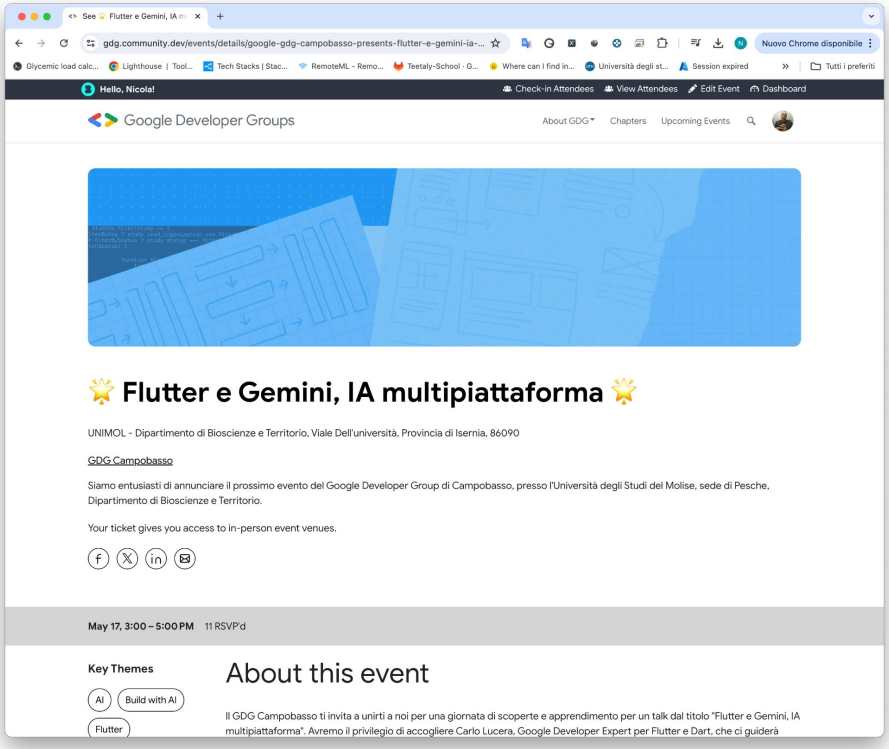
# Programmazione Mobile

**Nicola Noviello**
nicola.noviello@unimol.it

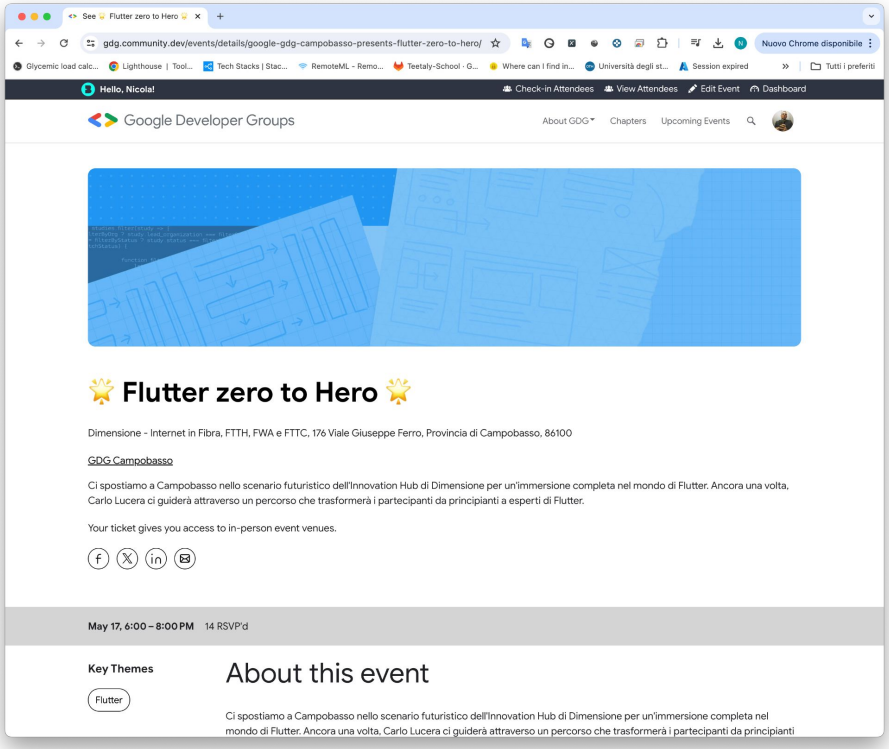
Corso di Laurea in Informatica
Dipartimento di Bioscienze e Territorio
Università degli Studi del Molise
Anno 2023/2024

# Eventi Flutter

# Evento Unimo 17 Maggio ore 15

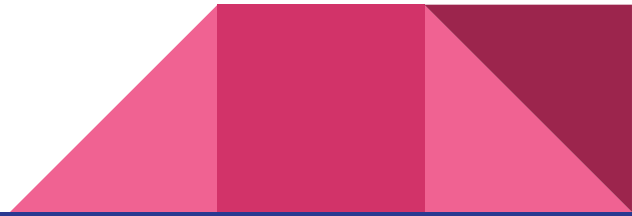


# Evento a Campobasso 17 Maggio ore 18



# Lezione: Temi, Gestione dell’asincronia, Navigazione e chiamate HTTP

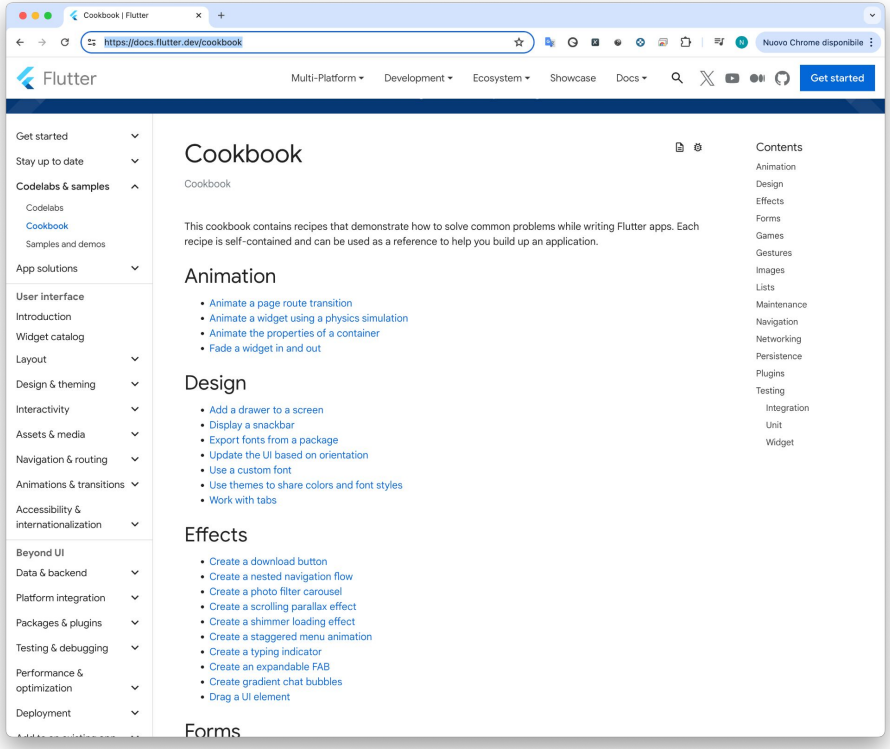
- Panoramica sull’uso dei temi
- Navigazione tra schermate
- Futures, Async & Await
- Gestione delle eccezioni
- Networking in flutter
- Parsing di JSON
- Gestione Location



# Uso dei temi

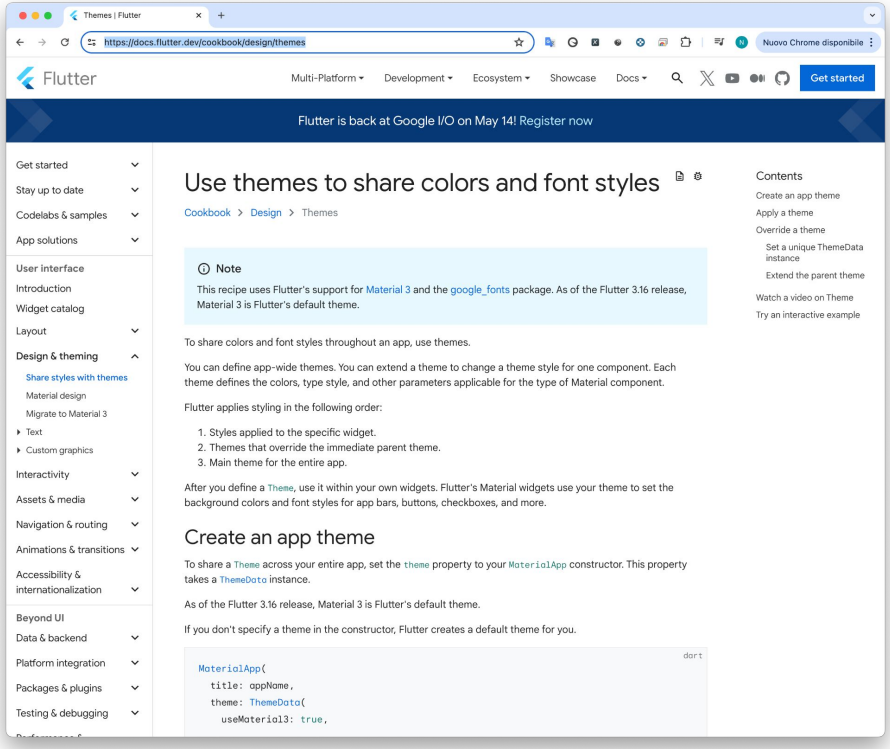
# Flutter Cookbook - https://docs.flutter.dev/cookbook

I Cookbook sono delle “ricette”. Possiamo usarli quando abbiamo necessità di implementare velocemente qualcosa senza scendere in maniera approfondita nella documentazione

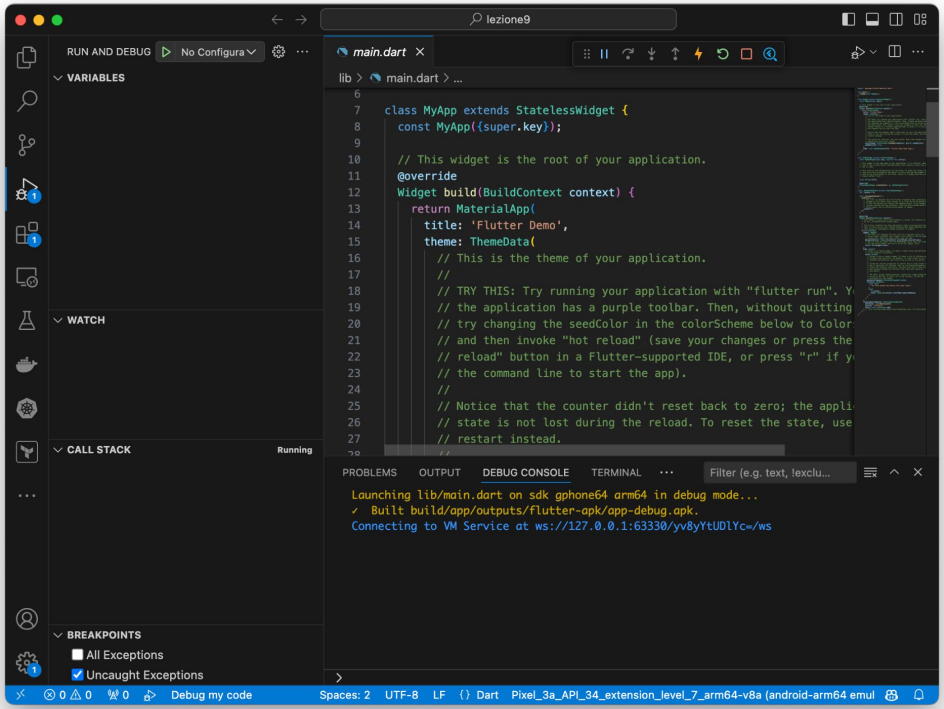


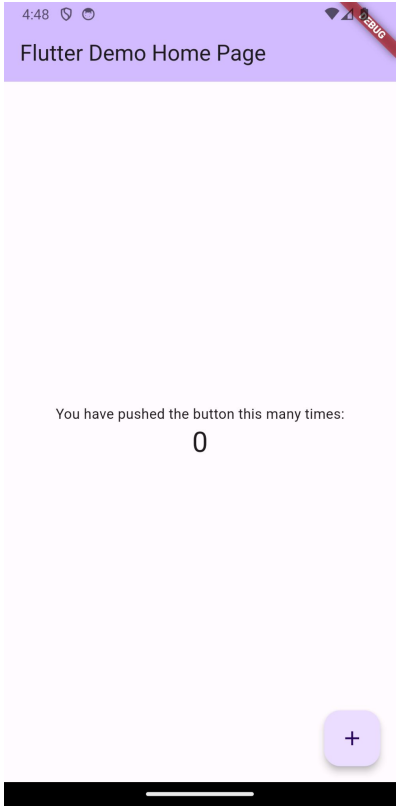
# Themes Cookbook - https://docs.flutter.dev/cookbook/design/themes

Una “ricetta” specifica sull’uso dei temi, ci aiuta ad essere subito operativi per rendere il design dell’Applicazione coerente tra tutte le componenti



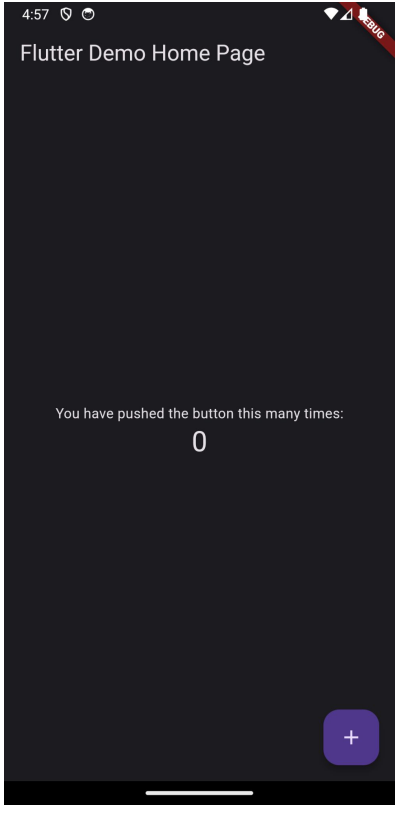
# Funzionalità principali riguardanti i temi




Progetto di default

# Theme.dark()



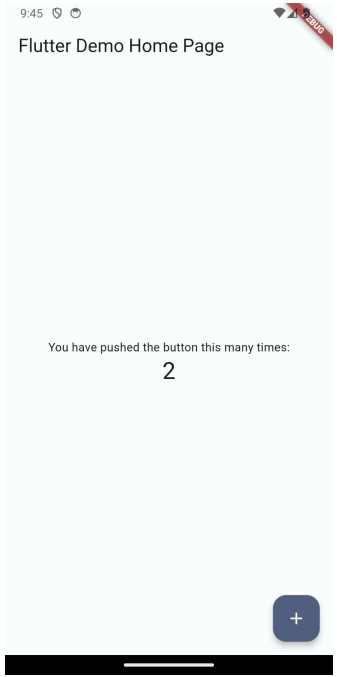

```
}

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.dark(),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    ); // MaterialApp
  }
}

```

# Theme - Impostazioni manuali

```
8   const MyApp({super.key});
9
10  // This widget is the root of your application.
11  @override
12  Widget build(BuildContext context) {
13    final ColorScheme colorScheme = ColorScheme.fromSeed(
14      brightness: MediaQuery.platformBrightnessOf(context),
15      seedColor: Color.fromARGB(255, 21, 195, 99),
16    ); // ColorScheme.fromSeed
17    return MaterialApp(
18      title: 'Flutter Demo',
19      theme: ThemeData(
20        colorScheme: colorScheme,
21        floatingActionButtonTheme: FloatingActionButtonThemeData(
22          backgroundColor: colorScheme.tertiary,
23          foregroundColor: colorScheme.onTertiary,
24        ), // FloatingActionButtonThemeData
25      ), // ThemeData
26      home: const MyHomePage(title: 'Flutter Demo Home Page'),
27    ); // MaterialApp
28  }
29 }
30
```

# Theme - Impostazioni manuali






```
seedColor: Colors.blueAccent,
```

```
seedColor: Colors.purpleAccent,
```

```
seedColor: Colors.grey,
```

# Theme - Impostazioni manuali

Leggermente più scuro

```dart
// This widget is the root of your application.
@override
Widget build(BuildContext context) {
  final ColorScheme colorScheme = ColorScheme.fromSeed(
    brightness: MediaQuery.platformBrightnessOf(context),
    seedColor: Colors.grey,
  );
  return MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
      colorScheme: colorScheme,
      scaffoldBackgroundColor: colorScheme.background.withOpacity(0.9),
      floatingActionButtonTheme: FloatingActionButtonThemeData(
        backgroundColor: colorScheme.tertiary,
        foregroundColor: colorScheme.onTertiary,
      ), // FloatingActionButtonThemeData
    ), // ThemeData
    home: const MyHomePage(title: 'Flutter Demo Home Page'),
  ); // MaterialApp
}
```

# Modalità Dark e Light

**Modalità Dark**

La modalità dark è una variante dell'interfaccia utente che utilizza uno schema di colori scuri anziché chiari;

Migliora l'esperienza dell'utente in condizioni di scarsa illuminazione e riduce l'affaticamento visivo;

È sempre più popolare nelle applicazioni moderne per la sua estetica elegante e la riduzione del consumo energetico su dispositivi con schermi OLED.

*(Linares-Vásquez, M., Bavota, G., Cárdenas, C. E. B., Oliveto, R., Di Penta, M., & Poshyvanyk, D. (2015, August). Optimizing energy consumption of guis in android apps: A multi-objective approach. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (pp. 143-154).)*

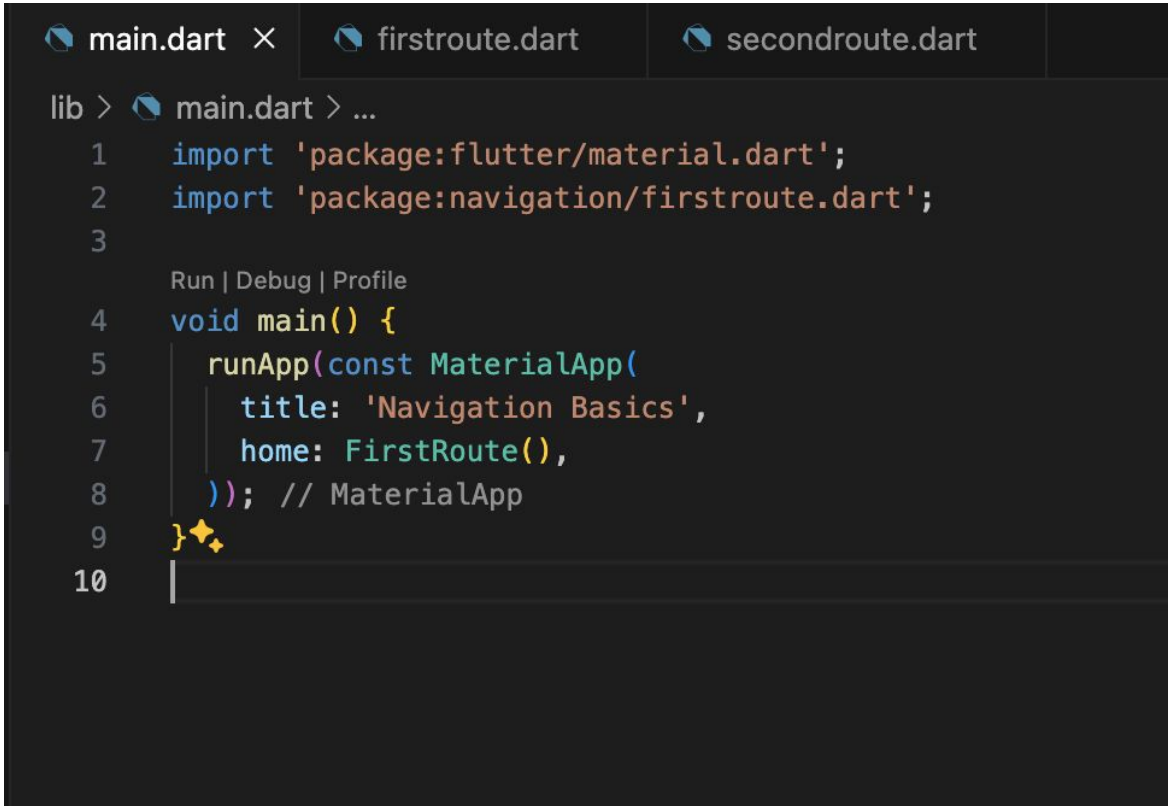
**Implementazione in Flutter**

Attraverso l'uso dei temi, utilizzando ThemeData, è possibile implementare in maniera selettiva l'uso delle due modalità in base alle impostazioni dell'utente o al tema di sistema del dispositivo.

# Navigazione

# Navigazione semplice tra due schermate

Un semplice main che chiama il Widget FirstRoute()

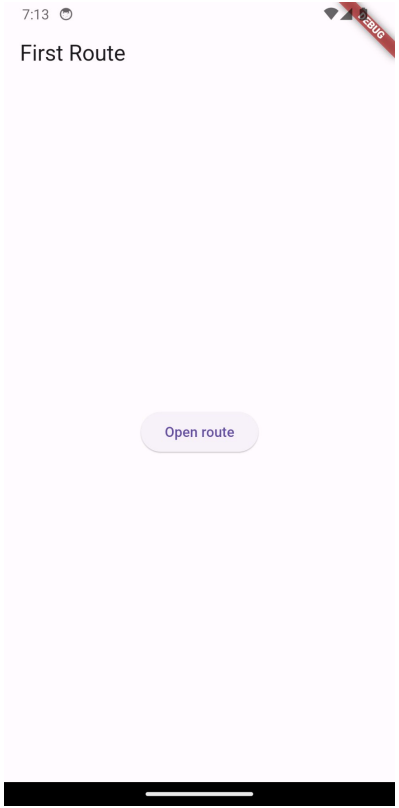


# Navigazione semplice tra due schermate

main.dart | firstroute.dart | secondroute.dart

lib > firstroute.dart > FirstRoute > build

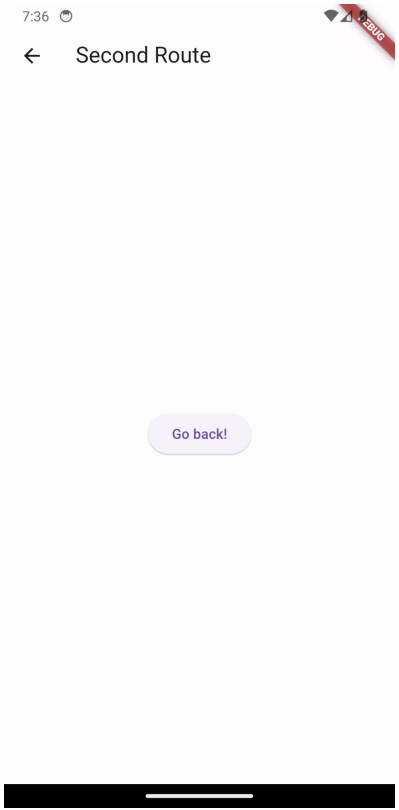
```dart
import 'package:flutter/material.dart';

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Route'),
      ), // AppBar
      body: Center(
        child: ElevatedButton(
          child: const Text('Open route'),
          onPressed: () {},
        ), // ElevatedButton
      ), // Center
    ); // Scaffold
  }
}

```



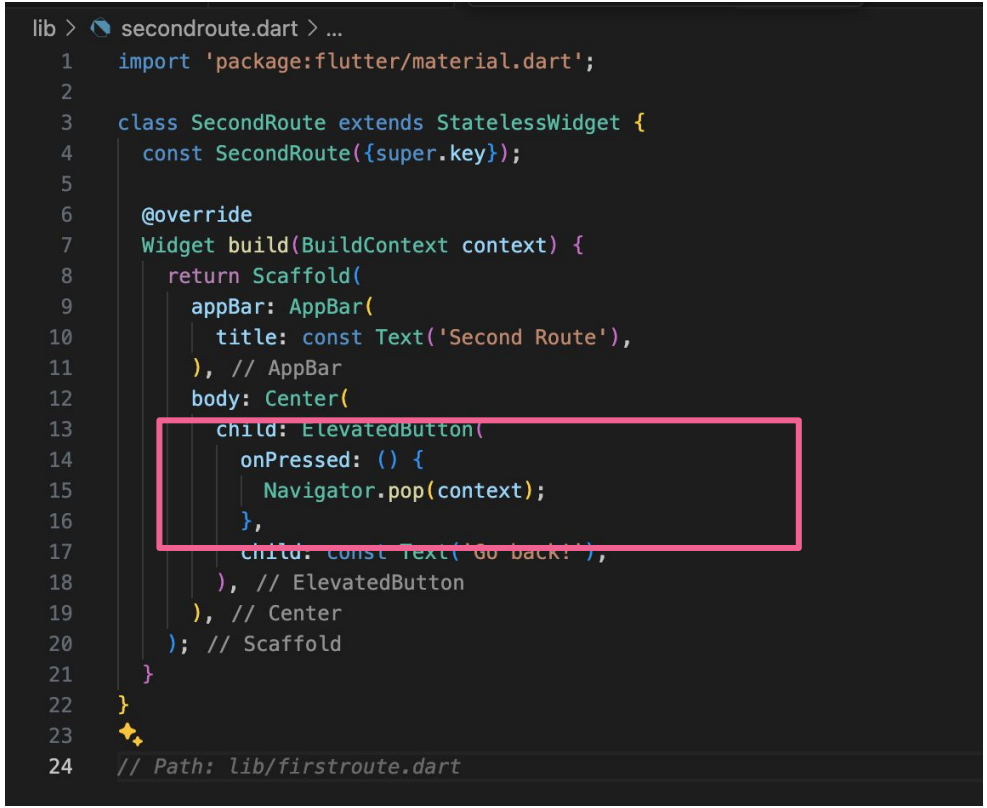

Creo un Widget SecondRoute() e voglio fare in modo di spostarmi una volta premuto “Open route”

# Navigazione semplice tra due schermate

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Route'),
      ), // AppBar
      body: Center(
        child: ElevatedButton(
          child: const Text('Open route'),
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => const SecondRoute()),
            );
          },
        ), // ElevatedButton
      ), // Center
    ); // Scaffold
  }
}
```



# E se voglio tornare indietro?



# Navigazione a Stack

- In Flutter, la navigazione a stack si riferisce alla gestione delle diverse schermate all'interno dell'App;
- Le schermate vengono sovrapposte una sopra l'altra, formando uno stack;
- Quando si passa da una schermata all'altra, si aggiungono o si rimuovono schermate dallo stack.

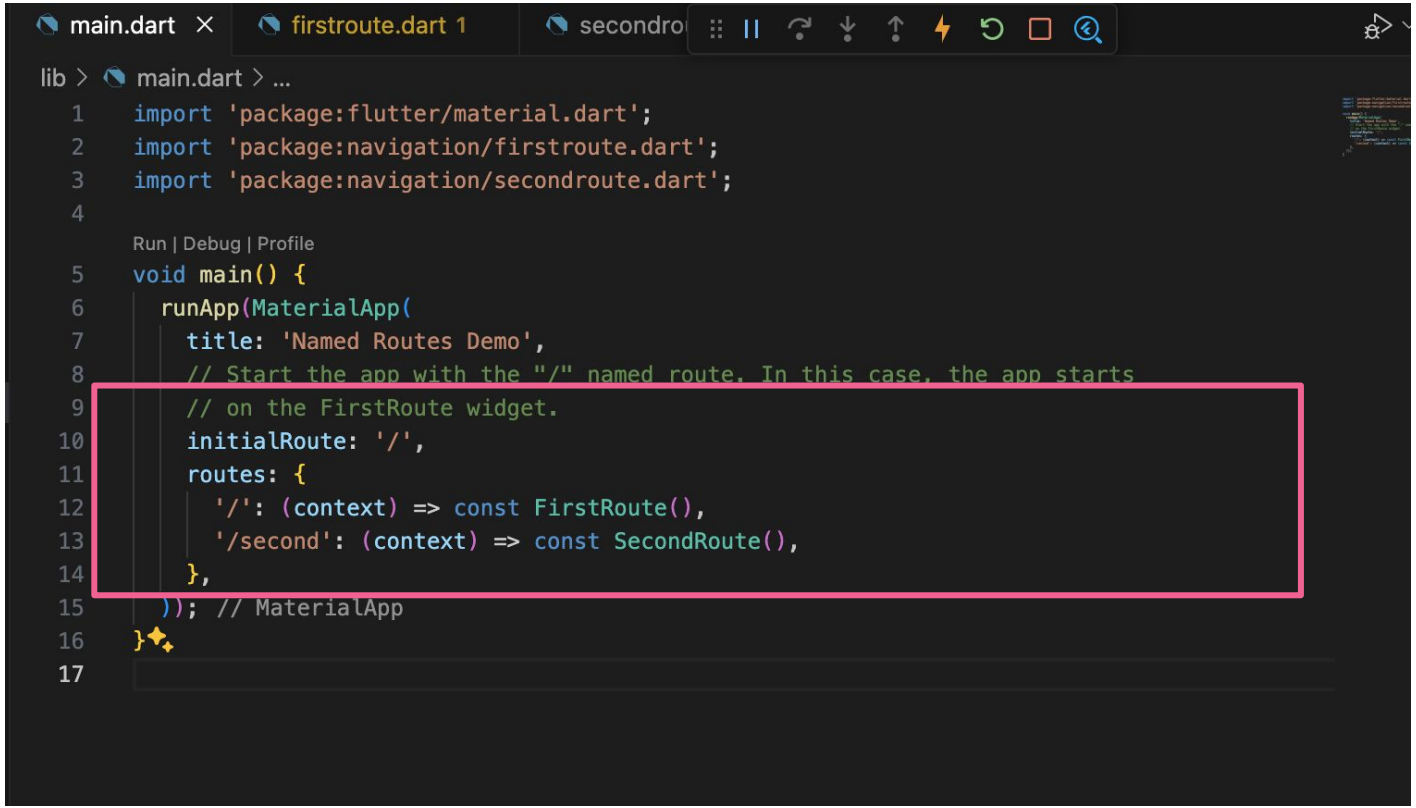
**Push e Passaggio di Context in MaterialPageRoute:**

- Il metodo push viene utilizzato per aggiungere una nuova schermata allo stack di navigazione;
- context in MaterialPageRoute rappresenta il "contesto", la posizione dell'attuale widget nell'albero dei widget;
- Viene utilizzato per ottenere informazioni sul tema, la localizzazione e altri aspetti dell'App;

**Pop:**

- Il metodo pop viene utilizzato per rimuovere la schermata corrente dallo stack e tornare alla schermata precedente;
- Quando viene chiamato pop, la schermata attuale viene rimossa dallo stack e il controllo torna alla schermata precedente nello stack;
- È comunemente utilizzato con pulsanti "Indietro" o altre azioni utente per tornare indietro nella navigazione.

# Navigazione con “rotte con nome” o denominate




Stesso funzionamento del precedente, ma approccio differente. Il main.dart viene trasformato in questo modo

# Navigazione con “rotte con nome” o denominate

lib > firstroute.dart > ...

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Route'),
      ), // AppBar
      body: Center(
        child: ElevatedButton(
          child: const Text('Open route'),
          onPressed: () {
            Navigator.pushNamed(context, '/second');
          },
        ), // ElevatedButton
      ), // Center
    ); // Scaffold
  }
}
```

Cambia qualcosa anche in FirstRoute…

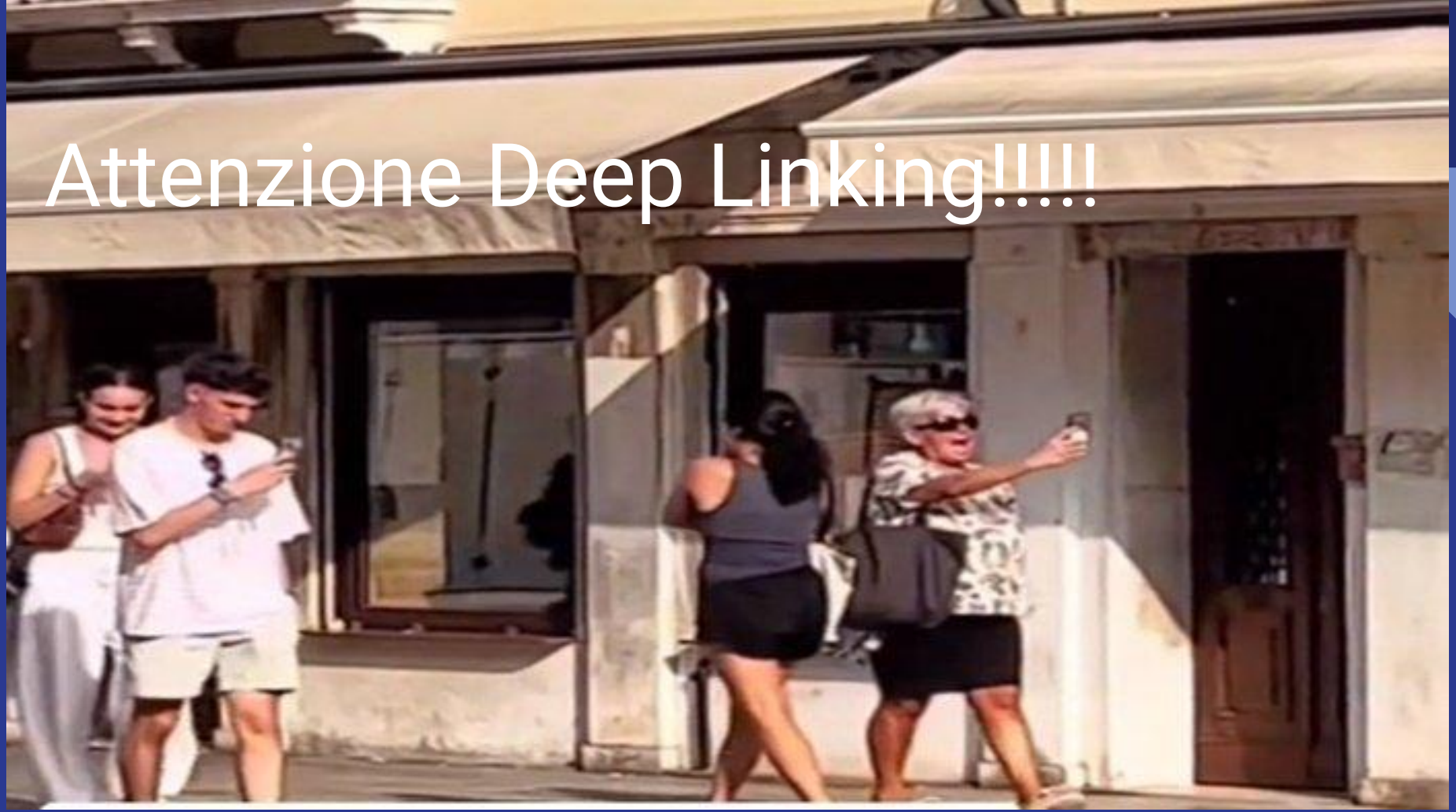
…SecondRoute rimane identico ma il funzionamento è analogo al precedente esempio

# Esercizio

Aprite dartpad al seguente url:
https://dartpad.dev/?id=a16d66a94b17becee11bcbd30fb2b171

Provate a navigare tra le rotte, modificatele, aggiungetene di nuove

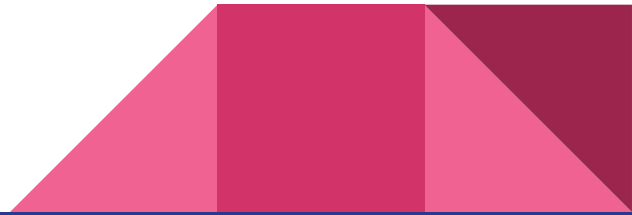
# Attenzione Deep Linking!!!!!



# Attenzione: Deep link warning

Come specificato nella documentazione ( https://docs.flutter.dev/ui/navigation ) l'utilizzo delle rotte con nome è sconsigliato in caso di App complesse che fanno un uso intensivo di Deep link.

In questo caso è consigliato l'uso di package esterni come go_router (https://pub.dev/packages/go_router) che permettono la navigazione verso rotte specifiche.



# Deep link

I Deep Link in Flutter si riferiscono alla capacità di gestire URL “diretti” all’interno dell'applicazione. Un deep URL è un URL che punta direttamente a un contenuto specifico all’interno dell’applicazione anziché alla schermata principale. Ad esempio, si potrebbe avere un deep URL come myunimol://pagina2 che apre direttamente la "pagina2" dell’applicazione anziché la schermata iniziale.

I deep link sono utili per fornire agli utenti un modo diretto per accedere a contenuti specifici all'interno dell’applicazione, consentendo loro di evitare la navigazione manuale attraverso le schermate. Sono particolarmente utili per collegare contenuti dell’applicazione da fonti esterne come link in messaggi, e-mail o siti web.

Ecco come funziona l'implementazione dei deep link in Flutter:

- **Definizione delle rotte con nome (o denominate)**: Le rotte denominate sono assegnate a URL specifici nell’applicazione. Ad esempio, si potrebbe assegnare l'URL myunimol://cambiopassword alla rotta denominata "cambiopassword" dell'applicazione;
- **Registrazione dei gestori di rotta**: All’avvio dell’applicazione, è necessario registrare i gestori delle rotte che determinano come interpretare i deep URL. Questi gestori di rotta possono essere definiti per rispondere a URL specifici o per elaborare URL generici e indirizzarli alle schermate appropriate utilizzando il sistema di navigazione di Flutter, come ad esempio Navigator.pushNamed(context, '/cambiopassword).
- **Parsing dei deep URL**: Quando l'applicazione riceve un deep URL, il framework Flutter lo analizza per determinare la rotta associata e inoltrare la richiesta alla schermata corretta;

# Ciclo di vita di un Widget Stateful - questa volta con esempio pratico

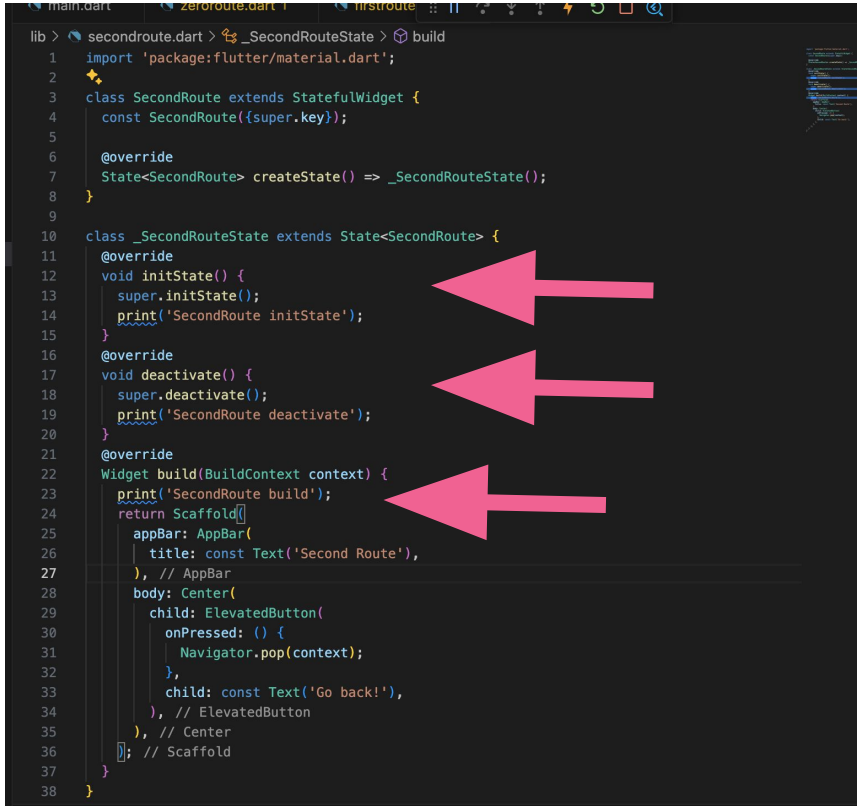
# Ciclo di vita dei Widget Stateful


Ogni Widget di Flutter ha un ciclo di vita integrato: una serie di metodi eseguiti automaticamente da Flutter (in determinati momenti).

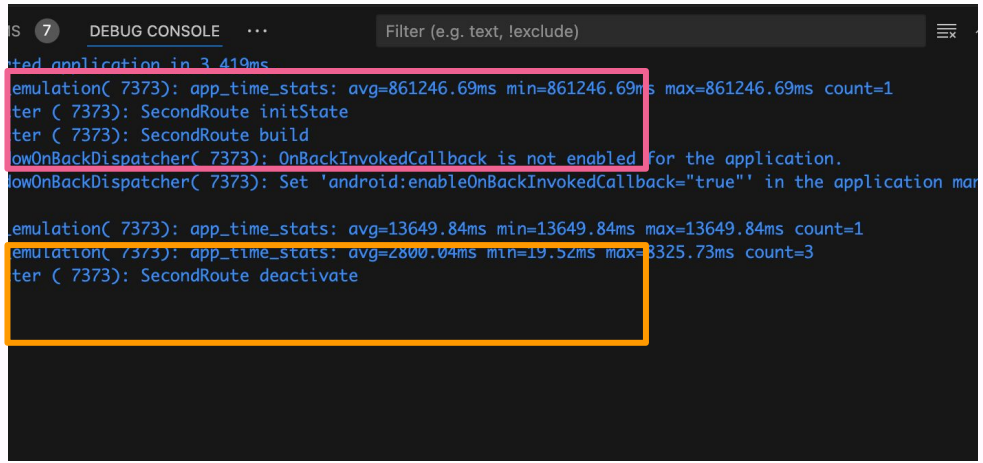
Ci sono tre metodi estremamente importanti del ciclo di vita di un **Widget Stateful**:

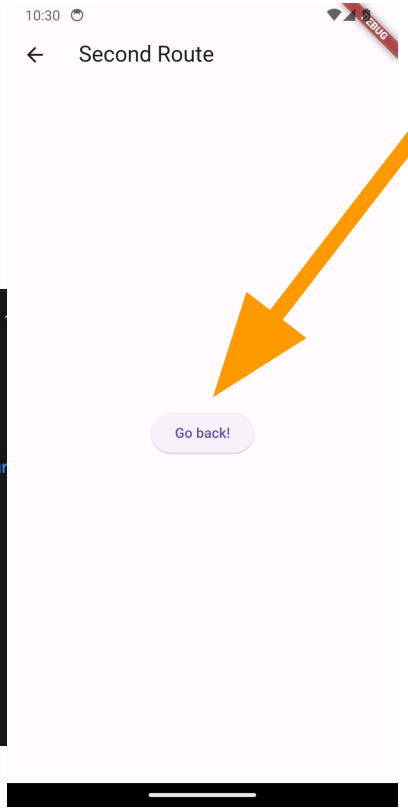
- **initState()**: Eseguito da Flutter quando lo State object del **Widget Stateful** viene **inizializzato**
- **build()**: Eseguito da Flutter quando il Widget viene costruito per la **prima volta** e ogni altra volta quando viene chiamato il metodo **setState()**
- **dispose()**: Eseguito da Flutter proprio prima che il Widget venga eliminato (ad es., perché è stato visualizzato condizionalmente)

# Rendiamo Stateful SecondRoute



# Log degli stati

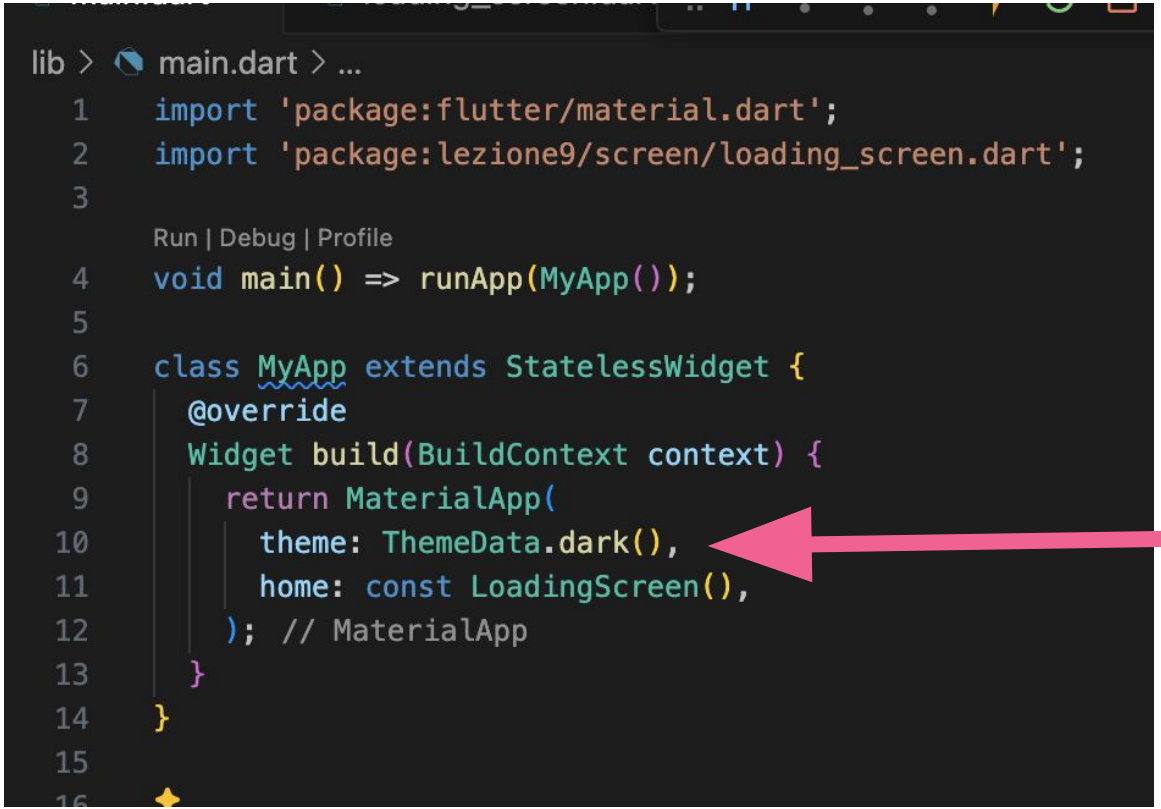



# Creiamo un’App Meteo

# Accesso al GPS

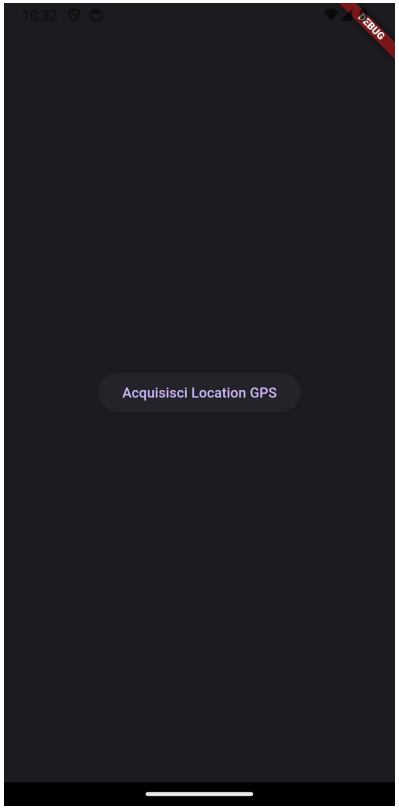
# Progetto di partenza



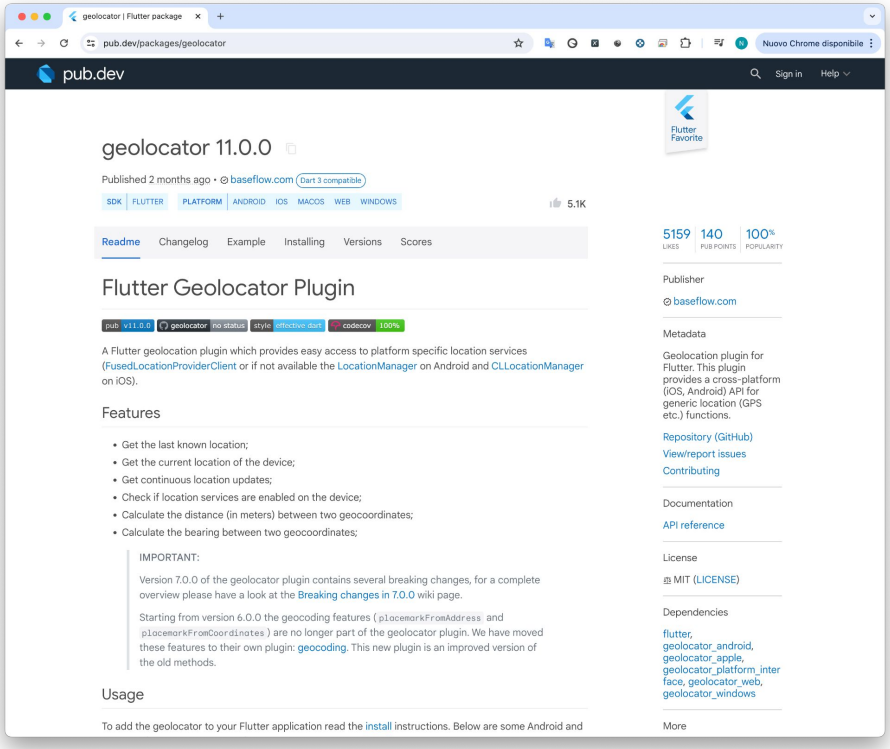
# Progetto di partenza

lib > screen > loading_screen.dart > ...

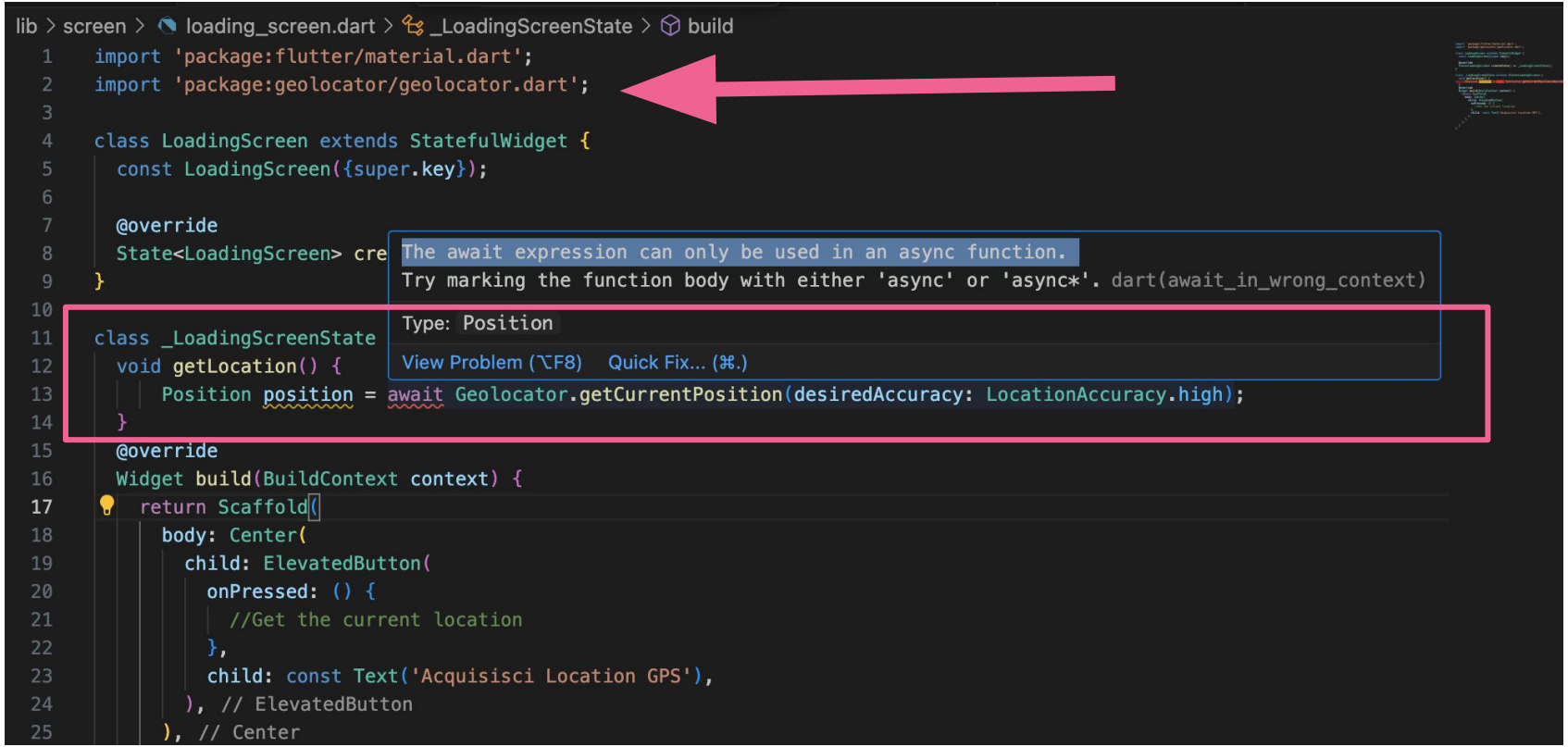
```dart
import 'package:flutter/material.dart';

class LoadingScreen extends StatefulWidget {
  const LoadingScreen({super.key});

  @override
  State<LoadingScreen> createState() => _LoadingScreenState();
}

class _LoadingScreenState extends State<LoadingScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            //Get the current location
          },
          child: const Text('Acquisisci Location GPS'),
        ), // ElevatedButton
      ), // Center
    ); // Scaffold
  }
}
```



# geolocator - https://pub.dev/packages/geolocator/



# Integrazione geolocator



# Integrazione geolocator

# Integrazione geolocator

```
}

class _LoadingScreenState extends State<LoadingScreen> {
  void getLocation() async {
    Position position = await Geolocator.getCurrentPosition(
        desiredAccuracy: LocationAccuracy.low);
    print(position);
  }

@override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: ElevatedButton(
          onPressed: getLocation,
          child: const Text('Acquisisci Location GPS'),
        ), // ElevatedButton
      ), // Center
    ); // Scaffold
  }
}
```

# Test da simulatore…

```
@override
State<LoadingScreen> createState() => _LoadingScreenState();
}

class _LoadingScreenState extends State<LoadingScreen> {
  void getLocation() async {
    Position position = await Geolocator.getCurrentPosition(
```

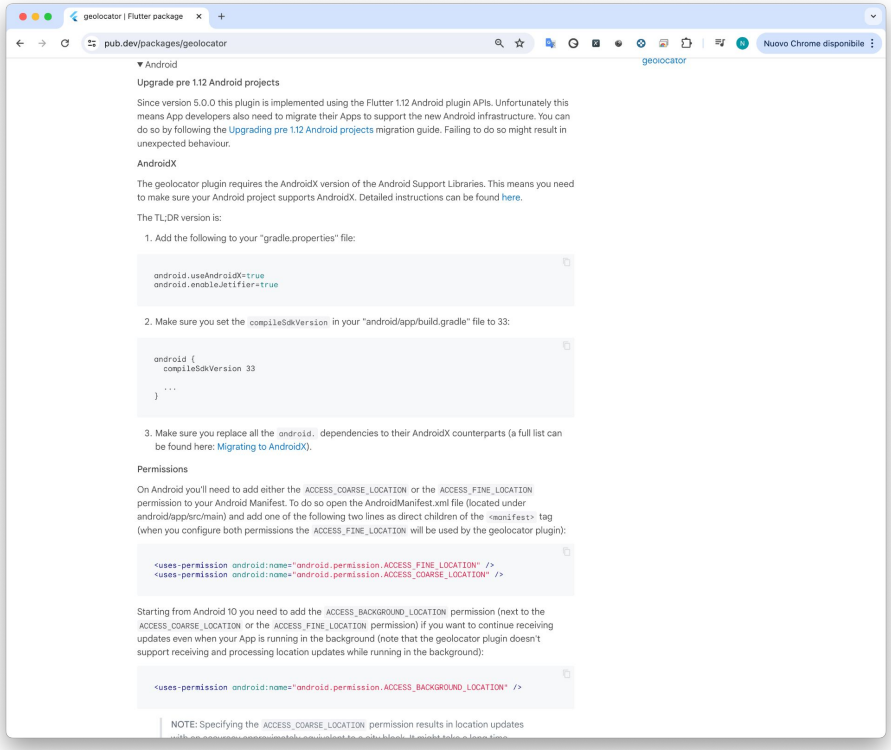
**Exception has occurred.** ×

MissingPluginException (MissingPluginException(No implementation found for method getCurrentPosition on channel flutter.baseflow.com/geolocator))

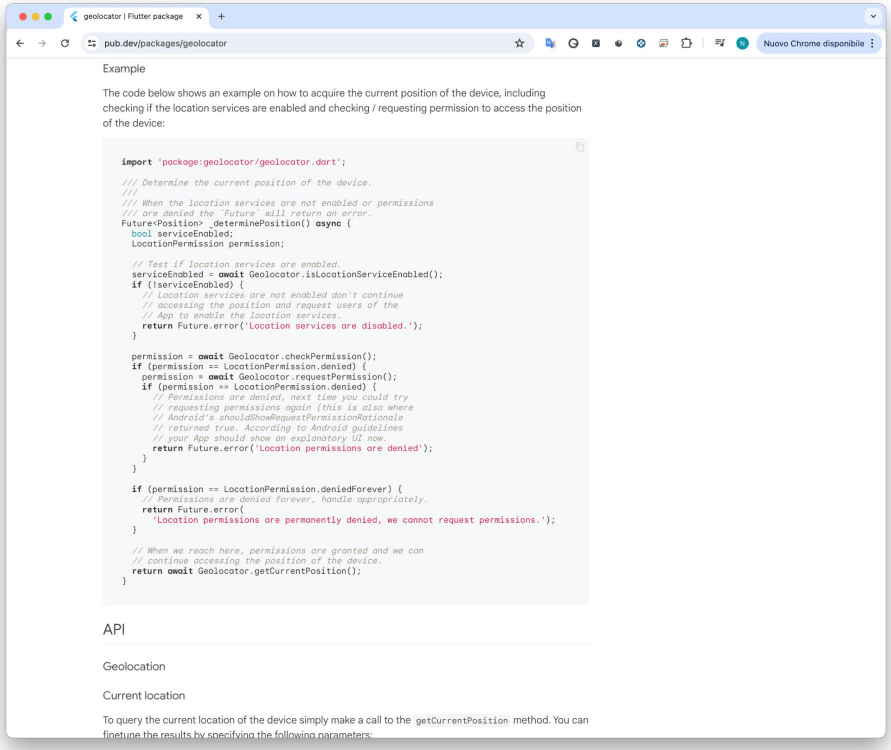
```
        desiredAccuracy: LocationAccuracy.low);
    print(position);
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: ElevatedButton(
          onPressed: getLocation,
          child: const Text('Acquisisci Location GPS'),
```

Abbiamo un errore! Cerchiamo nella documentazione del package…

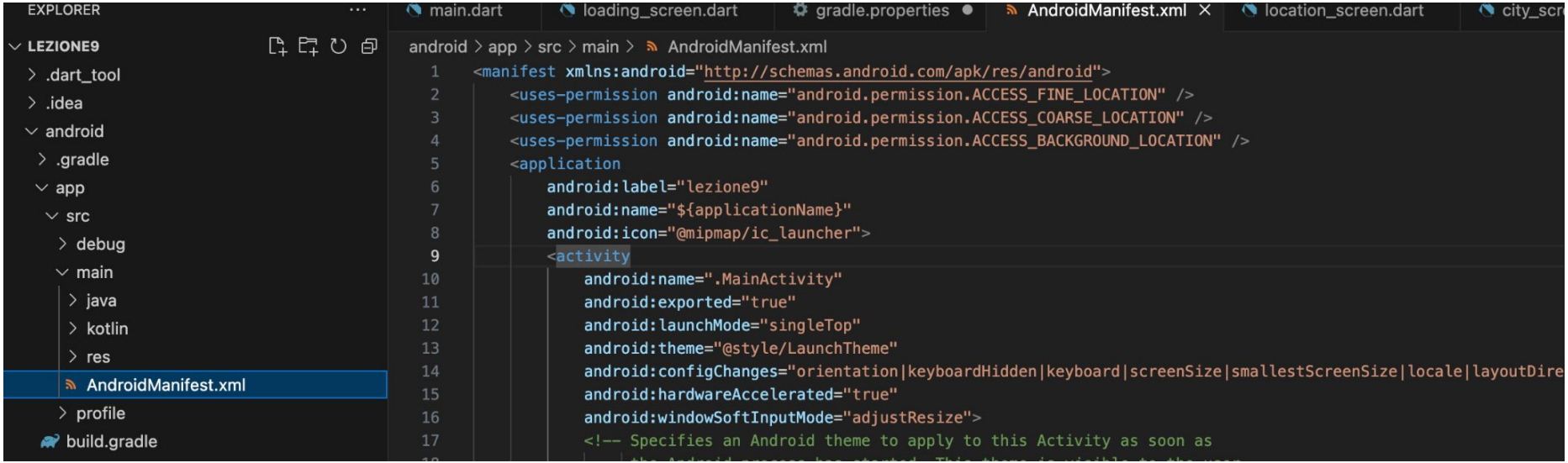
# Integrazione geolocator



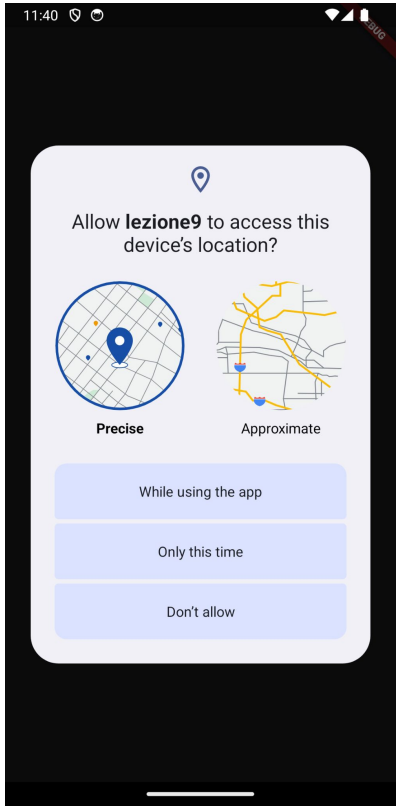
# Integrazione geolocator



# Integrazione geolocator



# Integrazione geolocator




```dart
class _LoadingScreenState extends State<LoadingScreen> {
  void getLocation() async {
    bool serviceEnabled;
    LocationPermission permission;

    // Test if location services are enabled.
    serviceEnabled = await Geolocator.isLocationServiceEnabled();
    if (!serviceEnabled) {
      // Location services are not enabled don't continue
      // accessing the position and request users of the
      // App to enable the location services.
      return Future.error('Location services are disabled.');
    }

    permission = await Geolocator.checkPermission();
    if (permission == LocationPermission.denied) {
      permission = await Geolocator.requestPermission();
      if (permission == LocationPermission.denied) {
        // Permissions are denied, next time you could try
        // requesting permissions again (this is also where
        // Android's shouldShowRequestPermissionRationale
        // returned true. According to Android guidelines
        // your App should show an explanatory UI now.
        return Future.error('Location permissions are denied');
      }
    }
```

```
BLEMS 3   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                Filter (e.g. text, !e
FlutterGeolocator(14792): Binding to location service.
FlutterGeolocator(14792): Geolocator foreground service connected
FlutterGeolocator(14792): Initializing Geolocator services
FlutterGeolocator(14792): Flutter engine connected. Connected engine count 1
nnecting to VM Service at ws://127.0.0.1:51225/Q30981C7VAw=/ws
xample.lezione9(14792): Verification of void androidx.collection.SimpleArrayMap.freeArrays(int[], java.lang.Object[], int
.75 bytecodes/s) (3832B approximate peak alloc)
CompatibilityChangeReporter(14792): Compat change id reported: 78294732; UID 10200; state: ENABLED
EGL_emulation(14792): app_time_stats: avg=1201.39ms min=1.83ms max=16561.71ms count=20
flutter (14792): Latitude: 37.4220936, Longitude: -122.083922
EGL_emulation(14792): app_time_stats: avg=1458.67ms min=1458.67ms max=1458.67ms count=1
flutter (14792): Latitude: 37.4220936, Longitude: -122.083922
```

# Programmazione asincrona

# Programmazione Sincrona vs. Asincrona

**Programmazione Sincrona**

- Le operazioni vengono eseguite una dopo l’altra, in ordine sequenziale;
- Un’operazione deve completarsi prima che un'altra inizi;
- Può causare il blocco dell’interfaccia utente o dei processi, specialmente con operazioni lunghe.

**Programmazione Asincrona**

- Le operazioni possono essere eseguite contemporaneamente senza dover attendere il completamento di quella precedente;
- Le operazioni asincrone non bloccano il thread principale e consentono all’interfaccia utente di rimanere reattiva;
- Sono utili per gestire operazioni che richiedono tempo, come chiamate di rete o accesso a database.

Esempi

- **Programmazione Sincrona**: Caricamento sequenziale di dati da un file.
- **Programmazione Asincrona**: Caricamento di immagini da Internet mentre l’utente continua a interagire con l'app.

# Esecuzione sequenziale

testsync.dart > ...

```dart
import 'dart:io';

Run | Debug | Profile
void main() {
  performTasks();
}

void performTasks() {
  task1();
  task2();
  task3();
}

void task1() {
  String result = 'task 1 data';
  print('Task 1 complete');
}

void task2() {
  String result = 'task 2 data';
  print('Task 2 complete');
}

void task3() {
  String result = 'task 3 data';
  print('Task 3 complete');
}
```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter (e.g. text, !exclu... Dart (Pixel_3a_API

```
Restarted application in 2.137ms.
I/flutter (21379): Task 1 complete
I/flutter (21379): Task 2 complete
I/flutter (21379): Task 3 complete
```

# Esecuzione sequenziale

testsync.dart > task2

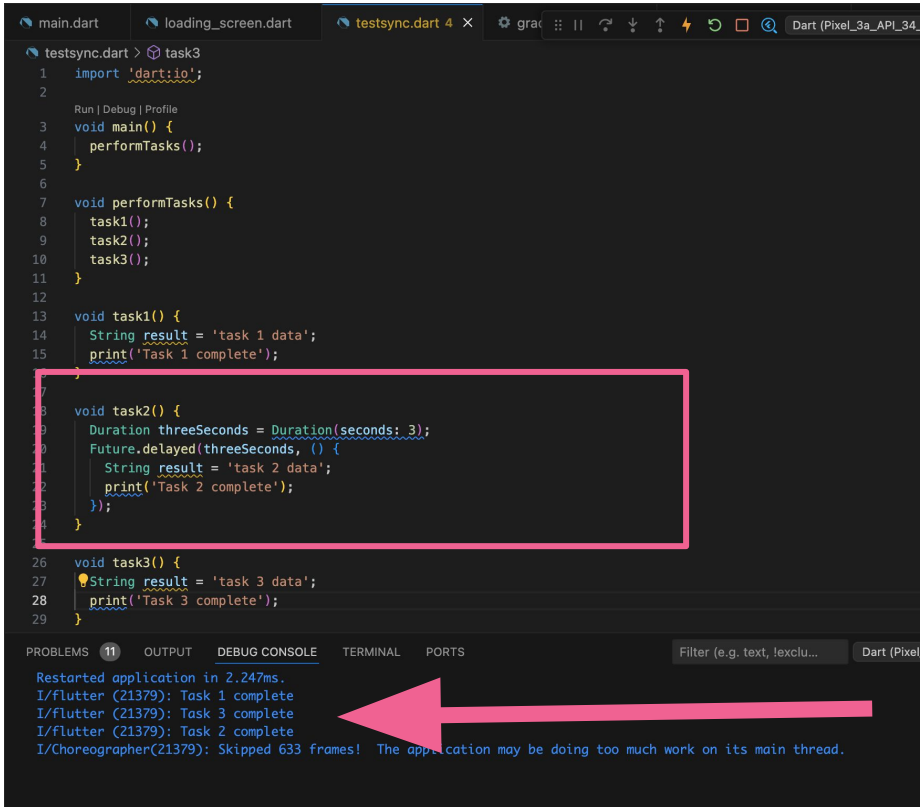
```dart
import 'dart:io';

Run | Debug | Profile
void main() {
  performTasks();
}

void performTasks() {
  task1();
  task2();
  task3();
}

void task1() {
  String result = 'task 1 data';
  print('Task 1 complete');
}

void task2() {
  Duration threeSeconds = Duration(seconds: 3);
  sleep(threeSeconds);
  String result = 'task 2 data';
  print('Task 2 complete');
}

void task3() {
  String result = 'task 3 data';
  print('Task 3 complete');
}
```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter (e.g. t

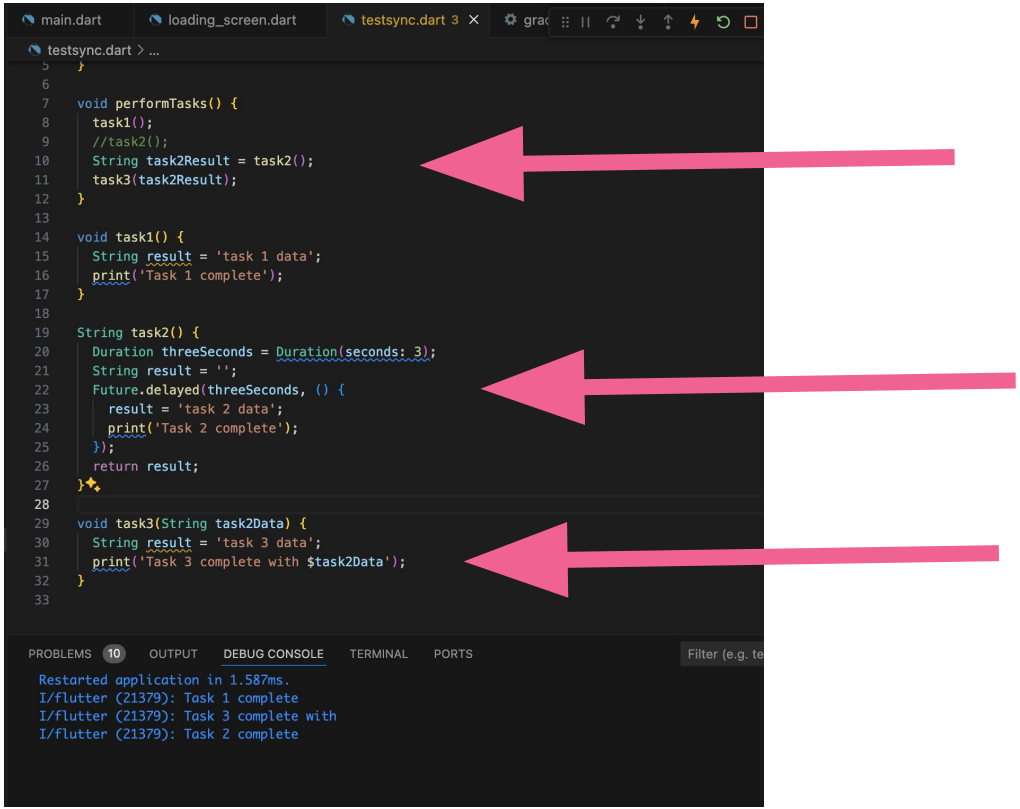
```
Restarted application in 1.372ms.
I/flutter (21379): Task 1 complete
I/flutter (21379): Task 2 complete
I/flutter (21379): Task 3 complete
```

Si ferma per 3 secondi

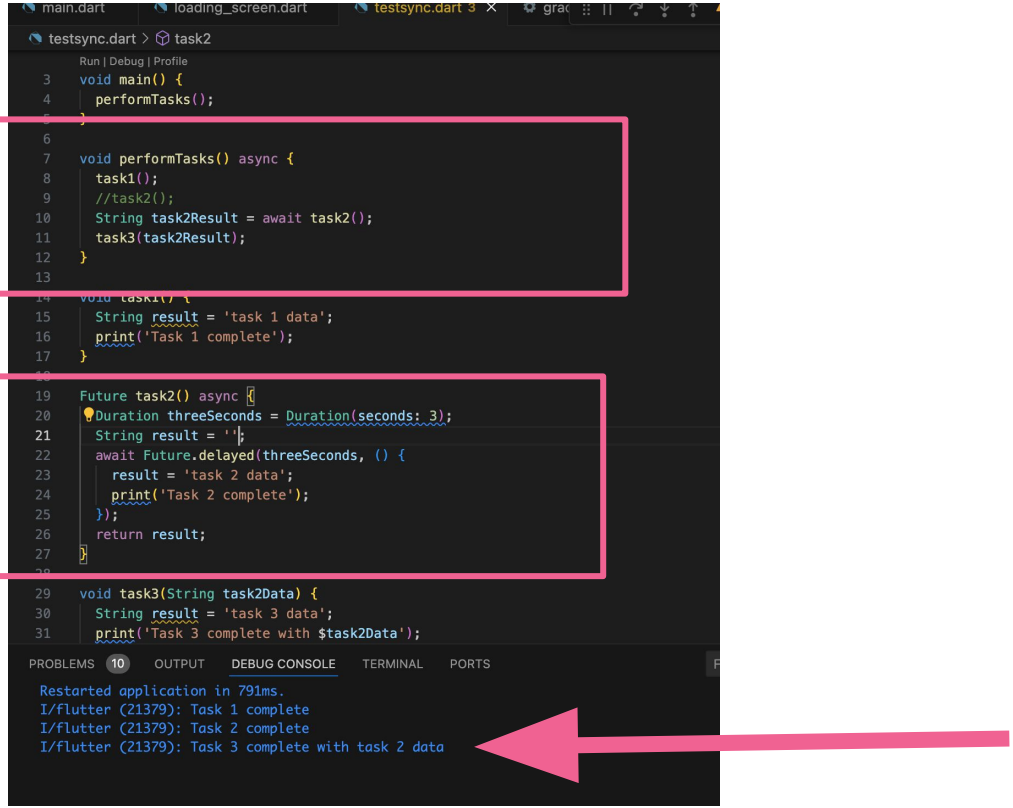
# Delay asincrono



# Delay asincrono



# Delay asincrono



# Gestione dell’Asincronia con Future in Flutter

**Cos'è un Future?**

- Un Future rappresenta un valore che potrebbe non essere disponibile immediatamente, ma in futuro;
- È utilizzato per eseguire operazioni asincrone come il recupero di dati da una rete o l’elaborazione di un’operazione costosa.

**Come funziona un Future?**

- Un Future è una promessa di un valore che sarà disponibile in un certo punto nel tempo;
- È possibile associare un Future a un’operazione e proseguire con altre attività mentre si aspetta il completamento di quell’operazione.

**Stati di un Future:**

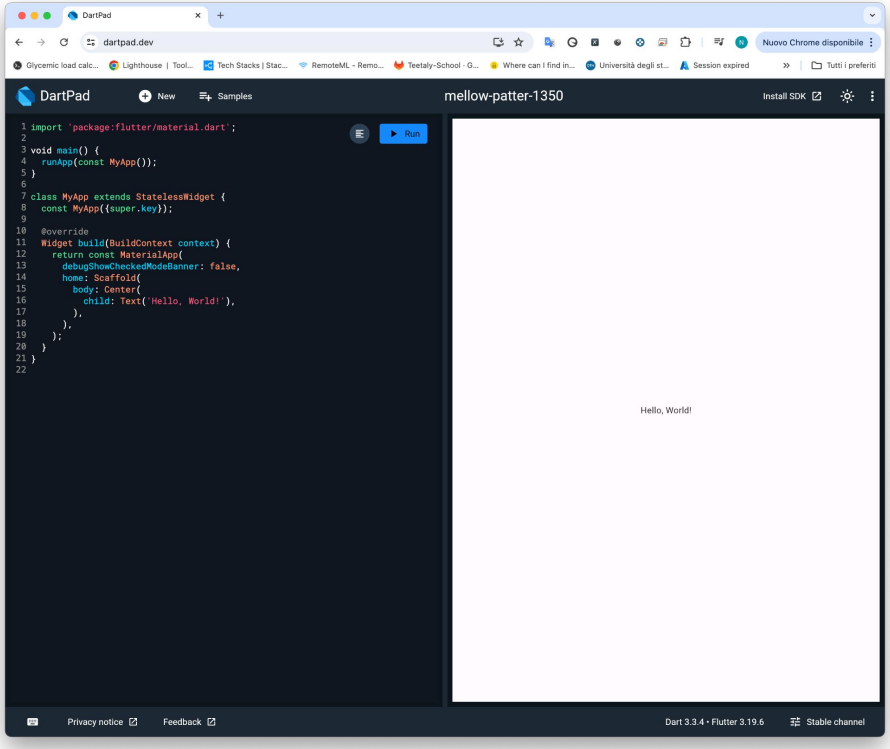
- **Pending**: Il Future è in attesa di completamento;
- **Completed**: Il Future ha restituito un valore;
- **Error**: Il Future ha restituito un errore.

# Esercizio 1

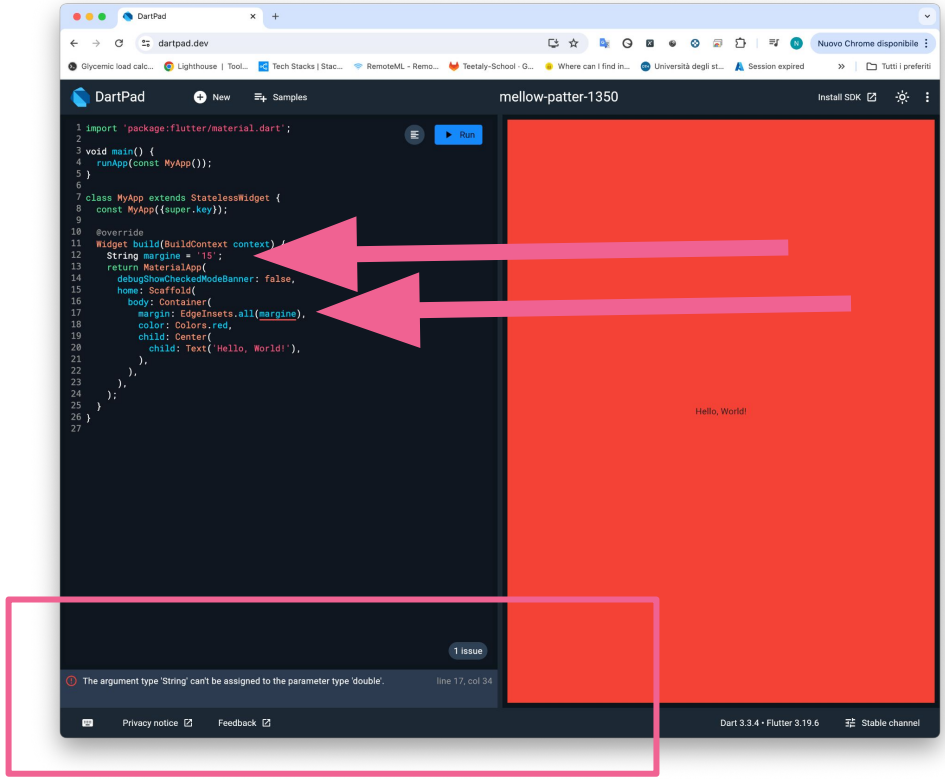
Partendo dal codice disponibile nel file esercizio1.zip, fate in modo di stampare la Location su console in maniera automatica, senza fare tap sul pulsante

# Gestione delle eccezioni

# Gestire le eccezioni a Runtime

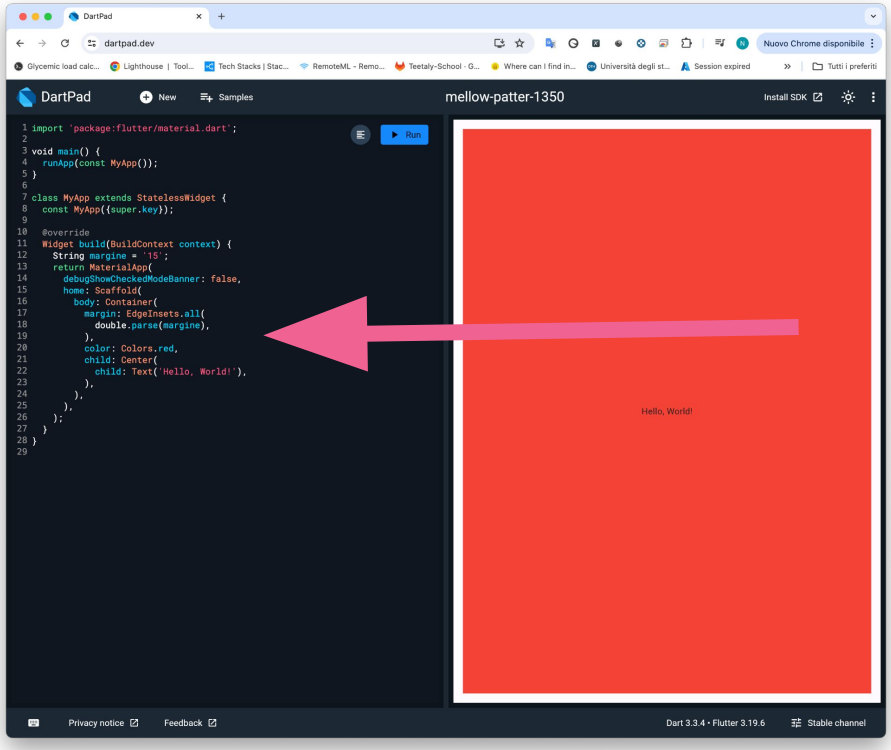


# Gestire le eccezioni a Runtime



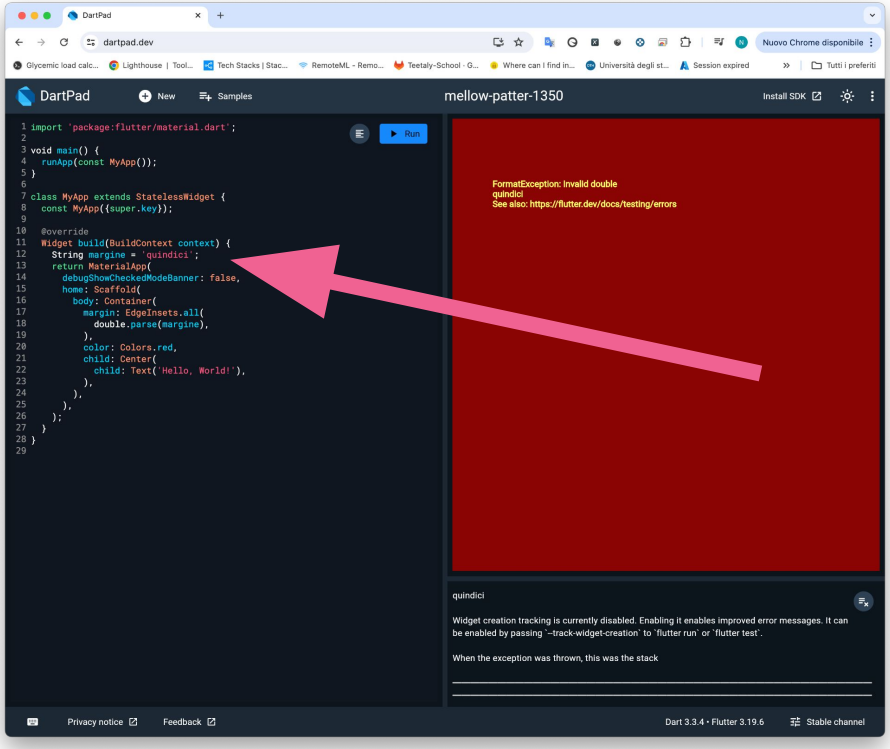

Finché è possibile, il Framework prova a non farmi sbagliare…

# Gestire le eccezioni a Runtime



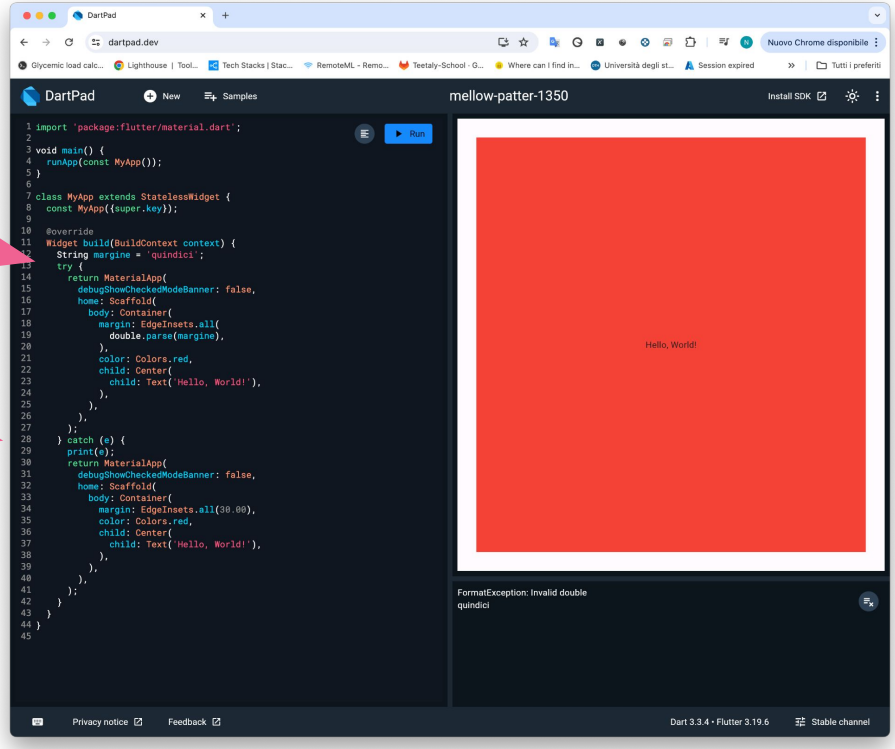

Ma io sono più testardo e fixo con grande maestria!

# Gestire le eccezioni a Runtime



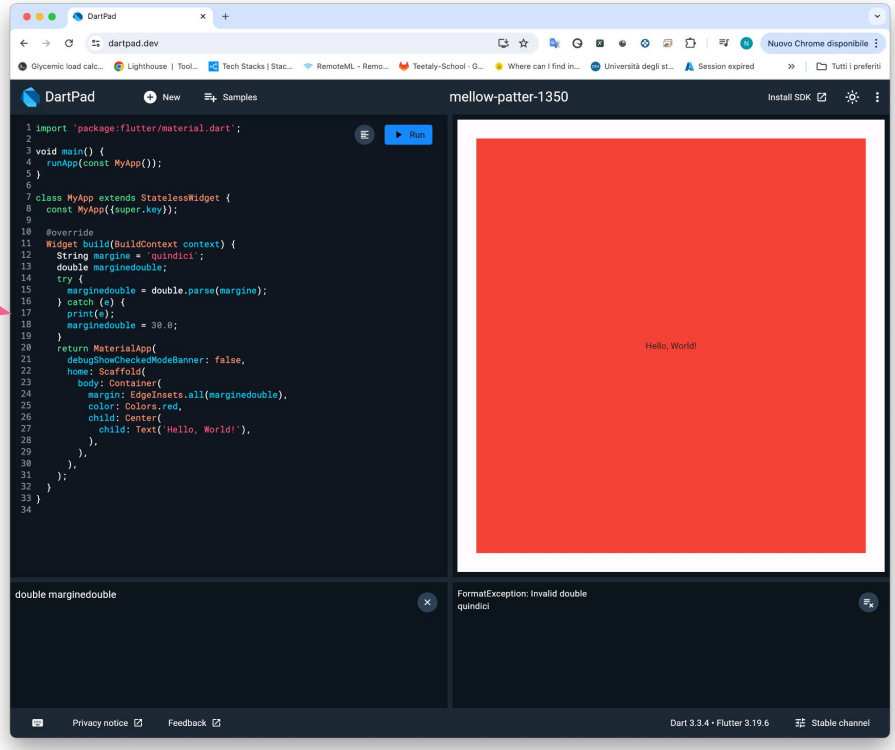

L'Applicazione va in crash. L'idea è quella di evitare per quanto possibile l'esperienza del crash all'utente e poter gestire l'eccezione

# Gestire le eccezioni a Runtime




Gestiamo l'eccezione in tutto il blocco…

# Gestire le eccezioni a Runtime




Oppure lo facciamo in maniera smart.

La gestione delle eccezioni integrata anche con un buon tracker di errori ci aiuta a migliorare nel tempo la nostra applicazione.

# Operatore di null-awareness (Null Aware Operator)

```
unaVariabile ?? valoreDiDefault
```

# Tornando al nostro progetto…

```
    }

    if (permission == LocationPermission.deniedForever) {
      // Permissions are denied forever, handle appropriately.
      return Future.error(
          'Location permissions are permanently denied, we cannot request permissions.');
    }
    try{
          Position position = await Geolocator.getCurrentPosition(
          desiredAccuracy: LocationAccuracy.low);
    } catch (e) {
      print(e);
    }
  }
```

# Organizzazione del codice: funzionalità distaccate dai widget

# Esercizio 2

Partendo sempre dal codice disponibile nel file esercizio1.zip, provate a spostare il codice per ottenere la posizione gps in una classe separata. Richiamate l'oggetto creato con la classe e stampate latitudine e longitudine.

# Creo una nuova classe

```dart
import 'package:geolocator/geolocator.dart';

class Location {
  double? latitude;
  double? longitude;

  void getCurrentLocation() async {
    bool serviceEnabled;
    LocationPermission permission;
    serviceEnabled = await Geolocator.isLocationServiceEnabled();
    if (!serviceEnabled) {
      return Future.error('Location services are disabled.');
    }

    permission = await Geolocator.checkPermission();
    if (permission == LocationPermission.denied) {
      permission = await Geolocator.requestPermission();
      if (permission == LocationPermission.denied) {
        return Future.error('Location permissions are denied');
      }
    }
    if (permission == LocationPermission.deniedForever) {
      return Future.error(
          'Location permissions are permanently denied, we cannot request permissions.');
    }
    try {
      Position position = await Geolocator.getCurrentPosition(
          desiredAccuracy: LocationAccuracy.low);
      latitude = position.latitude;
      longitude = position.longitude;
    } catch (e) {
      print(e);
    }
  }
}
```

# La richiamo dal widget della schermata

Perché?

loading_screen.dart 1

lib > screen > loading_screen.dart > _LoadingScreenState > getLocation

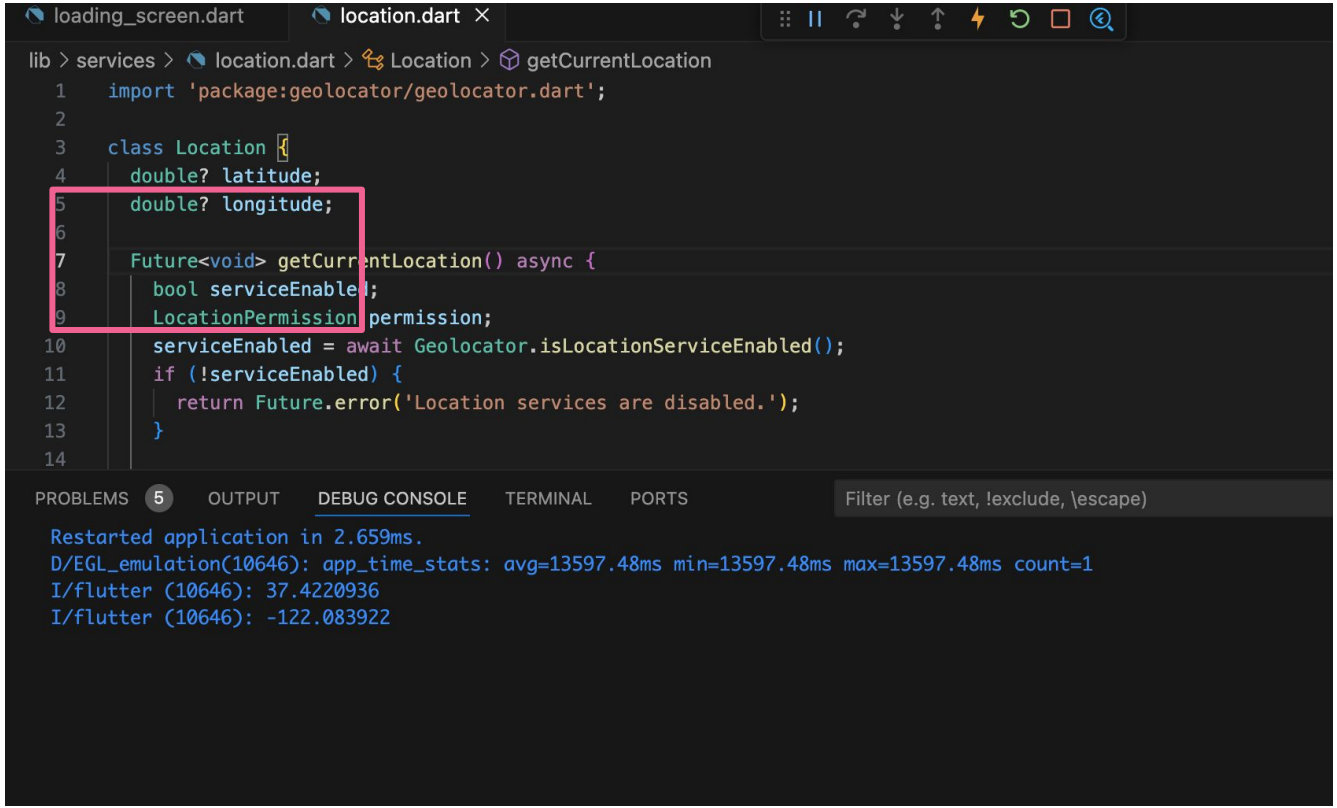
```dart
import 'package:flutter/material.dart';
import 'package:lezione9/services/location.dart';

class LoadingScreen extends StatefulWidget {
  const LoadingScreen({super.key});

  @override
  State<LoadingScreen> createState() => _LoadingScreenState();
}

class _LoadingScreenState extends State<LoadingScreen> {
  @override
  void initState() {
    super.initState();
    getLocation();
  }

  void getLocation() async {
    Location location = Location();
    await location.getCurrentLocation();
    print(location.latitude);
    print(location.longitude);
  }
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Filter (e.g. text, !exclude, \escape)

```
Restarted application in 1.217ms.
D/EGL_emulation(10646): app_time_stats: avg=1439.39ms min=16.93ms max=8426.61ms count=6
I/flutter (10646): 37.4220936
I/flutter (10646): -122.083922
lib/screen/loading_screen.dart:20:20: Error: This expression has type 'void' and can't be used.
    await location.getCurrentLocation();
                   ^
```

# Future <void>


loading_screen.dart
location.dart
lib > services > location.dart > Location > getCurrentLocation
1 import 'package:geolocator/geolocator.dart';
2
3 class Location {
4 double? latitude;
5 double? longitude;
6
7 Future<void> getCurrentLocation() async {
8 bool serviceEnabled;
9 LocationPermission permission;
10 serviceEnabled = await Geolocator.isLocationServiceEnabled();
11 if (!serviceEnabled) {
12 return Future.error('Location services are disabled.');
13 }
14
PROBLEMS 5
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, !exclude, \escape)
Restarted application in 2.659ms.
D/EGL_emulation(10646): app_time_stats: avg=13597.48ms min=13597.48ms max=13597.48ms count=1
I/flutter (10646): 37.4220936
I/flutter (10646): -122.083922