Flutter – First App – pt.2: Stateful vs Stateless

Dopo avere realizzato lo scheletro della nostra prima applicazione Flutter, procediamo analizzando alcuni punti chiave necessari per procedere con lo sviluppo.

Ricapitoliamo

Nella **precedente guida** abbiamo impostato lo scheletro della nostra applicazione e successivamente abbiamo fatto un semplice **refactoring** del codice. Analizziamo un attimo quanto scritto per introdurre anche alcuni concetti chiave di **Flutter** e della sintassi di **Dart**. Come per altri linguaggi e framework, tutto parte dal **main**, tramite l'istruzione: **void**

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

La funzione **main** (da notare come anche in Dart, abbiamo il costrutto per le **Lambda functions** / **Arrow functions**, presente anche in altri linguaggi di programmazione), richiamerà la nostra classe **MyApp** che in questo caso manderà, in primis, in esecuzione la funzione: Widget build(BuildContext context) Per dirla in due parole, questa funzione descrive ciò che verrà mostrato nell'interfaccia utente. Per i più curiosi ecco il LINK della doc ufficiale. Passiamo oltre ed analizziamo un concetto che ci accompagnerà durante tutto il percorso: le **StatelessWidget** e **StatefulWidget** class.

StatelessWidget e StatefulWidget

In due parole queste classi si differenziano principalmente **per l'abilità di ricaricare i widget a runtime**. Ma approfondiamo meglio la questione.

Stateless Widget

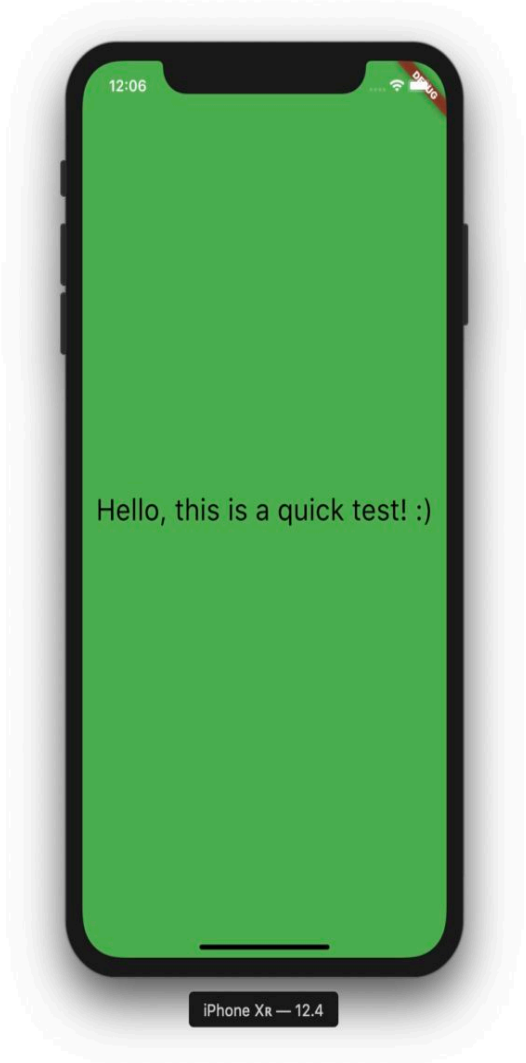
Uno StatelessWidget è un particolare tipo di Widget che **non subirà variazioni** nel corso del tempo. Alcuni esempi sono **Column**, **Row**, **Text**, **Container**. Per creare uno stateless widget basta estendere la classe **StatelessWidget**, che richiede l'**@override** del metodo **build()** che definisce l'interfaccia utente del Widget stesso e permettere l'inserimento del Widget all'interno del **widget tree** in uno specifico **BuildContext** (che sarebbe il context in cui uno specifico widget viene definito). Uno StatelessWidget è immutabile, perciò il solo modo per modificarlo è creare una nuova instanza. Da ciò, ne consegue che il suo ciclo di vita è piuttosto semplice:

- **inizializzazione** del widget;
- invocazione del metodo **build()**;

Passiamo ad un mini esempio concreto. Modifichiamo il file main.dart all'interno di un nuovo progetto ed inseriamo:

```
import 'package:flutter/material.dart'; void main() => runApp(MyApp()); class MyApp extends StatelessWidget { @override Widget build(BuildContext context) { return MaterialApp( title: 'StatelessWidget Test', home: StatelessWidgetTest(), ); }
} class StatelessWidgetTest extends StatelessWidget { @override Widget build(BuildContext context) { return Scaffold( body: Container( color: Colors.green, child: Center( child: Text( 'Hello, this is a quick test! :)', style: TextStyle(color: Colors.black, fontSize: 30), ), ), ), ); }
}
```

Ed ecco il risultato:



Stateful Widget

Contrariamente ai primi, gli stateless widget sono dinamici: il loro contenuto cambia in base alle azioni e agli eventi generati dall'utente. Alcuni esempi di stateful widget sono **Form**, **Image**, **Checkbox**. Sono utilizzati quando l'interfaccia utente cambierà in modo dinamico e vengono creati estendendo la classe **StatefulWidget**. Altro importante concetto è lo **State** che contiene i dati del widget. Esso, gestisce le informazioni per interagire con il widget, definendone il comportamento ed il layout, ed è associato permanentemente ad un **BuildContext** che non cambierà mai. Ogni cambiamento nello **State** forzerà una ricreazione del widget al fine di apportare le modifiche e rendenderle visibile all'utente finale. Lo StatefulWidget non ha il metodo **build()**, ma fa un @override del metodo

createState() che crea uno stato mutabile per il widget in base al **Widget Tree**. Se il widget viene inserito in diverse posizioni del Widget Tree, Flutter creerà istanze separate dell'oggetto State per ognuna di esse. Se, invece, il widget dovesse essere rimosso e reinserito nel Widget Tree, verrà creata una nuova istanza di State. Passiamo anche per lo StatefulWidget ad un mini esempio concreto. Modifichiamo sempre il file main.dart all'interno di un nuovo progetto ed inseriamo:

```
import 'package:flutter/material.dart'; void main() => runApp(MyApp()); class MyApp extends StatelessWidget { @override Widget build(BuildContext context) { print('test'); return MaterialApp( title: 'StatefulWidget Test', home: StatefulWidgetTest(), ); }
} class StatefulWidgetTest extends StatefulWidget { @override _StatefulWidgetTestState createState() => _StatefulWidgetTestState();
} class _StatefulWidgetTestState extends State<StatefulWidgetTest> { int _count = 0; @override Widget build(BuildContext context) { return Scaffold( backgroundColor: Colors.blue[100], body: Center( child: Text( 'Counter pressed: $_count times', style: TextStyle(fontSize: 25), ), ), floatingActionButton: FloatingActionButton( onPressed: () => setState( () { _count++; }, ), child: Icon(Icons.add), ), ); }
}
```

Ed ecco il risultato:



Come scegliere quale tipologia di Widget utilizzare?

Semplice: qualora sia necessario utilizzare un elemento che può cambiare nel tempo allora il widget da realizzare sarà di tipo stateful, contrariamente sarà stateless. Per ora ci fermiamo qui, nella prossima guida andremo finalmente a completare lo scheletro per costruire la nostra **UI** (che anticipo svilupperemo utilizzando **Figma**, di cui tratteremo in una guida a parte). https://italiancoders.it/flutter-first-app-pt-2-stateful-vs-stateless/