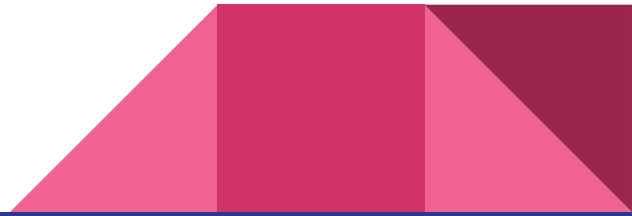
Programmazione Mobile

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

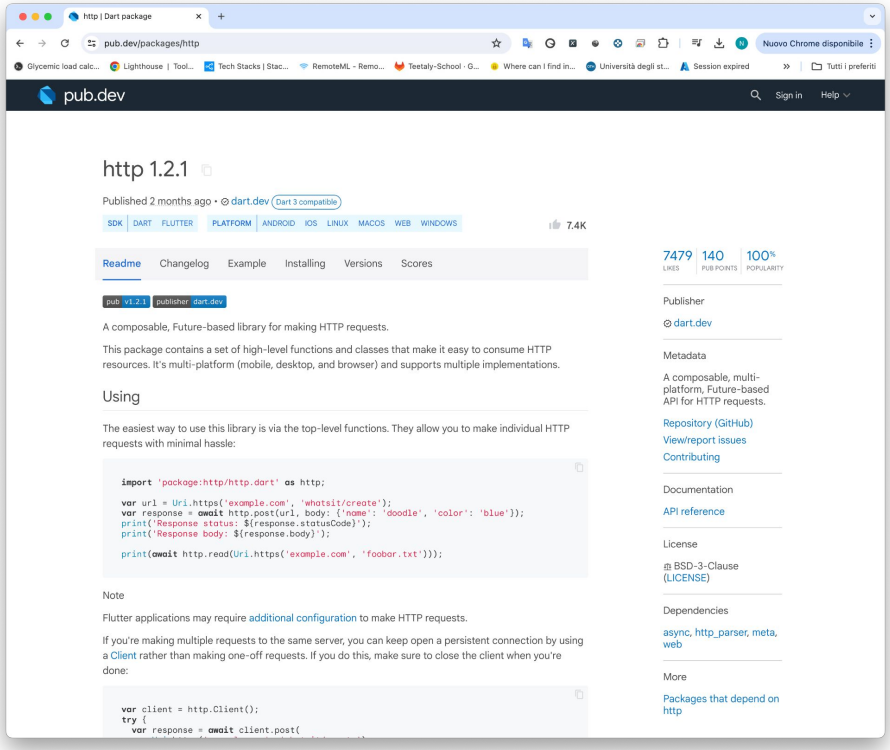
- Panoramica sull’uso dei temi
- Navigazione tra schermate
- Futures, Async & Await
- Gestione delle eccezioni
- Networking in flutter
- Parsing di JSON
- Gestione Location



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



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.

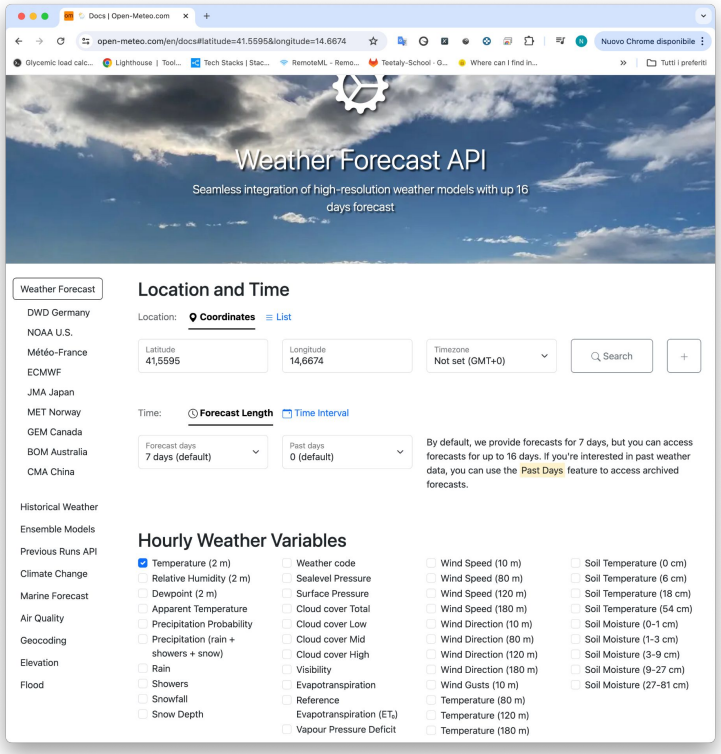
dio

- **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



Download XLSX Download CSV

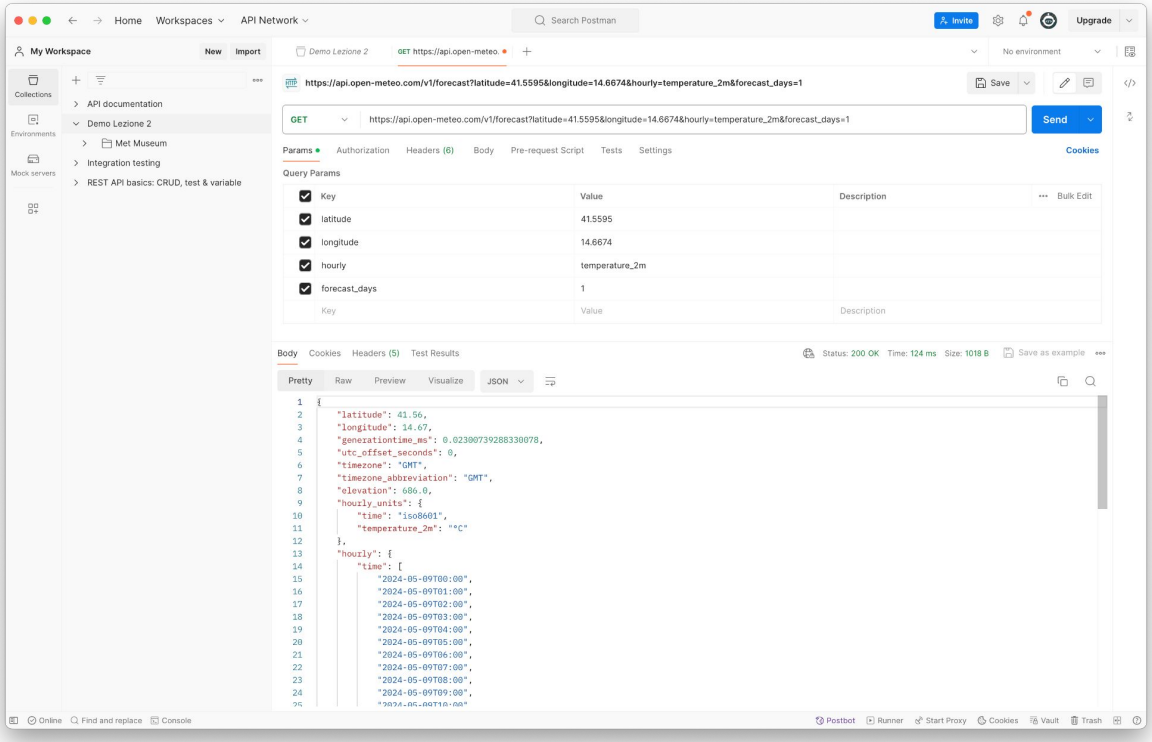
API URL (Open in new tab or copy this URL into your application).

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



Cominciamo l’integrazione del nostro codice

```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

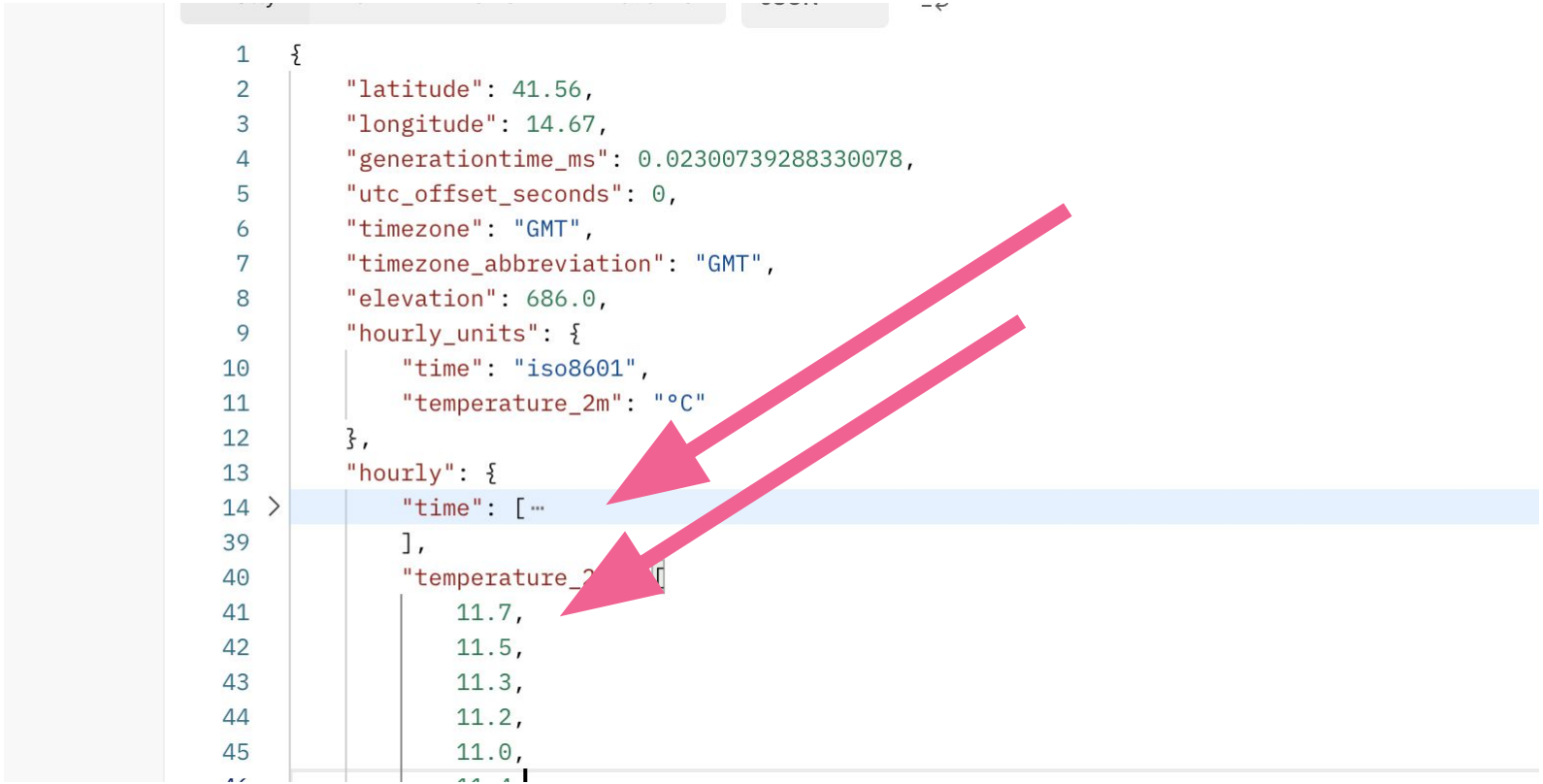
```dart
26  void getData() async {
27    Uri url = Uri.parse(
28        'https://api.open-meteo.com/v1/forecast?latitude=41.5595&longitude=14.6674&hourly=temperature_2m');
29    http.Response response = await http.get(url);
30    print('Response status: ${response.statusCode}');
31    print('Response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

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

```
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","2024-05-10T03:0
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","2024-05-10T10:0
0","2024-05-10T11:00","2024-05-10T12:00","2024-05-10T13:00","2024-05-10T14:00","2024-05-10T15:00","2024-05-
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":{"t
ime":["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","2024-05-09T13:0
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","2024-05-09T20:0
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","2024-05-10T03:0
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","2024-05-10T10:0
```

JSON Parsing

Acquisiamo data e temperatura

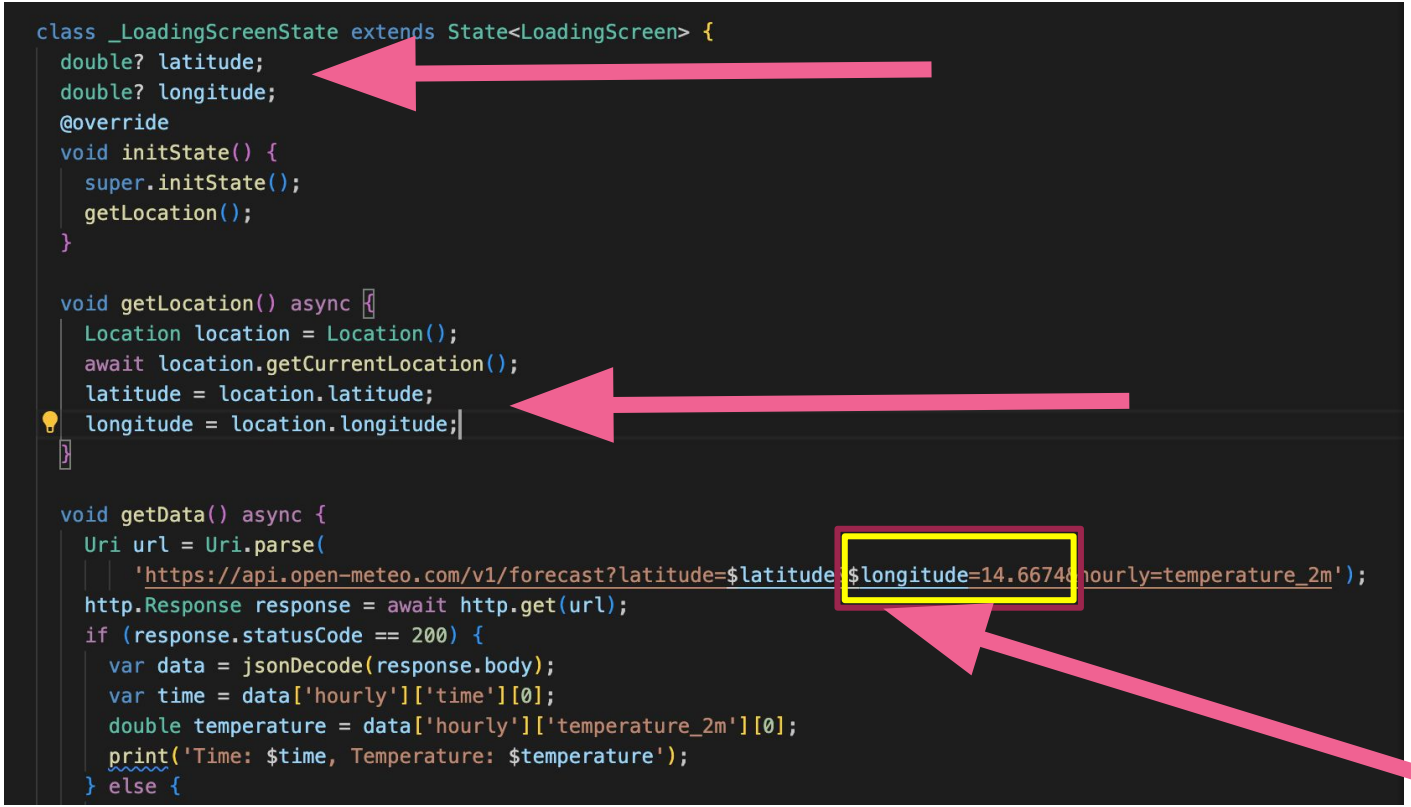


Integro il codice per il parsing del JSON

```
}

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);
  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 {
    print(response.statusCode);
  }
}
```

```
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=14.6674&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

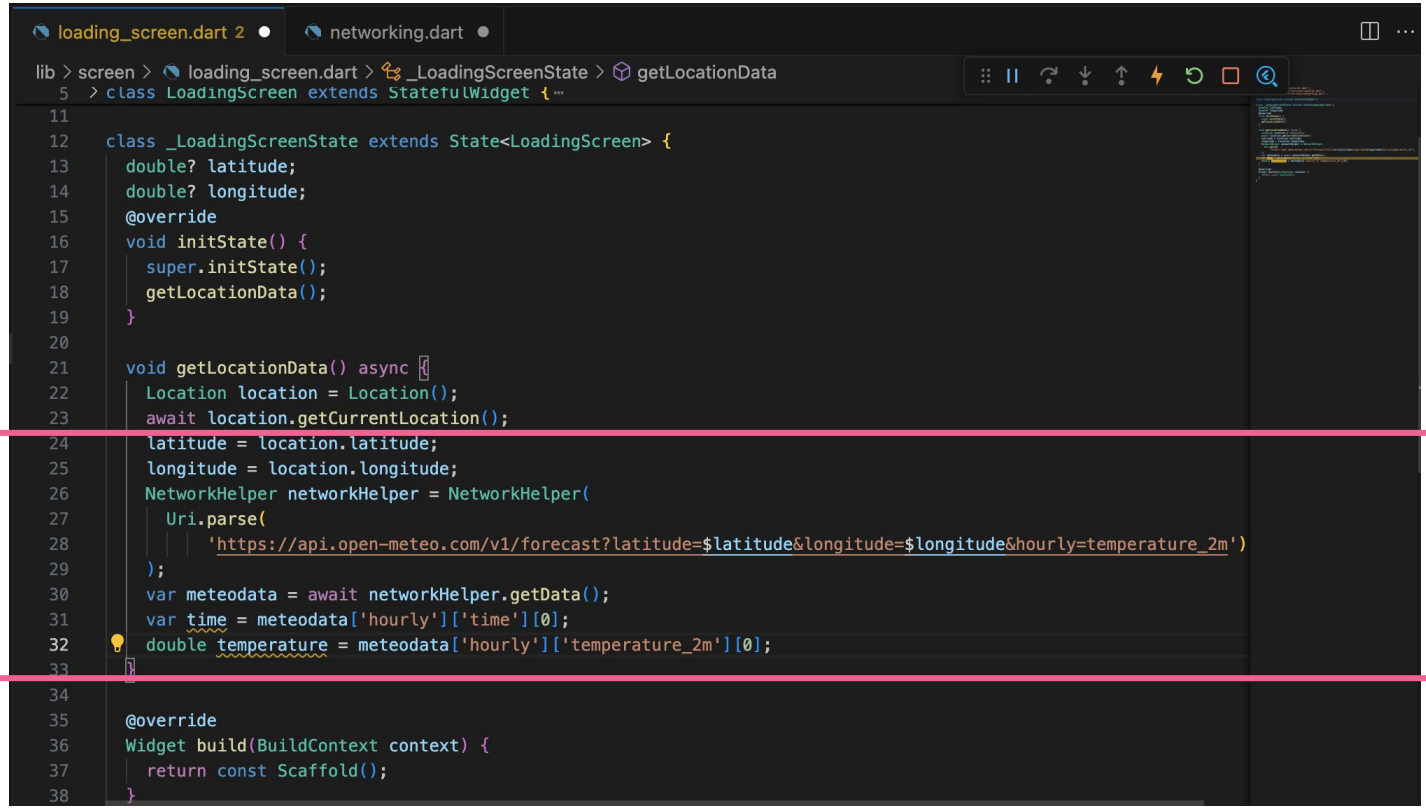
Network Service

loading_screen.dart 1 | networking.dart ●

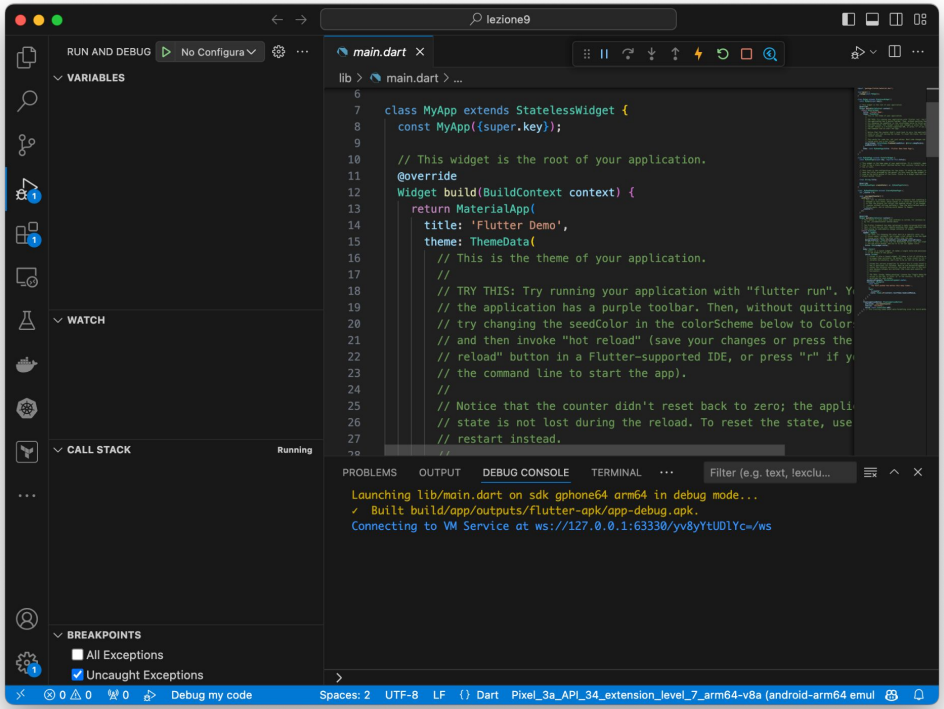
lib > services > networking.dart > ...

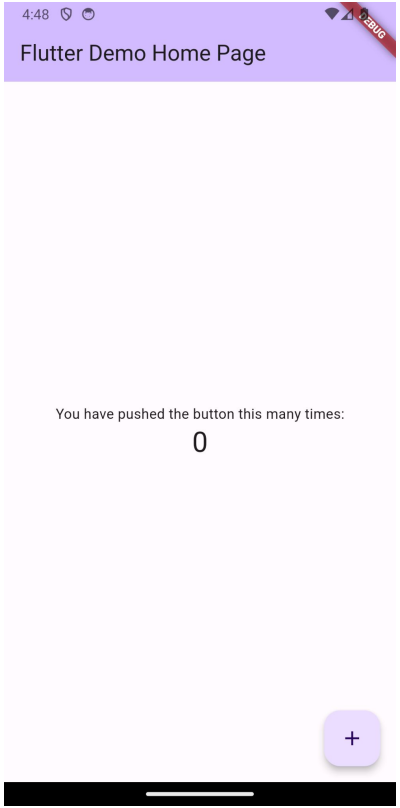
```dart
import 'package:http/http.dart' as http;
import 'dart:convert';

class NetworkHelper {
  NetworkHelper(this.url);

  final Uri url;

  Future getData() async {
    http.Response response = await http.get(url);
    if (response.statusCode == 200) {
      return jsonDecode(response.body);
    } else {
      print(response.statusCode);
    }
  }
}
```

Integrazione nel loading_screen



Funzionalità principali riguardanti i temi





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 > ...

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

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

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

class _LocationScreenState extends State<LocationScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.white,
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Image.asset('assets/comevestirmi.jpeg'),
            const Text(
              'Non so come dovrei vestirmi',
              style: TextStyle(
                fontSize: 24,
                color: Colors.black,
              ), // TextStyle
            ), // Text
          ], // <Widget>[]
        ), // Column
      ), // Center
    ); // Scaffold
  }
}
```

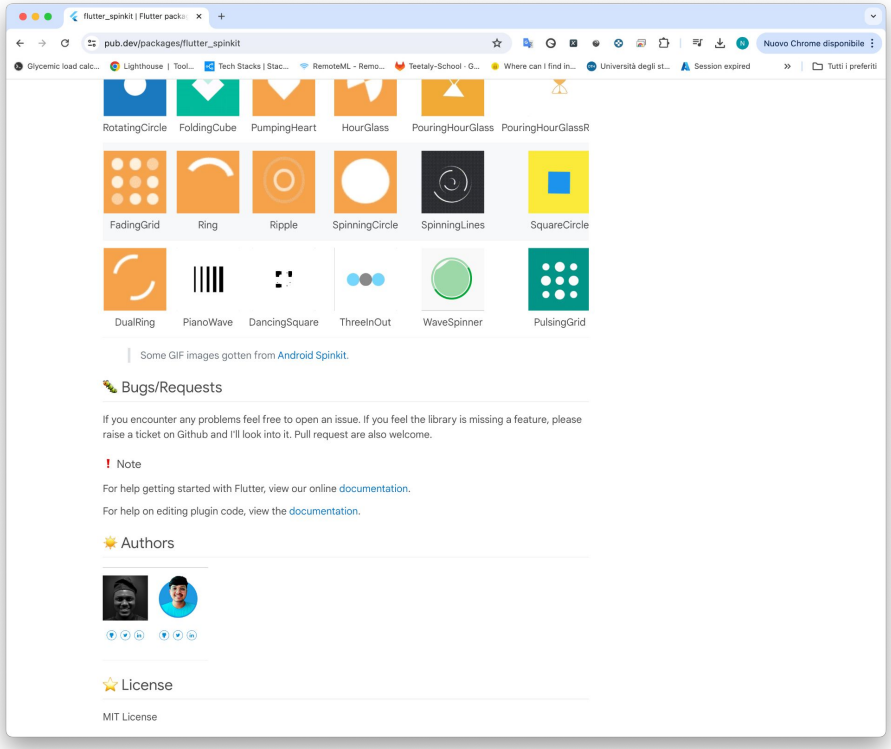
Mi sposto in automatico nella nuova schermata

```
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
}
```

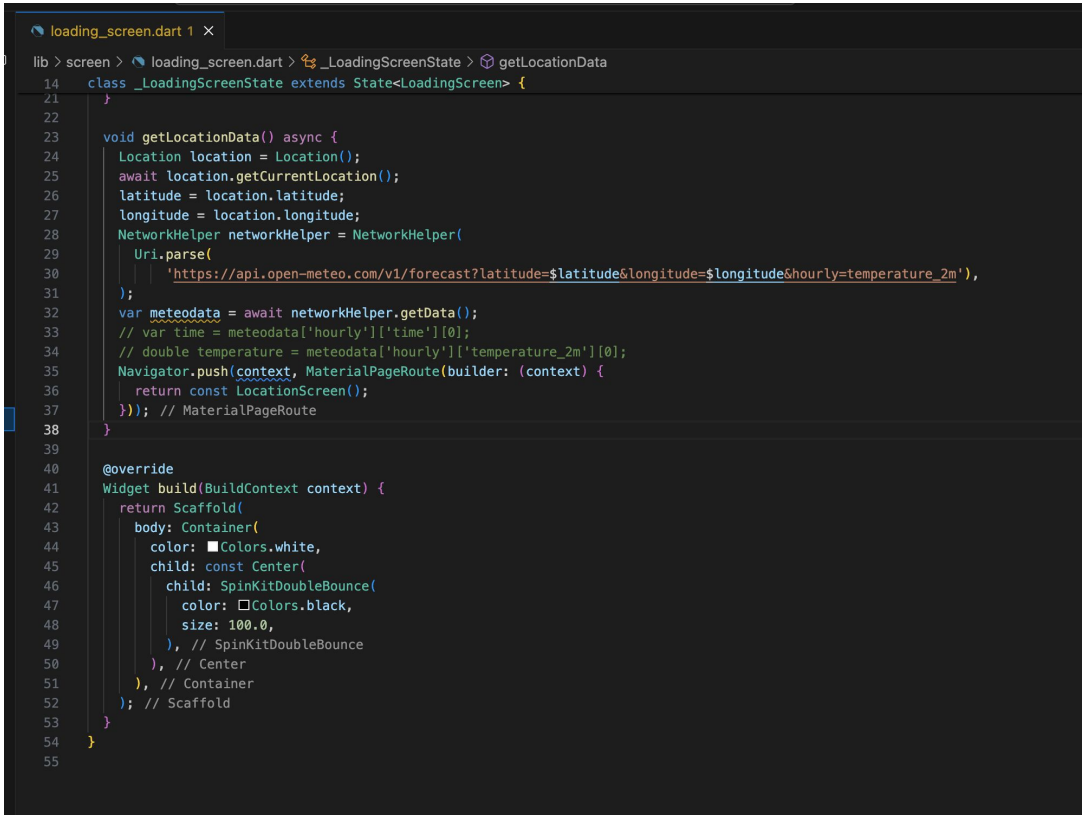
Il risultato?

Integriamo uno spinner nel passaggio tra schermate

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



Implemento flutter_spinkit

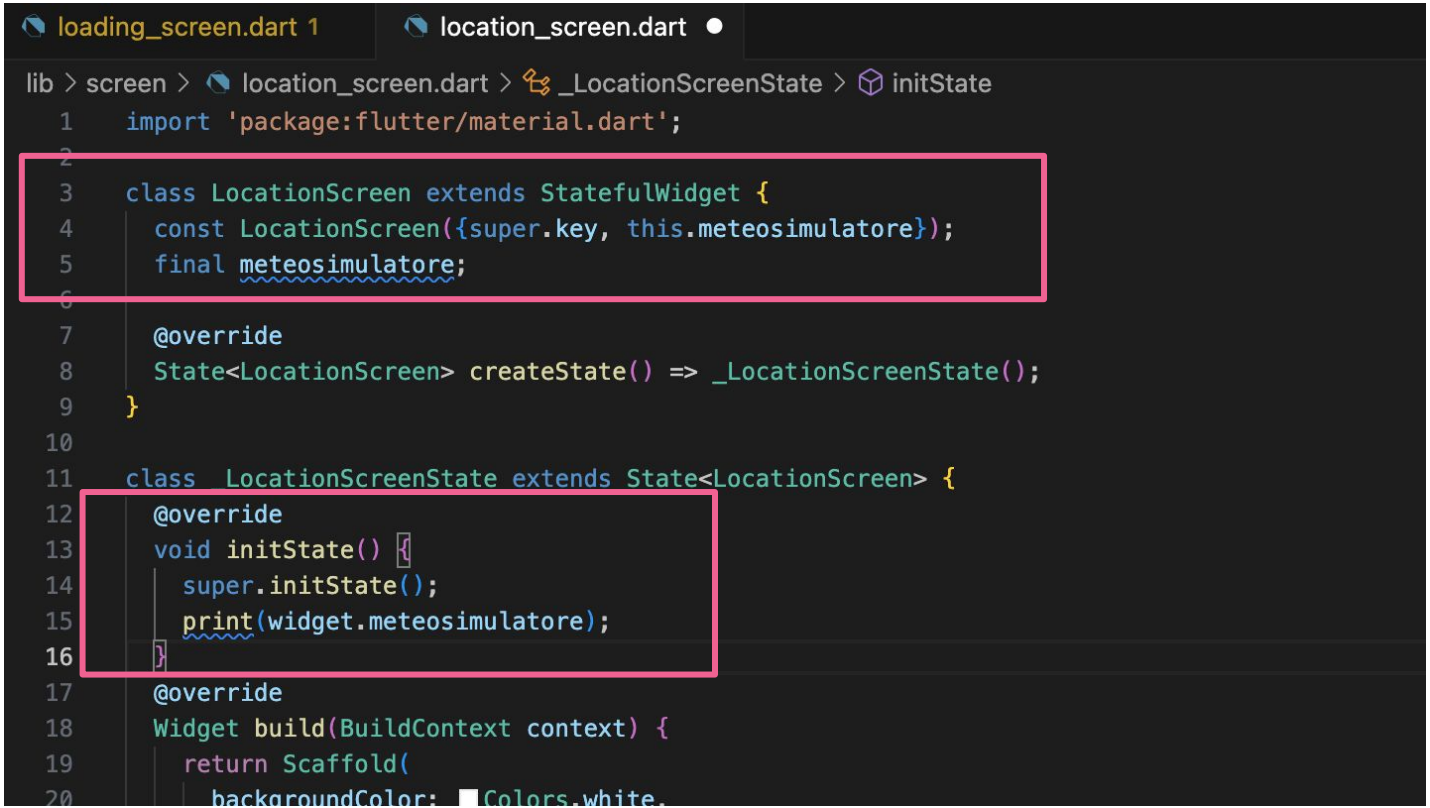


Esercizio

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



Integriamo anche la funzionalità nella loading_screen

lib > screen > loading_screen.dart > _LoadingScreenState

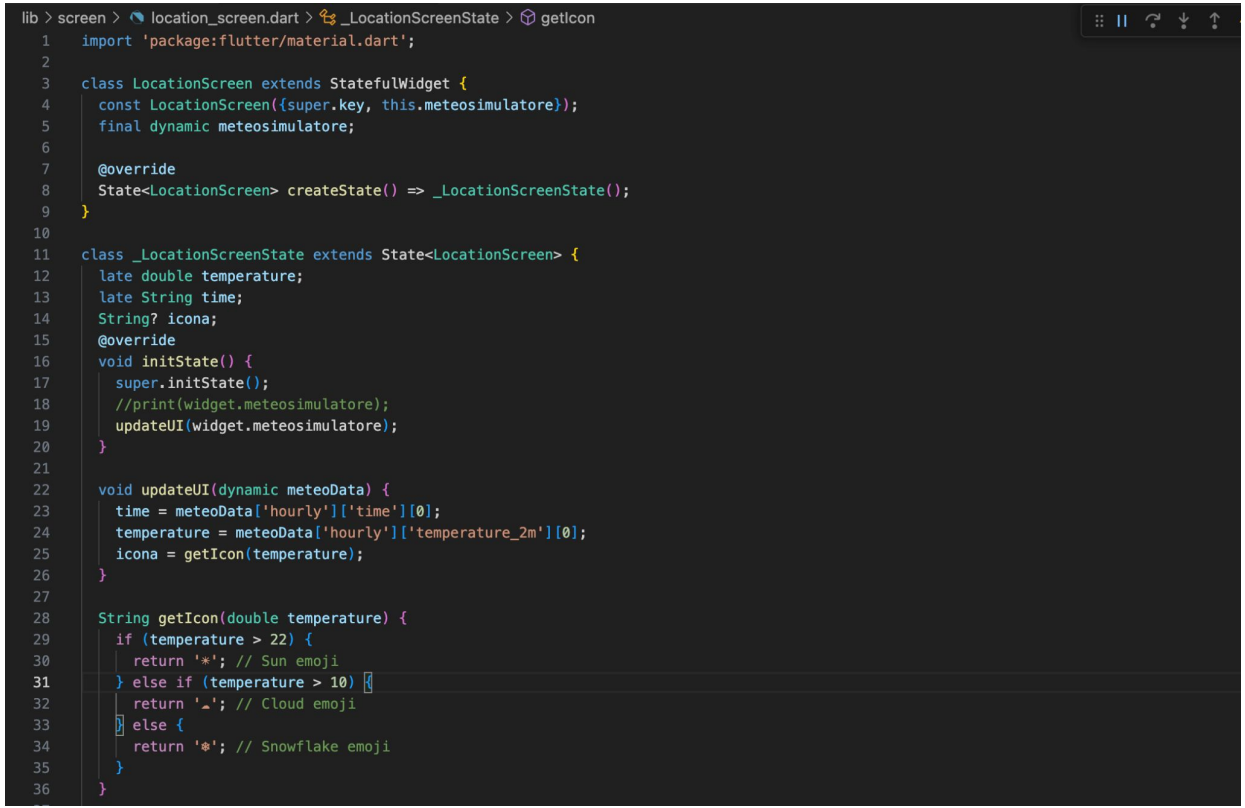
```
14  class _LoadingScreenState extends State<LoadingScreen> {
23    void getLocationData() async {
26      latitude = location.latitude;
27      longitude = location.longitude;
28      NetworkHelper networkHelper = NetworkHelper(
29        Uri.parse(
30            'https://api.open-meteo.com/v1/forecast?latitude=$latitude&longitude=$longitude&hourly=temperature_2m'),
31      );
32      var meteodata = await networkHelper.getData();
33      // var time = meteodata['hourly']['time'][0];
34      // double temperature = meteodata['hourly']['temperature_2m'][0];
35      Navigator.push(
36        context,
37        MaterialPageRoute(builder: (context) {
38          return LocationScreen(
39            meteosimulatore: meteodata,
40          ); // LocationScreen
41        }), // MaterialPageRoute
42      );
43    }
44
```

Finalizziamo l’acquisizione dei dati

```dart
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…



Il risultato finale

```dart
@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
}
}
```

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…