

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

The screenshot shows a Google Developer Groups event page. At the top, there's a navigation bar with links like 'Glycemic load calc...', 'Lighthouse | Tool...', 'Tech Stacks | Stac...', 'RemoteML - Remo...', 'Testaly-School - G...', 'Where can I find in...', 'Università degli st...', 'Session expired', and 'Tutti i preferiti'. Below the navigation, it says 'Hello, Nicola!' and 'Google Developer Groups'. There are tabs for 'Check-in Attendees', 'View Attendees', 'Edit Event', and 'Dashboard'. A large blue banner features a stylized diagram of a mobile application interface with arrows and text. Below the banner, the event title is displayed: **Flutter e Gemini, IA multipiattaforma**. The text below the title includes: 'UNIMOL - Dipartimento di Bioscienze e Territorio, Viale Dell'università, Provincia di Isernia, 86090', 'GDG Campobasso', and 'Siamo entusiasti di annunciare il prossimo evento del Google Developer Group di Campobasso, presso l'Università degli Studi del Molise, sede di Pesche, Dipartimento di Bioscienze e Territorio.' It also states 'Your ticket gives you access to in-person event venues.' At the bottom, there are social media sharing icons for Facebook, Twitter, LinkedIn, and Email. The footer shows the date and time: 'May 17, 3:00 – 5:00 PM' and '11 RSVP'd'. A section titled 'Key Themes' lists 'AI', 'Build with AI', and 'Flutter'. Another section titled 'About this event' contains a short description: 'Il GDG Campobasso ti invita a unirti a noi per una giornata di scoperte e apprendimento per un talk dal titolo "Flutter e Gemini, IA multipiattaforma". Avremo il privilegio di accogliere Carlo Lucera, Google Developer Expert per Flutter e Dart, che ci guiderà'.



Evento a Campobasso 17 Maggio ore 18

The screenshot shows a web browser window for the Google Developer Groups event page. The title of the event is "Flutter zero to Hero". Below the title, it says "Dimensione - Internet in Fibra, FTTH, FWA e FTTC, 176 Viale Giuseppe Ferro, Provincia di Campobasso, 86100". The description reads: "Ci spostiamo a Campobasso nello scenario futuristico dell'Innovation Hub di Dimensione per un'immersione completa nel mondo di Flutter. Ancora una volta, Carlo Lucera ci guiderà attraverso un percorso che trasformerà i partecipanti da principianti a esperti di Flutter." It also mentions that the ticket gives access to in-person event venues. At the bottom, it shows the date "May 17, 6:00 PM – 8:00 PM" and "14 RSVP'd".



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

The screenshot shows the Flutter Cookbook page at <https://docs.flutter.dev/cookbook>. The page has a dark blue header with the Flutter logo and navigation links for Multi-Platform, Development, Ecosystem, Showcase, Docs, and Get started. The main content area is titled "Cookbook" and contains a brief introduction: "This cookbook contains recipes that demonstrate how to solve common problems while writing Flutter apps. Each recipe is self-contained and can be used as a reference to help you build up an application." Below the introduction, there are several sections with bullet-pointed lists of recipes:

- Animation**:
 - Animate a page route transition
 - Animate a widget using a physics simulation
 - Animate the properties of a container
 - Fade a widget in and out
- Design**:
 - Add a drawer to a screen
 - Display a snackbar
 - Export fonts from a package
 - Update the UI based on orientation
 - Use a custom font
 - Use themes to share colors and font styles
 - Work with tabs
- Effects**:
 - Create a download button
 - Create a nested navigation flow
 - Create a photo filter carousel
 - Create a scrolling parallax effect
 - Create a shimmer loading effect
 - Create a staggered menu animation
 - Create a typing indicator
 - Create an expandable FAB
 - Create gradient chat bubbles
 - Drag a UI element
- Forms**

On the right side of the page, there is a sidebar with a tree view of the cookbook categories and a list of links corresponding to each category:

- Get started
- Stay up to date
- Codelabs & samples
 - Code labs
 - Cookbook**
 - Samples and demos
- App solutions
- User interface
 - Introduction
 - Widget catalog
- Layout
- Design & theming
- Interactivity
- Assets & media
- Navigation & routing
- Animations & transitions
- Accessibility & internationalization
- Beyond UI
- Data & backend
- Platform integration
- Packages & plugins
- Testing & debugging
- Performance & optimization
- Deployment

- Contents
 - Animation
 - Design
 - Effects
 - Forms
 - Games
 - Gestures
 - Images
 - Lists
 - Maintenance
 - Navigation
 - Networking
 - Persistence
 - Plugins
 - Testing
 - Integration
 - Unit
 - Widget

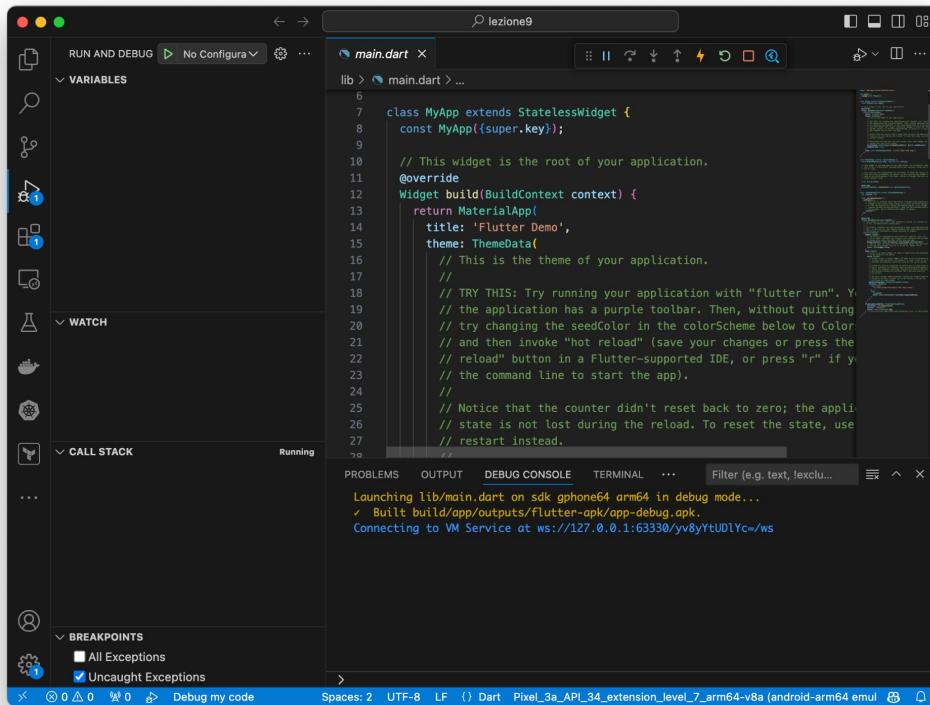
Themes Cookbook -

<https://docs.flutter.dev/cookbook/design/themes>

The screenshot shows a web browser displaying the Flutter Themes Cookbook. The URL in the address bar is <https://docs.flutter.dev/cookbook/design/themes>. The page has a dark blue header with the Flutter logo and navigation links for Multi-Platform, Development, Ecosystem, Showcase, Docs, and Get started. A banner at the top right says "Flutter is back at Google I/O on May 14! Register now". The main content area has a light gray background. On the left is a sidebar with a vertical navigation menu. The menu items include: Get started, Stay up to date, Codelabs & samples, App solutions, User interface (with sub-links: Introduction, Widget catalog, Layout), Design & theming (with sub-links: Share styles with themes, Material design, Migrate to Material 3, Text, Custom graphics, Interactivity, Assets & media, Navigation & routing, Animations & transitions, Accessibility & internationalization, Beyond UI, Data & backend, Platform integration, Packages & plugins, Testing & debugging). The main content area starts with the heading "Use themes to share colors and font styles" and includes a "Note" section, a "Create an app theme" section, and a code editor at the bottom showing a snippet of Dart code for a `MaterialApp` constructor.

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

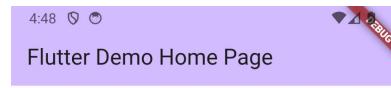


The screenshot shows the Flutter IDE interface. On the left, there's a sidebar with various icons for RUN AND DEBUG, VARIABLES, WATCH, CALL STACK, PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and BREAKPOINTS. The main area displays the code for main.dart:

```
lib > main.dart > ...
6
7 class MyApp extends StatelessWidget {
8   const MyApp({super.key});
9
10 // This widget is the root of your application.
11 @override
12 Widget build(BuildContext context) {
13   return MaterialApp(
14     title: 'Flutter Demo',
15     theme: ThemeData(
16       // This is the theme of your application.
17       //
18       // TRY THIS: Try running your application with "flutter run". You
19       // will see that the application has a purple toolbar. Then, without quitting
20       // the application, try changing the seedColor in the colorScheme below to Color
21       // .purple and then invoke "hot reload" (save your changes or press the
22       // "reload" button in a Flutter-supported IDE, or press "r" if you're
23       // running from the command line to start the app).
24       //
25       // Notice that the counter didn't reset back to zero; the application
26       // state is not lost during the reload. To reset the state, use
27       // restart instead.
28       //
29     ),
30   );
31 }
```

The DEBUG CONSOLE tab shows the output of the build process:

```
Launching lib/main.dart on sdk gphone64 arm64 in debug mode...
✓ Built build/app/outputs/flutter-apk/app-debug.apk.
Connecting to VM Service at ws://127.0.0.1:63330/yv8YtUDlYc=/ws
```

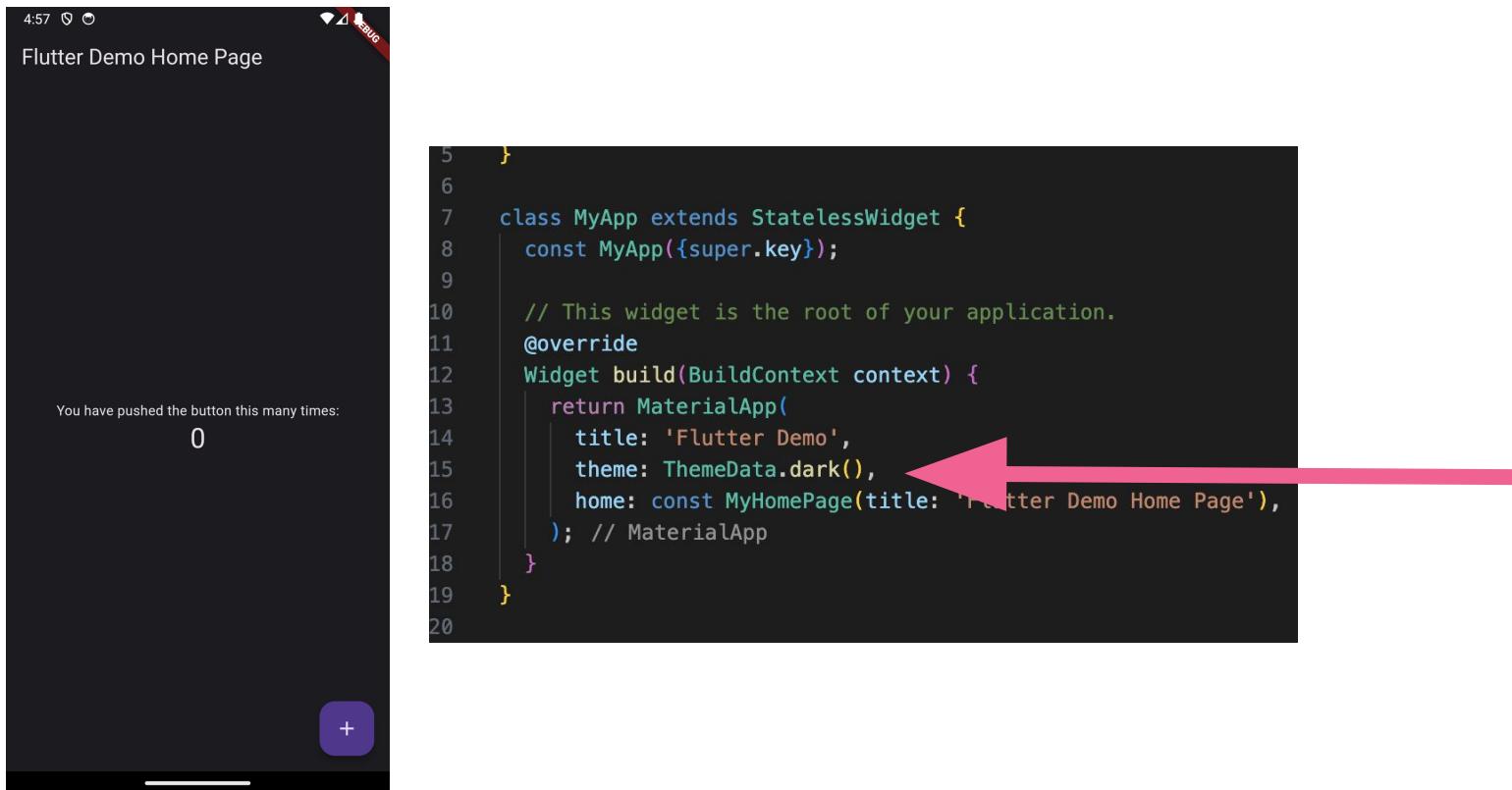


You have pushed the button this many times:

0

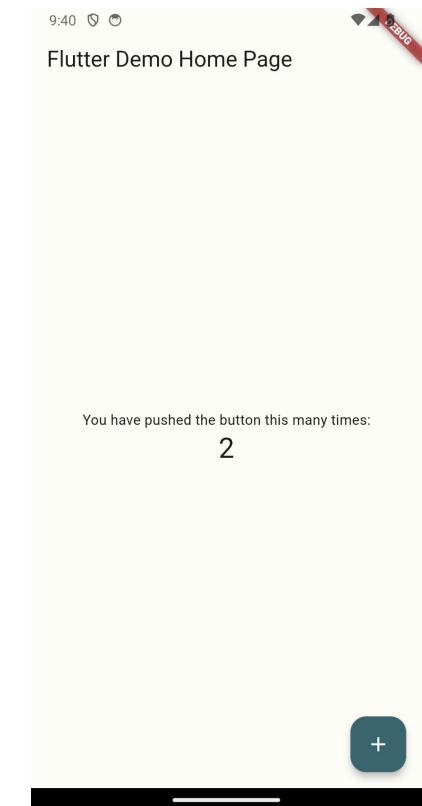
Progetto di default

Theme.dark()



Theme - Impostazioni manuali

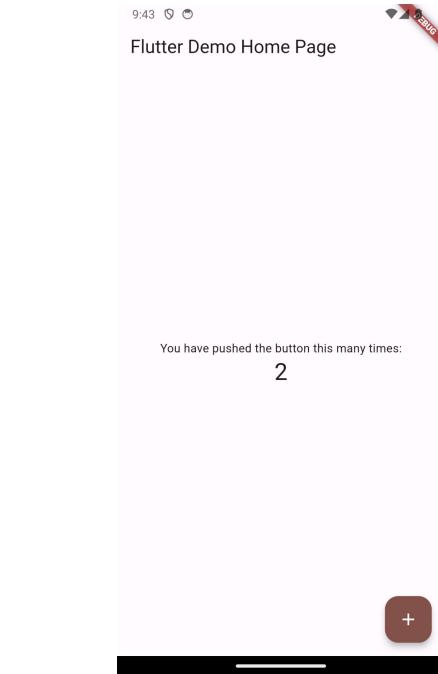
```
8  const MyApp({super.key});  
9  
10 // This widget is the root of your application.  
11 @override  
12 Widget build(BuildContext context) {  
13   final ColorScheme colorScheme = ColorScheme.fromSeed(  
14     brightness: MediaQuery.platformBrightnessOf(context),  
15     seedColor: Color.fromARGB(255, 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 }
```



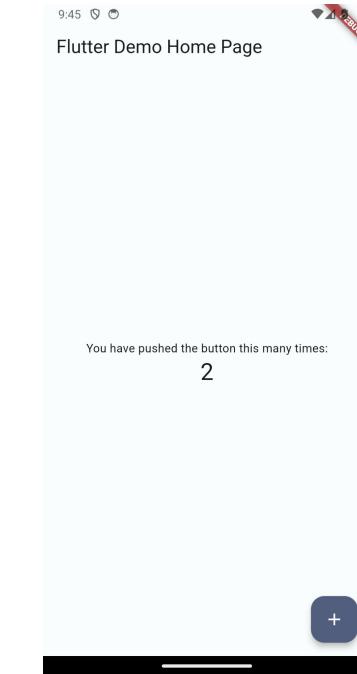
Theme - Impostazioni manuali



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



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

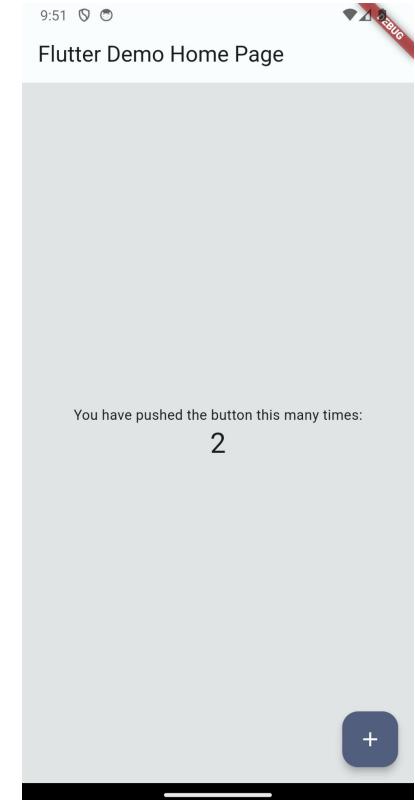
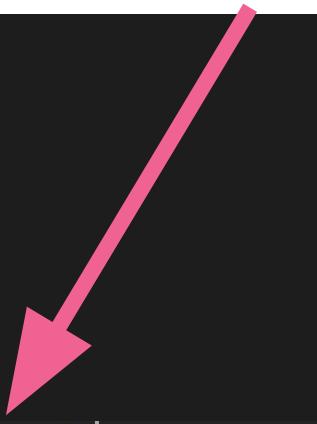


```
seedColor: Colors.grey,
```

Theme - Impostazioni manuali

Leggermente
più scuro

```
// 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;

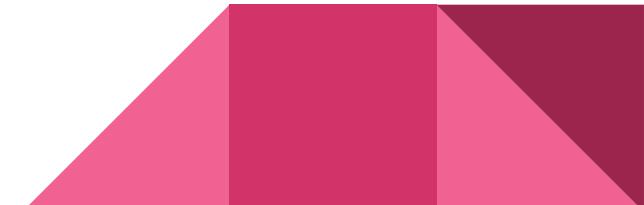
Migliora l'esperienza dell'utente in condizioni di scarsa illuminazione e riduce l'affaticamento visivo;

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

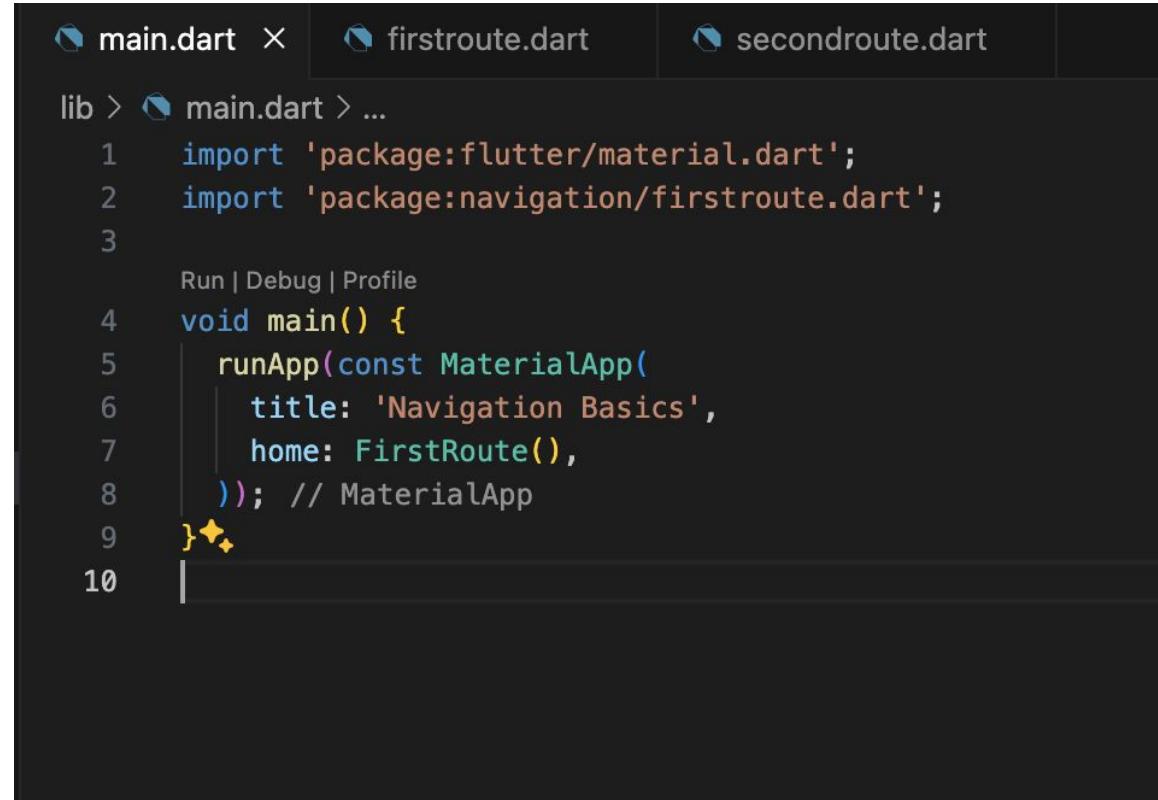
Attraverso l'uso dei temi, utilizzando ThemeData, è possibile implementare in maniera selettiva l'uso delle due modalità in base alle impostazioni dell'utente o al tema di sistema del dispositivo.



Navigazione

Navigazione semplice tra due schermate

Un semplice
main che
chiama il
Widget
FirstRoute()

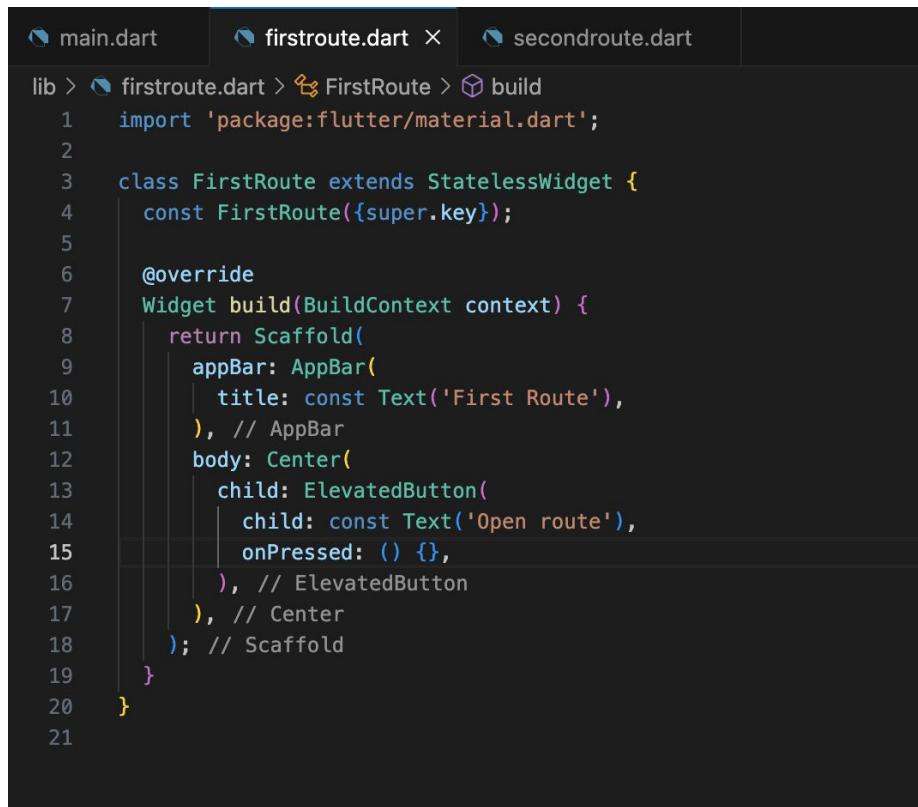


The image shows a code editor interface with three tabs at the top: 'main.dart' (selected), 'firstroute.dart', and 'secondroute.dart'. Below the tabs, the code for 'main.dart' is displayed:

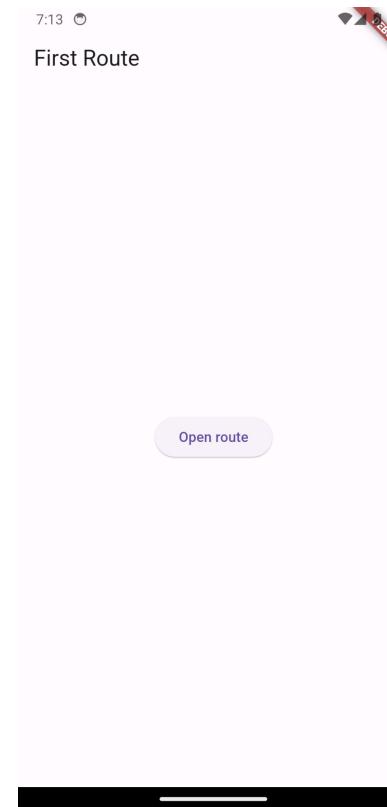
```
lib > main.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package:navigation/firstroute.dart';
3
4 void main() {
5     runApp(const MaterialApp(
6         title: 'Navigation Basics',
7         home: FirstRoute(),
8     )); // MaterialApp
9 }
10 |
```

The code imports the necessary packages and defines the main function which runs an instance of the MaterialApp class, setting the title to 'Navigation Basics' and the home page to FirstRoute().

Navigazione semplice tra due schermate



```
lib > firstroute.dart > FirstRoute > build
1 import 'package:flutter/material.dart';
2
3 class FirstRoute extends StatelessWidget {
4   const FirstRoute({super.key});
5
6   @override
7   Widget build(BuildContext context) {
8     return Scaffold(
9       appBar: AppBar(
10         title: const Text('First Route'),
11       ), // AppBar
12       body: Center(
13         child: ElevatedButton(
14           child: const Text('Open route'),
15           onPressed: () {},
16         ), // ElevatedButton
17       ), // Center
18     ); // Scaffold
19   }
20 }
```



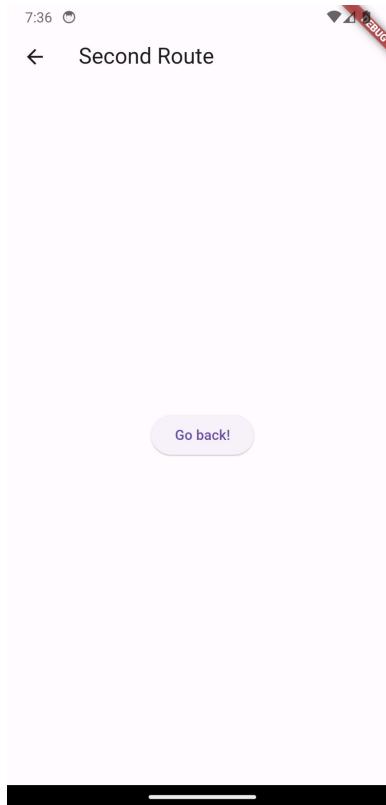
Creo un Widget SecondRoute() e voglio fare in modo di spostarmi una volta premuto "Open route"

Navigazione semplice tra due schermate

```
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?

```
lib > secondroute.dart > ...
1   import 'package:flutter/material.dart';
2
3   class SecondRoute extends StatelessWidget {
4     const SecondRoute({super.key});
5
6     @override
7     Widget build(BuildContext context) {
8       return Scaffold(
9         appBar: AppBar(
10            title: const Text('Second Route'),
11        ), // AppBar
12        body: Center(
13          child: ElevatedButton(
14            onPressed: () {
15              Navigator.pop(context);
16            },
17            child: const Text('Go back!'),
18          ), // ElevatedButton
19        ), // Center
20      ); // Scaffold
21    }
22  }
23  ✦
24 // Path: lib/firstroute.dart
```

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.

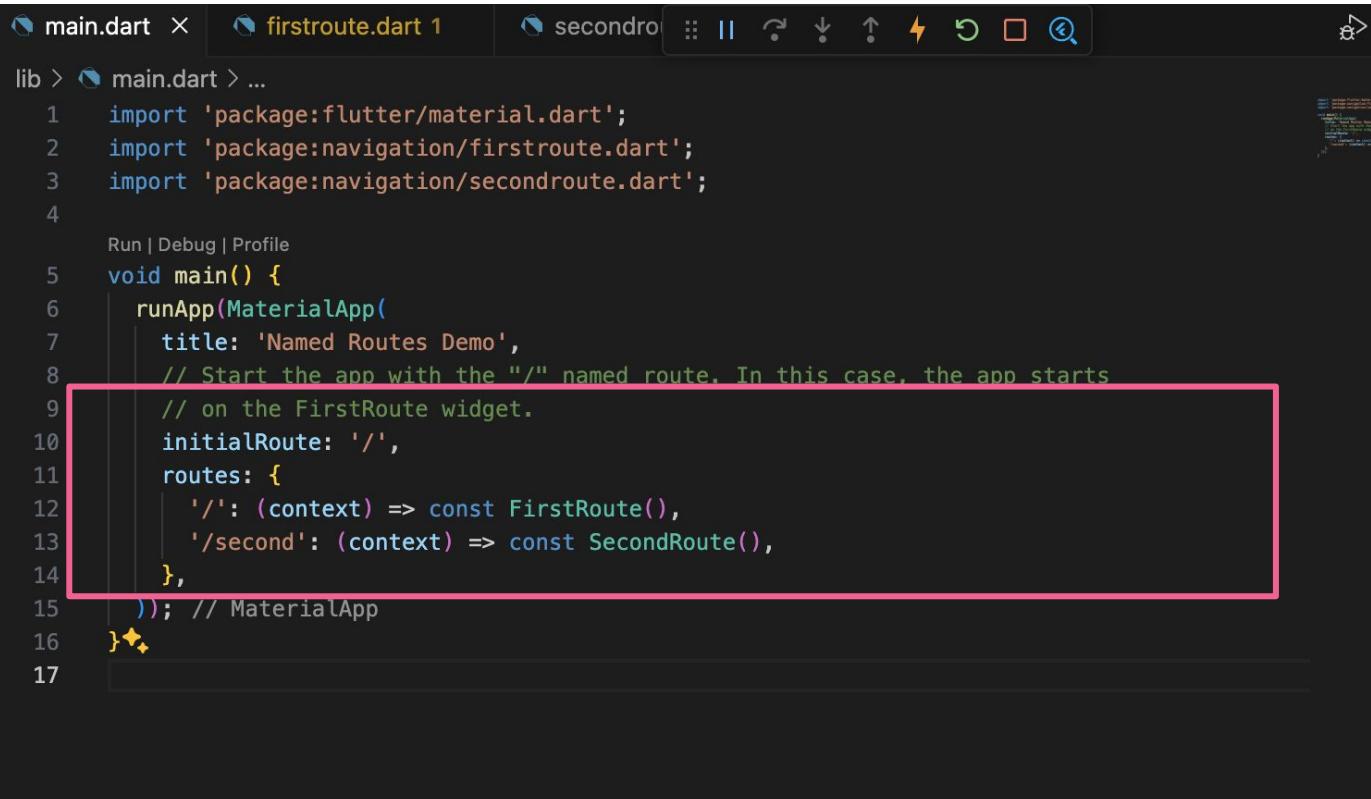
Push e Passaggio di Context in MaterialPageRoute:

- 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



The screenshot shows a code editor with three tabs: main.dart, firstroute.dart 1, and secondroute.dart. The main.dart tab is active, displaying the following code:

```
lib > main.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package:navigation/firstroute.dart';
3 import 'package:navigation/secondroute.dart';
4
5 void main() {
6   runApp(MaterialApp(
7     title: 'Named Routes Demo',
8     // Start the app with the "/" named route. In this case, the app starts
9     // on the FirstRoute widget.
10    initialRoute: '/',
11    routes: {
12      '/': (context) => const FirstRoute(),
13      '/second': (context) => const SecondRoute(),
14    },
15  )); // MaterialApp
16 }
17
```

A pink rectangular box highlights the routes section of the code, specifically the part where routes are defined with keys ('/' and '/second') and values (FirstRoute and SecondRoute constructors).

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 > ...
1 import 'package:flutter/material.dart';
2 import 'package:navigation/secondroute.dart';
3
4 class FirstRoute extends StatelessWidget {
5   const FirstRoute({super.key});
6
7   @override
8   Widget build(BuildContext context) {
9     return Scaffold(
10       appBar: AppBar(
11         title: const Text('First Route'),
12       ), // AppBar
13       body: Center(
14         child: ElevatedButton(
15           child: const Text('Open route'),
16           onPressed: () {
17             Navigator.pushNamed(context, '/second');
18           },
19         ), // ElevatedButton
20       ), // Center
21     ); // Scaffold
22   }
23 }
```

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.

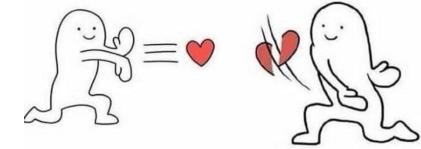
Ecco come funziona l'implementazione dei deep link in Flutter:

- **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 StatefulWidget SecondRoute

```
lib > secondroute.dart > _SecondRouteState > build
1 import 'package:flutter/material.dart';
2
3 class SecondRoute extends StatefulWidget {
4   const SecondRoute({super.key});
5
6   @override
7   State<SecondRoute> createState() => _SecondRouteState();
8 }
9
10 class _SecondRouteState extends State<SecondRoute> {
11   @override
12   void initState() {
13     super.initState();
14     print('SecondRoute initState');
15   }
16
17   @override
18   void deactivate() {
19     super.deactivate();
20     print('SecondRoute deactivate');
21   }
22
23   @override
24   Widget build(BuildContext context) {
25     print('SecondRoute build');
26     return Scaffold(
27       appBar: AppBar(
28         title: const Text('Second Route'),
29       ), // AppBar
30       body: Center(
31         child: ElevatedButton(
32           onPressed: () {
33             Navigator.pop(context);
34           },
35           child: const Text('Go back!'),
36         ), // ElevatedButton
37       ), // Center
38     ); // Scaffold
39   }
40 }
```

Log degli stati

10:30

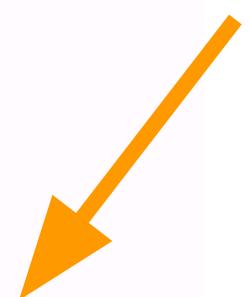
First Route

Second route

Zero route

10:30

← Second Route



```
IS 7 DEBUG CONSOLE ... Filter (e.g. text, !exclude)
*ed application in 3.419ms
emulation( 7373): app_time_stats: avg=861246.69ms min=861246.69ms max=861246.69ms count=1
ter ( 7373): SecondRoute initState
ter ( 7373): SecondRoute build
lowOnBackDispatcher( 7373): OnBackInvokedCallback is not enabled for the application.
lowOnBackDispatcher( 7373): Set 'android:enableOnBackInvokedCallback="true"' in the application man
emulation( 7373): app_time_stats: avg=13649.84ms min=13649.84ms max=13649.84ms count=1
emulation( 7373): app_time_stats: avg=2800.04ms min=19.52ms max=3325.73ms count=3
ter ( 7373): SecondRoute deactivate
```

Go back!



Creiamo un'App Meteo

Accesso al GPS

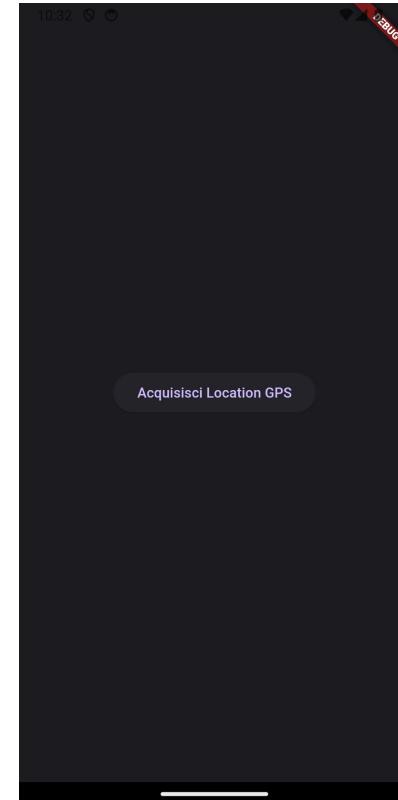
Progetto di partenza

```
lib > main.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package:lezione9/screen/loading_screen.dart';
3
4 void main() => runApp(MyApp());
5
6 class MyApp extends StatelessWidget {
7   @override
8   Widget build(BuildContext context) {
9     return MaterialApp(
10       theme: ThemeData.dark(),
11       home: const LoadingScreen(),
12     ); // MaterialApp
13   }
14 }
15
16 ━━━━
```



Progetto di partenza

```
lib > screen > loading_screen.dart > ...
1   import 'package:flutter/material.dart';
2
3   class LoadingScreen extends StatefulWidget {
4     const LoadingScreen({super.key});
5
6     @override
7     State<LoadingScreen> createState() => _LoadingScreenState();
8   }
9
10  class _LoadingScreenState extends State<LoadingScreen> {
11    @override
12    Widget build(BuildContext context) {
13      return Scaffold(
14        body: Center(
15          child: ElevatedButton(
16            onPressed: () {
17              //Get the current location
18            },
19            child: const Text('Acquisisci Location GPS'),
20          ), // ElevatedButton
21        ), // Center
22      ); // Scaffold
23    }
24  }◆
25 }
```



geolocator - <https://pub.dev/packages/geolocator/>

The screenshot shows the pub.dev package page for the geolocator plugin. At the top, it displays the package name "geolocator 11.0.0" with a "Flutter Favorite" badge. Below the title, it shows the publication date "Published 2 months ago" and the publisher "baseflow.com". The package is marked as "Dart 3 compatible". The "Flutter" tab is selected, showing compatibility with FLUTTER, PLATFORM, ANDROID, IOS, MACOS, WEB, and WINDOWS. It has 5.1K stars and 5159 likes. The "Publisher" section lists "baseflow.com". The "Metadata" section describes the plugin as a geolocation plugin for Flutter, providing cross-platform (IOS, Android) API for generic location (GPS etc.) functions. The "Repository" section links to GitHub, issues, and contributing. The "Documentation" section links to the API reference. The "License" section links to the MIT license. The "Dependencies" section lists flutter, geolocator_android, geolocator_apk, geolocator_platform_interface, geolocator_web, and geolocator_windows. The "Usage" section provides instructions for adding the package to a Flutter application. The bottom of the page includes a "More" link.

geolocator 11.0.0

Published 2 months ago • [baseflow.com](#) (Dart 3 compatible)

Flutter

5.1K

5159 LIKES | 140 PUB POINTS | 100% POPULARITY

Publisher
baseflow.com

Metadata

Geolocation plugin for Flutter. This plugin provides a cross-platform (IOS, Android) API for generic location (GPS etc.) functions.

Repository (GitHub)
[View/report issues](#)
[Contributing](#)

Documentation
[API reference](#)

License
[MIT \(LICENSE\)](#)

Dependencies

flutter,
geolocator_android,
geolocator_apk,
geolocator_platform_interface,
geolocator_web,
geolocator_windows

Usage

To add the geolocator to your Flutter application read the [install](#) instructions. Below are some Android and

More

Integrazione geolocator

```
lib > screen > loading_screen.dart > _LoadingScreenState > build
1 import 'package:flutter/material.dart';
2 import 'package:geolocator/geolocator.dart';
3
4 class LoadingScreen extends StatefulWidget {
5   const LoadingScreen({super.key});
6
7   @override
8   State<LoadingScreen> createState() => _LoadingScreenState();
9 }
10
11 class _LoadingScreenState extends State<LoadingScreen> {
12   void getLocation() {
13     Position position = await Geolocator.getCurrentPosition(desiredAccuracy: LocationAccuracy.high);
14   }
15   @override
16   Widget build(BuildContext context) {
17     return Scaffold(
18       body: Center(
19         child: ElevatedButton(
20           onPressed: () {
21             //Get the current location
22           },
23           child: const Text('Acquisisci Location GPS'),
24         ), // ElevatedButton
25       ), // Center
26     );
27   }
28 }
```



The await expression can only be used in an async function.
Try marking the function body with either 'async' or 'async*'. dart(await_in_wrong_context)

Type: Position
View Problem (F8) Quick Fix... (⌘.)

Integrazione geolocator

```
class _LoadingScreenState extends State<LoadingScreen> {
    void getLocation() async {
        Position position = await Geolocator.getCurrentPosition(desiredAccuracy: LocationAccuracy.low);
    }
}
@Override
Widget build(BuildContext context) {
    return Scaffold(
        body: Center(
            child: ElevatedButton(
                onPressed: () {
                    //Get the current location
                },
                child: const Text('Acquisisci Location GPS'),
            ), // ElevatedButton
    ),
}
```

- best LocationAccuracy
- bestForNavigation LocationAccuracy
- high LocationAccuracy
- low LocationAccuracy**
- lowest LocationAccuracy
- medium LocationAccuracy
- reduced LocationAccuracy
- values List<LocationAccuracy>

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...

```
7  @override
8  State<LoadingScreen> createState() => _LoadingScreenState();
9 }
10
11 class _LoadingScreenState extends State<LoadingScreen> {
12   void getLocation() async {
13     Position position = await Geolocator.getCurrentPosition(
```

Exception has occurred. ×

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

```
14   desiredAccuracy: LocationAccuracy.low);
15   print(position);
16 }
17
18 @override
19 Widget build(BuildContext context) {
20   return Scaffold(
21     body: Center(
22       child: ElevatedButton(
23         onPressed: getLocation,
24         child: const Text('Acquisisci Location GPS'),
```

Abbiamo un errore! Cerchiamo nella documentazione del package...

Integrazione geolocator

The screenshot shows a Chrome browser window with the title "geolocator | Flutter package". The URL bar shows "pub.dev/packages/geolocator". The page content is a migration guide for upgrading from version 5.0.0 to 1.12. It includes sections for "Upgrade pre 1.12 Android projects", "AndroidX", "Permissions", and a note about background location updates.

Upgrade pre 1.12 Android projects

Since version 5.0.0 this plugin is implemented using the Flutter 1.12 Android plugin APIs. Unfortunately this means App developers also need to migrate their Apps to support the new Android infrastructure. You can do so by following the [Upgrading pre 1.12 Android projects](#) migration guide. Failing to do so might result in unexpected behaviour.

AndroidX

The geolocator plugin requires the AndroidX version of the Android Support Libraries. This means you need to make sure your Android project supports AndroidX. Detailed instructions can be found [here](#).

The TL;DR version is:

1. Add the following to your "gradle.properties" file:

```
android.useAndroidX=true  
android.enableJetifier=true
```
2. Make sure you set the `compileSdkVersion` in your "android/app/build.gradle" file to 33:

```
android {  
    compileSdkVersion 33  
    ...  
}
```
3. Make sure you replace all the `android.` dependencies to their AndroidX counterparts (a full list can be found here: [Migrating to AndroidX](#)).

Permissions

On Android you'll need to add either the `ACCESS_COARSE_LOCATION` or the `ACCESS_FINE_LOCATION` permission to your Android Manifest. To do so open the `AndroidManifest.xml` file (located under `android/app/src/main`) and add one of the following two lines as direct children of the `<manifest>` tag (when you configure both permissions the `ACCESS_FINE_LOCATION` will be used by the geolocator plugin):

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />  
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Starting from Android 10 you need to add the `ACCESS_BACKGROUND_LOCATION` permission (next to the `ACCESS_COARSE_LOCATION` or the `ACCESS_FINE_LOCATION` permission) if you want to continue receiving updates even when your App is running in the background (note that the geolocator plugin doesn't support receiving and processing location updates while running in the background):

```
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
```

NOTE: Specifying the `ACCESS_COARSE_LOCATION` permission results in location updates

Integrazione geolocator

The screenshot shows a web browser window displaying the documentation for the `geolocator` Flutter package on pub.dev. The page title is "geolocator | Flutter package". The main content area is titled "Example" and contains a code snippet demonstrating how to acquire the current position of a device, including handling service enablement and permission requests.

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

/// Determine the current position of the device.
///
/// When the location services are not enabled or permissions
/// are denied, the future will return an error.
Future<Position> determinePosition() 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 settings screen is launched
      // returned true. According to Android guidelines
      // your App should show an explanatory UI now.
      return Future.error('Location permissions are denied');
    }
  }

  if (permission == LocationPermission.deniedForever) {
    // Permissions are denied forever, handle appropriately.
    return Future.error(
      'Location permissions are permanently denied, we cannot request permissions.'
    )
  }

  // When we reach here, permissions are granted and we can
  // continue accessing the position of the device.
  return await Geolocator.getCurrentPosition();
}
```

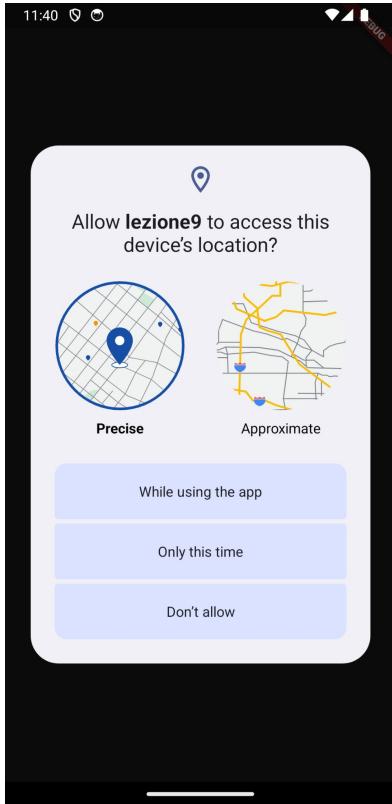
Below the example code, there is a section titled "API" which includes "Geolocation" and "Current location". A note under "Current location" states: "To query the current location of the device simply make a call to the `getCurrentPosition` method. You can finetune the results by specifying the following parameters:"

Integrazione geolocator

The screenshot shows a code editor interface with several tabs at the top: main.dart, loading_screen.dart, gradle.properties, AndroidManifest.xml (which is currently selected), location_screen.dart, and city_scr. The left sidebar is an Explorer view showing the project structure under 'LEZIONE9'. The 'AndroidManifest.xml' file is open in the main editor area, displaying XML code for an Android application manifest. The manifest includes declarations for location permissions (ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION, ACCESS_BACKGROUND_LOCATION) and an activity named '.MainActivity' with specific launch mode and theme settings.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
    <application
        android:label="lezione9"
        android:name="${applicationName}"
        android:icon="@mipmap/ic_launcher">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:launchMode="singleTop"
            android:theme="@style/LaunchTheme"
            android:configChanges="orientation|keyboardHidden|keyboard|screenSize|smallestScreenSize|locale|layoutDirection|fontScale|density|ActionBar|color"
            android:hardwareAccelerated="true"
            android:windowSoftInputMode="adjustResize">
            <!-- Specifies an Android theme to apply to this Activity as soon as
                the Android system creates its window. This theme need not be the same
                as the application's global theme -->
        </activity>
    </application>
</manifest>
```

Integrazione geolocator



```
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');
      }
    }

    if (permission == LocationPermission.deniedForever) {
```

BLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter (e.g. text, le

```
flutter_geolocator(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
Connecting to VM Service at ws://127.0.0.1:51225/Q30981C7VAw=/ws
kample.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=1
Flutter (14792): Latitude: 37.4220936, Longitude: -122.08392
EGL_emulation(14792): app_time_stats: avg=1458.67ms min=1458.67ms max=1458.67ms count=1
flutter (14792): Latitude: 37.4220936, Longitude: -122.08392
```

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

The screenshot shows a Dart code editor with a dark theme. A file named `testsync.dart` is open, containing the following code:

```
testsync.dart
1 import 'dart:io';
2
3 void main() {
4   performTasks();
5 }
6
7 void performTasks() {
8   task1();
9   task2();
10  task3();
11 }
12
13 void task1() {
14   String result = 'task 1 data';
15   print('Task 1 complete');
16 }
17
18 void task2() {
19   String result = 'task 2 data';
20   print('Task 2 complete');
21 }
22
23 void task3() {
24   String result = 'task 3 data';
25   print('Task 3 complete');
26 }◆
27
```

The code defines a `main` function that calls `performTasks`. `performTasks` contains three tasks: `task1`, `task2`, and `task3`. Each task prints its result and a completion message to the console.

Below the code editor, there is a terminal window showing the output of the application:

```
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

The screenshot shows a code editor with a Dart file named `testsync.dart`. The code defines three tasks: `task1`, `task2`, and `task3`. The `task2` function is highlighted with a pink rectangle. The `task1` function prints 'Task 1 complete'. The `task2` function prints 'Task 2 complete' after a 3-second delay. The `task3` function prints 'Task 3 complete'. The `DEBUG CONSOLE` tab is selected at the bottom, showing the output of the application's execution.

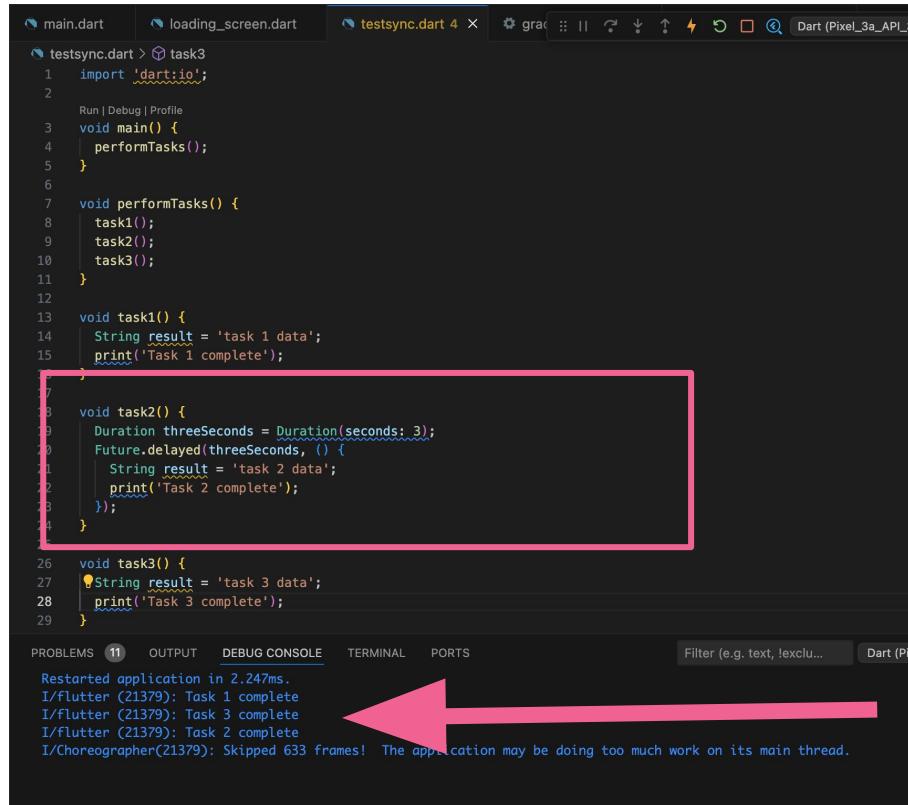
```
testsync.dart > task2
1 import 'dart:io';
2
3 void main() {
4   performTasks();
5 }
6
7 void performTasks() {
8   task1();
9   task2();
10  task3();
11 }
12
13 void task1() {
14   String result = 'task 1 data';
15   print('Task 1 complete');
16 }
17
18 void task2() {
19   Duration threeSeconds = Duration(seconds: 3);
20   sleep(threeSeconds);
21   String result = 'task 2 data';
22   print('Task 2 complete');
23 }
24
25 void task3() {
26   String result = 'task 3 data';
27   print('Task 3 complete');
28 }
```

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

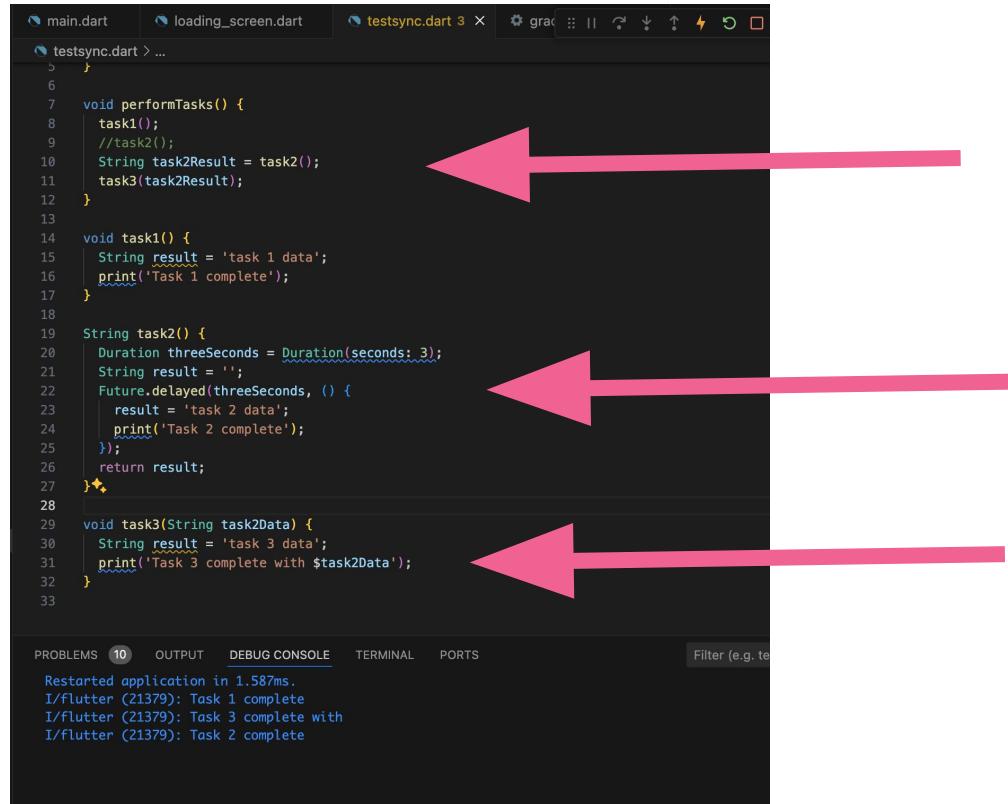


The screenshot shows a Dart code editor with several tabs open: main.dart, loading_screen.dart, testsync.dart (active), and Dart (Pixel_3a_API_34). The code in testsync.dart demonstrates three tasks: task1, task2, and task3. Task1 is synchronous. Task2 uses Future.delayed to delay its execution by three seconds. Task3 is also synchronous. A pink rectangle highlights the code for task2(). An arrow points from this highlighted area to the terminal output at the bottom, which shows the logs for each task's completion.

```
main.dart    loading_screen.dart    testsync.dart 4 X    Dart (Pixel_3a_API_34)
  testsync.dart > task3
  import 'dart:io';
  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);
  | Future.delayed(threeSeconds, () {
  | | String result = 'task 2 data';
  | | print('Task 2 complete');
  | });
  }
  void task3() {
  | String result = 'task 3 data';
  | print('Task 3 complete');
  }

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter (e.g. text, !exclu... Dart (Pixel_3a_API_34)
Restarted application in 2.247ms.
I/Flutter (21379): Task 1 complete
I/Flutter (21379): Task 3 complete
I/Flutter (21379): Task 2 complete
I/Choreographer(21379): Skipped 633 frames! The application may be doing too much work on its main thread.
```

Delay asincrono

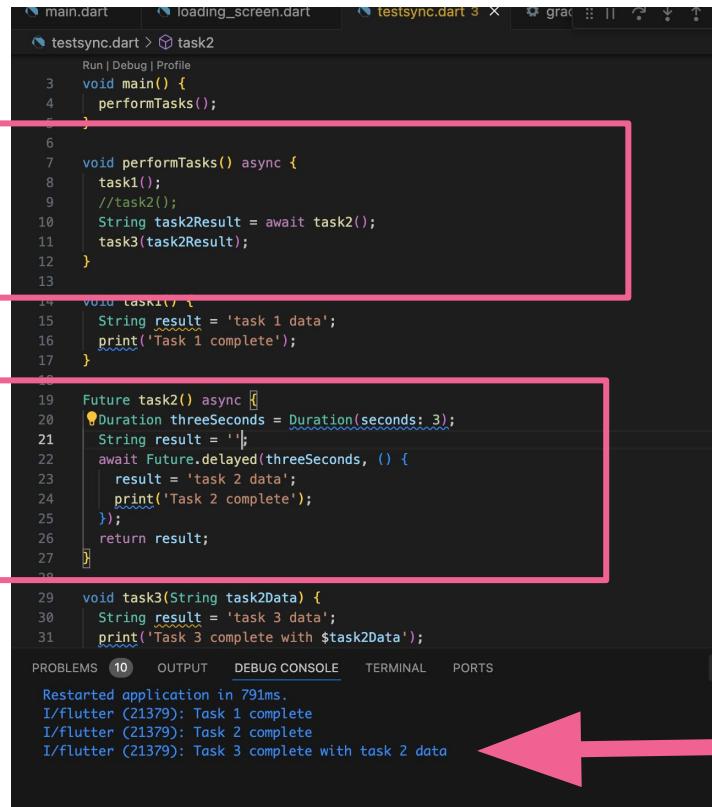


```
main.dart loading_screen.dart testsync.dart 3
5
6
7 void performTasks() {
8   task1();
9   //task2();
10  String task2Result = task2();
11  task3(task2Result);
12 }
13
14 void task1() {
15   String result = 'task 1 data';
16   print('Task 1 complete');
17 }
18
19 String task2() {
20   Duration threeSeconds = Duration(seconds: 3);
21   String result = '';
22   Future.delayed(threeSeconds, () {
23     result = 'task 2 data';
24     print('Task 2 complete');
25   });
26   return result;
27 }
28
29 void task3(String task2Data) {
30   String result = 'task 3 data';
31   print('Task 3 complete with $task2Data');
32 }
33
```

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

```
Restarted application in 1.587ms.
I/flutter (21379): Task 1 complete
I/flutter (21379): Task 3 complete with task 2 data
I/flutter (21379): Task 2 complete
```

Delay asincrono



```
main.dart loading_screen.dart testsync.dart 3 X graf :: || ? ⓘ
testsuite.dart > task2
Run | Debug | Profile
3 void main() {
4     performTasks();
5 }
6
7 void performTasks() async {
8     task1();
9     //task2();
10    String task2Result = await task2();
11    task3(task2Result);
12 }
13
14 void task1() {
15     String result = 'task 1 data';
16     print('Task 1 complete');
17 }
18
19 Future task2() async {
20     Duration threeSeconds = Duration(seconds: 3);
21     String result = '';
22     await Future.delayed(threeSeconds, () {
23         result = 'task 2 data';
24         print('Task 2 complete');
25     });
26     return result;
27 }
28
29 void task3(String task2Data) {
30     String result = 'task 3 data';
31     print('Task 3 complete with $task2Data');
```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Restarted application in 791ms.

I/flutter (21379): Task 1 complete

I/flutter (21379): Task 2 complete

I/flutter (21379): Task 3 complete with task 2 data

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.

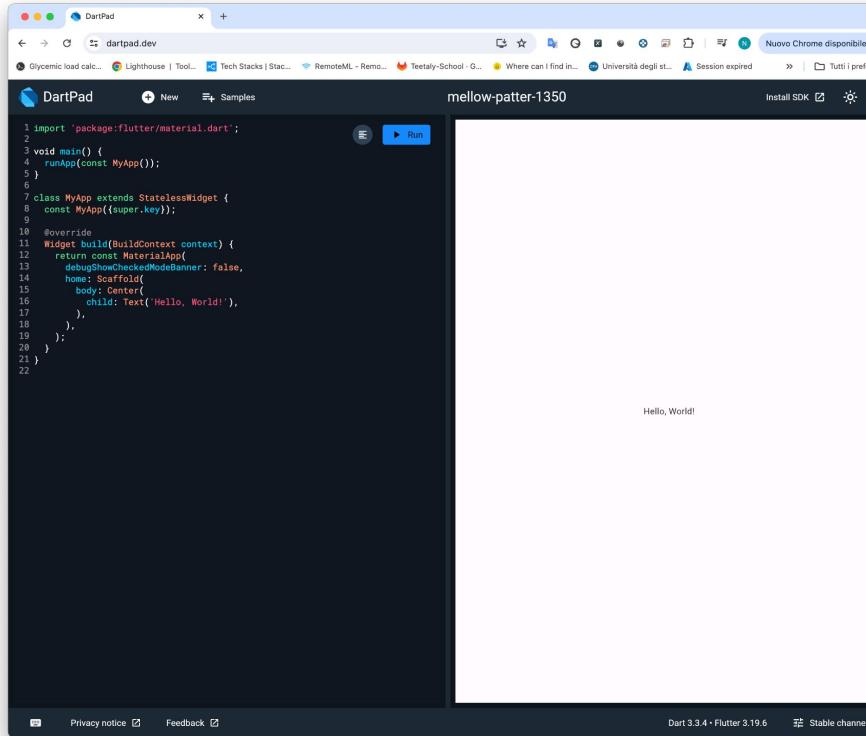
Esercizio 1

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

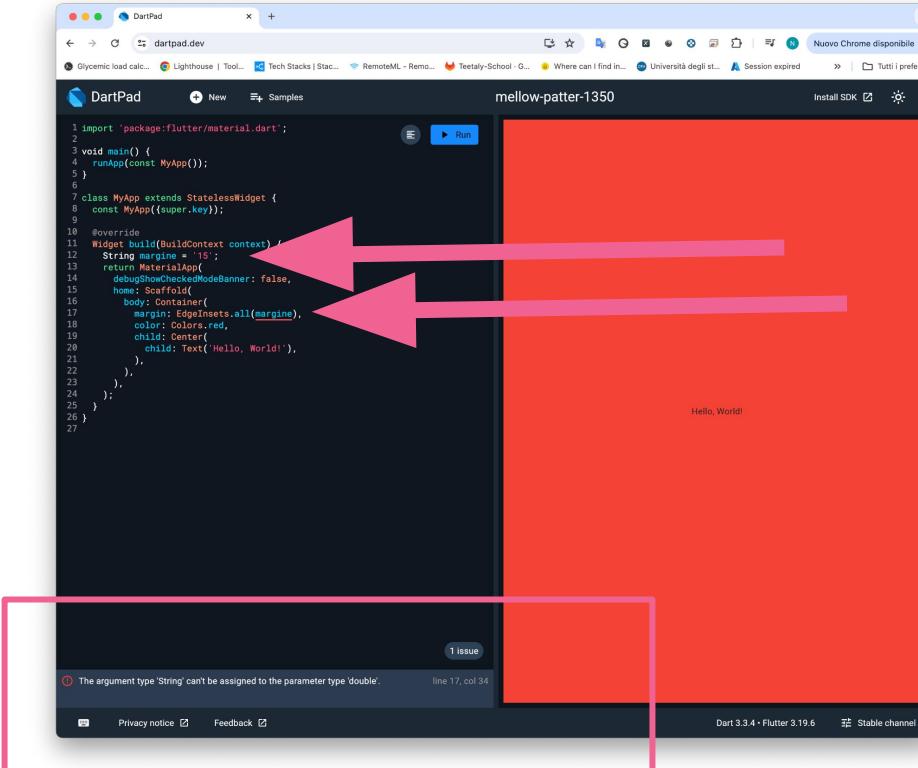
Gestione delle eccezioni



Gestire le eccezioni a Runtime

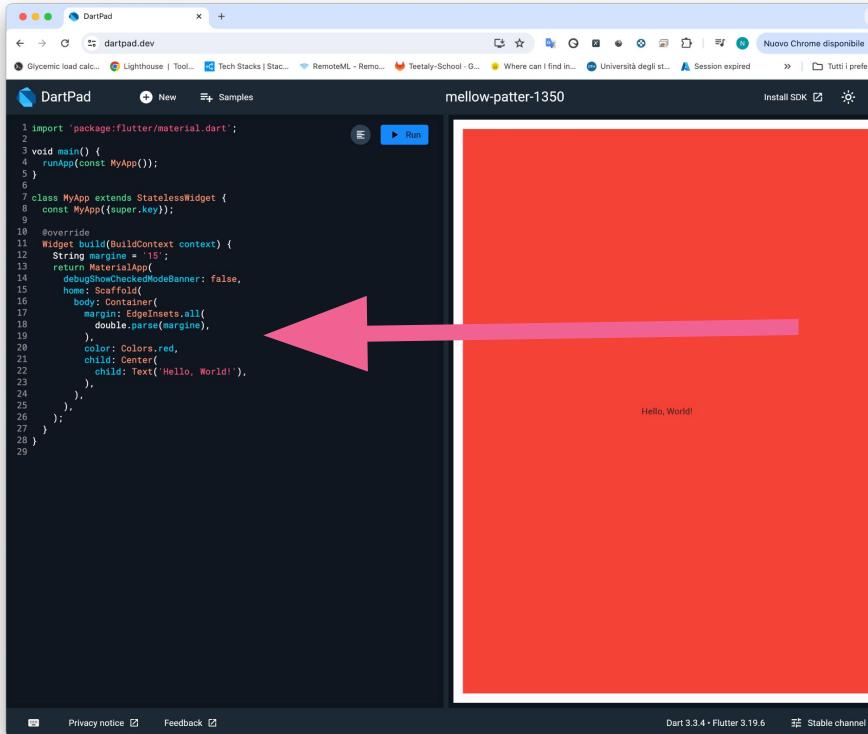


Gestire le eccezioni a Runtime



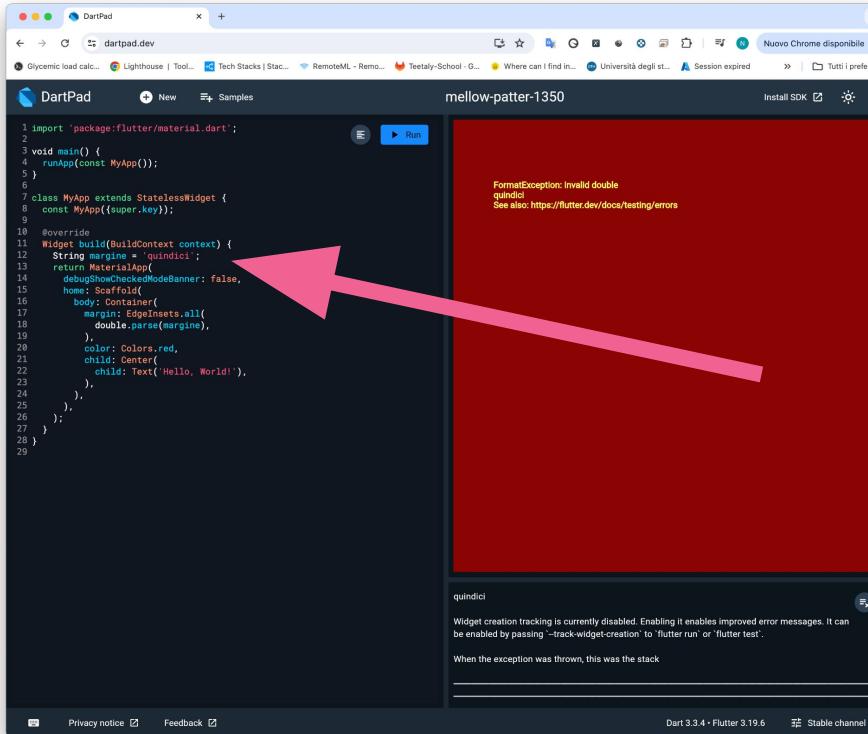
Finché è possibile, il Framework prova a non farmi sbagliare...

Gestire le eccezioni a Runtime



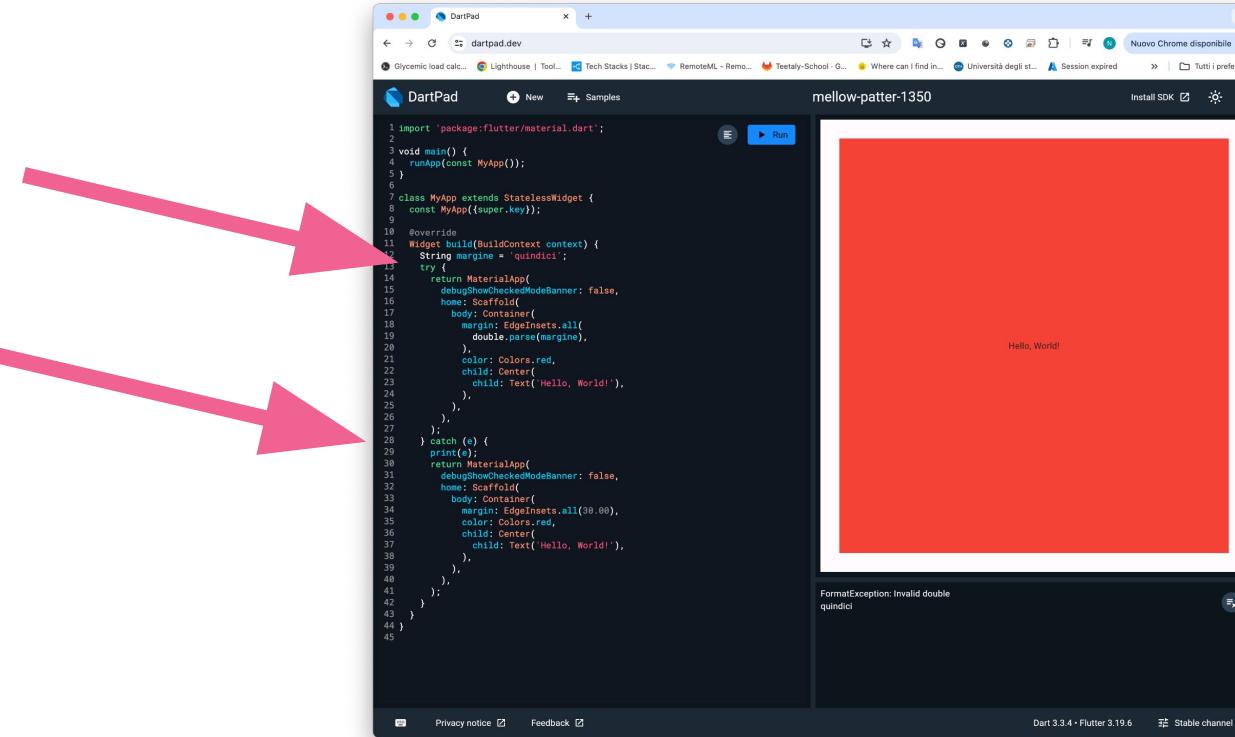
Ma io sono più
testardo e fixo
con grande
maestria!

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



The screenshot shows a DartPad session running on dartpad.dev. The code is a simple Flutter application. It defines a `MyApp` class that overrides the `build` method. Inside, it creates a `MaterialApp` with a red background and a central text widget. A `try` block attempts to parse the string `'quindici'` into a double. If successful, it prints the result and returns the `MaterialApp`. If there's a `FormatException`, it catches the error, prints its message, and returns a new `MaterialApp` with a margin of 30.0. The application runs successfully, displaying "Hello, World!" on a red screen. In the bottom right corner of the app's output area, the error message "FormatException: Invalid double quindici" is visible.

```
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(const MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8   const MyApp({super.key});
9
10 @override
11 Widget build(BuildContext context) {
12   String margine = 'quindici';
13   try {
14     return MaterialApp(
15       debugShowCheckedModeBanner: false,
16       home: Scaffold(
17         body: Container(
18           margin: EdgeInsets.all(
19             double.parse(margine),
20           ),
21           color: Colors.red,
22           child: Center(
23             child: Text('Hello, World!'),
24           ),
25         ),
26       ),
27     );
28   } catch (e) {
29     print(e);
30     return MaterialApp(
31       debugShowCheckedModeBanner: false,
32       home: Scaffold(
33         body: Container(
34           margin: EdgeInsets.all(30.0),
35           color: Colors.red,
36           child: Center(
37             child: Text('Hello, World!'),
38           ),
39         ),
40       ),
41     );
42   }
43 }
44 }
```

Gestiamo
l'eccezione in
tutto il blocco...

Gestire le eccezioni a Runtime



The screenshot shows a DartPad session running on dartpad.dev. The code is as follows:

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    String margin = 'quindici';
    double marginedouble;
    try {
      marginedouble = double.parse(margin);
    } catch (e) {
      print(e);
      marginedouble = 30.0;
    }
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        body: Container(
          margin: EdgeInsets.all(marginedouble),
          color: Colors.red,
          child: Center(
            child: Text('Hello, World!'),
          ),
        ),
      ),
    );
  }
}
```

A tooltip at the bottom left of the code editor says "double marginedouble". At the bottom right of the screen, there is an error message: "FormatException: Invalid double quindici".

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

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

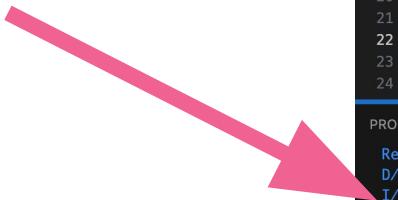
class Location {
    double? latitude;
    double? longitude;

    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?



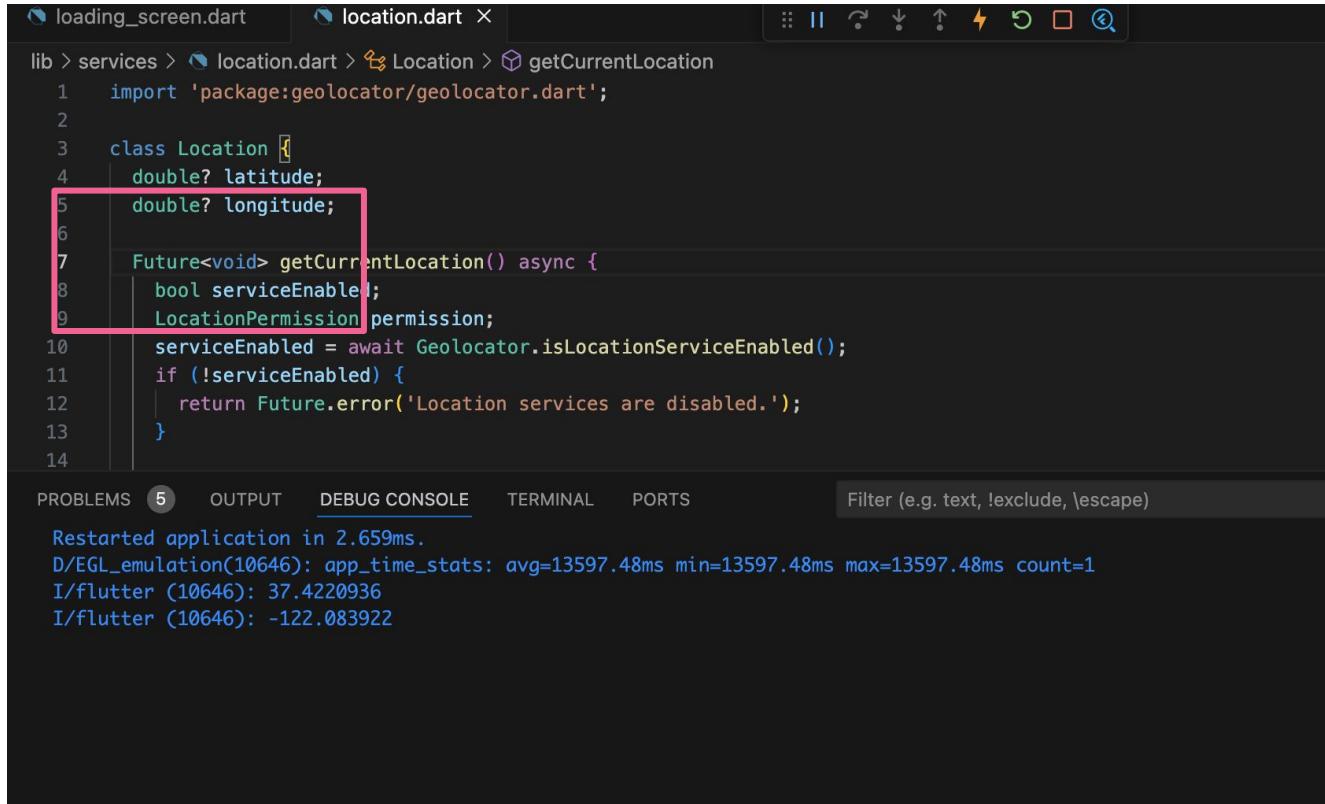
The screenshot shows a code editor with a file named `loading_screen.dart`. The code defines a `LoadingScreen` stateful widget and its corresponding `_LoadingScreenState` state class. The `getLocation` method uses `await` to call `location.getCurrentLocation()` and prints the latitude and longitude to the console. A red arrow points from the word "Perché?" in the text box below to the `await` keyword in the code.

```
lib > screen > loading_screen.dart 1 X
lib > screen > loading_screen.dart > _LoadingScreenState > getLocation
1 import 'package:flutter/material.dart';
2 import 'package:lezione9/services/location.dart';
3
4 class LoadingScreen extends StatefulWidget {
5   const LoadingScreen({super.key});
6
7   @override
8   State<LoadingScreen> createState() => _LoadingScreenState();
9 }
10
11 class _LoadingScreenState extends State<LoadingScreen> {
12   @override
13   void initState() {
14     super.initState();
15     getLocation();
16   }
17
18   void getLocation() async {
19     Location location = Location();
20     await location.getCurrentLocation();
21     print(location.latitude);
22     print(location.longitude);
23   }
24 }
```

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>



```
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
```