Flutter

Di: Edoardo Lo Iacono

Ultima modifica: 19/04/2026

Appunti sul corso di Edoardo Midali + integrazione della documentazione

Contents

1 Introduzione a Flutter

Flutter è un **framework** di Google che ci permette di andare a sviluppare applicazioni **multi-piattaforma**. Questo significa che mediante l'utilizzo di Flutter non dovremo andare a scrivere codici diversi per tipologie di applicazioni diverse (ios, android, web) ma sarà il framework stesso a prendere il nostro codice e ad andare a convertirlo nei diversi formati.

Flutter si basa sul linguaggio di programmazione ad oggetti **Dart**, anch'esso di proprietà di Google, è quindi necessario avere una base in Dart per poter comprendere quanto segue.

Una delle caratteristiche più importanti di flutter è l'**hot reload** che ci permette di andare a ricaricare solamente la parte di codice interessata andando a rendere la modifica sostanzialmente istantanea.

1.1 Installazione di Flutter

Per una visione più completa e costantemente aggiornata sull'installazione di Flutter si faccia riferimento alla pagina ufficiale al seguente link: pagina di Dowload, seguono le installazioni per Windows e per Mac:

Windows Per installare Flutter su dispositivi windows sono necessari alcuni requisiti, specificati meglio al link citato in precedenza; è in particolare necessario:

- Avere Windows 10 o una versione successiva
- E' necessario installare Git for Windows alla seguente pagina: link

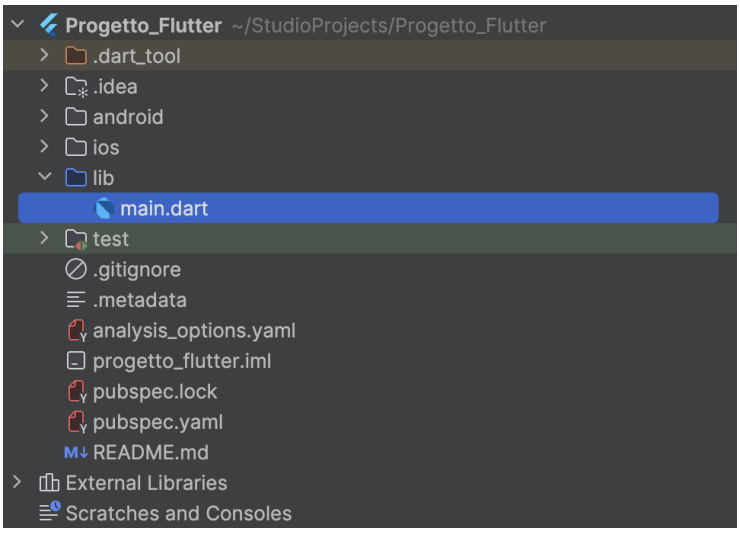
Di seguito gli effettivi passaggi:

1. Installare lo zip contenente l'SDK di flutter (NB: SDK è l'acronimo di Software Development Kit, una collezione di librerie necessari per lo sviluppo)

Mac

1.2 Struttura di un progetto

Una volta avviato il nostro progetto flutter ci troveremo davanti ad una struttra dei file simile alla seguente:



Andiamo di seguito a vedere quanto contenuto in ognuno di questi file e cartelle:

- **.dart_tool** = Contiene file di configurazione e pacchetti dipendenti di Dart
- **.idea** = Cartella generata direttamente da Andorid Studio
- **android** = Cartella del progetto lato Android
- **ios** = Cartella del progetto lato Ios
- **test** = Cartella usata per strutturare eventuali test al progetto
- **.gitignore** = File in cui vengono specificati i file e le cartelli che non vanno sincronizzati su git
- **analysis_options** = File in cui possiamo inserire il linting di Dart
- **pubspec.yaml** = File in cui possiamo andare a definire le caratteristiche di un progetto
- **pubspec.lock** = Insieme di tutte le dipendenze usate nel progetto
- **main.dart** = E' il file su cui andiamo a scrivere il codice effettivo del nostro progetto, vediamolo meglio di seguito.

1.3 Struttura del film main.dart

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

Questo import ci permette di lavorare in primo luogo con dart e anche con *material*, il framework grafico di Google.

```
void main(){
  runApp(const MyApp());
}
```

Questo è l'entry point, ovvero le prime istruzioni che vengono eseguite appena viene fatto partire il programma, in questo caso vediamo una funzione che lancia l'applicazione alla cui passiamo MyApp(), un widget che contiene tutta la nostra applicazione. Parleremo meglio in seguito di cosa siano i Widget

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        // This is the theme of your application.
        //
        // TRY THIS: Try running your application with "flutter run". You
            'll see
        // the application has a purple toolbar. Then, without quitting
            the app,
        // try changing the seedColor in the colorScheme below to Colors.
            green
        // and then invoke "hot reload" (save your changes or press the "
            hot
        // reload" button in a Flutter-supported IDE, or press "r" if you
             used
        // the command line to start the app).
        //
        // Notice that the counter didn't reset back to zero; the
            application
        // state is not lost during the reload. To reset the state, use
            hot
        // restart instead.
        //
        // This works for code too, not just values: Most code changes
            can be
        // tested with just a hot reload.
        colorScheme: .fromSeed(seedColor: Colors.deepPurple),
      ),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
```

Nella sezione di codice copiata qui sopra troviamo il nostro widget di cui si parlava precedentemente, troviamo al suo interno un costruttore, un metodo

build che va a costruire un widget e di cui facciamo l'override. All'interno della build andiamo a creare e ritornare la nostra material app dandole un titolo, un tema (in cui al momento abbiamo solamente un colore primario) ed una home, ovvero una schermata di ingresso che in questo caso rimanda ad un altro widget.

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

  // This widget is the home page of your application. It is stateful,
      meaning
  // that it has a State object (defined below) that contains fields that
       affect
  // how it looks.

  // This class is the configuration for the state. It holds the values (
      in this
  // case the title) provided by the parent (in this case the App widget)
       and
  // used by the build method of the State. Fields in a Widget subclass
      are
  // always marked "final".

  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}
```

Il widget della home page, a differenza di quello dell'applicazione in generale è di tipo *statefull*, questo significa che mantiene uno stato. Parleremo meglio più avanti di widget stateless e statefull, per ora è sufficiente sapere che la loro composizione è diversa, troviamo infatti un widget diviso in myHome e myHomeState a cui facciamo accesso tramite l'ultima riga di codice.

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

  void _incrementCounter() {
    setState(() {
      // This call to setState tells the Flutter framework that something
           has
      // changed in this State, which causes it to rerun the build method
           below
      // so that the display can reflect the updated values. If we
          changed
      // _counter without calling setState(), then the build method would
           not be
      // called again, and so nothing would appear to happen.
      _counter++;
    });
```

```
}

@override
Widget build(BuildContext context) {
  // This method is rerun every time setState is called, for instance
      as done
  // by the _incrementCounter method above.
  //
  // The Flutter framework has been optimized to make rerunning build
      methods
  // fast, so that you can just rebuild anything that needs updating
      rather
  // than having to individually change instances of widgets.
  return Scaffold(
    appBar: AppBar(
      // TRY THIS: Try changing the color here to a specific color (to
      // Colors.amber, perhaps?) and trigger a hot reload to see the
          AppBar
      // change color while the other colors stay the same.
      backgroundColor: Theme.of(context).colorScheme.inversePrimary,
      // Here we take the value from the MyHomePage object that was
          created by
      // the App.build method, and use it to set our appbar title.
      title: Text(widget.title),
    ),
    body: Center(
      // Center is a layout widget. It takes a single child and
          positions it
      // in the middle of the parent.
      child: Column(
        // Column is also a layout widget. It takes a list of children
            and
        // arranges them vertically. By default, it sizes itself to fit
            its
        // children horizontally, and tries to be as tall as its parent.
        //
        // Column has various properties to control how it sizes itself
            and
        // how it positions its children. Here we use mainAxisAlignment
            to
        // center the children vertically; the main axis here is the
            vertical
        // axis because Columns are vertical (the cross axis would be
        // horizontal).
        //
        // TRY THIS: Invoke "debug painting" (choose the "Toggle Debug
            Paint"
        // action in the IDE, or press "p" in the console), to see the
        // wireframe for each widget.
        mainAxisAlignment: .center,
```

```
        children: [
          const Text('You have pushed the button this many times:'),
          Text(
            '$_counter',
            style: Theme.of(context).textTheme.headlineMedium,
          ),
        ],
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: _incrementCounter,
      tooltip: 'Increment',
      child: const Icon(Icons.add),
    ),
  );
 }
}
```

Quello che ora ci interessa di questo widget è lo **scaffold**, ovvero il widget che ci permette di dare una veste grafica alla nostra applicazione, vedremo di seguito diverse caratteristiche dello scaffold.

1.4 I widget

Abbiamo fino ad ora parlato di **widget**, è importante capire bene il concetto alla base di questi elementi poiché costituiscono il paradigma di programmazione in flutter.

I Widget sono dei blocchi di codice, più o meno tangibili, alcune volte sono visibili graficamente come un testo o un bottone, altre volte sono più astratti, come possono essere dei contenitori. I widget sono annidati tra di loro come vedremo meglio insieme e vanno quindi a creare una struttura del codice a matrioska.

I widget possono essere divisi in diversi categorie in base a quello che fanno, è possibile approfondire meglio queste categorie al seguente link: widget.

Capiamo meglio cosa sono i widget andando a prendere in considerazione i più importanti!

2 I Widget principali

2.1 AppBar

L'AppBar rappresenta l'header della schermata. Ecco le proprietà principali e i tipi di dati richiesti:

- **leading** → *Widget* (solitamente `Icon` o `IconButton`).

  - `leadingWidth`: **double** (larghezza dell'area leading).
  - `automaticallyImplyLeading`: **bool** (mostra/nasconde il tasto back automatico).

- **title** → *Widget* (solitamente un widget `Text`).
  - `centerTitle`: **bool** (centra il titolo nella barra).
  - `titleTextStyle`: **TextStyle** (definisce lo stile del testo del titolo).
  - `titleSpacing`: **double** (spazio orizzontale attorno al titolo).
- **actions** → *List<Widget>* (accetta una lista di widget, tipicamente `IconButton`).
- **foregroundColor** → *Color* (colore per icone e testo).
- **backgroundColor** → *Color* (colore di sfondo della barra).
- **elevation** → **double** (intensità dell'ombra inferiore).
- **toolbarHeight** → **double** (altezza complessiva della barra).
- **toolbarOpacity** → **double** (valore da 0.0 a 1.0).

2.2 Testo e TextStyle

Per formattare il testo, utilizziamo la classe **TextStyle**. Ecco i parametri più comuni:

- **color** → *Color* (es. `Colors.blue`).
- **backgroundColor** → *Color* (colore dello sfondo del testo/evidenziatore).
- **fontSize** → **double** (dimensione del carattere).
- **fontWeight** → *FontWeight* (es. `FontWeight.bold` o `FontWeight.w700`).
- **fontStyle** → *FontStyle* (es. `FontStyle.italic`).
- **letterSpacing** → **double** (spazio extra tra le lettere).
- **wordSpacing** → **double** (spazio extra tra le parole).
- **shadows** → *List<Shadow>* (una lista di oggetti `Shadow`).
  - *Shadow* richiede: `offset` (**Offset**), `blurRadius` (**double**) e `color` (**Color**).
- **decoration** → *TextDecoration* (es. `TextDecoration.underline`).

Scaricare un font da Google fonts

1. Andare su google fonts
2. Scaricare il font di nostro interesse tramite il bottone "download all" o "download family"
3. Unzippare il file scaricato
4. Creare una nuova cartella "fonts" nel nostro progetto e trascinare al suo interno il file
5. In *pubspec.yaml* scorrere in basso fino all'elenco dei font e togliere i commenti dai font
6. Inserire in *family:* il nome della nostra famiglia di font e in *asset:* il percorso

2.3 Bottoni ed icone

Per quanto questi due widget siano di natura separata e possano essere usati in maniera indipendente l'uno dall'altro soon spesso considerati in maniera atomica.

Il material design mette a nostra disposizione 3 tipi di bottoni:

- **textButton**
- **elevatedButton**
- **outlinedButton**

Tutti i bottoni richiedono di base due proprietà: *onPressed*, che richiede una funzione oppure *null* nel caso in cui si voglia disabilitare il bottone. La funzione viene ovviamente eseguita in corrispondenza del click sul bottone; e *child* in cui si può inserire quanto vediamo all'interno del bottone, nei bottoni presenti nella foto precedente al child viene passato un widgwt di tipo Text con al suo interno il testo relativo.

Possiamo andare ad utilizzare delle icone in maniera atomica ai bottoni, per farlo dovremo andare a modificare il nostro codice che da così

```
ElevatedButton(onPressed: (){}, child: Text('ElevatedButton'))
```

diventerà:

```
ElevatedButton.icon(
  onPressed: (){},
  icon: Icon(Icons.delete),
  label: Text('ElevatedButton')
),
```

il discorso sullo stile dei bottoni è un po' complicato e richiede la comprensione degli stati che qui non abbiamo ancora trattato, per il momento basti sapere che per cambiare lo stile ad un bottone è sufficiente inserire all'interno di esso il codice seguente.

```
style: ElevatedButton.styleFrom(...)
```

2.4 Le immagini

Per inserire immagini all'interno della nostra applicazione dobbiamo in primo luogo andare a modificare il nostro file *pubspec.yaml*, come abbiamo fatto in precedenza per aggiungere il font da google font. In questo file inseriamo solo le immagini che dobbiamo avere nell'applicazione caricati direttamente dal programmatore. Procediamo quindi nel seguente modo:

1. Creare una cartella images all'interno del progetto ed inserire al suo interno le nostre immagini
2. Specificare all'interno del file pubsec.yaml, sotto ad asset: il path dell'immagine o definire tutta la cartella
3. All'interno di *main.dart* possiamo inserire il widget Image.asset() e passare al suo interno il path della nostra immagine
    - Per prendere invece un'immagine dal web possiamo usare il widget Image.network e passare l'url al suo interno
    - Possiamo anche inserire delle immagini presenti nella galleria del telefono tramite Image.file

Vediamo di seguito alcune proprietà comuni alle immagini, le quali vanno passate all'interno del widget, indipendetemente dalla fonte da cui viene presa l'immagine stessa.

- **width** → **double** (Larghezza dell'immagine).
- **height** → **double** (Altezza dell'immagine).
- **color** → **Color** (Colore dell'immagine).
- **colorBlendMode** → **BlendMode** (La blend mode ci permette di unire il colore di overlay inserito con color a quello dell'immagine).
- **fit** → **BoxFit** (Cambia il modo in cui l'immagine si adatta nel container).

2.5 Row e Column

Passiamo ora a vedere qualcosa di un po' più astratto, ovvero dei widget di layout, questi widget ci permettono di formattare i nostri elementi nella pagina in un determinato modo. Column, come Row va inserito come child di un Container, a differenza del padre che richiede un solo child, Column richiede un array di child, dentro cui possiamo inserire vari widget. Per comprendere bene i widget di column e di row dobbiamo prima chiarire il concetto di *mainAxis* e di *crossAxis*, nel caso della colonna, l'asse principale sarà quello verticale e quello che trapassa sarà l'orizzontale, viceversa per la row, capito questo concetto basilare possiamo vedere alcuni widget:

- **mainAxisSize** → **MainAxisSize** (Altezza della colonna o della riga nel caso di Row).
- **mainAxisAlignment** → **MainAxisAlignment** (Allinea gli elementi in diversi modi).

2.6 Container e Padding

Container Container è il contenitore più basilare, possiamo dargli proprietà come:

- **color** → **Color** (Colore del container).
- **width** → **double** (Larghezza del container).
- **height** → **double** (Altezza del container).
- **padding** → **EdgeInserts** (Spazio interno tra la fine dell'elemento ed il suo contenuto).
- **margin** → **EdgeInserts** (Spazio esterno tra la fine dell'elemento ed altri elementi).

- **alignment** → **Alignment** (Allinea il contenuto interno ad un contenitore).
- **transform** → **Matrix4** (Permette di modellare in diversi modi il nostro container).

Padding Il widget padding accetta solamente un widget padding che crea dello spazio bianco e altri widget figli, questo widget viene usato solo in casi marginali e di test.

2.7 Stack e scroll

Il widget stack ci permette di andare a posizionare i nostri elementi in maniera assoluta, è il corrispettivo del position.absoulute nel css. Sono quindi elementi fissi anche quando viene effettuato uno scroll.

E' importante all'interno di stack il widget **Positioned** che si aspetta un child ed una posizione specifica (in cui i valori double positivi rappresentano la distanza da quella direzione verso l'altra).

Per abilitare lo scroll nella pagina è necessario wrappare la nostra pagina con il widget **SingleChildScrollView**, al suo interno possiamo impostare alcuni widget come la scrollDirection o la reverse. La barra dello scroll è un widget separato, per metterlo è necessario wrappare tutto dentro il widget **Scrollbar**, per far sì che sia sempre visibile è necessario mettere *thumbVisibility:true*

2.8 ListView

ListView è una View che permette di gestire in maniera automatica una lista, ci permette di andare a eliminare codice ripetitivo scrivendolo una volta sola e generandolo per quanti elementi sono presenti in una lista di partenza, con i quali sarà ovviamente possibile interagine, vediamo un esempio per capirlo meglio, data la lista:

```
List<int> lista = [1,2,3,4,5,6,7];
```

possiamo scrivere il seguente codice:

```
   body:ListView(
       padding: EdgeInsert.all(8),
       children: [
           for(var i in lista)
           Container(
               alignment: Alignment.center,
               color: Colors.grey,
               width: double.infinity,
               height: 100,
               margin: EdgeInsets.only(bottom:8),
               child: Text('$i')
```

```
        )
      ]
  )
```

2.9 ListTile

Il widget **ListTile** è uno dei componenti più utilizzati all'interno delle ListView. È progettato per rispettare gli standard del Material Design per le righe di una lista, offrendo una struttura predefinita per icone e testo.

- **leading** → *Widget* (Posiziona un widget, solitamente un'icona o un `CircleAvatar`, all'inizio della riga).
- **title** → *Widget* (Il contenuto principale, solitamente un widget `Text` in grassetto).
- **subtitle** → *Widget* (Contenuto secondario posto sotto il titolo, con font più piccolo).
- **trailing** → *Widget* (Widget posizionato alla fine della riga, come icone di frecce o checkbox).
- **onTap** → *Function* (Callback richiamata quando l'utente preme sulla riga).
- **tileColor** → *Color* (Definisce il colore di sfondo della singola riga).
- **selected** → *bool* (Se impostato su `true`, cambia il colore del testo e delle icone per indicare la selezione).

2.10 GridView

Mentre la ListView dispone gli elementi in una sola colonna (o riga), la **GridView** permette di creare layout a griglia (bidimensionali). Il modo più comune per utilizzarla è tramite il costruttore `count`.

Vediamo i parametri principali per gestire la griglia:

- **crossAxisCount** → **int** (Proprietà obbligatoria; definisce il numero di colonne se lo scroll è verticale).
- **mainAxisSpacing** → **double** (Spazio vuoto tra gli elementi lungo l'asse principale).
- **crossAxisSpacing** → **double** (Spazio vuoto tra gli elementi lungo l'asse trasversale).
- **childAspectRatio** → **double** (Rapporto tra larghezza e altezza di ogni elemento; `1.0` crea elementi quadrati).

- **shrinkWrap** → **bool** (Se impostato su `true`, la griglia occuperà solo lo spazio necessario invece di espandersi al massimo).

Esempio di una griglia a due colonne:

```
GridView.count(
  crossAxisCount: 2,
  crossAxisSpacing: 10,
  mainAxisSpacing: 10,
  children: [
    Container(color: Colors.red, height: 100),
    Container(color: Colors.blue, height: 100),
    Container(color: Colors.green, height: 100),
    Container(color: Colors.yellow, height: 100),
  ],
)
```

2.11 Card

Il widget card è già preimpostato nel materila design, rappresenta una particolare categoria di contenitore cun un effetto di ombra sottostante che lo fa sembrare rialzato, vediamo di seguito un esempio di Card:

```
   Expanded(
          child: ListView(
            children: [
              for(String i in post)
                SizedBox(
                  height: 370,
                  child: Card(

                    shape: RoundedRectangleBorder(
                        borderRadius: BorderRadius.circular(20)
                    ),
                    margin: EdgeInsets.all(20),
                    clipBehavior: Clip.hardEdge,
                    child: Column(
                      children: [
                        Image.network("https://images.unsplash.com/photo
                            -1740507619572-ac180ca2630f?fm=jpg&q=60&w
                            =3000&ixlib=rb-4.1.0&ixid=
                            M3wxMjA3fDB8MHxwaG90by1yZWxhdGVkfDE1fHx8ZW58MHx8fHx8
                            "),
                        ListTile(
                          leading: CircleAvatar(
                            backgroundImage: NetworkImage("https://
                                avataaars.io/?avatarStyle=Circle&topType=
                                LongHairStraight&accessoriesType=Blank&
                                hairColor=BrownDark&facialHairType=Blank&
                                clotheType=BlazerShirt&eyeType=Default&
```

```
                            eyebrowType=Default&mouthType=Default&
                            skinColor=Light"),
                        ),
                        title: Text('$i'),
                        subtitle: Text('Ciao, sono nuov*, mi chiamo $i
                            !!'),
                        trailing: Icon(Icons.favorite),
                      )
                    ],
                  ),
                ),
              ),
            ]
          ),
        )
      ],
    ),
```

3 Gli stati