

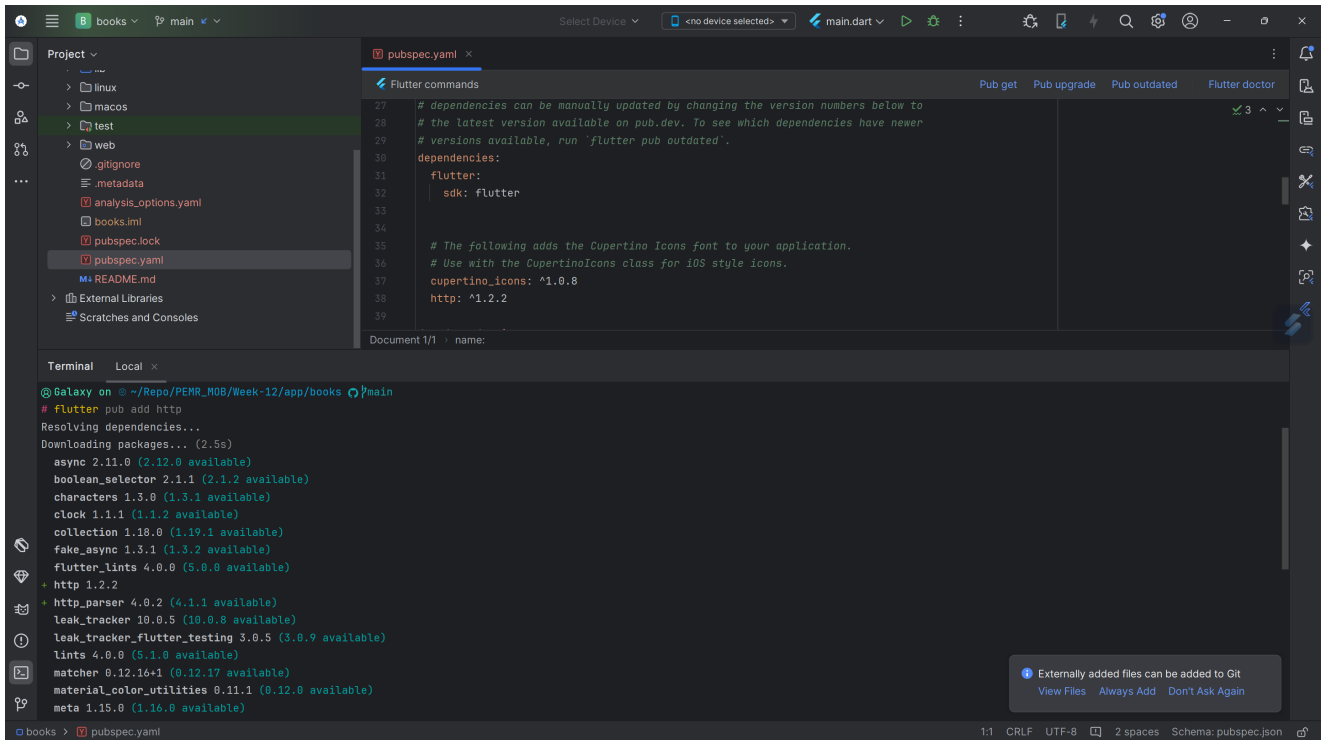
# 12 | Pemrograman Asynchronous

## Praktikum 1: Mengunduh Data dari Web Service (API)

### Langkah 1: Buat Project Baru

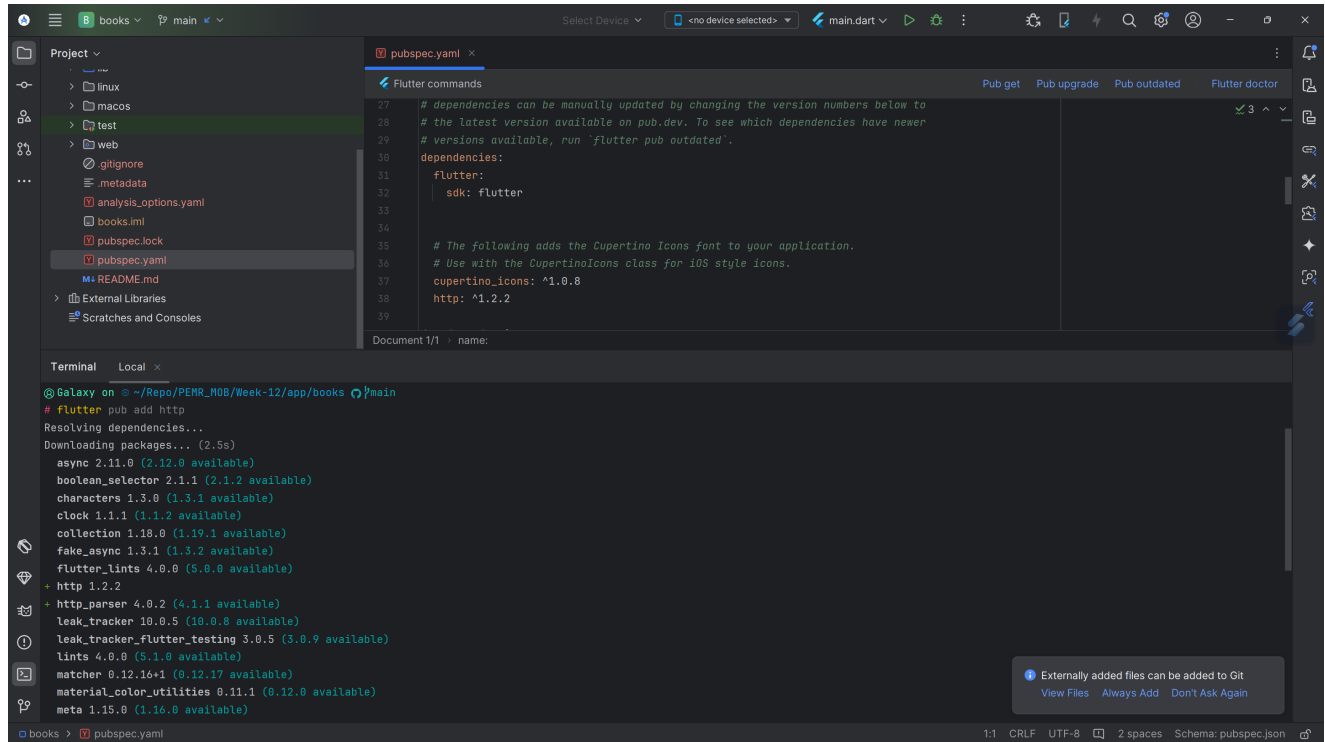
Buatlah sebuah project flutter baru dengan nama **books** di folder src **week-12** repository GitHub Anda.

Kemudian Tambahkan dependensi http dengan mengetik perintah **flutter pub get http**



### Langkah 2: Cek file pubspec.yaml

Jika berhasil install plugin, pastikan plugin http telah ada di file pubspec.



### Langkah 3: Buka file main.dart

Ketiklah kode seperti berikut ini.

#### Soal 1

Tambahkan nama panggilan Anda pada title app sebagai identitas hasil pekerjaan Anda.

```
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:http/http.dart';
import 'package:http/http.dart' as http;

void main() {
  runApp(const MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Naufal',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: ,
    );
  }
}
```

```

}

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

  @override
  State<FuturePage> createState() => _FuturePageState();
}

class _FuturePageState extends State<FuturePage> {
  String result = '';

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Back from the future'),
      ),
      body: Center(
        child: Column(children: [
          const Spacer(),
          ElevatedButton(
            onPressed: () {

              }, child: const Text('GO!')),
          const Spacer(),
          Text(result),
          const Spacer(),
          const CircularProgressIndicator(),
          const Spacer()
        ],),
      ),
    );
  }
}

```

#### Langkah 4: Tambah method getData()

Tambahkan method berikut ke dalam class `_FuturePageState` yang berguna untuk mengambil data dari *API Google Books*.

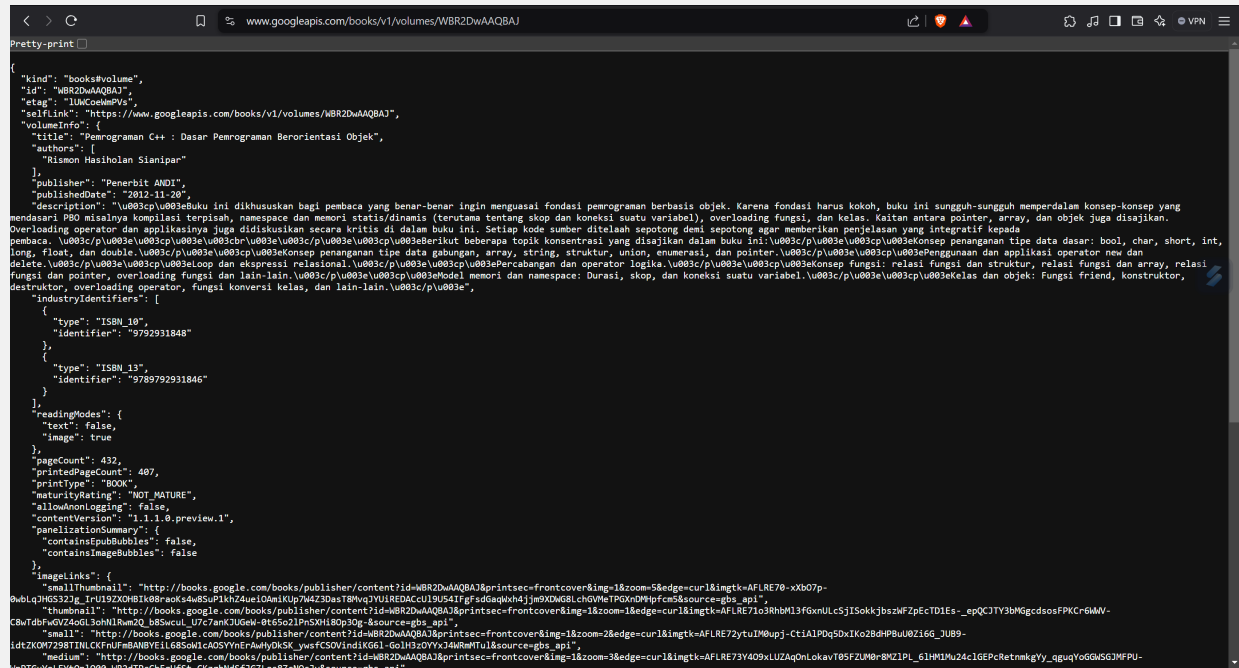
```

Future<Response> getData() async {
  const authority = 'www.googleapis.com';
  const path = '/books/v1/volumes/WBR2DwAAQBAJ';
  Uri url = Uri.https(authority, path);
  return http.get(url);
}

```

## Soal 2

Carilah judul buku favorit Anda di Google Books, lalu ganti ID buku pada variabel path di kode tersebut.



## Langkah 5: Tambah kode di ElevatedButton

Tambahkan kode pada onPressed di ElevatedButton.

```
ElevatedButton(  
  onPressed: () {  
    setState(() {});  
    getData().then((value) {  
      result = value.body.toString().substring(0, 450);  
      setState(() {});  
    }).catchError((_) {  
      result = 'An error occurrrred';  
      setState(() {});  
    });  
  }, child: const Text('GO!')),
```

Lakukan run aplikasi Flutter Anda.

## Soal 3

- Jelaskan maksud kode langkah 5 tersebut terkait substring dan catchError!
  - `substring(0, 450)` berguna untuk membatasi panjang teks yang akan ditampilkan, sehingga hanya 450 karakter pertama dari data yang diambil yang akan ditampilkan dalam variabel result.
  - `catchError` adalah metode yang digunakan untuk menangani kesalahan yang terjadi saat pemanggilan fungsi getData().

- Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan "W12: Soal 3".



## Praktikum 2: Menggunakan await/async untuk menghindari callbacks

## Langkah 1: Buka file main.dart

Tambahkan tiga method berisi kode seperti berikut di dalam class `_FuturePageState`.

```
Future<int> returnOneAsync() async {  
  await Future.delayed(const Duration(seconds: 3));  
  return 1;  
}  
  
Future<int> returnTwoAsync() async {  
  await Future.delayed(const Duration(seconds: 3));  
  return 2;  
}  
  
Future<int> returnThreeAsync() async {  
  await Future.delayed(const Duration(seconds: 3));  
  return 3;  
}
```

## Langkah 2: Tambah method count()

Lalu tambahkan lagi method ini di bawah ketiga method sebelumnya.

```
Future count() async {  
  int total = 0;  
  total = await returnOneAsync();  
  total += await returnTwoAsync();  
  total += await returnThreeAsync();  
  setState(() {  
    result = total.toString();  
  });  
}
```

## Langkah 3: Panggil count()

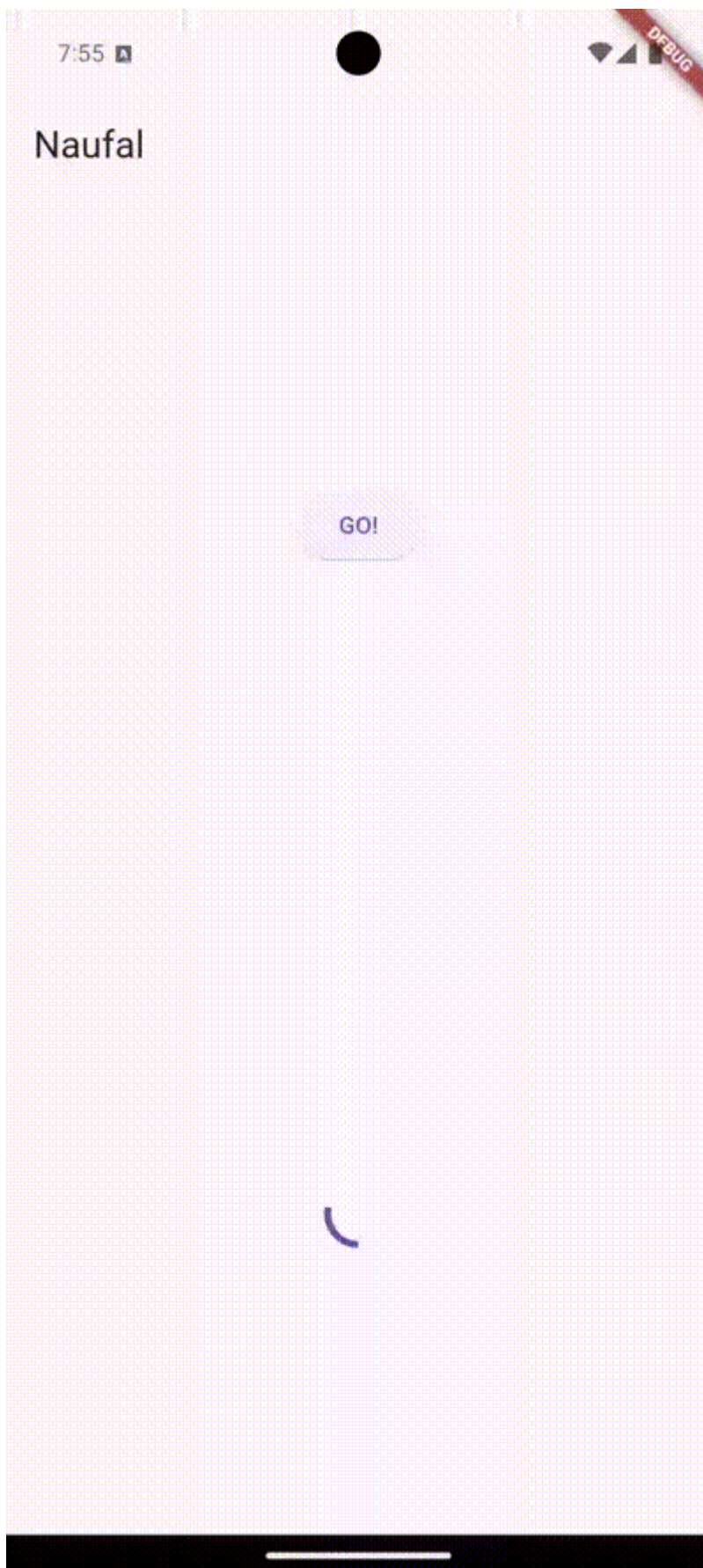
Lakukan comment kode sebelumnya, ubah isi kode `onPressed()` menjadi seperti berikut.

```
ElevatedButton(  
  onPressed: () {  
    // setState(() {});  
    // getData().then((value) {  
    //   result = value.body.toString().substring(0, 450);  
    //   setState(() {});  
    // }).catchError(_) {  
    //   result = 'An error occurred';  
    //   setState(() {});  
    // });  
  },  
)
```

```
count();  
}, child: const Text('GO!'))
```

#### Langkah 4: Run

Akhirnya, run atau tekan F5 jika aplikasi belum running. Maka Anda akan melihat seperti gambar berikut, hasil angka 6 akan tampil setelah delay 9 detik.



#### Soal 4

- Jelaskan maksud kode langkah 1 dan 2 tersebut!  
kode pada langkah 1 dan 2 di atas menunjukkan cara kerja pemanggilan **asynchronous**



secara berurutan di Flutter menggunakan Future dan await. Dengan mendefinisikan tiga fungsi `asynchronous` yang masing-masing mengembalikan nilai setelah jeda waktu, kemudian memanggil fungsi-fungsi tersebut secara berurutan dalam fungsi `count()`, kita dapat menghitung total hasil dari ketiga fungsi tersebut. Fungsi `count()` memastikan setiap pemanggilan `asynchronous` selesai sebelum melanjutkan ke pemanggilan berikutnya, menghasilkan total akhir yang ditampilkan di UI setelah semua operasi selesai.

- Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan "W12: Soal 4".

## Praktikum 3: Menggunakan Completer di Future

### Langkah 1: Buka main.dart

Pastikan telah impor package async berikut.

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

### Langkah 2: Tambahkan variabel dan method

Tambahkan variabel late dan method di class `_FuturePageState` seperti ini.

```
late Completer completer;

Future getNumber() {
  completer = Completer<int>();
  calculate();
  return completer.future;
}

Future calculate() async {
  await Future.delayed(const Duration(seconds : 5));
  completer.complete(42);
}
```

### Langkah 3: Ganti isi kode onPressed()

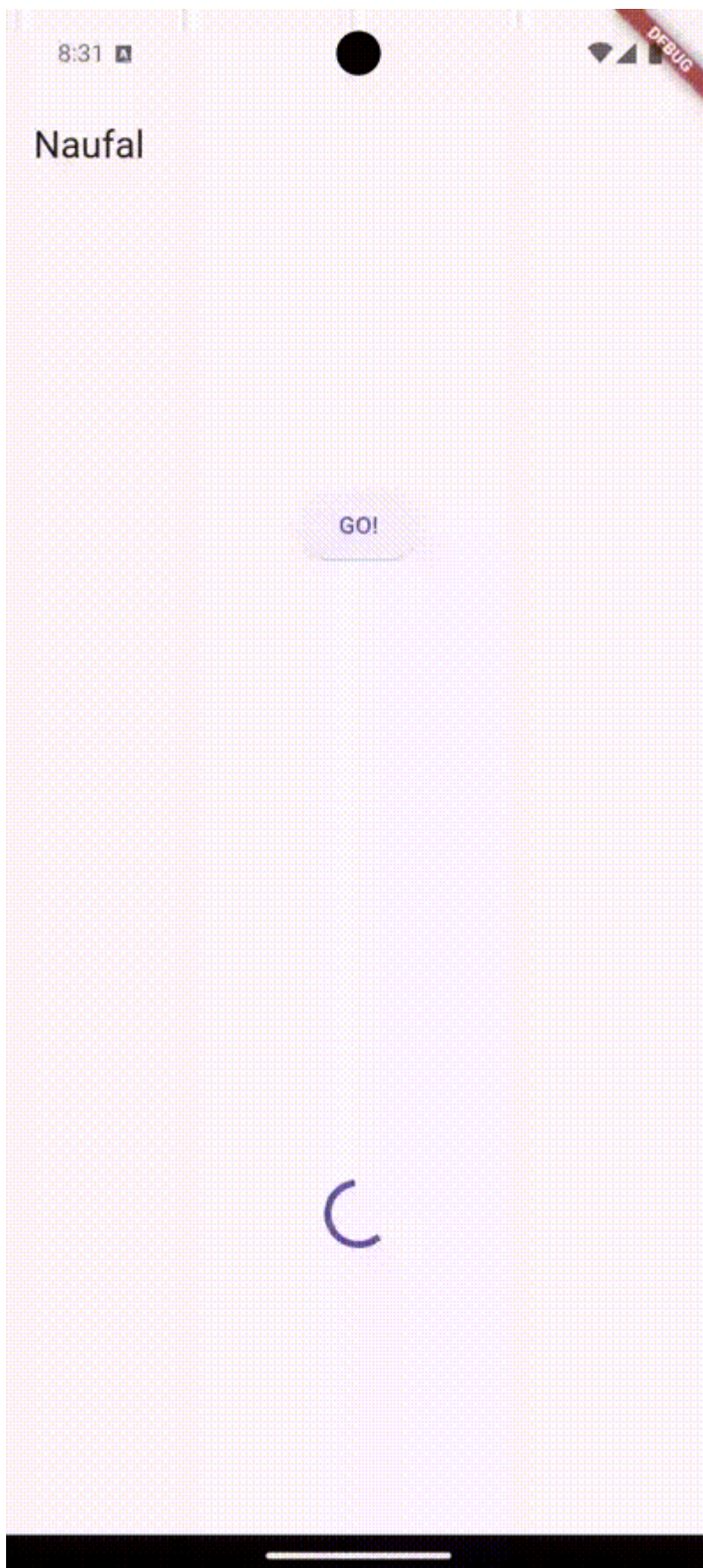
Tambahkan kode berikut pada fungsi `onPressed()`. Kode sebelumnya bisa Anda comment.

```
ElevatedButton(
  onPressed: () {
    // setState(() {});
    // getData().then((value) {
    //   result = value.body.toString().substring(0, 450);
    //   setState(() {});
    // }).catchError(_) {
    //   result = 'An error occurred';
    // }
```

```
//   setState(() {});  
// });  
// count();  
getNumber().then((value) {  
  setState(() {  
    result = value.toString();  
  });  
},);  
, child: const Text('GO!'))
```

#### Langkah 4:

Terakhir, run atau tekan F5 untuk melihat hasilnya jika memang belum running. Bisa juga lakukan hot restart jika aplikasi sudah running. Maka hasilnya akan seperti gambar berikut ini. Setelah 5 detik, maka angka 42 akan tampil.



#### Soal 5

- Jelaskan maksud kode langkah 2 tersebut!  
Kode tersebut menggunakan Completer untuk menghasilkan Future yang bisa dikendalikan

secara manual. Method `getNumber()` memulai proses asynchronous `calculate()`, yang setelah jeda waktu 5 detik, menyelesaikan Future dengan nilai 42. Teknik ini bermanfaat jika kita perlu menyelesaikan Future dengan cara atau waktu yang khusus, di luar kontrol otomatis dari mekanisme `async/await`.

- Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan "W12: Soal 5".

### Langkah 5: Ganti method calculate()

Gantilah isi code method `calculate()` seperti kode berikut, atau Anda dapat membuat `calculate2()`

```
calculate() async {
  try {
    await new Future.delayed(const Duration(seconds: 5));
    completer.complete(42);
  } catch (_) {
    completer.completeError({});
  }
}
```

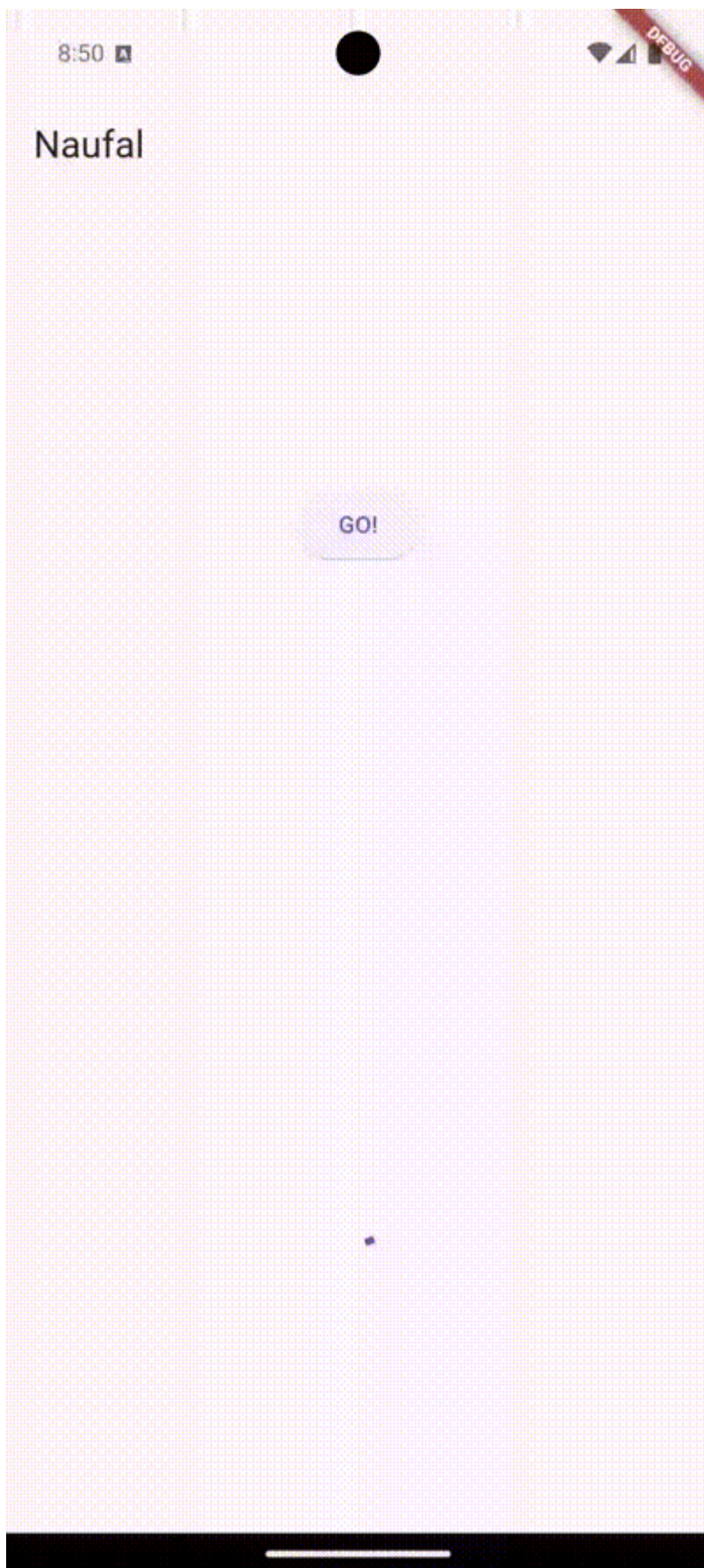
### Langkah 6: Pindah ke onPressed()

Ganti menjadi kode seperti berikut.

```
getNumber().then((value) {
  setState(() {
    result = value.toString();
  });
}).catchError((e) {
  result = 'An error occurred';
});
```

### Soal 6

- Jelaskan maksud perbedaan kode langkah 2 dengan langkah 5-6 tersebut! Modifikasi pada langkah 5 dan 6 lebih memfokuskan pada pemrosesan hasil Future di UI dan menambah penanganan error, sementara langkah 2 hanya membuat Future dengan Completer untuk mengatur penyelesaiannya secara manual tanpa langsung menampilkan hasil di UI atau menangani error.
- Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan "W12: Soal 6".



## Praktikum 4: Memanggil Future secara paralel

Langkah 1: Buka file main.dart

Tambahkan method ini ke dalam class `_FuturePageState`

```
void returnFG() {
    FutureGroup<int> futureGroup = FutureGroup<int>();
    futureGroup.add(returnOneAsync());
    futureGroup.add(returnTwoAsync());
    futureGroup.add(returnThreeAsync());
    futureGroup.close();
    futureGroup.future.then((value) {
        int total = 0;
        for (var element in value) {
            total += element;
        }
        setState(() {
            result = total.toString();
        });
    },);
}
```

## Langkah 2: Edit onPressed()

Anda bisa hapus atau comment kode sebelumnya, kemudian panggil method dari langkah 1 tersebut.

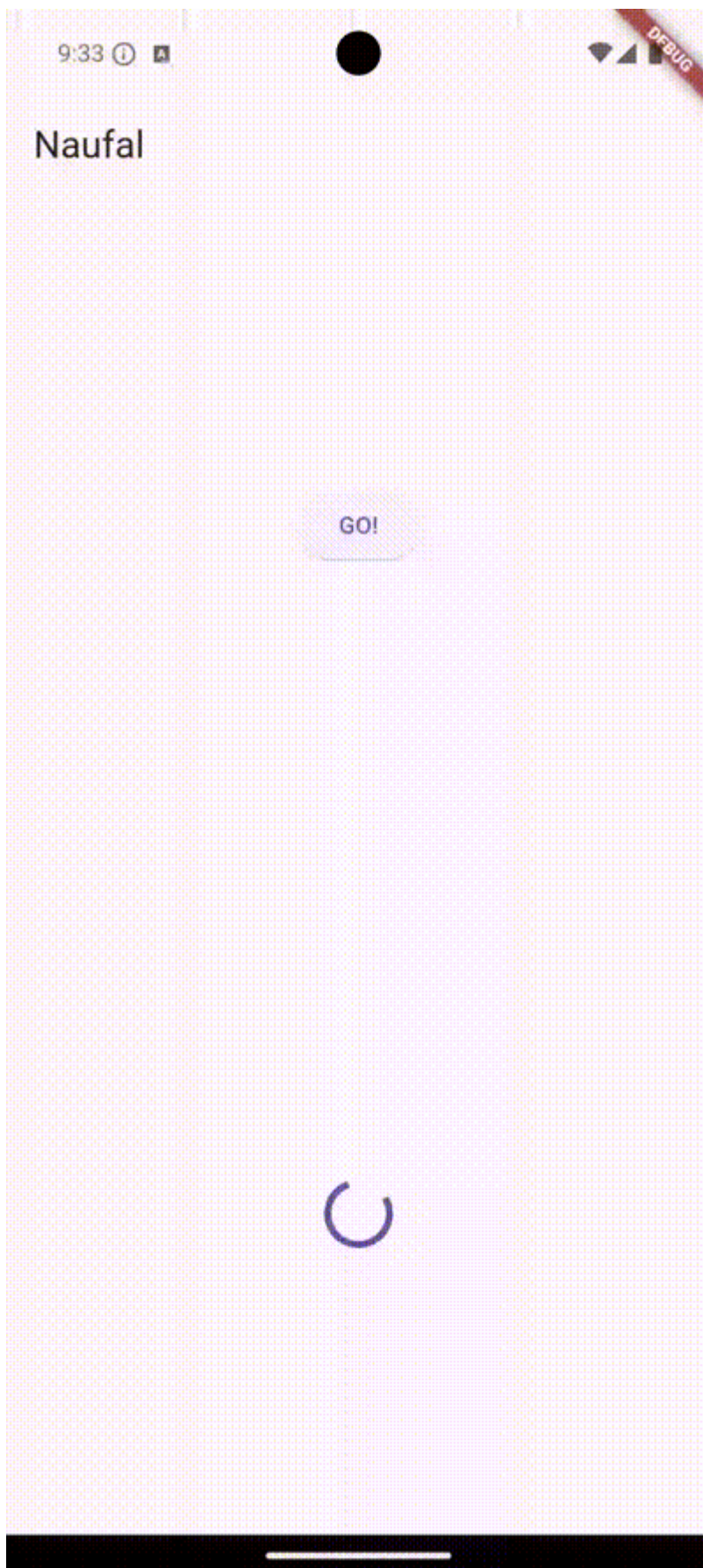
```
ElevatedButton(
    onPressed: () {
        returnFG();
    }, child: const Text('GO!')),
```

## Langkah 3: Run

Anda akan melihat hasilnya dalam 3 detik berupa angka 6 lebih cepat dibandingkan praktikum sebelumnya menunggu sampai 9 detik.

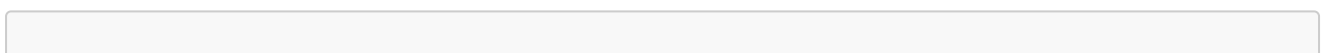
### Soal 7

Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan "W12: Soal 7".



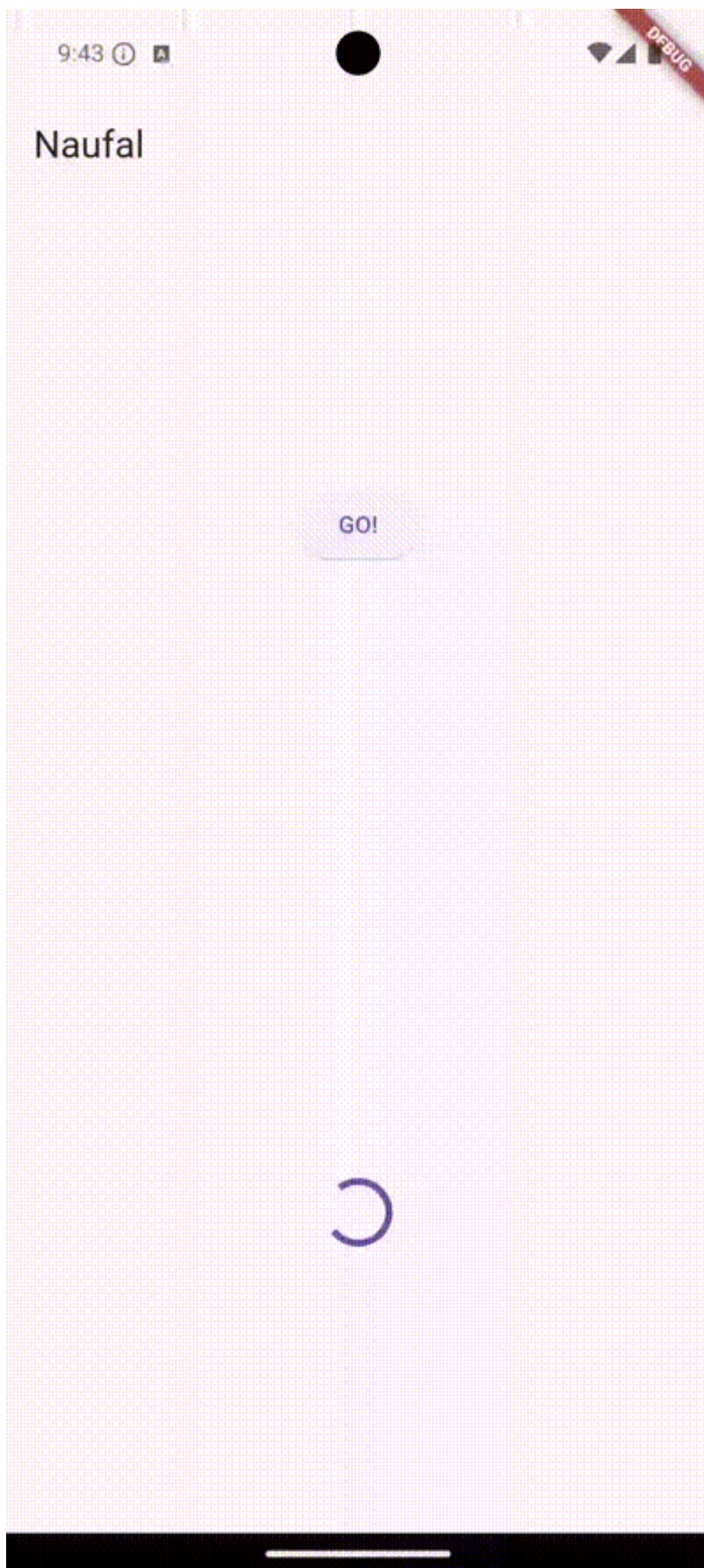
Langkah 4: Ganti variabel futureGroup

Anda dapat menggunakan FutureGroup dengan `Future.wait` seperti kode berikut.



```
final futures = Future.wait<int>([  
  returnOneAsync(),  
  returnTwoAsync(),  
  returnThreeAsync(),  
]);
```





#### Soal 8

Jelaskan maksud perbedaan kode langkah 1 dan 4!

- Fleksibilitas: FutureGroup memungkinkan penambahan Future secara dinamis, sedangkan Future.wait bekerja dengan daftar Future yang tetap.
- Kompleksitas dan Kinerja: Future.wait lebih sederhana dan cocok untuk tugas paralel yang sudah terstruktur dengan baik, sementara FutureGroup lebih cocok untuk kasus di mana jumlah Future bisa berubah selama eksekusi.

Keduanya sama-sama menjalankan Future secara paralel, namun FutureGroup memberikan kontrol lebih dinamis, sedangkan Future.wait lebih langsung dan efisien untuk situasi dengan daftar Future yang tetap.

Praktikum 5: Menangani Respon Error pada Async Code

Praktikum 6: Menggunakan Future dengan StatefulWidget

Praktikum 7: Manajemen Future dengan FutureBuilder

Praktikum 8: Navigation route dengan Future Function

Praktikum 9: Memanfaatkan async/await dengan Widget Dialog