

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

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

Integrazione con HTTP

http 1.2.1 - <https://pub.dev/packages/http>

Installiamo una libreria per la gestione delle chiamate http.
Http è creata direttamente dal team di sviluppo di dart

The screenshot shows the pub.dev package page for the `http` package version 1.2.1. The page is titled "http 1.2.1" and includes the following details:

- Published 2 months ago • @dart.dev (Dart 3 compatible)
- SDK | DART | FLUTTER | PLATFORM | ANDROID | IOS | LINUX | MACOS | WEB | WINDOWS
- 7.4K
- 7479 LIKES | 140 PUB POINTS | 100% POPULARITY
- Publisher: @dart.dev
- Metadata: A composable, multi-platform, Future-based API for HTTP requests.
- Repository (GitHub): View/report issues | Contributing
- Documentation: API reference
- License: BSD-3-Clause (LICENSE)
- Dependencies: asynt, http_parser, meta, web
- More: Packages that depend on http

The main content area displays the package's description, usage instructions, and code examples. It also features sections for "Using", "Note", and "Flutter applications may require additional configuration to make HTTP requests".

```
import 'package:http/http.dart' as http;
var url = Uri.https('example.com', 'whatsit/create');
var response = await http.post(url, body: {'name': 'doodle', 'color': 'blue'});
print('Response status: ${response.statusCode}');
print('Response body: ${response.body}');

print(await http.read(Uri.https('example.com', 'foobar.txt')));
```

```
Note
Flutter applications may require additional configuration to make HTTP requests.
If you're making multiple requests to the same server, you can keep open a persistent connection by using a Client rather than making one-off requests. If you do this, make sure to close the client when you're done.

var client = http.Client();
try {
  var response = await client.post(
```

Confronto tra http e dio in Flutter

http

- **Semplice:** Il package http fornisce una semplice interfaccia per effettuare richieste HTTP;
- **Facile da usare:** È facile da integrare nelle applicazioni Flutter esistenti grazie alla sua semplice sintassi;
- **Leggero:** Ha una dimensione più piccola rispetto a dio, rendendolo una scelta appropriata per progetti con requisiti di dimensione ridotta;
- **Più basic:** Offre funzionalità di base per effettuare richieste HTTP come GET, POST, PUT, DELETE, ecc.;
- **Gestione manuale degli errori:** Richiede una gestione manuale degli errori per catturare e gestire le eccezioni durante le richieste.

dio

- **Potente:** Il package dio offre una vasta gamma di funzionalità avanzate per effettuare richieste HTTP;
- **Interoperabile:** Supporta funzionalità come l'intercettazione delle richieste, il salvataggio delle sessioni, la gestione dei cookie e molto altro;
- **Facilità di gestione degli errori:** Fornisce un meccanismo integrato per la gestione degli errori tramite l'uso di try-catch o async-await;
- **Personalizzabile:** È altamente personalizzabile e consente di configurare varie opzioni come timeout, intestazioni personalizzate, autenticazione, ecc.;
- **Supporto per i tipi di dati:** Supporta la conversione diretta dei dati JSON in oggetti Dart e viceversa;
- **Dimensione:** A causa della sua vasta gamma di funzionalità, il pacchetto dio ha una dimensione maggiore rispetto a http.

Quale scegliere?

- Se il progetto richiede solo operazioni di base come GET, POST, PUT, ecc., **http** potrebbe essere la scelta migliore per la sua semplicità e leggerezza.
- Per progetti che richiedono funzionalità avanzate come l'intercettazione delle richieste, la gestione avanzata degli errori e la personalizzazione delle richieste, **dio** potrebbe essere la scelta migliore per la sua potenza e flessibilità.

Scelgo un servizio in grado di offrirmi le informazioni

The screenshot shows the Open-Meteo Weather Forecast API interface. At the top, there's a banner with a gear icon and the text "Weather Forecast API" and "Seamless integration of high-resolution weather models with up to 16 days forecast". Below this is a "Location and Time" section where users can enter coordinates or select from a list. The coordinates entered are Latitude 41,5595 and Longitude 14,6674. The "Forecast Length" dropdown is set to "7 days (default)". The "Hourly Weather Variables" section contains a large grid of checkboxes for various weather parameters like Temperature, Wind Speed, and Soil Moisture at different heights.

This screenshot shows the API documentation page for the Open-Meteo API. It features two prominent blue buttons: "Download XLSX" and "Download CSV". Below these buttons is the API URL: "https://api.open-meteo.com/v1/forecast?latitude=41.5595&longitude=14.6674&hourly=temperature_2m".

Data Source

Open-Meteo weather forecast APIs use weather models from multiple national weather providers. For each location worldwide, the best models will be combined to provide the best possible forecast.

Integro in Postman i miei servizi per verificare

The screenshot shows the Postman application interface. In the top navigation bar, there are icons for Home, Workspaces, API Network, and a search bar labeled "Search Postman". Below the navigation is a header for "Demo Lezione 2" with a GET method and the URL "https://api.open-meteo.com/v1/forecast?latitude=41.5595&longitude=14.6674&hourly=temperature_2m&forecast_days=1". The main workspace is titled "My Workspace" and contains sections for Collections, Environments, and Mock servers. Under Environments, "Demo Lezione 2" is selected, and under Mock servers, "Met Museum" is listed. The central area displays the API request configuration. The "Params" tab is active, showing query parameters: latitude (41.5595), longitude (14.6674), hourly (temperature_2m), and forecast_days (1). Other tabs include Authorization, Headers (6), Body, Pre-request Script, Tests, and Settings. Below the parameters, the "Body" tab is selected, showing the JSON response received from the API. The response is a detailed weather forecast for the specified location and time period, including current conditions and hourly forecasts for the next day. The status bar at the bottom indicates "Status: 200 OK", "Time: 124 ms", and "Size: 1018 B".

```
1 {  
2   "latitude": 41.56,  
3   "longitude": 14.67,  
4   "generationtime_ms": 0.023800739288330078,  
5   "utc_offset_seconds": 0,  
6   "timezone": "GMT",  
7   "timezone_abbreviation": "GMT",  
8   "elevation": 0,  
9   "hourly": {  
10     "time": "2024-05-09T00:00",  
11     "temperature_2m": 10  
12   },  
13   "hourly": {  
14     "time": [  
15       "2024-05-09T00:00",  
16       "2024-05-09T01:00",  
17       "2024-05-09T02:00",  
18       "2024-05-09T03:00",  
19       "2024-05-09T04:00",  
20       "2024-05-09T05:00",  
21       "2024-05-09T06:00",  
22       "2024-05-09T07:00",  
23       "2024-05-09T08:00",  
24       "2024-05-09T09:00",  
25       "2024-05-09T10:00"  
26     ]  
27   }  
28 }
```

Cominciamo l'integrazione del nostro codice

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

> class LoadingScreen extends StatefulWidget { ...

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);
  }

  void getData() async {
    Uri url = Uri.parse(
      'https://api.open-meteo.com/v1/forecast?latitude=41.5595&longitude=14.6674&hourly=temperature_2m');
    http.Response response = await http.get(url);
    print(response);
  }
}

@override
Widget build(BuildContext context) {
  getData();
  return const Scaffold();
}
```

Reloaded 1 of 842 libraries in 1.007ms (compile: 17 ms, reload: 594 ms, reassemble: 314 ms).
I/flutter (16263): Instance of 'Response'

Verifichiamo statusCode e body

```
25
26     void getData() async {
27         Uri url = Uri.parse(
28             'https://api.open-meteo.com/v1/forecast?latitude=41.5595&longitude=14.6674&hourly=temperature_2m');
29         http.Response response = await http.get(url);
30         print('Response status: ${response.statusCode}');
31         print('Response body: ${response.body}');
32     }
33
34     @override
```

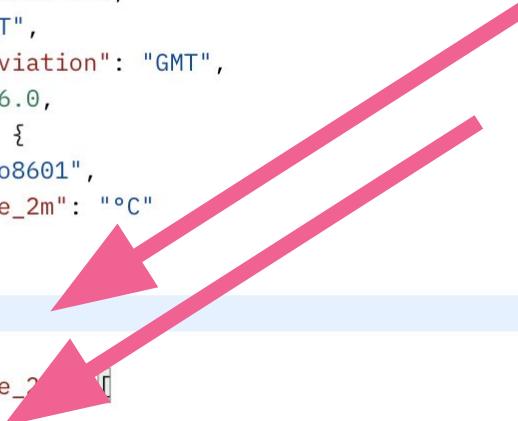
PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter (e.g. text, !exclude, \escape) ✖ ⌂ ^ X

```
0 , 2024-05-09T14:00 , 2024-05-09T15:00 , 2024-05-09T16:00 , 2024-05-09T17:00 , 2024-05-09T18:00 , 2024-05-09T19:00 , 2024-05-09T20:00
0 , "2024-05-09T21:00", "2024-05-09T22:00", "2024-05-09T23:00", "2024-05-10T00:00", "2024-05-10T01:00", "2024-05-10T02:00", "2024-05-10T03:00
0 , "2024-05-10T04:00", "2024-05-10T05:00", "2024-05-10T06:00", "2024-05-10T07:00", "2024-05-10T08:00", "2024-05-10T09:00", "2024-05-10T10:00
0 , "2024-05-10T11:00", "2024-05-10T12:00", "2024-05-10T13:00", "2024-05-10T14:00", "2024-05-10T15:00", "2024-05-10T16:00
Restarted application in 5.272ms.
D/EGL_emulation(16263): app_time_stats: avg=7752.76ms min=7752.76ms max=7752.76ms count=1
I/flutter (16263): 37.4220936
I/flutter (16263): -122.083922
I/flutter (16263): Response status: 200
I/flutter (16263): Response body: {"latitude":41.56,"longitude":14.67,"generationtime_ms":0.049948692321777344,"utc_offset_seconds":0,"timezone":"GMT","timezone_abbreviation":"GMT","elevation":686.0,"hourly_units":{"time":"iso8601","temperature_2m":°C},"hourly":[{"time":["2024-05-09T00:00","2024-05-09T01:00","2024-05-09T02:00","2024-05-09T03:00","2024-05-09T04:00","2024-05-09T05:00","2024-05-09T06:00","2024-05-09T07:00","2024-05-09T08:00","2024-05-09T09:00","2024-05-09T10:00","2024-05-09T11:00","2024-05-09T12:00","2024-05-09T13:00","2024-05-09T14:00","2024-05-09T15:00","2024-05-09T16:00","2024-05-09T17:00","2024-05-09T18:00","2024-05-09T19:00","2024-05-09T20:00","2024-05-09T21:00","2024-05-09T22:00","2024-05-09T23:00"],"temperature_2m":[]}]}
```

JSON Parsing

Acquisiamo data e temperatura

```
1  {
2      "latitude": 41.56,
3      "longitude": 14.67,
4      "generationtime_ms": 0.02300739288330078,
5      "utc_offset_seconds": 0,
6      "timezone": "GMT",
7      "timezone_abbreviation": "GMT",
8      "elevation": 686.0,
9      "hourly_units": {
10         "time": "iso8601",
11         "temperature_2m": "°C"
12     },
13     "hourly": {
14 >         "time": [...]
39         ],
40         "temperature_2m": [
41             11.7,
42             11.5,
43             11.3,
44             11.2,
45             11.0,
..             ...
..         ]
..     }
.. }
```



Integro il codice per il parsing del JSON

```
5
6
7 void getData() async {
8     Uri url = Uri.parse(
9         'https://api.open-meteo.com/v1/forecast?latitude=41.5595&longitude=14.6674&hourly=temperature_2m');
10    http.Response response = await http.get(url);
11    if (response.statusCode == 200) {
12        var data = jsonDecode(response.body);
13        var time = data['hourly']['time'][0];
14        double temperature = data['hourly']['temperature_2m'][0];
15        print('Time: $time, Temperature: $temperature');
16    } else {
17        print(response.statusCode);
18    }
19 }
```

```
Restarted application in 1.775ms.
D/EGL_emulation(16263): app_time_stats: avg=57870.26ms min=57870.26ms max=57870.26ms count=1
I/flutter (16263): Time: 2024-05-10T00:00, Temperature: 12.8
I/flutter (16263): 37.4220936
I/flutter (16263): -122.083922
```

Parametrizzo l'url della mia chiamata

```
class _LoadingScreenState extends State<LoadingScreen> {
    double? latitude;
    double? longitude;
    @override
    void initState() {
        super.initState();
        getLocation();
    }

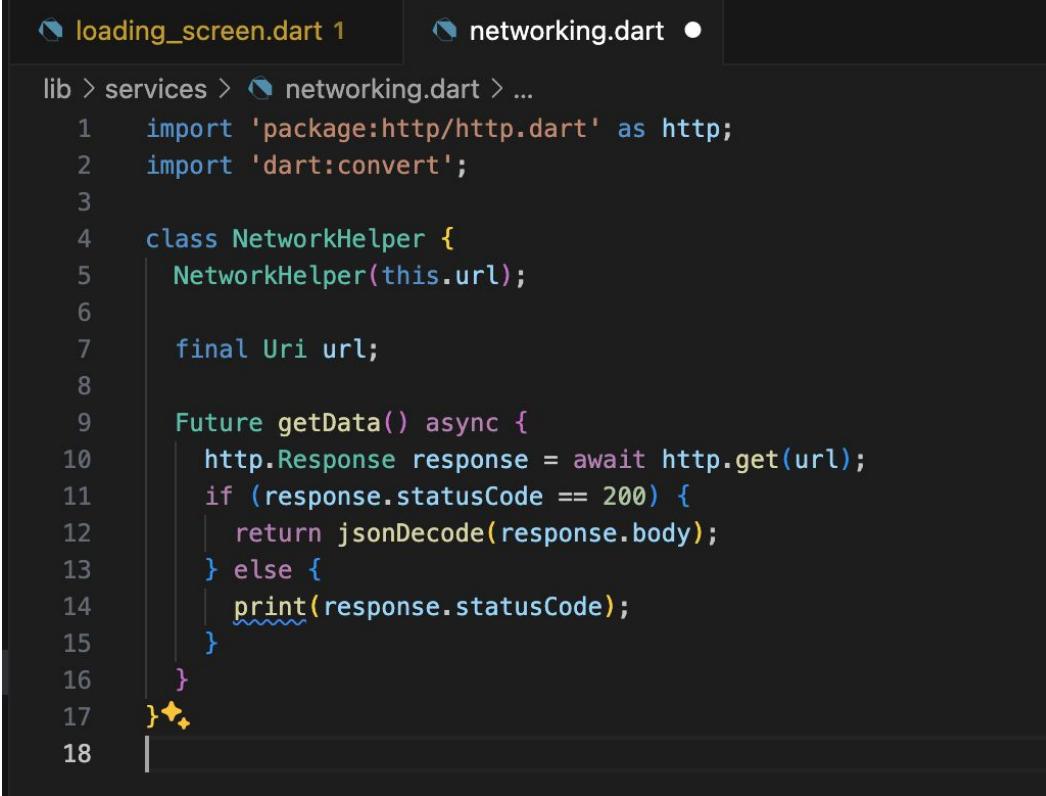
    void getLocation() async {
        Location location = Location();
        await location.getCurrentLocation();
        latitude = location.latitude;
        longitude = location.longitude;
    }

    void getData() async {
        Uri url = Uri.parse(
            'https://api.open-meteo.com/v1/forecast?latitude=$latitude&longitude=$longitude&hourly=temperature_2m');
        http.Response response = await http.get(url);
        if (response.statusCode == 200) {
            var data = jsonDecode(response.body);
            var time = data['hourly']['time'][0];
            double temperature = data['hourly']['temperature_2m'][0];
            print('Time: $time, Temperature: $temperature');
        } else {
    }
```



Facciamo
refactoring della
componente di
networking

Network Service

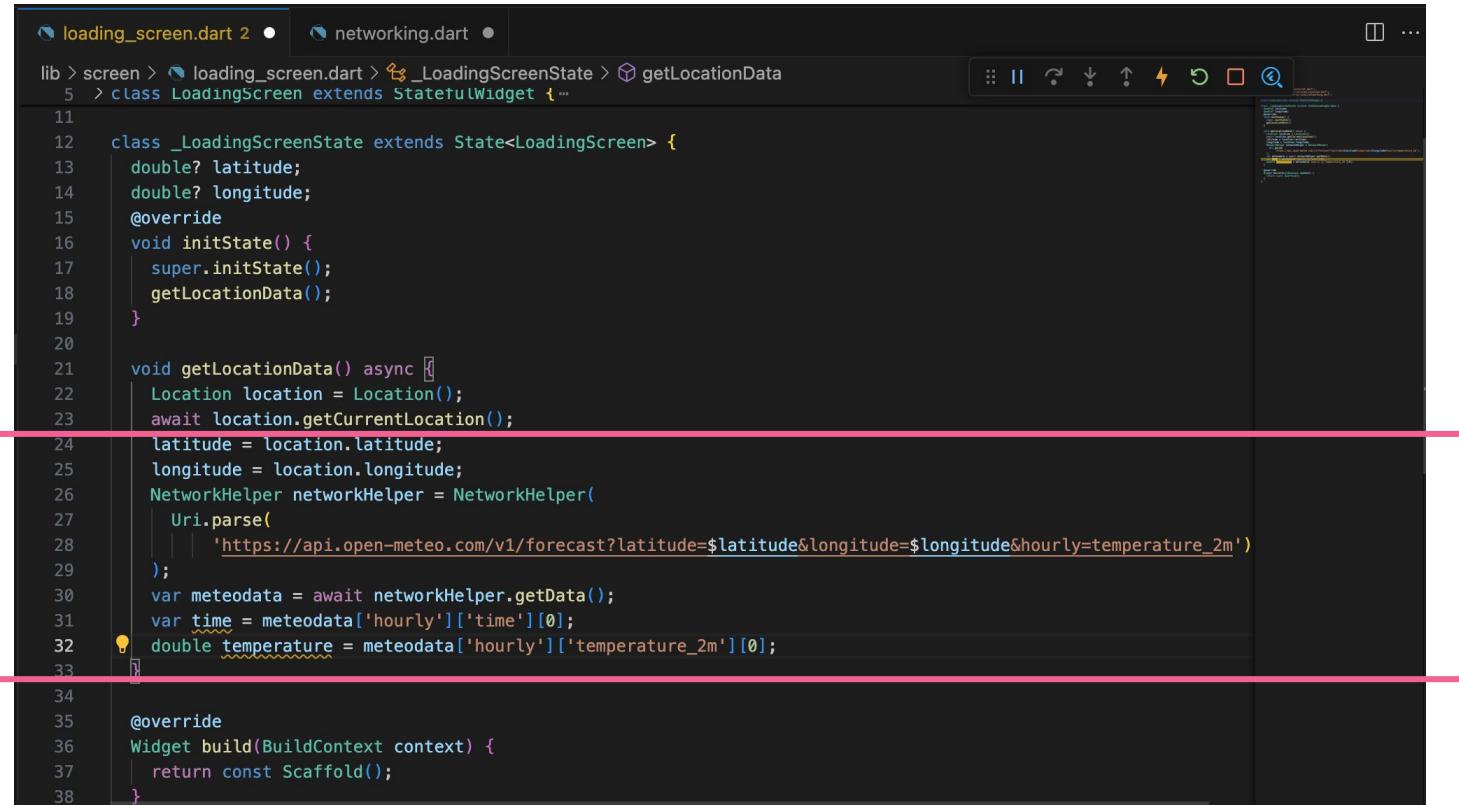


The screenshot shows a code editor with two tabs at the top: "loading_screen.dart 1" and "networking.dart". The "networking.dart" tab is active, indicated by a blue dot. The code in the editor is as follows:

```
lib > services > networking.dart > ...
1 import 'package:http/http.dart' as http;
2 import 'dart:convert';
3
4 class NetworkHelper {
5     NetworkHelper(this.url);
6
7     final Uri url;
8
9     Future getData() async {
10         http.Response response = await http.get(url);
11         if (response.statusCode == 200) {
12             return jsonDecode(response.body);
13         } else {
14             print(response.statusCode);
15         }
16     }
17 }
18
```

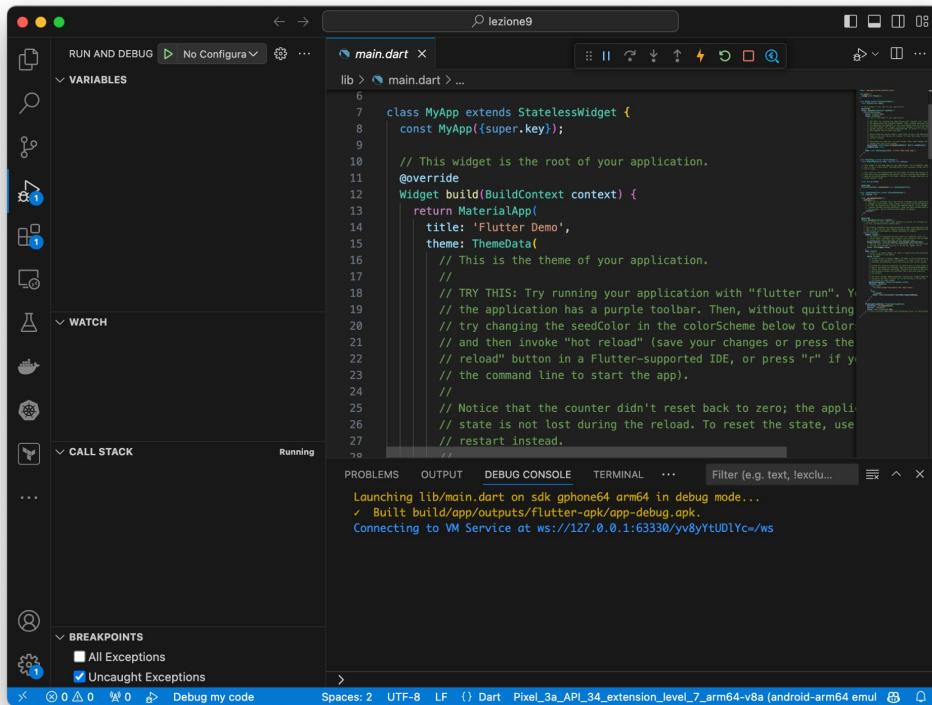
The code defines a class `NetworkHelper` that takes a `Uri` as a parameter. It has a `final Uri url;` field. The `getData()` method makes a `http.get(url)` request and returns the JSON decoded body if the status code is 200, or prints the status code otherwise.

Integrazione nel loading_screen



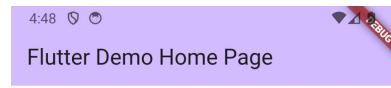
```
lib > screen > loading_screen.dart > _LoadingScreenState > getLocationData
5 > class LoadingScreen extends StatefulWidget { ...
11
12     class _LoadingScreenState extends State<LoadingScreen> {
13         double? latitude;
14         double? longitude;
15         @override
16         void initState() {
17             super.initState();
18             getLocationData();
19         }
20
21         void getLocationData() async {
22             Location location = Location();
23             await location.getCurrentLocation();
24             latitude = location.latitude;
25             longitude = location.longitude;
26             NetworkHelper networkHelper = NetworkHelper(
27                 Uri.parse(
28                     'https://api.open-meteo.com/v1/forecast?latitude=$latitude&longitude=$longitude&hourly=temperature_2m'
29                 );
30             var meteodata = await networkHelper.getData();
31             var time = meteodata['hourly']['time'][0];
32             double temperature = meteodata['hourly']['temperature_2m'][0];
33
34
35             @override
36             Widget build(BuildContext context) {
37                 return const Scaffold();
38             }
39         }
40     }
41 }
```

Funzionalità principali riguardanti i temi



The screenshot shows the Android Studio interface with the following details:

- Toolbar:** RUN AND DEBUG (No Configuration), Variables, Watch, Call Stack, Breakpoints.
- Main Dart File:** The code defines a `MyApp` widget that extends `StatelessWidget`. It sets the title to 'Flutter Demo' and the theme to `ThemeData` with a purple color scheme. A note in the code suggests trying a hot reload with the command "flutter run".
- Output Tab:** Shows the terminal output for launching the app on an `arm64` emulator, including the APK build and connection to the VM service.
- Emulator Preview:** Displays the 'Flutter Demo Home Page' with a purple toolbar and a red 'DEBUG' indicator.



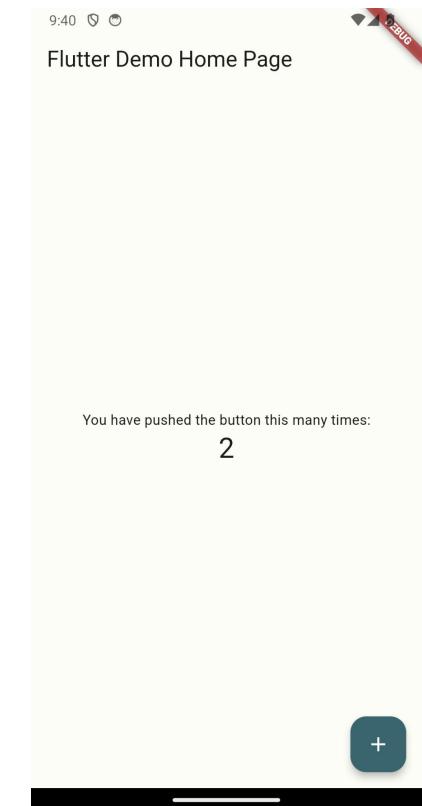
You have pushed the button this many times:

0

Progetto di default

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 }
```



Esercizio

Partendo dal codice disponibile nel file esercizio2.zip, provate ad integrare un endpoint a vostra scelta (non strettamente legato al meteo) e stampate in console dei dati acquisiti da JSON.



Mostriamo il
risultato

LocationScreen

```
lib > screen > location_screen.dart > ...
1  import 'package:flutter/material.dart';
2
3  class LocationScreen extends StatefulWidget {
4    const LocationScreen({super.key});
5
6    @override
7    State<LocationScreen> createState() => _LocationScreenState();
8  }
9
10 class _LocationScreenState extends State<LocationScreen> {
11   @override
12   Widget build(BuildContext context) {
13     return Scaffold(
14       backgroundColor: Colors.white,
15       body: Center(
16         child: Column(
17           mainAxisAlignment: MainAxisAlignment.center,
18           children: <Widget>[
19             Image.asset('assets/comevestirmi.jpeg'),
20             const Text(
21               'Non so come dovrei vestirmi',
22               style: TextStyle(
23                 fontSize: 24,
24                 color: Colors.black,
25               ), // TextStyle
26             ), // Text
27             ], // <Widget>[]
28           ), // Column
29         ), // Center
30       ); // Scaffold
31   }
32 }◆
33 |
```

Mi sposto in automatico nella nuova schermata

```
void getLocationData() async {
  Location location = Location();
  await location.getCurrentLocation();
  latitude = location.latitude;
  longitude = location.longitude;
  NetworkHelper networkHelper = NetworkHelper(
    Uri.parse(
      'https://api.open-meteo.com/v1/forecast?latitude=$latitude&longitude=$longitude&hourly=temperature_2m'),
  );
  var meteodata = await networkHelper.getData();
  // var time = meteodata['hourly']['time'][0];
  // double temperature = meteodata['hourly']['temperature_2m'][0];
  Navigator.push(context, MaterialPageRoute(builder: (context) {
    return const LocationScreen();
  })); // MaterialPageRoute
}
```

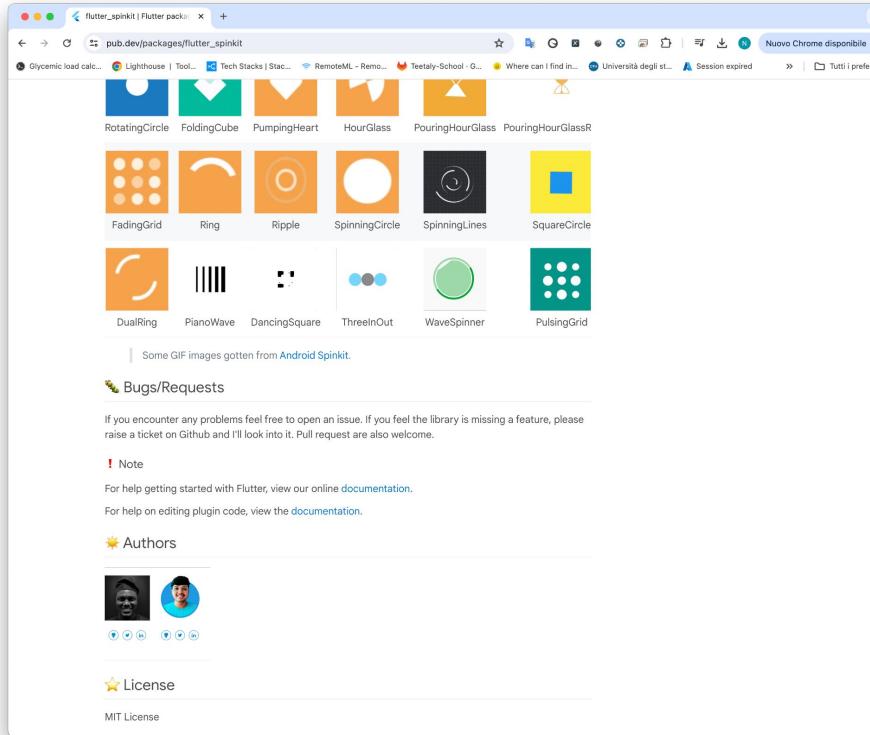
Il risultato?



Non so come dovrei vestirmi

Integriamo uno spinner nel passaggio tra schermate

https://pub.dev/packages/flutter_spinkit/install



Implemento flutter_spinkit

```
lib > screen > loading_screen.dart > _LoadingScreenState > getLocationData
14 class _LoadingScreenState extends State<LoadingScreen> {
21 }
22
23 void getLocationData() async {
24     Location location = Location();
25     await location.getCurrentLocation();
26     latitude = location.latitude;
27     longitude = location.longitude;
28     NetworkHelper networkHelper = NetworkHelper(
29         Uri.parse(
30             'https://api.open-meteo.com/v1/forecast?latitude=$latitude&longitude=$longitude&hourly=temperature_2m'),
31     );
32     var metedata = await networkHelper.getData();
33     // var time = metedata['hourly']['time'][0];
34     // double temperature = metedata['hourly']['temperature_2m'][0];
35     Navigator.push(context, MaterialPageRoute(builder: (context) {
36         return const LocationScreen();
37     }));
38 }
39
40 @override
41 Widget build(BuildContext context) {
42     return Scaffold(
43         body: Container(
44             color: Colors.white,
45             child: const Center(
46                 child: SpinKitDoubleBounce(
47                     color: Colors.black,
48                     size: 100.0,
49                 ), // SpinKitDoubleBounce
50             ), // Center
51         ), // Container
52     ); // Scaffold
53 }
54 }
```

Esercizio

Non potendo mostrare il risultato pratico, effettuate una build del progetto partendo dal codice disponibile nel file esercizio3.zip.



Passiamo i dati al
secondo widget

Modifichiamo il widget per accettare un parametro

```
lib > screen > location_screen.dart > _LocationScreenState > initState
1   import 'package:flutter/material.dart';
2
3   class LocationScreen extends StatefulWidget {
4       const LocationScreen({super.key, this.meteosimulatore});
5       final meteosimulatore;
6
7       @override
8       State<LocationScreen> createState() => _LocationScreenState();
9   }
10
11  class _LocationScreenState extends State<LocationScreen> {
12      @override
13      void initState() {
14          super.initState();
15          print(widget.meteosimulatore);
16      }
17      @override
18      Widget build(BuildContext context) {
19          return Scaffold(
20              backgroundColor: Colors.white,
```

Integriamo anche la funzionalità nella loading_screen

```
lib > screen > loading_screen.dart > _LoadingScreenState
14   class _LoadingScreenState extends State<LoadingScreen> {
15     void getLocationData() async {
16       await location.getCurrentLocation(),
17       latitude = location.latitude;
18       longitude = location.longitude;
19       NetworkHelper networkHelper = NetworkHelper(
20         Uri.parse(
21           'https://api.open-meteo.com/v1/forecast?latitude=$latitude&longitude=$longitude&hourly=temperature_2m'),
22         );
23       var meteodata = await networkHelper.getData();
24       // var time = meteodata['hourly']['time'][0];
25       // double temperature = meteodata['hourly']['temperature_2m'][0];
26       Navigator.push(
27         context,
28         MaterialPageRoute(builder: (context) {
29           return LocationScreen(
30             meteosimulatore: meteodata,
31           ); // LocationScreen
32         }, // MaterialPageRoute
33         );
34       );
35     }
36   }
```



Finalizziamo l'acquisizione dei dati

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

class LocationScreen extends StatefulWidget {
  const LocationScreen({super.key, this.meteosimulatore});
  final dynamic meteosimulatore;

  @override
  State<LocationScreen> createState() => _LocationScreenState();
}

class _LocationScreenState extends State<LocationScreen> {
  double? temperature;
  String? time;
  @override
  void initState() {
    super.initState();
    //print(widget.meteosimulatore);
    updateUI(widget.meteosimulatore);
  }

  void updateUI(dynamic meteoData) {
    time = meteoData['hourly']['time'][0];
    temperature = meteoData['hourly']['temperature_2m'][0];
    print(time);
    print(temperature);
  }
}
```

I/flutter (25013): 2024-05-10T00:00
I/flutter (25013): 28.5

Una nota di colore...

```
lib > screen > location_screen.dart > _LocationScreenState > getIcon
  1 import 'package:flutter/material.dart';
  2
  3 class LocationScreen extends StatefulWidget {
  4   const LocationScreen({super.key, this.meteosimulatore});
  5   final dynamic meteosimulatore;
  6
  7   @override
  8   State<LocationScreen> createState() => _LocationScreenState();
  9 }
10
11 class _LocationScreenState extends State<LocationScreen> {
12   late double temperature;
13   late String time;
14   String? icona;
15   @override
16   void initState() {
17     super.initState();
18     //print(widget.meteosimulatore);
19     updateUI(widget.meteosimulatore);
20   }
21
22   void updateUI(dynamic meteoData) {
23     time = meteoData['hourly'][0]['time'];
24     temperature = meteoData['hourly'][0]['temperature_2m'];
25     icona = getIcon(temperature);
26   }
27
28   String getIcon(double temperature) {
29     if (temperature > 22) {
30       return '*'; // Sun emoji
31     } else if (temperature > 10) {
32       return '☁'; // Cloud emoji
33     } else {
34       return '❄'; // Snowflake emoji
35     }
36   }

```

Il risultato finale

```
@override  
Widget build(BuildContext context) {  
  return Scaffold(  
    backgroundColor: Colors.white,  
    body: Center(  
      child: Column(  
        mainAxisAlignment: MainAxisAlignment.center,  
        children: <Widget>[  
          Image.asset('assets/comevestirmi.jpeg'),  
          Text(  
            'Il giorno $time ci saranno $temperature gradi $icona',  
            style: const TextStyle(  
              fontSize: 24,  
              color: Colors.black,  
            ), // TextStyle  
            ), // Text  
          ], // <Widget>[]  
        ), // Column  
      ), // Center  
    ); // Scaffold  
}
```



Il giorno 2024-05-10T00:00 ci
saranno 28.5 gradi ☀

Esercizio

Trovate l'ultima versione del codice nel file esercizio4.zip . Nella schermata finale la data è davvero brutta, provate a renderla human-readable, magari sfruttando un package...