**Corso**

**Flutter**

**Dedalo’s notes – Una serie di appunti da non prendere sul serio**

Serie di appunti sugli argomenti più svariati, fatti per imparare, apprendere e divertirsi, navigando per il magico mondo del…tutto!

Ogni errore o imprecisione sarà sistemata, oppure no 😊.

Attenzione, può contenere boiate, amenità o stramberie!

https://github.com/GiacomoBorsellino

Scritto da

**Giacomo Borsellino**

27-12-2024

Flutter è un **framework** open-source sviluppato da Google per la creazione di applicazioni **multipiattaforma** con un'unica base di codice. Permette di sviluppare app per Android, iOS, Web, Windows, macOS e Linux utilizzando il linguaggio Dart.

Caratteristiche principali:

* **Single Codebase**: Un unico codice per più piattaforme.
* **Hot Reload**: Aggiornamenti rapidi durante lo sviluppo senza perdere lo stato dell'app.
* **Widget-Based UI**: Interfaccia completamente costruita con widget altamente personalizzabili.
* **Prestazioni Elevate**: Usa il motore grafico Skia per un rendering fluido e veloce.
* **Material** & **Cupertino Design**: Supporta nativamente i design di Android e iOS.

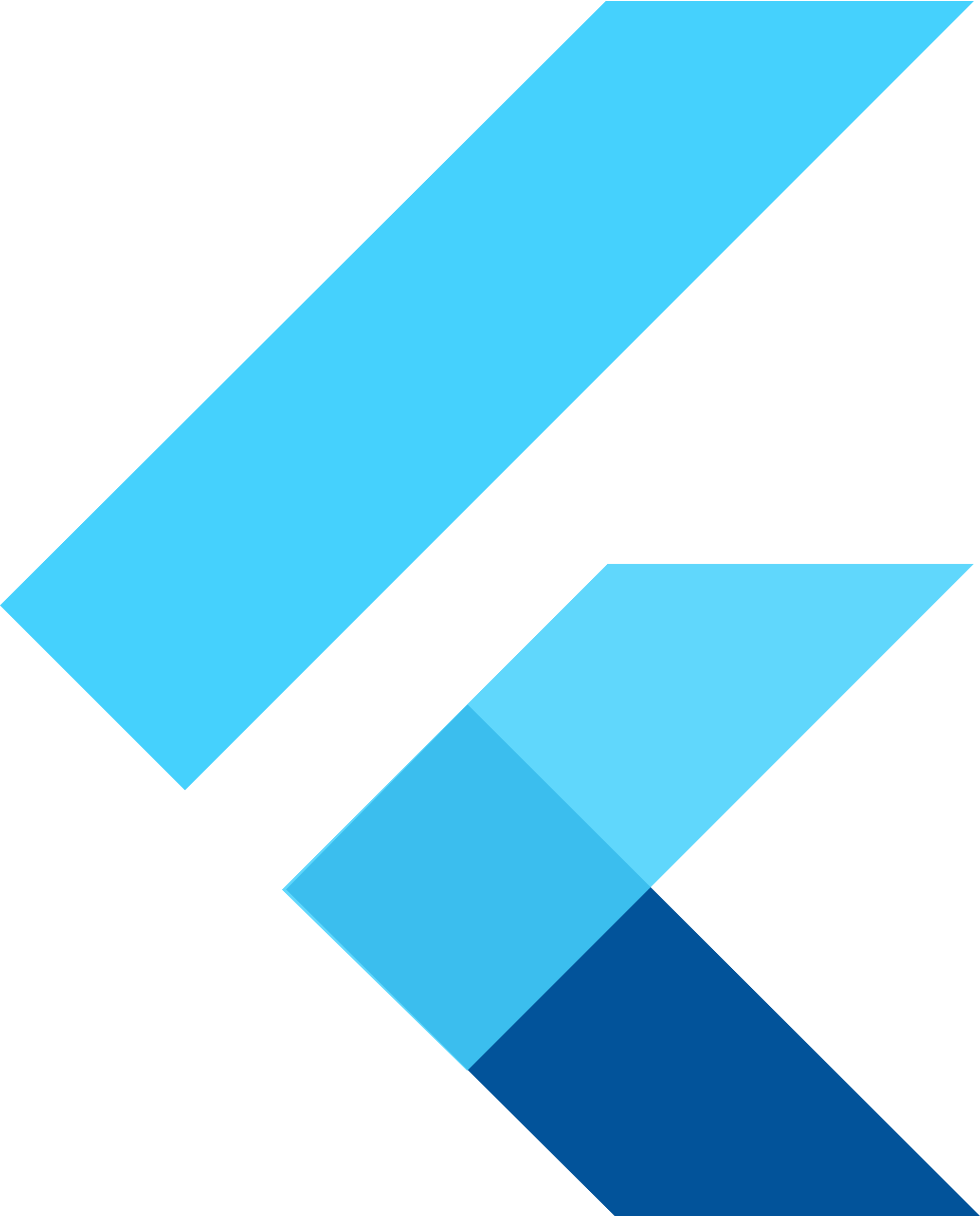
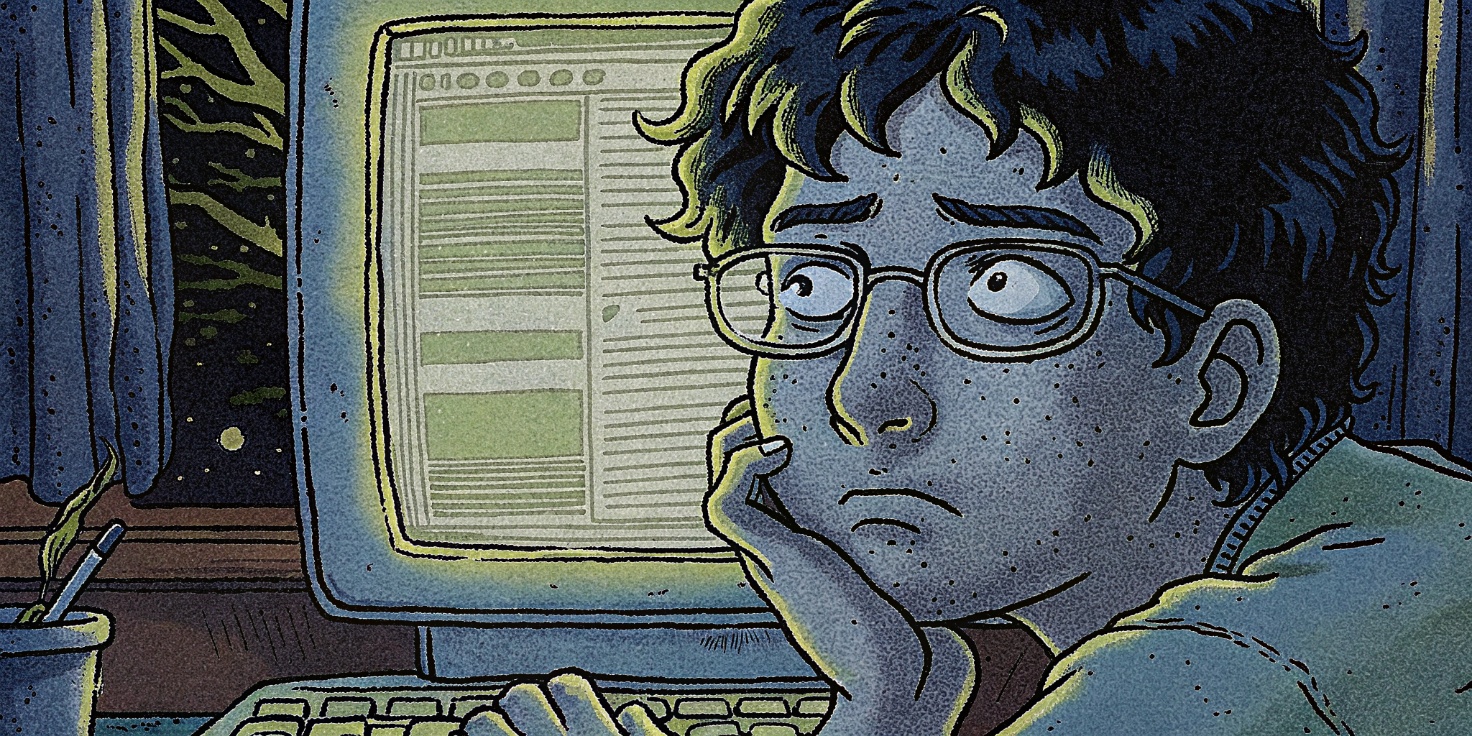


Figura Link al corso:

https://www.youtube.com/watch?v=d7j6vZHskNY&list=PLP5MAKLy8lP93TzdGqBnuc2dK02Mz6-VH

Capitolo 1 - Installazione

**Prerequisiti**:

**Installazione Dart**

1. Installa **powershell** (se non già presente)
2. Installare **Chocolatey\*** (<https://chocolatey.org/install#individual>)
3. Installa sdk dart (**choco install dart-sdk** **\*\***)
4. Installa PowerShell
5. Installa git cli

**Installazione Flutter**

1. Scarica Flutter SDK
2. Crea nel disco **C** la cartella **src** dove inserire il contenuto dello zip
3. Avvia la flutter\_console
4. Scrivi nel terminale … per verificare se l’installazione è andata a buon fine

flutter doctor

1. Installa le relative mancanze

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

1. Aggiungi ai **PATH** di account il path relativo flutter

C:\src\flutter\bin

1. Installa Android Studio

Installa plugin Flutter in esso

Installa plugin Dart in esso

Vai in customize 🡪All settings 🡪 Appearance 🡪 System Settings 🡪 Android SDK

Vado in **SDK Tools** e aggiungi questi tools

 Immagine che contiene testo, schermata, software, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Vai in More Actions e su Virtual Device manager per definire **l’emulatore** e avviarlo con play

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

1. Installa con flutter console le licenze android

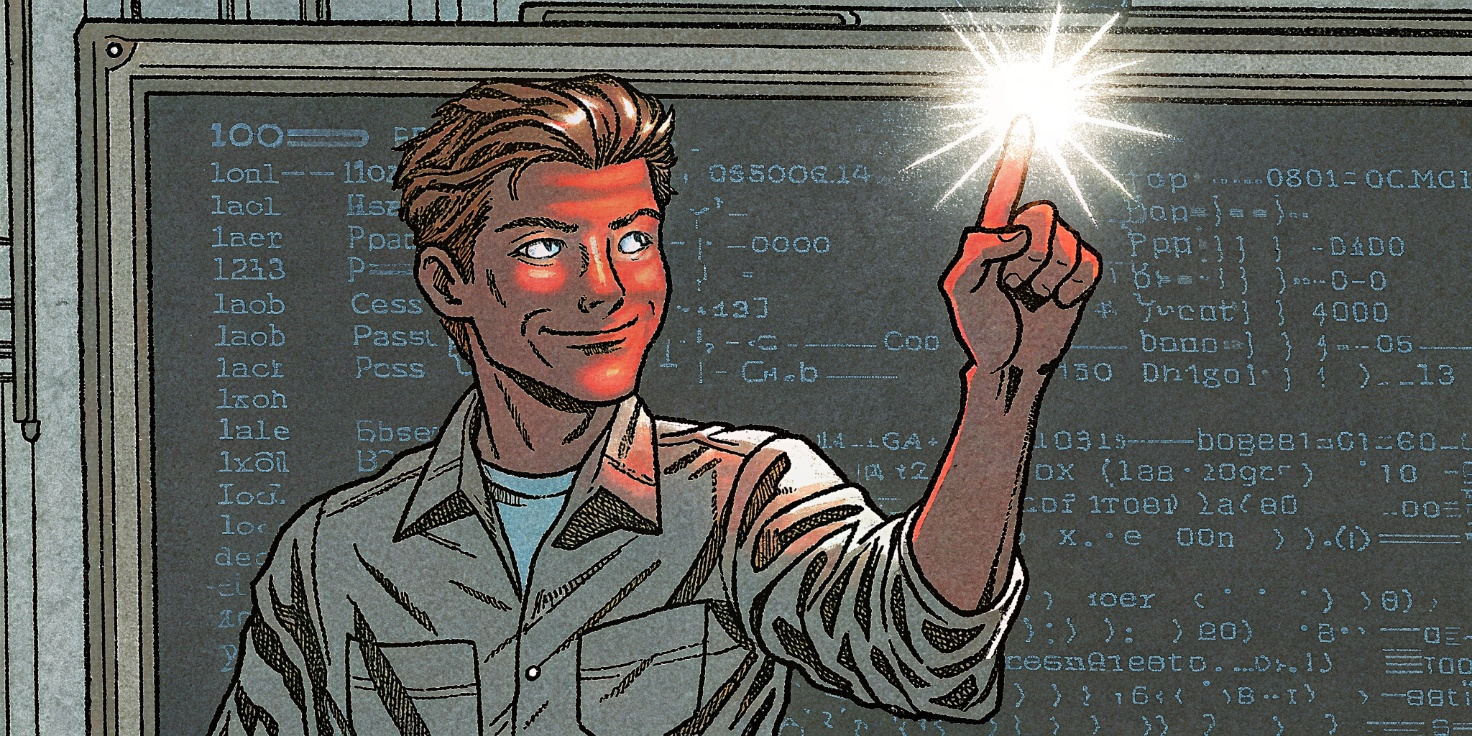
flutter doctor --android-licenses

1. Per sviluppare è possibile usare **android studio** o **VSCode**, eventualmente installa le estensioni di **Dart** e **Flutter**.

**\*** **Chocolatey** è un gestore di pacchetti, al pari di apt su linux

**\*\*** Un **SDK** (Software Development Kit) è un insieme di strumenti, librerie, documentazione e esempi forniti da un fornitore per aiutare gli sviluppatori a creare applicazioni o integrare servizi specifici all'interno di un determinato ecosistema o piattaforma.

Capitolo 2 – Creazione nuovo progetto



1. Apri Android Studio
2. Crea un nuovo progetto in Flutter

Immagine che contiene testo, schermata, Carattere, design

Il contenuto generato dall'IA potrebbe non essere corretto.

1. Vai su Generators
2. Selezione la cartella con l’SDK appena installato

Immagine che contiene schermata, testo, software, Software multimediale

Il contenuto generato dall'IA potrebbe non essere corretto.

1. Seleziona il path e le piattaforme su cui poi esportare il programma

Immagine che contiene testo, schermata, software, Software multimediale

Il contenuto generato dall'IA potrebbe non essere corretto.

**Struttura progetto:**

**📂 code**

**│── 📂 dart\_tool**: riferimento a dart

**│── 📂 idea:** cartella di android studio per config. I l workspace

**│── 📂 lib:** cartella con il file **main.dart**, entry point dove viene eseguito il vero e proprio sviluppo

**│── 📂 android:** cartella effettiva del progetto in android, per fare modifiche specifiche a quell’ambiente

**│── 📂 ios:** cartella effettiva del progetto in ios, per fare modifiche specifiche a quell’ambiente

**│── 📂 test:** cartella per mettere gli scripts di test

**│── 📄 .gitignore:** ignora I files da caricare nel repo

**│── 📄 metadata:** metadati di flutter

**│── 📄 analysis\_options.yaml:** file in cui possiamo mettere le regole per il **linting** di Dart

**│── 📄 corso\_flutter.iml:** tipo index.html

**│── 📄 pubspec.lock:** tutte le dependecies effettive (pubspec.yaml, ma specifico)

**│── 📄 pubspec.yaml:** possiamo andare a definire i dati principali del progetto (nome proj., environment, dependecies, assets, ecc…)

**│── 📄 README.md**

**📂 External Libraries**

**│── 📂 Dart Packages**

**│── 📂 Dart SDK**

**Uso VSCode:**

1. Vai su vscode e installa le estensioni Flutter e Dart
2. Apri terminale e usa doctor per vedere lo stato di flutter

flutter doctor

1. Controlla la presenza di emulatori

flutter emulators

1. Controlla la presenza di devices

flutter devices

1. Crea un Progetto in flutter

flutter create my\_app

1. Avvia emulatore

flutter emulators –launch nome\_emulatore

1. Avvia progetto con emulatore

flutter run -d nome\_emulatore

1. Se non funziona, forza la pulizia della cache

flutter clean

Capitolo 3 – Hello World



Avvia l’emulatore del dispositivo che ti interessa (nel nostro caso un dispositivo mobile)

Facciamo il run dell’applicazione

La struttura che ci si parerà di fronte sarà:

Immagine che contiene testo, schermata, software, Software multimediale

Il contenuto generato dall'IA potrebbe non essere corretto.

* Import delle librerie utili (es. material di flutter, che importa lo stile material per Flutter)
* Entry point del programma (**main()**) da cui viene avviato il programma,
* richiamando, per prima, la funzione **runApp()**,
* che ha come argument la classe **MyApp**, classe estesa
* da una super classe **stateless widget**

**📌 MyApp()**

Questa classe ha un **costruttore**

const MyApp({super.key});

E un metodo che effettua l’override (sovrascrittura), della classe di provenienza.

Il metodo in questione, che restituisce un dato di **tipo Widget**, crea, appunto, un **widget**, sfruttando il metodo **build()**.

Cioè che verrà restituito è un widget di tipo **Material**, un tipo di widget proprio del modulo importato Material.

📍 La classe (widget) **MaterialApp** ha delle proprietà,

* **title**: il titolo
* **theme**: lo stile
* **home**: il punto di ingresso per l’app

Widget build(BuildContext context) {  
 return MaterialApp(  
 title: 'Flutter Demo 8',  
 theme: ThemeData(  
 colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),  
 ),  
 home: const MyHomePage(title: 'Flutter Demo Home Page 18'),  
 );  
}

**State of Widgets**

MaterialApp è un **widget** **stateless**, ovvero che **non** permette di **salvare uno stato** (state)!

A differenza della classe **MyHomePage**, che è un **widget** **statefull**, ovvero che **permette di salvare uno stato** (state)!

📌 La classe (widget) **MyHomePage**, ha un costruttore e, al pari della classe MyApp, un metodo che deve essere sovrascritto.

class MyHomePage extends StatefulWidget {  
 const MyHomePage({super.key, required this.title});  
  
 final String title;  
  
 @override State<MyHomePage> createState() => \_MyHomePageState(); **\*\*\***  
}

Come possiamo notare, il **title** nel costruttore è **required**, ciò obbliga, in fase di instanziamento, l’inserimento forzato di una stringa.

Per il richiamo dello state è sovrascritto con override un metodo **createState()** che richiama il gestore dello state **\_MyHomePageState**.

**NOTA**: mentre i **stateless widget** sono un “pezzo unico” (classe con costruttore e metodi), i **stateful widget** sono composti da “2 parti”, blocco principale e classe di gestione dello state, **MyHomePage** e **\_MyHomePageState**

📌 Struttura del gestore dello state **\_MyHomePageState**

* **\_counter:** variabile inizializzata del counter a 0
* **\_incrementCounter: setta lo stato aumentato il conter che lo incremnta di +1**
* **Build:** viene chiamato un altro metodo di **build()** che crea uno **Scaffold**, un elemento in view che verrà mostrato a schermo.

**\*\*\***

class \_MyHomePageState extends State<MyHomePage> {  
 int \_counter = 0;  
  
 void \_incrementCounter() {  
 setState(() {  
 \_counter++;  
 });  
 }  
  
 @override Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
  
 backgroundColor: Theme.of(context).colorScheme.inversePrimary,  
  
 title: Text(widget.title),  
 ),  
 body: Center(  
  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: <Widget>[  
 const Text(  
 'You have pushed the button this many times, bravo11 :) : ',  
 ),  
 Text(  
 '$\_counter',  
 style: Theme.of(context).textTheme.headlineMedium,  
 ),  
 ],  
 ),  
 ),  
 floatingActionButton: FloatingActionButton(  
 onPressed: \_incrementCounter,  
 tooltip: 'Increment',  
 child: const Icon(Icons.add),  
 ), // This trailing comma makes auto-formatting nicer for build methods.  
 );  
 }  
}

📍**Scaffold**

Una classe costituita, in questo caso, da

* **appBar**: Barra in alto, che contiene
  + **background**
  + **title**
* **Body**: corpo della view, centrato (**center()**), che contiene un
  + **Child**, che incolonna gli elemnti all’interno, con all’interno delle proprietà
* **floatingActionButton**: pulsante flottante, contenente a sua volta delle proprietà e dei metodi, come **\_incrementCounter()**.

Capitolo 4 – Cosa è un Widget?

Immagine che contiene dipinto, Viso umano, uomo, disegno

Il contenuto generato dall'IA potrebbe non essere corretto.

I Widget sono degli atomi che costruiscono il programma.

In Flutter è l'elemento base dell'interfaccia utente, un po' come i **componenti** in un framework web come **React** o gli **elementi HTML** in una pagina web.

Immagina di costruire un sito web:

* Un **div** o un **button** in HTML sono elementi visivi della pagina.
* In React o Angular, usi componenti come <Card /> o <Button /> per creare interfacce più dinamiche.

In Flutter, invece, tutto è un **widget**:

* Un **Text** widget → equivalente a un <p> o <span> in HTML.
* Un **Container** widget → simile a un <div>, perché può contenere altri elementi e avere stili.
* Un **Column** o **Row** → equivalenti a un **flexbox** in CSS per impaginare gli elementi.
* Un **ElevatedButton** → come un <button> in HTML.

Quindi, in Flutter costruisci l'interfaccia componendo widget dentro altri widget, un po' come faresti con i componenti in React o con i tag HTML e CSS in una pagina web.

A volte questi sono dei contenuti (es. widget MaterialApp) a volte possono essere dei container con delle specifiche che raccolgono un contenuto (es. Scaffold).

Capitolo 5 – AppBar

Immagine che contiene arte, dipinto, disegno, schizzo

Il contenuto generato dall'IA potrebbe non essere corretto.

a **AppBar** è un widget che rappresenta la barra superiore di un'applicazione, tipicamente utilizzata per mostrare il titolo della pagina, azioni rapide e un'icona di navigazione. Fa parte del widget **Scaffold** e viene definita attraverso la proprietà appBar, solitamente con un'istanza di AppBar(). Tra le proprietà più comuni troviamo title, che permette di impostare un titolo (spesso un widget Text), actions, una lista di widget (come icone per azioni rapide), e leading, che di solito contiene un'icona per il drawer o un pulsante di ritorno. È possibile personalizzare l'aspetto della AppBar con proprietà come backgroundColor per il colore di sfondo e elevation per l'ombra.

**Esempio base di AppBar:**

Scaffold(

appBar: AppBar(

title: Text('Home'),

backgroundColor: Colors.blue,

actions: [

IconButton(

icon: Icon(Icons.search),

onPressed: () {

// Azione di ricerca

},

),

],

),

);

Insieme ad altri componenti (nell’app di test che stiamo vedendo insieme a **Body** e **floatingActionButton**), è un componente che andrà a costruire la pagina HomePage della nostra app.

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Abdando sopra del nostro componente, avremo un’antrprima degli attributi che vengono accettati da esso, con **tipo** e relativo nome del **subcomponente**.

**Es.**

backgroundColor: Color(0xDDDDDDDD)

per definire il suo sfondo.

actions: [  
Icon(Icons.ice\_skating),  
 Icon(Icons.import\_contacts)],

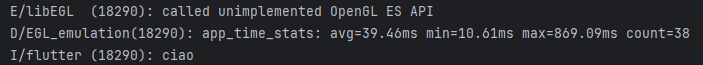
per definire dei tasti azione.

Tra le actions è possibile definire anche delle funzione all’interno dei nostri pulsanti usando delle **callback**

…

IconButton(onPressed: () => {print('ciao')}, icon: Icon(Icons.import\_contacts)),

…]

**Output:** 

**Immagine che contiene schermata, testo, telecomando

Il contenuto generato dall'IA potrebbe non essere corretto.NOTA**: Per visualizzare il terminale, clicca il tasto run

**NOTA 2**: Una nota importante è legata alla dichiarazione costante o dinamica delle variabili.

const Icon(Icons.ice\_skating), ( Icon(Icons.accessible\_sharp),

Il const permette di far capire, in fase di reload, all’applicazione che quell’istanza sarà costante nel tempo, non inserendola nel pool del refresh!

Ma esiste una quantità enorme di proprietà per modificare l’UI della nostra appBar.

Immagine che contiene testo, schermata, software, Software multimediale

Il contenuto generato dall'IA potrebbe non essere corretto.

Capitolo 6 – Testo e stile



**The End**

**Dedalo’s notes – Una serie di appunti da non prendere sul serio**

Serie di appunti sugli argomenti più svariati, fatti per imparare, apprendere e divertirsi, navigando per il magico mondo del…tutto!

Ogni errore o imprecisione sarà sistemata, oppure no 😊.

Attenzione, può contenere boiate, amenità o stramberie!

https://github.com/GiacomoBorsellino

Scritto da

**Giacomo Borsellino**

20-02-2025