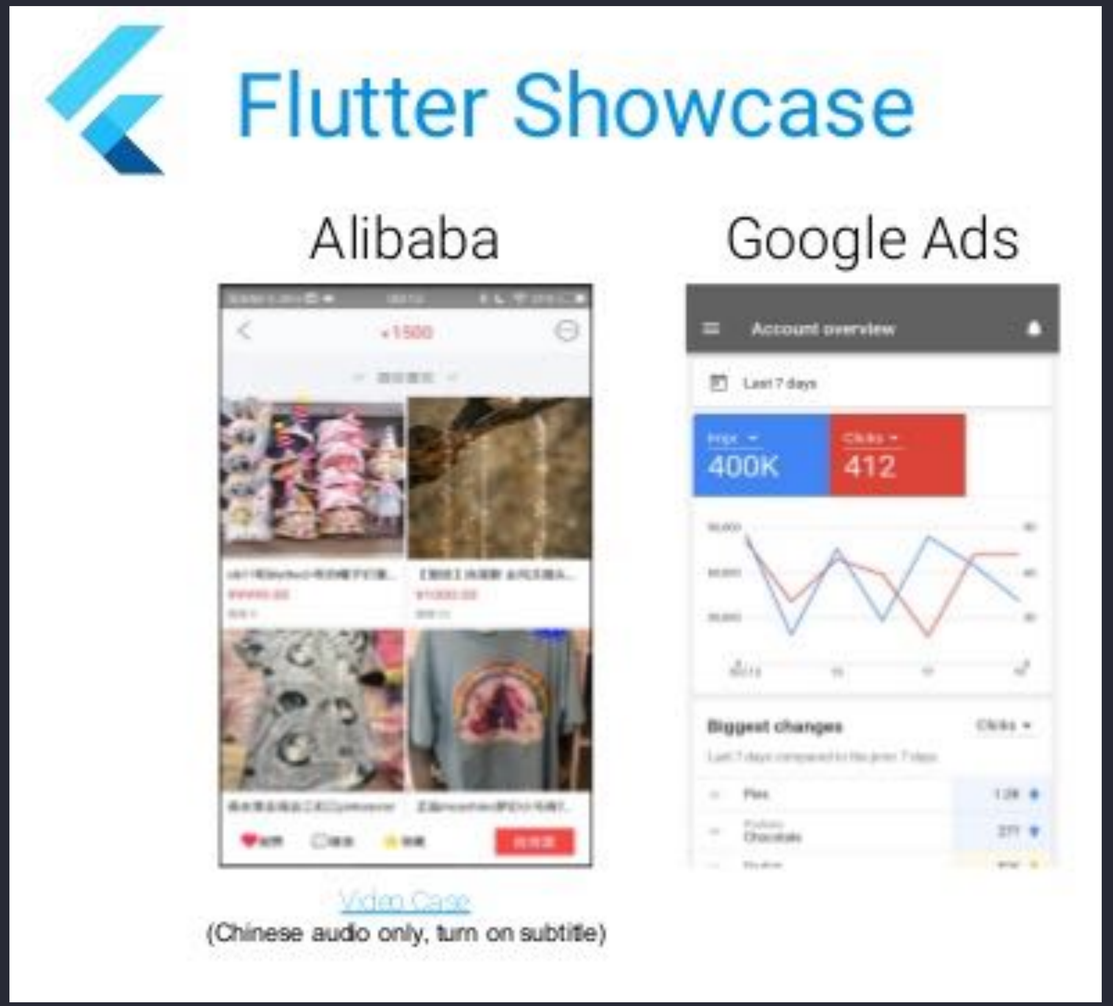
Kotlin/Native

Firebase

Material Components

Flutter

Flutter


Flutter Showcase
Alibaba
Google Ads
Account overview
Last 7 days
400K
412
Biggest changes
Xvlem Case
(Chinese audio only, turn on subtitle)

Flutter

https://buildflutter.com/how-flutter-works/

Flutter

```
    appBar: AppBar(
      leading: GestureDetector(
          child: Icon(Icons.arrow_back),
          onTap: () => Navigator.pop(context)
      ), // GestureDetector
      title: new Text(this.title),
      textTheme: TextTheme(
          title: TextStyle(
              color: Color(0xFF9BFFA0),
              fontSize: 20.0,
              fontFamily: "Castledown"
          ) // TextStyle
      ), // TextTheme
    ), // AppBar

    body: SingleChildScrollView(
      child: Card(
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.all(Radius.circular(12.0))
        ), // RoundedRectangleBorder
        margin: EdgeInsets.all(8.0),
        child: Column(
            mainAxisAlignment: MainAxisAlignment.start,
            crossAxisAlignment: CrossAxisAlignment.start,
            children: <Widget>[
              createPhoto(),

              Container(
                padding: EdgeInsets.only(
                    top: 6.0, bottom: 6.0,
                    left: 12.0, right: 12.0
                ), // EdgeInsets.only
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.start,
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: buildDetailFields()
                ), // Column
              ) // Container

            ], // <Widget>[]
          ) // Column
```


https://buildflutter.com/how-flutter-works/

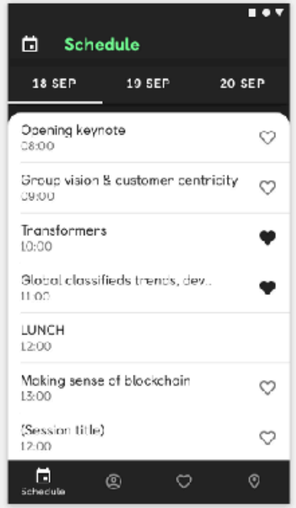
The Story

Loading Screen

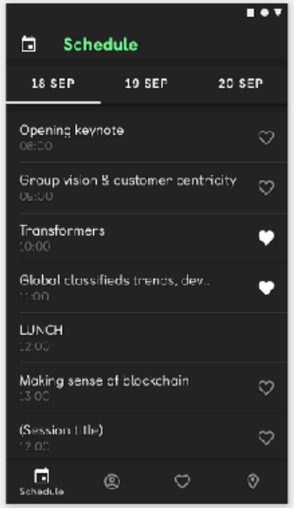
Login Page

Schedule - A

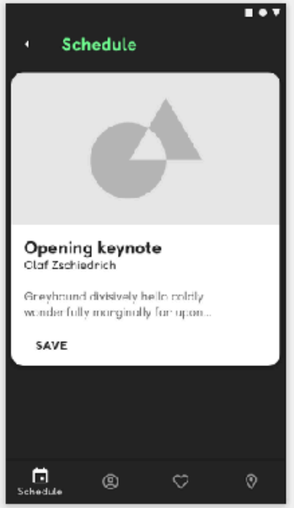
Schedule - B



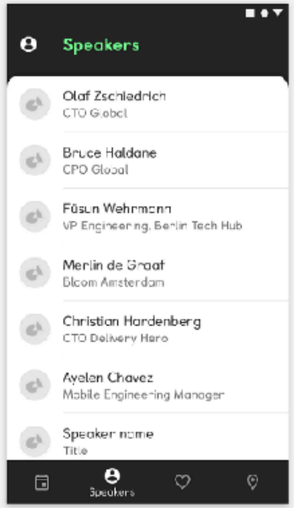
Schedule - C



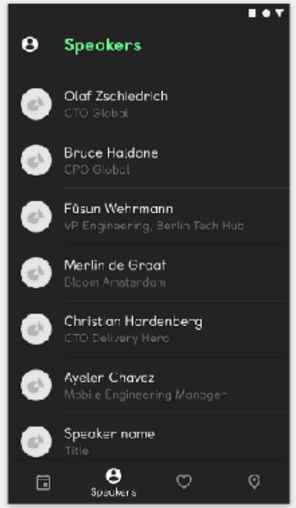
Schedule - Details



Speakers - A

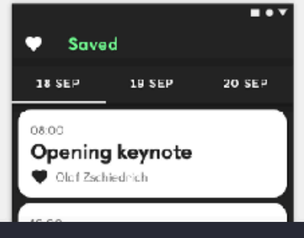


Speakers - B



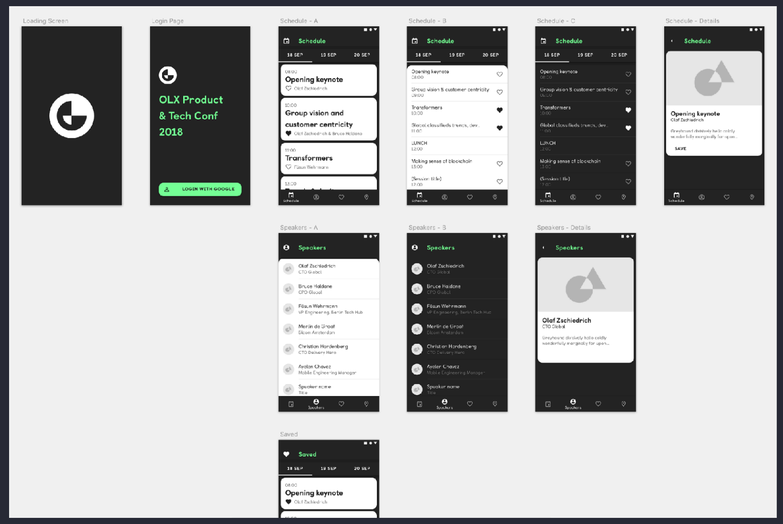
Speakers - Details

Saved



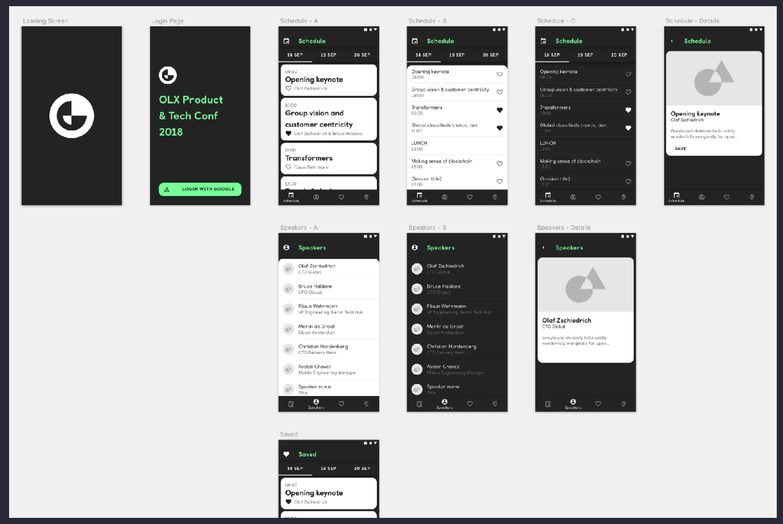
@bennegeek

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
```



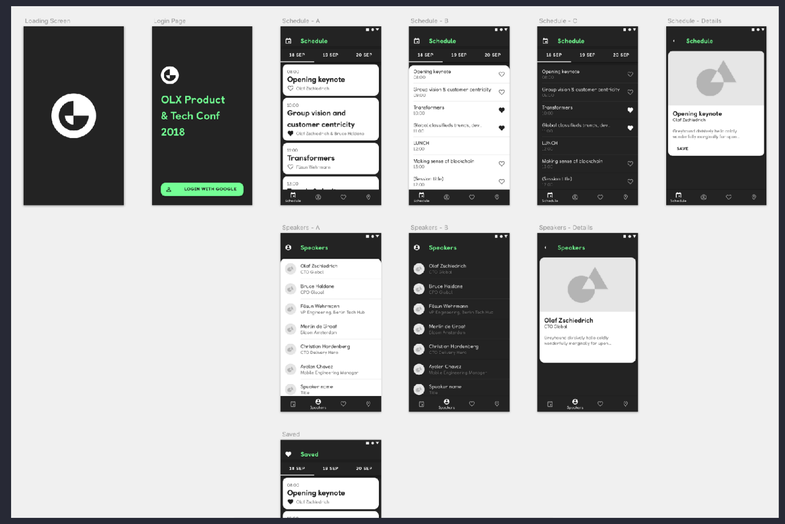
@bennegeek

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
```



@bennegeek

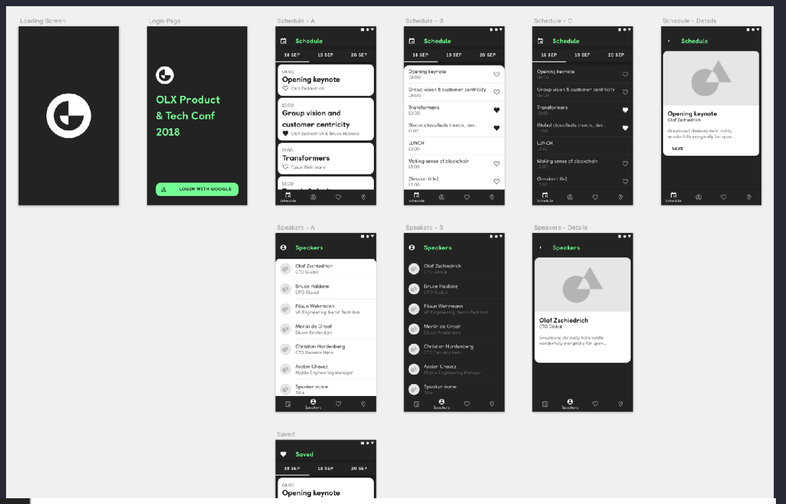
```
ThemeData(
    primarySwatch: Colors.blue,
),
```

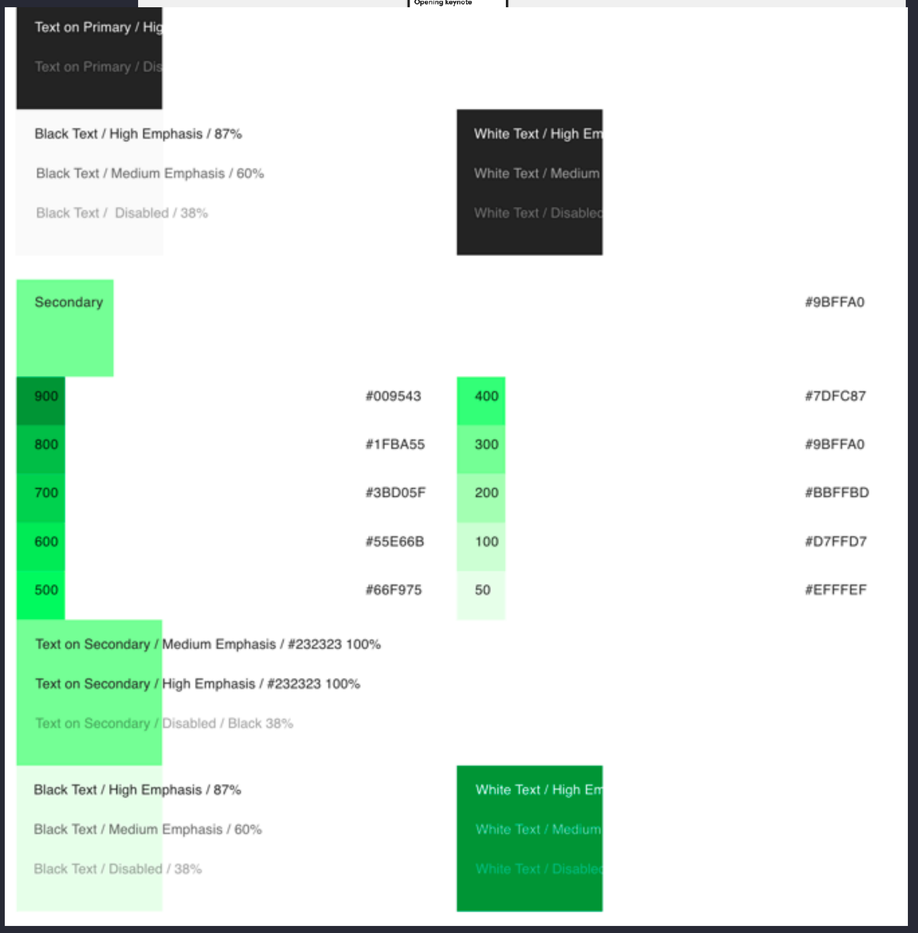


@bennegeek

```
ThemeData(
  fontFamily: "Castledown",
  primarySwatch: MaterialColor(
    0xFF232323,
    const <int, Color>{
      50: const Color(0xFFFAFAFA),
      100: const Color(0xFFF5F5F5),
      ...
      900: const Color(0xFF232323),
    }),
  accentColor: MaterialColor(
    0xFF9BFFA0,
    const <int, Color>{
      50: const Color (0xFFEFEFEF),
      100: const Color(0xFFD7FFD7),
      ...
      900: const Color(0xFF009543),
  }),
  canvasColor: Color(0xFFFF2323),
),
```

@bennegeek





What you gonna do with all that *Kotlin/Native* code?

Kotlin/Native

👍 *Google*

👍 *Know*

👍 *Null Safety* (compile-time)

👍 *Write* multiplatform code

😈 *multiplatform UI*

😈 *Beta*

Flutter

- 👍 *Google*
- 😈 *Know*
- 😈 *Null Safety* (compile-time)
- 👍 *Write* multiplatform code
- 👍 multiplatform *UI*
- 👍 *Beta*

Kotlin/Native

👍? *Google*

👍😈? *Know*

👍😈? *Null Safety* (compile-time)

👍? *Write* multiplatform code

😈👍? multiplatform *UI*

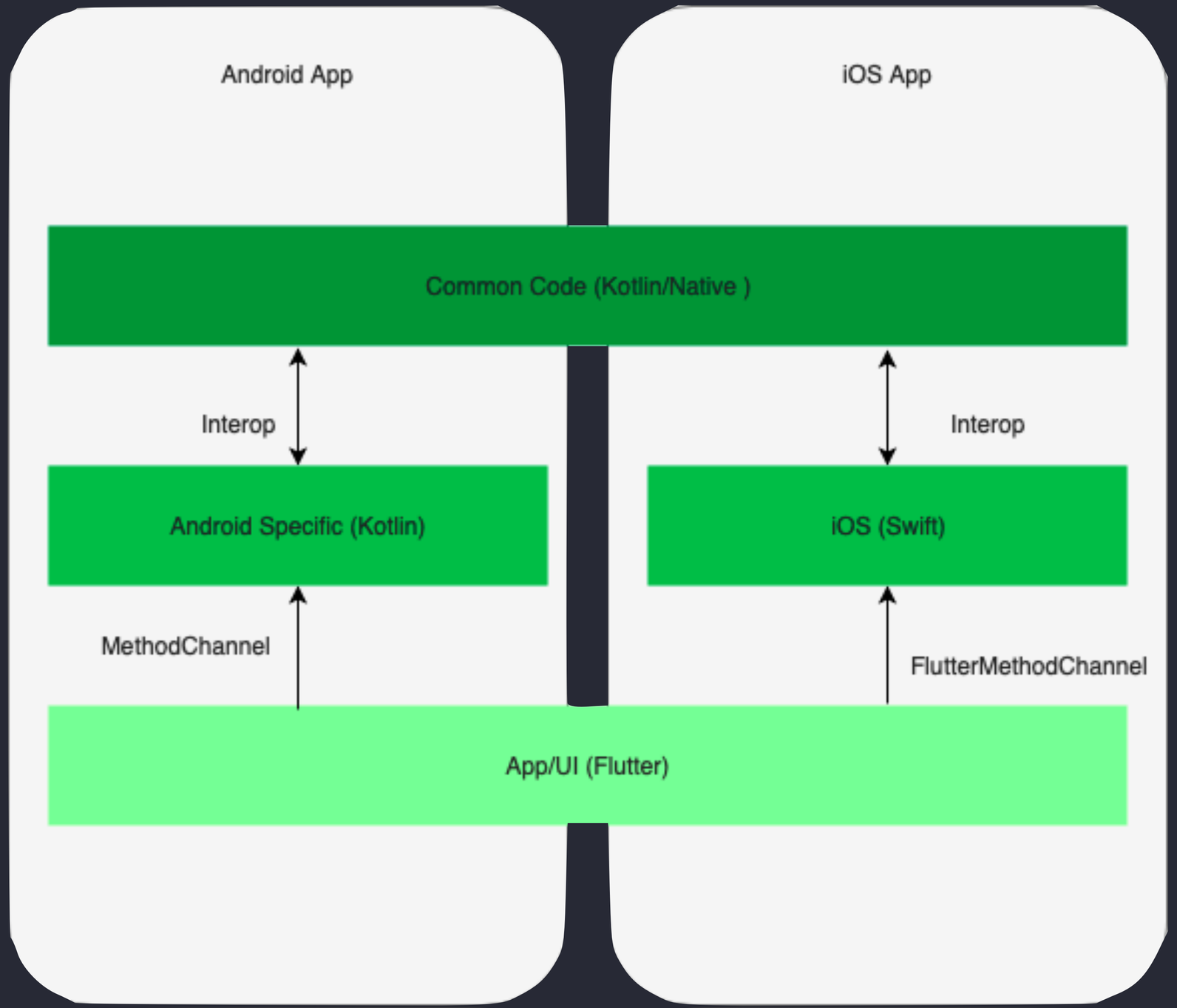
😈👍? *Beta*

Kotlin/Native

Kotlin/Native

Why Flutter + Kotlin/Native

Architecture

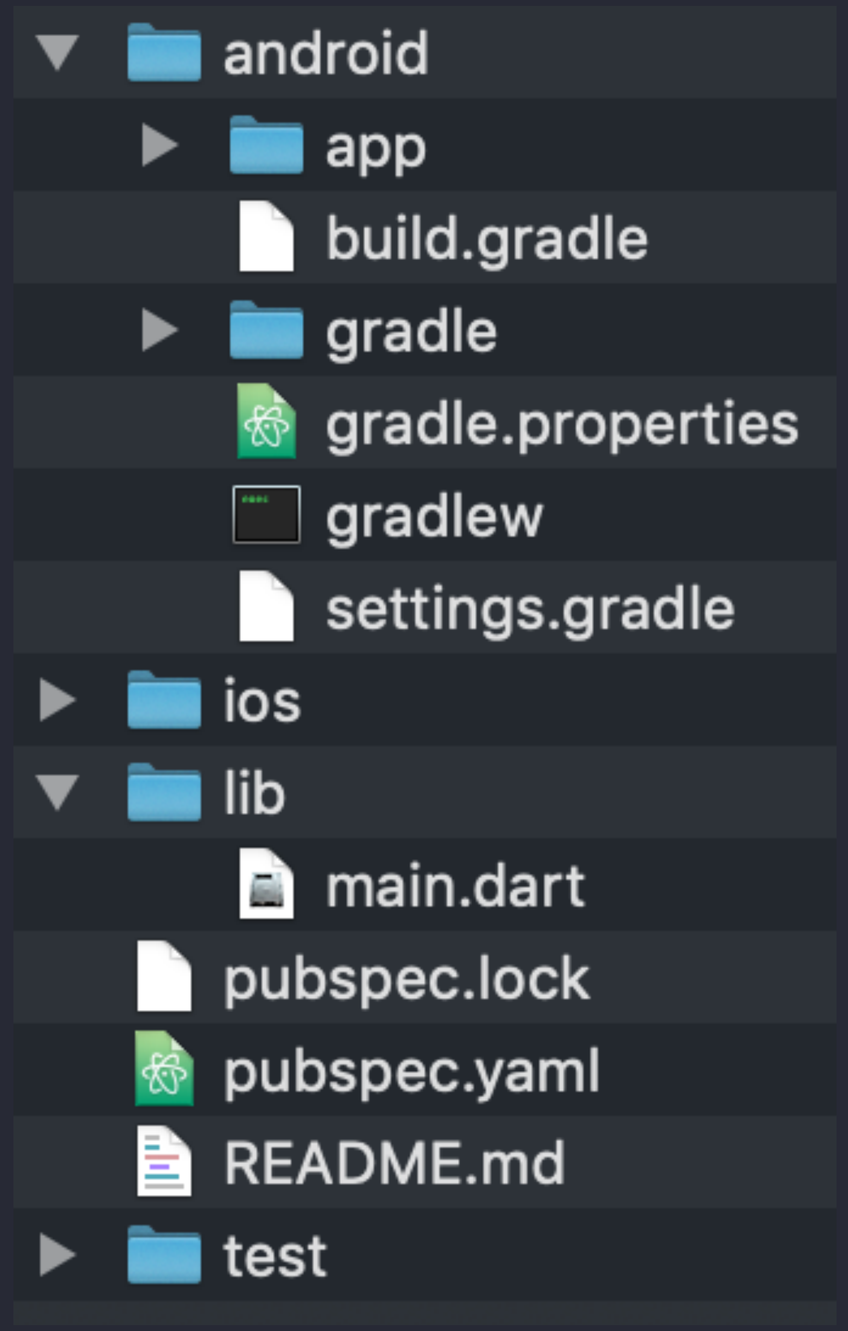


How do I start

What project structure works?

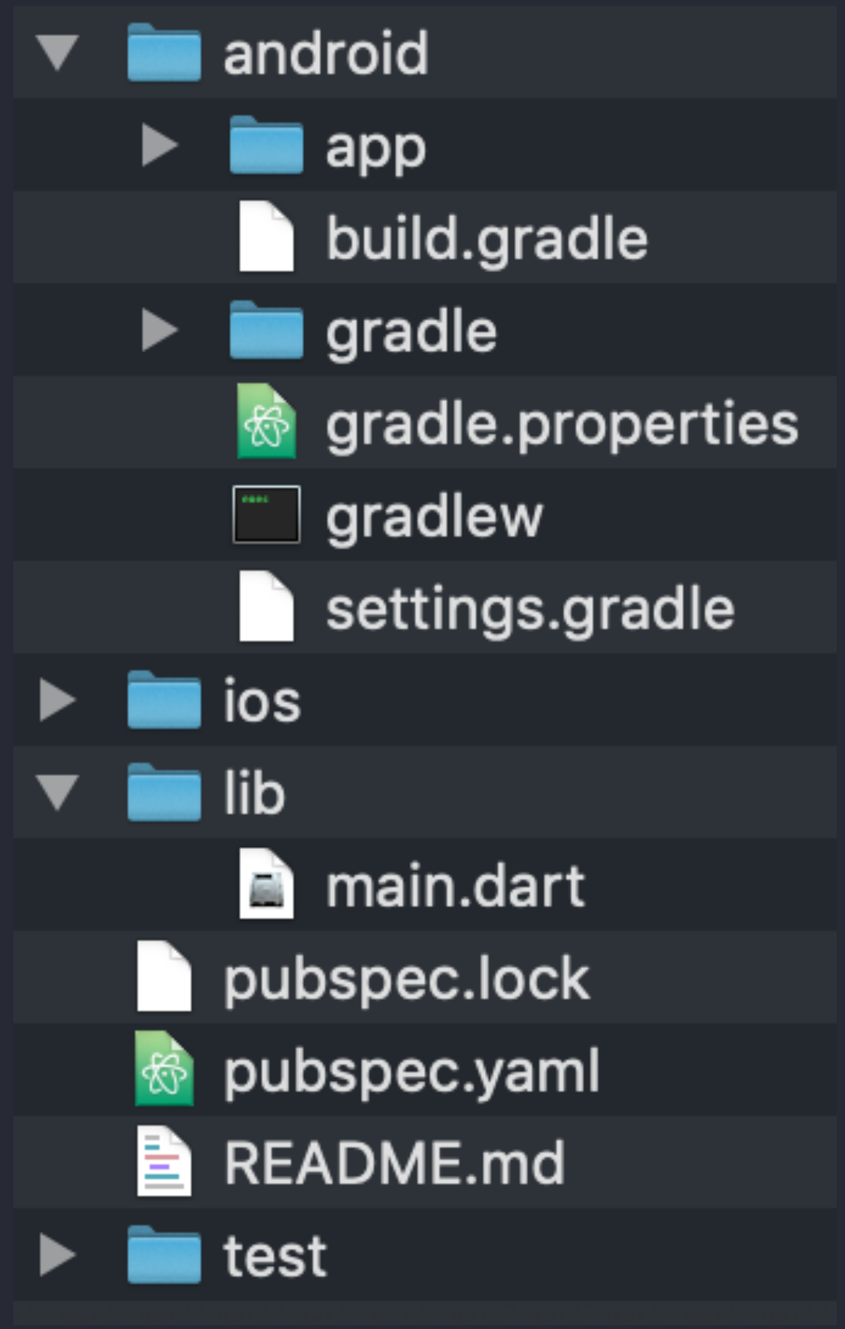
```
> flutter create  -i swift -a kotlin YourFlutterProjectName
```

Flutter project structure



Flutter project structure

How do I + Kotlin/Native



Flutter
project
structure

- android
  - app
  - build.gradle
  - gradle
  - gradle.properties
  - gradlew
  - settings.gradle
- ios
- lib
  - main.dart
- pubspec.lock
- pubspec.yaml
- README.md
- test

Kotlin/Native

- android
- build
- build.gradle
- calculator.iml
- common
- gradle
- gradle.properties
- gradlew
- gradlew.bat
- ios
- jvm
- local.properties
- README.md
- settings.gradle

```
android
 \-- app
  \-- build.gradle
 \-- build.gradle
 \-- settings.gradle
common
 \-- build.gradle
ios
 \-- Runner.xcodeproj
lib
```

```
android
 \-- app
  \-- build.gradle
 \-- build.gradle
 \-- settings.gradle
common
 \-- build.gradle
ios
 \-- Runner.xcodeproj
lib
```

```
// settings.gradle

include ':app'

def flutterProjectRoot = rootProject.projectDir.parentFile.toPath()

def plugins = new Properties()
def pluginsFile = new File(flutterProjectRoot.toFile(), '.flutter-plugins')
if (pluginsFile.exists()) {
    pluginsFile.withReader('UTF-8') { reader -> plugins.load(reader) }
}

include ':common'

plugins.each { name, path ->
    def pluginDirectory = flutterProjectRoot.resolve(path).resolve('android').toFile()
    include ":$name"
    project(":$name").projectDir = pluginDirectory
}

project(":common").projectDir = new File(“../common”)

enableFeaturePreview('GRADLE_METADATA')
```

```
// settings.gradle

include ':app'

def flutterProjectRoot = rootProject.projectDir.parentFile.toPath()

def plugins = new Properties()
def pluginsFile = new File(flutterProjectRoot.toFile(), '.flutter-plugins')
if (pluginsFile.exists()) {
    pluginsFile.withReader('UTF-8') { reader -> plugins.load(reader) }
}

include ':common'

plugins.each { name, path ->
    def pluginDirectory = flutterProjectRoot.resolve(path).resolve('android').toFile()
    include ":$name"
    project(":$name").projectDir = pluginDirectory
}

project(":common").projectDir = new File("../common")

enableFeaturePreview('GRADLE_METADATA')
```

```
// settings.gradle

include ':app'

def flutterProjectRoot = rootProject.projectDir.parentFile.toPath()

def plugins = new Properties()
def pluginsFile = new File(flutterProjectRoot.toFile(), '.flutter-plugins')
if (pluginsFile.exists()) {
    pluginsFile.withReader('UTF-8') { reader -> plugins.load(reader) }
}

include ':common'

plugins.each { name, path ->
    def pluginDirectory = flutterProjectRoot.resolve(path).resolve('android').toFile()
    include ":$name"
    project(":$name").projectDir = pluginDirectory
}

project(":common").projectDir = new File(“../common”)

enableFeaturePreview('GRADLE_METADATA')
```

```
// settings.gradle

include ':app'

def flutterProjectRoot = rootProject.projectDir.parentFile.toPath()

def plugins = new Properties()
def pluginsFile = new File(flutterProjectRoot.toFile(), '.flutter-plugins')
if (pluginsFile.exists()) {
    pluginsFile.withReader('UTF-8') { reader -> plugins.load(reader) }
}

include ':common'

plugins.each { name, path ->
    def pluginDirectory = flutterProjectRoot.resolve(path).resolve('android').toFile()
    include ":$name"
    project(":$name").projectDir = pluginDirectory
}

project(":common").projectDir = new File(“../common”)

enableFeaturePreview('GRADLE_METADATA')
```

```
// settings.gradle

include ':app'

def flutterProjectRoot = rootProject.projectDir.parentFile.toPath()

def plugins = new Properties()
def pluginsFile = new File(flutterProjectRoot.toFile(), '.flutter-plugins')
if (pluginsFile.exists()) {
    pluginsFile.withReader('UTF-8') { reader -> plugins.load(reader) }
}

include ':common'

plugins.each { name, path ->
    def pluginDirectory = flutterProjectRoot.resolve(path).resolve('android').toFile()
    include ":$name"
    project(":$name").projectDir = pluginDirectory
}

project(":common").projectDir = new File("../common")

enableFeaturePreview('GRADLE_METADATA')
```

```
android
 \-- app
  \-- build.gradle
 \-- build.gradle
 \-- settings.gradle
common
 \-- build.gradle
ios
 \-- Runner.xcodeproj
lib
```

```
android
 \-- app
  \-- build.gradle
 \-- build.gradle
 \-- settings.gradle
common
 \-- build.gradle
ios
 \-- Runner.xcodeproj
lib
```

```
// build.gradle

buildscript {
    ext.kotlin_version = '1.3.0'

    repositories {
        google()
        jcenter()
    }

    dependencies {
        classpath 'com.android.tools.build:gradle:3.3.0'
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
    }
}

// ...
```

```
// build.gradle

buildscript {
    ext.kotlin_version = '1.3.0'

    repositories {
        google()
        jcenter()
    }

    dependencies {
        classpath 'com.android.tools.build:gradle:3.3.0'
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
    }
}

// ...
```

```
// build.gradle

buildscript {
    ext.kotlin_version = '1.3.0'

    repositories {
        google()
        jcenter()
    }

    dependencies {
        classpath 'com.android.tools.build:gradle:3.3.0'
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
    }
}

// ...
```

```
android
 \-- app
  \-- build.gradle
 \-- build.gradle
 \-- settings.gradle
common
 \-- build.gradle
ios
 \-- Runner.xcodeproj
lib
```

```
android
 \-- app
  \-- build.gradle
 \-- build.gradle
 \-- settings.gradle
common
 \-- build.gradle
ios
 \-- Runner.xcodeproj
lib
```

```
// app/build.gradle

dependencies {
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation project(“:common”)
}
```

```
android
 \-- app
  \-- build.gradle
 \-- build.gradle
 \-- settings.gradle
common
 \-- build.gradle
ios
 \-- Runner.xcodeproj
lib
```

```
android
 \-- app
  \-- build.gradle
 \-- build.gradle
 \-- settings.gradle
common
 \-- build.gradle
ios
 \-- Runner.xcodeproj
lib
```

```
// common/build.gradle

apply plugin: 'kotlin-multiplatform'

kotlin {
    targets {
        final def iOSTarget = System.getenv('SDK_NAME')?.startsWith("iphoneos") \
                              ? presets.iosArm64 : presets.iosX64

        fromPreset(iOSTarget, 'iOS') {
            compilations.main.outputKinds('FRAMEWORK')
        }

        fromPreset(presets.jvm, 'android')
    }

    sourceSets {
        commonMain.dependencies {
            api 'org.jetbrains.kotlin:kotlin-stdlib-common'
        }

        androidMain.dependencies {
            api ‘org.jetbrains.kotlin:kotlin-stdlib' [1]
        }
    }
}
// …
```


[1] Dissecting Stdlib https://www.youtube.com/watch?v=Fzt_9I733Yg

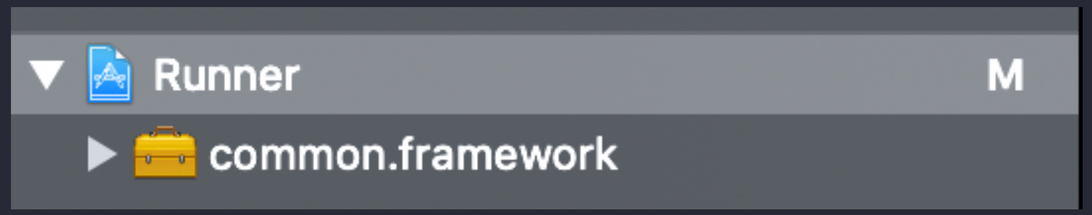
```
// common/build.gradle

// ...

task packForXCode(type: Sync) {
    final File frameworkDir = new File(buildDir, "xcode-frameworks")
    final String mode = project.findProperty("XCODE_CONFIGURATION")?.toUpperCase() ?: 'DEBUG'

    inputs.property "mode", mode
    dependsOn kotlin.targets.iOS.compilations.main.linkTaskName("FRAMEWORK", mode)

    from { kotlin.targets.iOS.compilations.main.getBinary("FRAMEWORK", mode).parentFile }
    into frameworkDir

    doLast {
        new File(frameworkDir, 'gradlew').with {
            text = "#!/bin/bash\nexport 'JAVA_HOME=${System.getProperty("java.home")}'\ncd '$
{rootProject.rootDir}'\n./gradlew \$@\n"
            setExecutable(true)
        }
    }
}

tasks.build.dependsOn packForXCode
```

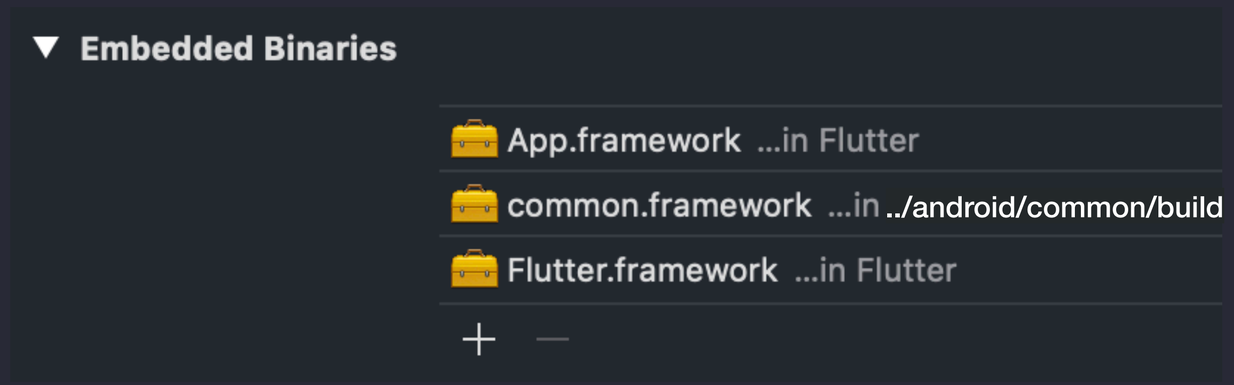
```
android
 \-- app
  \-- build.gradle
 \-- build.gradle
 \-- settings.gradle
common
 \-- build.gradle
ios
 \-- Runner.xcodeproj
lib
```

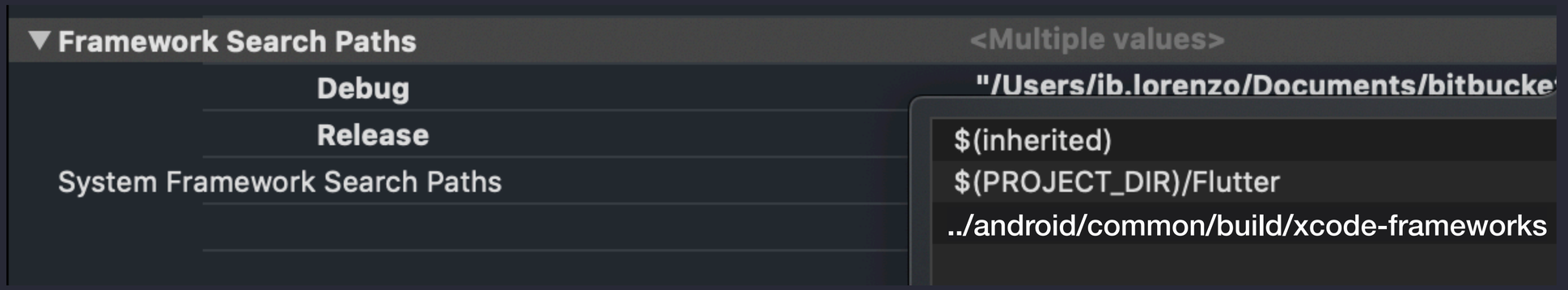
```
android
 \-- app
  \-- build.gradle
 \-- build.gradle
 \-- settings.gradle
common
 \-- build.gradle
ios
 \-- Runner.xcodeproj
lib
```

```
# Top-level directory
> cd android
> ./gradlew :common:build


# Runner.xcodeproj
# Build Phase

cd \"$SRCROOT/../android/build/common/xcode-frameworks\"\n
./gradlew :common:packForXCode -PXCODE_CONFIGURATION=${CONFIGURATION}
```

```
# Runner.xcodeproj
# Build Phase

cd \"$SRCROOT/../android/build/common/xcode-frameworks\"\n
./gradlew :common:packForXCode -PXCODE_CONFIGURATION=${CONFIGURATION}
```







Now we've setup

- android/settings.gradle
- android/build.gradle
- ios/Runner.xcodeproj
- common/build.gradle
- framework in iOS

Kotlin/Native

Now we can start coding in *Flutter* + *Kotlin/Native*

...

How?

For gluing Flutter and Kotlin

Android

```
// MainActivity.kt
import common.doSomethingWith
...

GeneratedPluginRegistrant.registerWith(this)

MethodChannel(flutterView, CHANNEL).setMethodCallHandler { call, result ->
  when (call.method) {
    "methodName" -> doSomethingWith(result)
    "methodName2" -> doSomethingWith2(result)
    "methodName3" -> doSomethingWith3(result)
    ...
  }
}
```

iOS

```
// AppDelegate.swift

import YourFrameworkName // from Kotlin-Native

let channel = FlutterMethodChannel.init(name: "/channel", binaryMessenger: controller)
channel.setMethodCallHandler({(call: FlutterMethodCall, result: @escaping FlutterResult)
-> Void in

    if call.method == "methodName" {
        doSomethingWith(result)
    } else if call.method == "methodName2" {
        doSomethingWith2(result)
    } ...

});

GeneratedPluginRegistrant.register(with: self)
```

```
iOS
// AppDelegate.swift

import YourFrameworkName // from Kotlin-Native

let channel = FlutterMethodChannel.init(name: "/channel", binaryMessenger: controller)
channel.setMethodCallHandler({(call: FlutterMethodCall, result: @escaping FlutterResult) -
> Void in

    if call.method == "methodName" {
        doSomethingWith(result)
    } else if call.method == "methodName2" {
        doSomethingWith2(result)
    } ...

^ Looks familiar . . .
});

GeneratedPluginRegistrant.register(with: self)
```

Common Glue Code

```
// common/src/commonMain/kotlin/common.kt
fun processMethodChannel(method: String,
                         params: Any,
                         success: (Any) -> Unit,
                         error: (String?, String?, String?) -> Unit) {
    if (method == "yourMethodNameHere") {
        var response = aCommonFunc(params)
        success(response)
    } else if ... {
        ...
    } else { // unknown method
        error(some, error, message)
    }
}
```

```kotlin
// android/.../MainActivity.kt
import com.olxgroup.kotlinmultiplatformworkshop.*

GeneratedPluginRegistrant.registerWith(this)

MethodChannel(flutterView, "/api").setMethodCallHandler { methodCall, result ->
    processMethodChannel(methodCall.method ?: "",
            methodCall.arguments,
            { s -> result.success(s) },
            { e, e1, e2 -> result.error(e, e1, e2) }
)}
```

```swift
// ios/Runner/AppDelegate.swift
import common

let CHANNEL_NAME = "/api"
let channel = FlutterMethodChannel.init(name: CHANNEL_NAME, binaryMessenger: controller)

channel.setMethodCallHandler({
    (call: FlutterMethodCall, result: @escaping FlutterResult) -> Void in

    CommonKt.processMethodChannel(method: call.method ,
                                  params: call.arguments ?? [:],
                                  success: { s in result(s); return KotlinUnit() },
                                  error: { _, _, _ in
result(FlutterMethodNotImplemented); return KotlinUnit() })

});
```

from Flutter to Common

```dart
// lib/somefile.dart
const CHANNEL_NAME = '/api'
const channel = MethodChannel(CHANNEL_NAME);
final List<dynamic> result = channel.invokeMethod(‘getSpeakers’, params);

var castedResult = result.cast<Map<dynamic, dynamic>>();
castedResult.sort((a, b) => a['id'].compareTo(b['id']));
```

from Flutter to Common

```dart
// lib/somefile.dart
const CHANNEL_NAME = '/api'
const channel = const MethodChannel(CHANNEL_NAME);
final List<dynamic> result = await channel.invokeMethod('getSpeakers', params);

var castedResult = result.cast<Map<dynamic, dynamic>>();
castedResult.sort((a, b) => a['id'].compareTo(b['id']));
```

from Flutter to Common

```dart
// lib/somefile.dart
const CHANNEL_NAME = '/api'
const channel = const MethodChannel(CHANNEL_NAME);
final List<dynamic> result = await channel.invokeMethod('getSpeakers', params);

var castedResult = result.cast<Map<dynamic, dynamic>>();
castedResult.sort((a, b) => a['id'].compareTo(b['id']));
```

Got a parser?

from Common to Flutter

```
// android/.../MainActivity.kt
var channel = MethodChannel(flutterView, “/api")

channel.invokeMethod("config", null, object: MethodChannel.Result {
  override fun notImplemented() {
    Log.e("MSG", "not implemented")
  }

  override fun error(code: String?, msg: String?, details: Any?) {
    Log.e("MSG", "failed: $msg")
  }

  override fun success(r: Any?) {
    TrackSomethingHere()
  }
}
```

from Common to Flutter

```dart
// lib/somefile.dart
const MethodChannel _channel = const MethodChannel('ninja');
_channel.setMethodCallHandler((MethodCall call) async {
  switch (call.method) {
    case 'methodName':
      return yourDataHere;
    ...
  }
});
```

Pitfalls

Kotlin/Native

Serializing and *Deserializing*

Companion in *Swift*

arm32/arm64

No Bitcode

K/N changes don’t *Hot* reload

JVM libraries

Bigger app (it’s Flutter)

Serializing/Deserializing

https://app.quicktype.io

Name: Talk

Source type: JSON

```
{
    "talks": [{
        "day": "18 SEP",
        "description": "* Share purpose of the con
        "endTime": 1537262100,
        "id": "01",
        "name": "Introduction",
        "room": "Nikolai I",
        "speakers": ["Max Mustermann", "Erika Must
        "startTime": 1537261200,
        "topic": ""
    }]
}
```

```
// To parse this JSON data, do
//
//     final talk = talkFromJson(jsonStri

import 'dart:convert';

Talk talkFromJson(String str) {
    final jsonData = json.decode(str);
    return Talk.fromJson(jsonData);
}

String talkToJson(Talk data) {
    final dyn = data.toJson();
    return json.encode(dyn);
}

class Talk {
    List<TalkElement> talks;

    Talk({
        this.talks,
    });

    factory Talk.fromJson(Map<String, dynamic> json) => new Talk(
        talks: new List<TalkElement>.from(json["talks"].map((x) => TalkEle
    );

    Map<String, dynamic> toJson() => {
        "talks": new List<dynamic>.from(talks.map((x) => x.toJson())),
    };
}

class TalkElement {
    String day;
    String description;
    int endTime;
    String id;
    String name;
    String room;
    List<String> speakers;
    int startTime;
    String topic;
```

Language: Dart | Other

Dart

Classes only

Make all properties optional

Share Link to Code

Copy Code

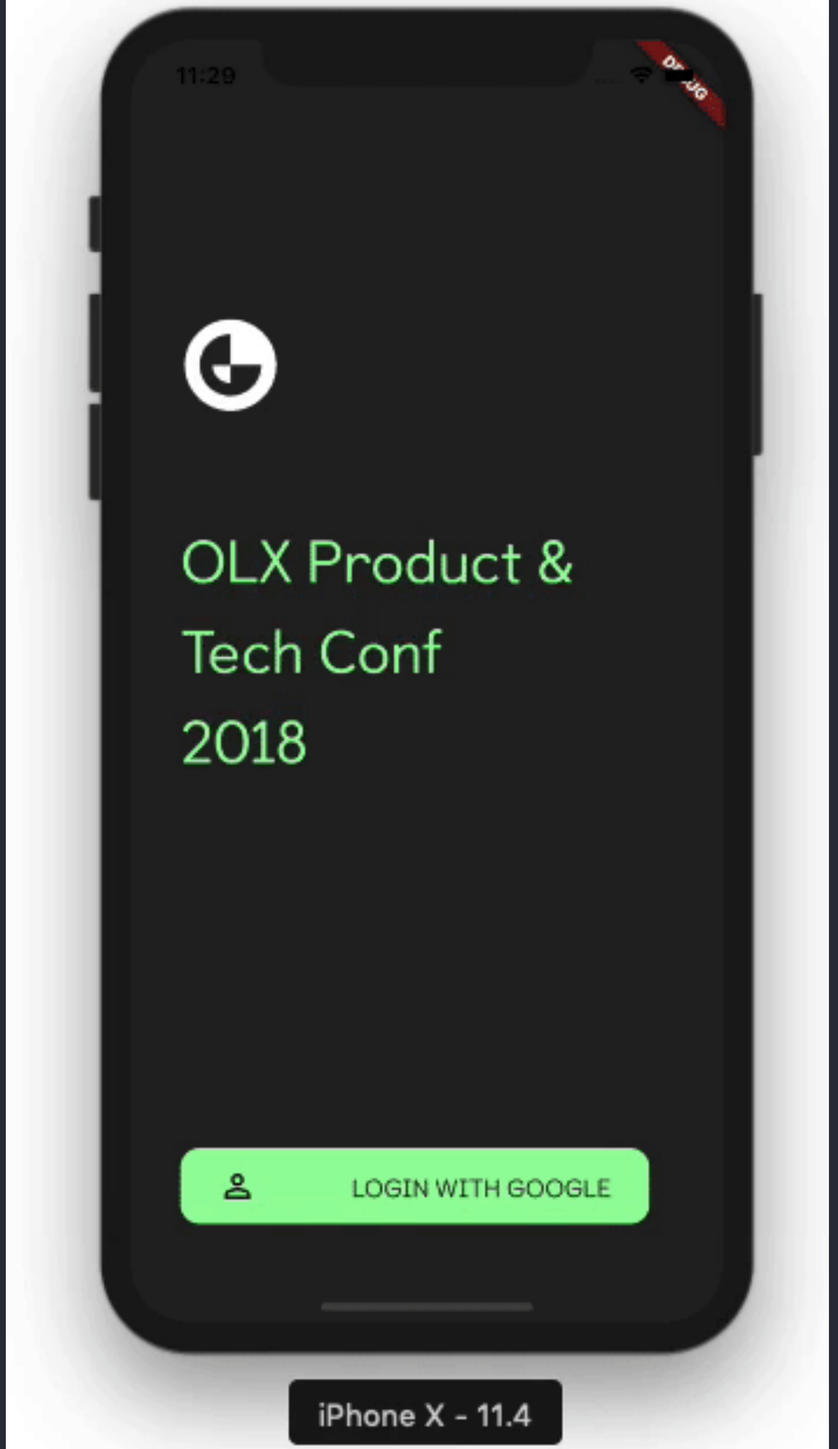
arm32/arm64

https://github.com/AlecStrong/kotlin-native-cocoapods

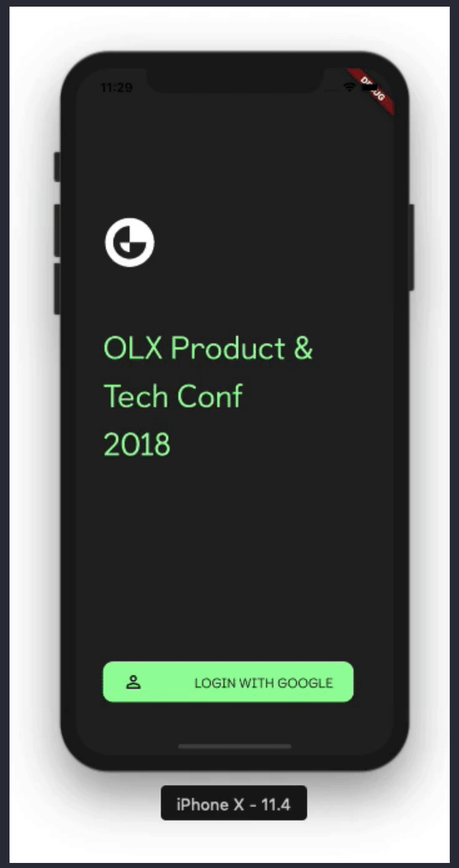
```
apply plugin: 'com.alecstrong.cocoapods'

// Optional configuration of plugin.
cocoapods {
  version = "1.0.0-LOCAL" // Defaults to "1.0.0-LOCAL"
  homepage = www.mywebsite.com  // Default to empty
  deploymentTarget = "10.0" // Defaults to "10.0"
  authors = "Ben Asher" // Defaults to empty
  license = "..." // Defaults to empty
  summary = "..." // Defaults to empty
  daemon = true // Defaults to false
}
```

Flutter Community 👏👏👏

The Result/Demo



Takeaways



Flutter is awesome *for* fast prototypes

Kotlin/Native can reduce platform/glue code

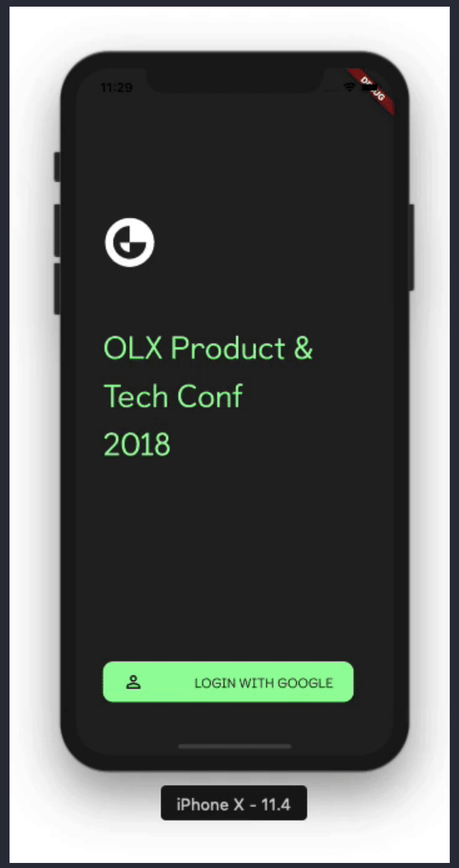
MethodChannels helps reuse existing code

Need glue-code for f(x) + obj.toJson()

Don't be afraid to try out new technologies

(but …)

Next Steps



try pure *Flutter* ***(newbie friendly)***

try pure *Kotlin/Native* or *Multiplatform*

try making *Flutter Plugin*

follow their *Roadmaps*

create an issue or a *PR*

4:00 PM - 4:30 PM

Flutter, a designer and a developer walk into a project

Nancy and Edward Roberts Studio Theatre • Stage 2 • Rainbow

SPEAKERS

Carmen Gonzalez
Design Lead, Media Managers

Adrian Catalan
Lead of Innovation Lab, GDE, Galileo University

Recommended Talks

Example project:

github.com/jblorenzo/flutter-kotlin-native-example.git

tech.olx.com

Thanks! Questions?

Get in touch with us

olxgroup.com
tech.olx.com
olxtechberlin

jb.lorenzo@olx.com
jblorenzo