Empezando con Flutter

Fernando Souto Gonzalez Follow

Dec 27, 2017 · 3 min read

Una tecnología híbrida que pinta los widgets en un canvas nativo

. . .

¿Es Flutter una opción?

Hace unos meses en el trabajo empezamos a mirar una solución para hacer pequeños *POCs para Android e iOS* y casi por casualidad encontramos **Flutter**. Al principio eran todo dudas: aún está en alpha, hay que aprender **Dart** y demás *bullshit*…

La magia de Flutter

La gran ventaja de **Flutter** es que es diferente a la mayoría de soluciones para crear apps multiplataforma y es que **Flutter** no usa *Webviews* o *widgets OEM* sino que usa su propio mecanismo para renderizar la interfaz gráfica a través de un *canvas* nativo gracias a su capa de *C++*. Además **Flutter** implementa la mayoría de los *widgets* y animaciones nativas de cada plataforma pero permite crear *custom views* muy fácilmente. Todo esto fue definitivo para no utilizar otras soluciones como **React Native** que tienen una mucho peor *performance*.

En cuanto a **Dart** es un lenguaje con una curva de aprendizaje muy rápida que para los que venimos de **Java** es como viajar en el tiempo a

un futuro que pensábamos que era imposible de alcanzar y para los que vienen de **Javascript** es prácticamente automática la conversión.

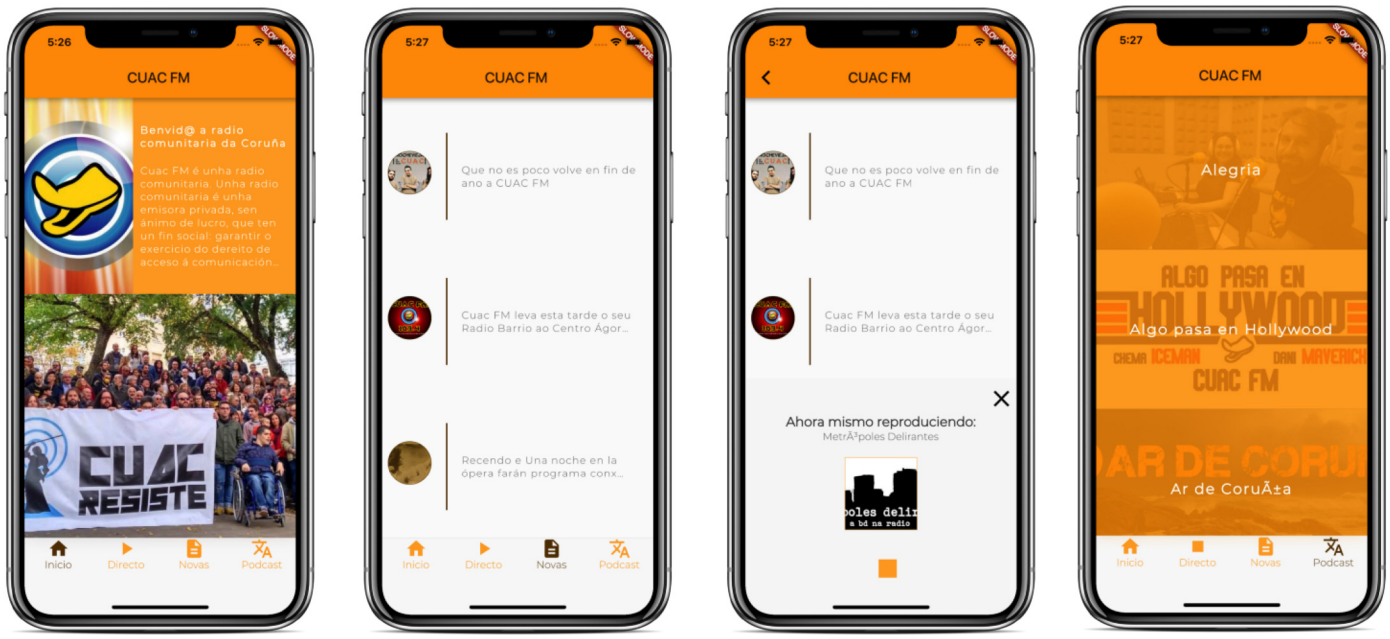
La comunicación a través de *Events o Channels* con el código nativo es posible dado que con Flutter al crear el proyecto también crea un proyecto para **Android Studio** y otro para **Xcode**. Esto resulta muy útil si por ejemplo hay alguna pantalla de la aplicación que requiere el uso de la cámara para escanear un código QR y devolver el resultado de dicho escaneo a **Flutter**.

El pintado de la **UI** es muy sencillo a través de los *widgets* y la actualización de la misma ocurre al estilo *reactivo(*como en **React**) y cuando invocamos el método *setState(){}* hacemos que se modifique la parte la interfaz que ha sufrido un cambio dentro del árbol de widgets.

Uno de los puntos negativos que he encontrado es la gestión de los recursos del proyecto puesto que hay que definirlos en un archivo de configuración para poder usarlos en el proyecto lo cual hace un poco engorroso importar un archivo, pero en general estoy muy contento con este **SDK** y espero que pase de estado *Alpha* cuanto antes para poder convencer a otros *developers* que es una opción para usar en **apps en producción**,dado que el ahorro de tiempo en tiempo de desarrollo es descomunal.

¿Cómo pruebo todo esto?

Actualemente estoy desarrollando una app para la emisora comunitaria de A Coruña **CUAC FM** aplicando *Clean Architecture* que está disponible en mi **github** y aunque aún se encuentra en proceso de desarrollo sirve para hacerse una idea de la potencia de este **SDK multiplataforma**.



Interfaz actual de la app de iOS

Clean Architecture en Flutter

Para implementar *Clean* y hacer código *testable* implementaremos capas con responsabilidades únicas y que cumplen los principios SOLID. A modo de resumen explico los principales componentes de la app:

La interfaz de la vista

Representa una abstracción o contrato de lo que puede realizar la vista, siempre teniendo en cuenta que la única responsabilidad de la vista es implementar esa interfaz e interactuar con el presenter.

```
1  abstract class HomeView {
2    void onLoadRadioStation(RadioStation station);
3    void onLoadNews(List<New> news);
4    void onLoadLiveData(Now now);
5    void onLoadPodcasts(List<Program> podcasts);
6    void onPlayerReady();
```

La vista

Interactúa con el usuario e implementa el contrato delegando al presenter para realizar las operaciones con el modelo.

```
class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  final String title;

  @override
  _MyHomePageState createState() => new _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> implements

  HomePresenter _presenter;
  MediaQueryData queryData;

  _MyHomePageState() {
    _presenter = new HomePresenter(this);
  }

  //UI creation

  getBody(){
    return new Text("My awesome view");
  }

  @override
  Widget build(BuildContext context) {
    queryData = MediaQuery.of(context);
    _margin = RadiocomUtils.getMargin(
        queryData.size.height, queryData.devicePixelRatio);

    return new IPhoneXPadding(child: new Scaffold(
      key: scaffoldKey,
      primary: true,
      resizeToAvoidBottomPadding: true,
      appBar: new AppBar(
        title: new Text(_station.station_name),
      ),
      body: getBody(),
      bottomNavigationBar: new CupertinoTabBar(items: getBu
          currentIndex: _currentIndex,
          onTap: (index) => getData(index)),
    ));
  }
```

```
45
46    @override
47    void initState() {
48      super.initState();
49      _nowProgram = new Now.mock();
50      _presenter.getRadioStationData();
51    }
52
53    //view actions
54
55    @override
56    void onLoadLiveData(Now now) {
57      setState(() {
58        _nowProgram = now;
59      });
60    }
```

El presenter

Es el encargado de coordinar la implementación de la vista y el modelo, actualiza la vista y actúa sobre los eventos de usuario que se envían por la vista. El presenter también recupera los datos del modelo y los prepara para su visualización.

```
class HomePresenter {

  HomePresenter(this._homeView, [CuacRepository repository]
    audioPlayer = new AudioPlayer();
    _repository = _repository != null ? repository : new Cu
  }

  getRadioStationData() {
    _repository.getRadioStationData()
        .catchError((err) {
      //todo use error
    })
        .then((station) {
      _homeView.onLoadRadioStation(station);
    });
  }

  getLiveProgram() {
    _repository.getLiveBroadcast()
        .catchError((err) {
      //todo use error
    })
        .then((now) {
      _homeView.onLoadLiveData(now);
    });
  }

  getAllPodcasts() {
    _repository.getAllPodcasts()
        .catchError((err) {
      //todo use error
    })
```

El repositorio

Es la capa de datos donde nuestra app obtiene los datos. Al utilizar el patrón *repository* y gracias a este, será indiferente si los datos provienen de red, de una base de datos o de la persistencia de los propios dispositivos.

```
class CuacRepository {

  final CuacClient client = new CuacClient();

  CuacRepository({client});

  Future<RadioStation> getRadioStationData() {
    Uri url = Uri.parse(NetworkUtils.baseUrl + NetworkUtils
    return this.client.get(url)
        .then((res) {
      return new RadioStation.fromInstance(res);
    });
  }

  Future<Now> getLiveBroadcast() {
    Uri url = Uri.parse(NetworkUtils.baseUrl + NetworkUtils
    return this.client.get(url)
        .then((res) {
      return new Now.fromInstance(res);
    });
  }

```