#13 | Lanjutan State Management dengan Streams

Praktikum 1: Dart Stream

Langkah 1: Buat Project Baru

Buatlah sebuah project flutter baru dengan nama `stream_nama` (beri nama panggilan Anda) di folder `week-13/src/` repository GitHub Anda.

Langkah 2: Buka file main.dart

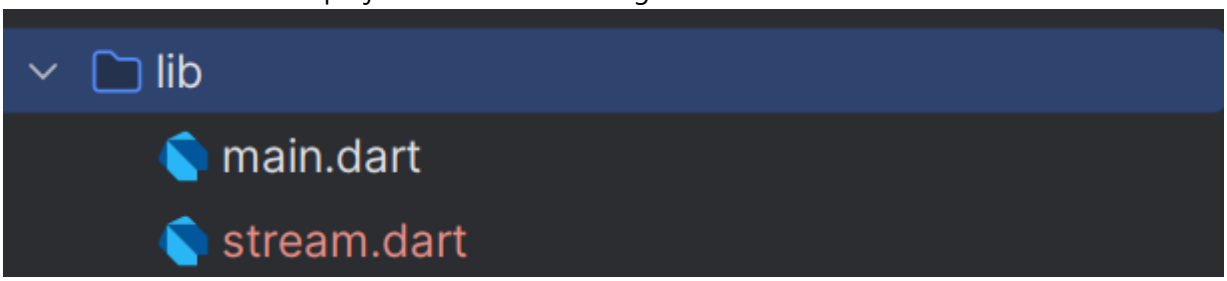
```
import 'package:flutter/material.dart';

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Stream [Naufal]',
      theme: ThemeData(primarySwatch: Colors.deepPurple),
      home: const StreamHomePage()
    );
  }
}

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

  @override
  State<StreamHomePage> createState() => _StreamHomePageState();
}

class _StreamHomePageState extends State<StreamHomePage> {
  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

> Soal 1
>
> - Tambahkan nama panggilan Anda pada title app sebagai identitas hasil pekerjaan Anda.
> - Gantilah warna tema aplikasi sesuai kesukaan Anda.

- Lakukan commit hasil jawaban Soal 1 dengan pesan "W13: Jawaban Soal 1"

Langkah 3: Buat file baru stream.dart

Buat file baru di folder lib project Anda. Lalu isi dengan kode berikut.




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

class ColorStream {

}
```

Langkah 4: Tambah variabel colors

Tambahkan variabel di dalam class ColorStream seperti berikut.

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

class ColorStream {
  final List<Color> colors = [
    const Color(0xffE63946),
    const Color(0xfff1faee),
    const Color(0xffa8dadc),
    const Color(0xff457b9d),
    const Color(0xff1d3557),
  ];
}
```

Soal 2

- Tambahkan 5 warna lainnya sesuai keinginan Anda pada variabel colors tersebut.
- Lakukan commit hasil jawaban Soal 2 dengan pesan "W13: Jawaban Soal 2"

Langkah 5: Tambah method getColors()

Di dalam class ColorStream ketik method seperti kode berikut. Perhatikan tanda bintang di akhir keyword `async*` (ini digunakan untuk melakukan Stream data)

```
Stream<Color> getColors() async* {

}
```

Langkah 6: Tambah perintah yield*

Tambahkan kode berikut ini.

```
Stream<Color> getColors() async* {
    yield* Stream.periodic(
        const Duration(seconds: 1), (int t) {
        int index = t % colors.length;
        return colors[index];
    });
}
```

> Soal 3
>
> - Jelaskan fungsi keyword yield* pada kode tersebut!
>   `yield*` meneruskan seluruh elemen dari *stream* `Stream.periodic(...)` ke dalam *stream* `getColors()` secara otomatis, menghasilkan nilai tanpa harus menulis `yield` berulang kali.
> - Apa maksud isi perintah kode tersebut?
>   Kode tersebut menghasilkan warna dari daftar `colors` setiap detik, berulang dari awal daftar setelah mencapai warna terakhir, sehingga membentuk pola warna berulang.
> - Lakukan commit hasil jawaban Soal 3 dengan pesan "W13: Jawaban Soal 3"

Langkah 7: Buka main.dart

Ketik kode impor file ini pada file main.dart

```
import 'stream.dart';
```

Langkah 8: Tambah variabel

Ketik dua properti ini di dalam `class _StreamHomePageState`

```
class _StreamHomePageState extends State<StreamHomePage> {
  Color bgColor = const Color(0xffE63946);
  late ColorStream colorStream;

  @override
  Widget build(BuildContext context) {
    return Container();
```

```
  }
}
```

Langkah 9: Tambah method changeColor()

Tetap di file main, Ketik kode seperti berikut

```
void changeColor() async {
    await for (var eventColor in colorStream.getColors()) {
        setState(() {
        bgColor = eventColor;
        });
    }
}
```

Langkah 10: Lakukan override initState()

Ketika kode seperti berikut

```
@override
void initState() {
    super.initState();
    colorStream = ColorStream();
    changeColor();
}
```

Langkah 11: Ubah isi Scaffold()

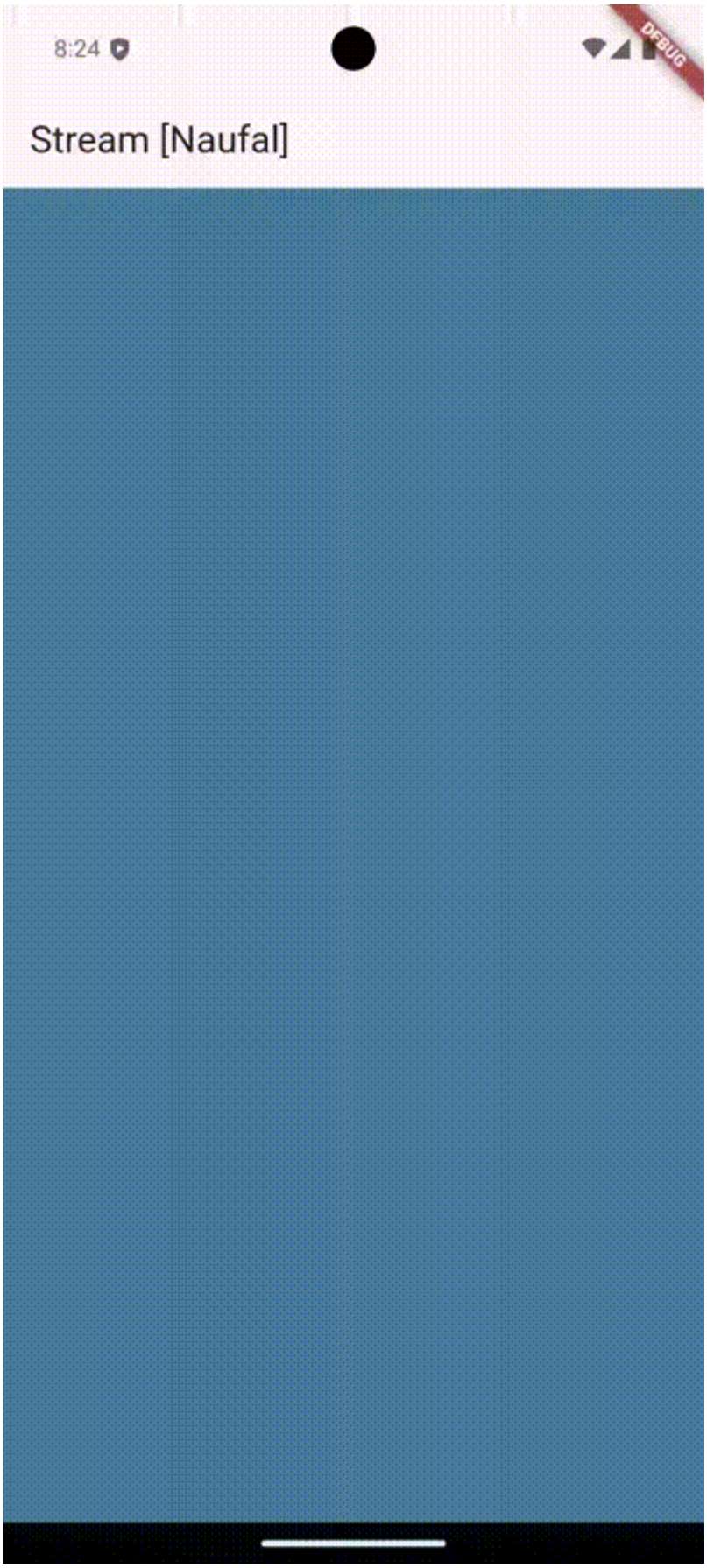
Sesuaikan kode seperti berikut.

```
@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
        title: const Text("Stream [Naufal]"),
        ),
        body: Container(
        decoration: BoxDecoration(
            color: bgColor
        ),
        ),
    );
}
```

Langkah 12: Run

Lakukan running pada aplikasi Flutter Anda, maka akan terlihat berubah warna background setiap detik.

> Soal 4
>
> - Capture hasil praktikum Anda berupa GIF dan lampirkan di README.
> - Lakukan commit hasil jawaban Soal 4 dengan pesan "W13: Jawaban Soal 4"



Langkah 13: Ganti isi method changeColor()

Anda boleh comment atau hapus kode sebelumnya, lalu ketika kode seperti berikut.

```
void changeColor() async {
    colorStream.getColors().listen((eventColor) {
        setState(() {
        bgColor = eventColor;
        });
    },);
}
```

> Soal 5
>
> - Jelaskan perbedaan menggunakan listen dan await for (langkah 9) !
>   - `await for` digunakan untuk mengiterasi setiap nilai *stream* secara berurutan, sambil menunggu nilai berikutnya diproses satu per satu secara sinkron.
>   - `listen` menggunakan *callback* untuk menangani data secara asinkron, memungkinkan UI atau variabel diperbarui segera saat data diterima, tanpa perlu menunggu setiap nilai selesai diproses.
> - Lakukan commit hasil jawaban Soal 5 dengan pesan "W13: Jawaban Soal 5"

Praktikum 2: Stream controllers dan sinks

Langkah 1: Buka file stream.dart

Lakukan impor dengan mengetik kode ini.

```
import 'dart:async';
```

Langkah 2: Tambah class NumberStream

Tetap di file `stream.dart` tambah class baru seperti berikut.

```
class NumberStream {

}
```

Langkah 3: Tambah StreamController

Di dalam `class NumberStream` buatlah variabel seperti berikut.

```
final StreamController<int> controller = StreamController<int>();
```

Langkah 4: Tambah method addNumberToSink

Tetap di `class NumberStream` buatlah method ini

```
void addNumberToSink(int newNumber) {
    controller.sink.add(newNumber);
}
```

Langkah 5: Tambah method close()

```
void close() {
    controller.close();
}
```

Langkah 6: Buka main.dart

Ketik kode import seperti berikut

```
import 'dart:async';
import 'dart:math';
```

Langkah 7: Tambah variabel

Di dalam `class _StreamHomePageState` ketik variabel berikut

```
int lastNumber = 0;
late StreamController numberStreamController;
late NumberStream numberStream;
```

Langkah 8: Edit initState()

```
@override
void initState() {
    numberStream = NumberStream();
    numberStreamController = numberStream.controller;
    Stream stream = numberStreamController.stream;
    stream.listen((event) {
        setState(() {
        lastNumber = event;
        });
    },);
    super.initState();
}
```

Langkah 9: Edit dispose()

```
@override
void dispose() {
    numberStreamController.close();
    super.dispose();
}
```

Langkah 10: Tambah method addRandomNumber()

```
void addRandomNumber() {
  Random random = Random();
  int myNum = random.nextInt(10);
  numberStream.addNumberToSink(myNum);
}
```

Langkah 11: Edit method build()

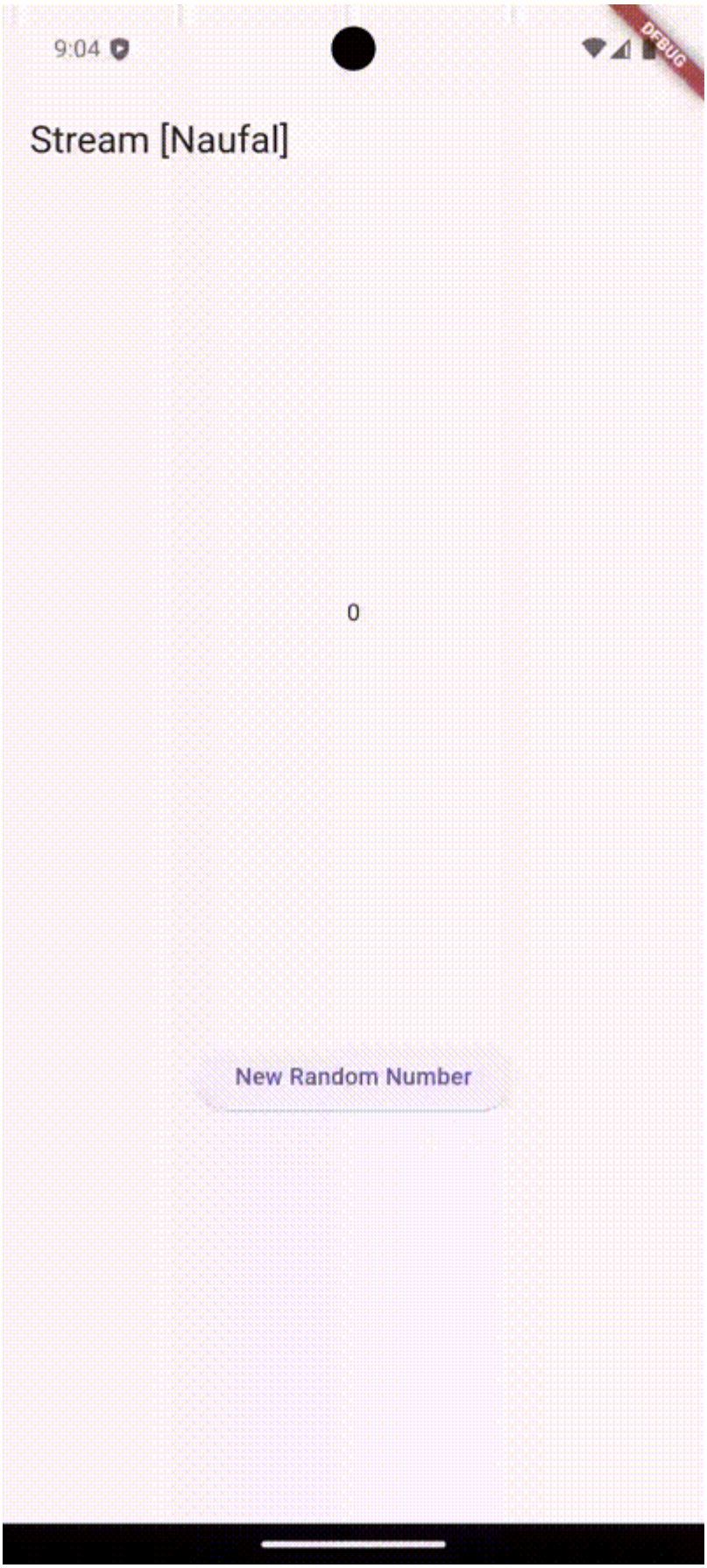
```
@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
        title: const Text("Stream [Naufal]"),
        ),
        body: SizedBox(
        width: double.infinity,
        child: Column(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
            Text(lastNumber.toString()),
            ElevatedButton(
                onPressed: () => addRandomNumber(),
                child: const Text('New Random Number')
            )
            ],
        ),
        ),
    );
}
```

Langkah 12: Run

Lakukan running pada aplikasi Flutter Anda, maka akan terlihat seperti gambar berikut.

> Soal 6

- Jelaskan maksud kode langkah 8 dan 10 tersebut!
  Jawab:
  - Langkah 8: initState mendengarkan stream untuk memperbarui UI setiap kali angka baru ditambahkan, memastikan nilai lastNumber selalu tampak terkini.
  - Langkah 10: addRandomNumber menghasilkan angka acak dan menambahkannya ke stream, memungkinkan angka baru ditampilkan di UI setiap kali tombol ditekan.
- Capture hasil praktikum Anda berupa GIF dan lampirkan di README.
- Lalu lakukan commit dengan pesan "W13: Jawaban Soal 6".



Langkah 13: Buka stream.dart

Tambahkan method berikut ini.

```
addError() {
    controller.sink.addError('error');
}
```

Langkah 14: Buka main.dart

Tambahkan method onError di dalam class StreamHomePageState pada method listen di fungsi initState() seperti berikut ini.

```
@override
void initState() {
    numberStream = NumberStream();
    numberStreamController = numberStream.controller;
    Stream stream = numberStreamController.stream;
    stream.listen((event) {
        setState(() {
        lastNumber = event;
        });
    },).onError((error){
        setState(() {
        lastNumber = -1;
        });
    });
    super.initState();
}
```

Langkah 15: Edit method addRandomNumber()

Lakukan comment pada dua baris kode berikut, lalu ketik kode seperti berikut ini.

```
void addRandomNumber() {
    Random random = Random();
    // int myNum = random.nextInt(10);
    // numberStream.addNumberToSink(myNum);
    numberStream.addError();
}
```

> Soal 7
>
> - Jelaskan maksud kode langkah 13 sampai 15 tersebut!
>
>   **Jawab**:
>   - Langkah 13: `addError` menambahkan kesalahan ke dalam stream, yang nantinya dapat ditangani oleh `listener`.
>   - Langkah 14: `onError` pada listen menangani kesalahan di stream dan memperbarui `lastNumber` menjadi -1 untuk menampilkan indikator kesalahan di UI.

    - Langkah 15: `addRandomNumber` diubah untuk memicu kesalahan pada stream alih-alih menambahkan angka, sehingga UI menampilkan indikator kesalahan setiap kali tombol ditekan.
- Kembalikan kode seperti semula pada Langkah 15, comment `addError()` agar Anda dapat melanjutkan ke praktikum 3 berikutnya.
- Lalu lakukan commit dengan pesan "W13: Jawaban Soal 7".

Praktikum 3: Injeksi data ke streams

Langkah 1: Buka main.dart

Tambahkan variabel baru di dalam `class _StreamHomePageState`

```
late StreamTransformer transformer;
```

Langkah 2: Tambahkan kode ini di initState

```
transformer = StreamTransformer<int, int>.fromHandlers(
    handleData: (value, sink) {
    sink.add(value * 10);
    },
    handleError: (error, trace, sink) {
    sink.add(-1);
    },
    handleDone: (sink) {
    sink.close();
    },
);
```

Langkah 3: Tetap di initState

Lakukan edit seperti kode berikut.

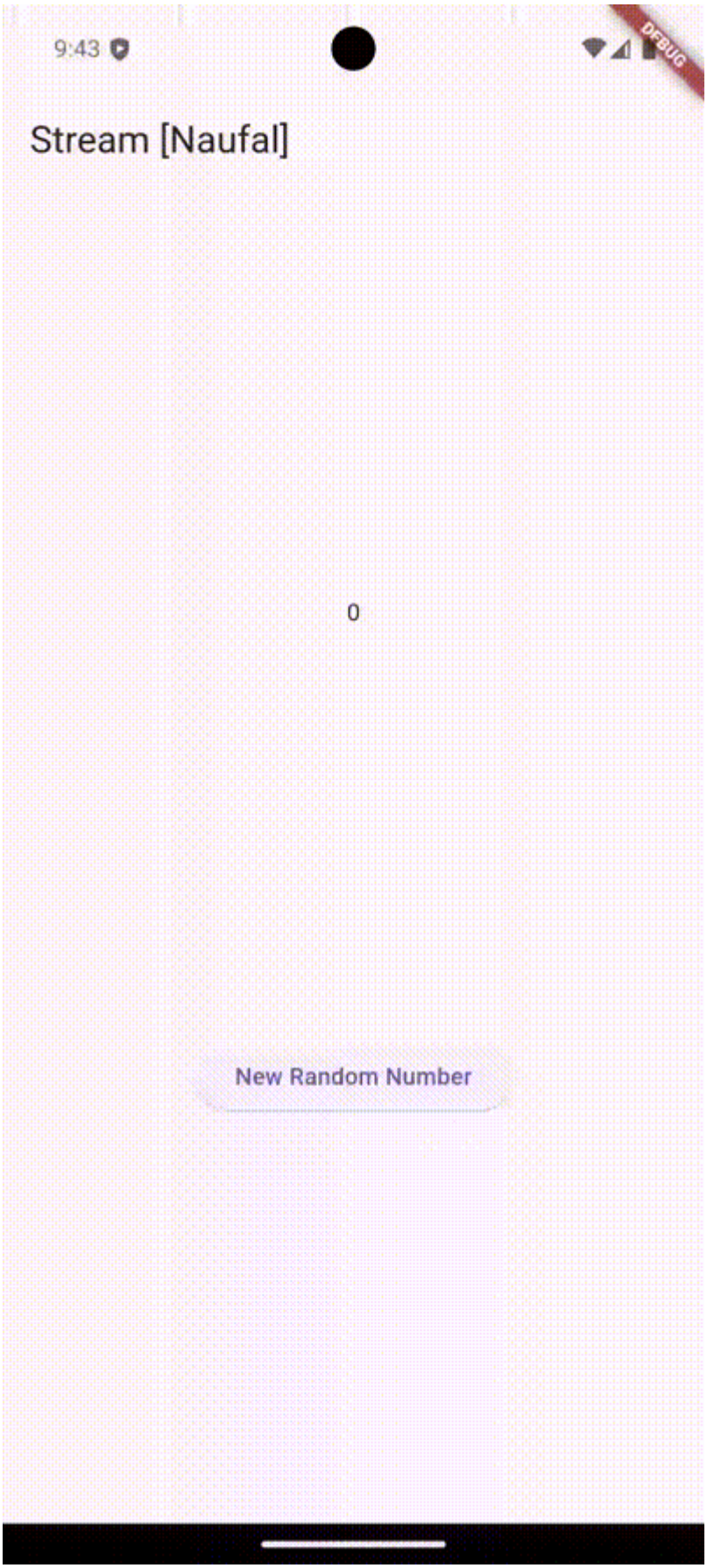
```
@override
void initState() {
    numberStream = NumberStream();
    numberStreamController = numberStream.controller;
    Stream stream = numberStreamController.stream;
    stream.transform(transformer).listen((event) {
        setState(() {
        lastNumber = event;
        });
    },).onError((error){
        setState(() {
        lastNumber = -1;
        });
```

```
    });
    transformer = StreamTransformer<int, int>.fromHandlers(
        handleData: (value, sink) {
        sink.add(value * 10);
        },
        handleError: (error, trace, sink) {
        sink.add(-1);
        },
        handleDone: (sink) {
        sink.close();
        },
    );
    super.initState();
}
```

Langkah 4: Run

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. Anda akan melihat tampilan angka dari 0 hingga 90.

> Soal 8
>
> - Jelaskan maksud kode langkah 1-3 tersebut!
>   **Jawab**:
>   1. Deklarasi Transformer: `StreamTransformer` dideklarasikan untuk memproses data stream sebelum diterima oleh listener.
>   2. Inisialisasi Transformer: Mengalikan data dengan 10 (jika ada kesalahan, mengirimkan -1), dan menutup stream saat selesai.
>   3. Penerapan Transformer pada Stream: Stream dimodifikasi oleh transformer sebelum hasilnya ditampilkan di UI.
> - Capture hasil praktikum Anda berupa GIF dan lampirkan di README.
> - Lalu lakukan commit dengan pesan "W13: Jawaban Soal 8".



Praktikum 4: Subscribe ke stream events

Langkah 1: Tambah variabel

Tambahkan variabel berikut di `class _StreamHomePageState`

```
late StreamSubscription subscription;
```

Langkah 2: Edit `initState()`

Edit kode seperti berikut ini.

```
@override
  void initState() {
    numberStream = NumberStream();
    numberStreamController = numberStream.controller;
    Stream stream = numberStreamController.stream;
    subscription = stream.listen(
      (event) {
        setState(() {
          lastNumber = event;
        });
      },
    );
}
```

Langkah 3: Tetap di `initState()`

Tambahkan kode berikut ini.

```
@override
  void initState() {
    numberStream = NumberStream();
    numberStreamController = numberStream.controller;
    Stream stream = numberStreamController.stream;
    subscription = stream.listen(
      (event) {
        setState(() {
          lastNumber = event;
        });
      },
    );
    subscription.onError((error) {
      setState(() {
        lastNumber = -1;
      });
    });
}
```

Langkah 4: Tambah properti `onDone()`

Tambahkan dibawahnya kode ini setelah `onError`

```
@override
void initState() {
    numberStream = NumberStream();
    numberStreamController = numberStream.controller;
    Stream stream = numberStreamController.stream;
    subscription = stream.listen(
        (event) {
        setState(() {
            lastNumber = event;
        });
        },
    );
    subscription.onError((error) {
        setState(() {
        lastNumber = -1;
        });
    });
    subscription.onDone(() {
        print('OnDone was called');
    });
    super.initState();
}
```

Langkah 5: Tambah method baru

Ketik method ini di dalam `class _StreamHomePageState`

```
void stopStream() {
    numberStreamController.close();
}
```

Langkah 6: Pindah ke method `dispose()`

Jika method `dispose()` belum ada, Anda dapat mengetiknya dan dibuat override. Ketik kode ini didalamnya.

```
@override
void dispose() {
    numberStreamController.close();
    subscription.cancel();
    super.dispose();
}
```

Langkah 7: Pindah ke method `build()`

Tambahkan button kedua dengan isi kode seperti berikut ini.

```
children: [
    Text(lastNumber.toString()),
    ElevatedButton(
        onPressed: () => addRandomNumber(),
        child: const Text('New Random Number')),
    ElevatedButton(
        onPressed: () => stopStream(),
        child: const Text('Stop Subscription')),
],
```

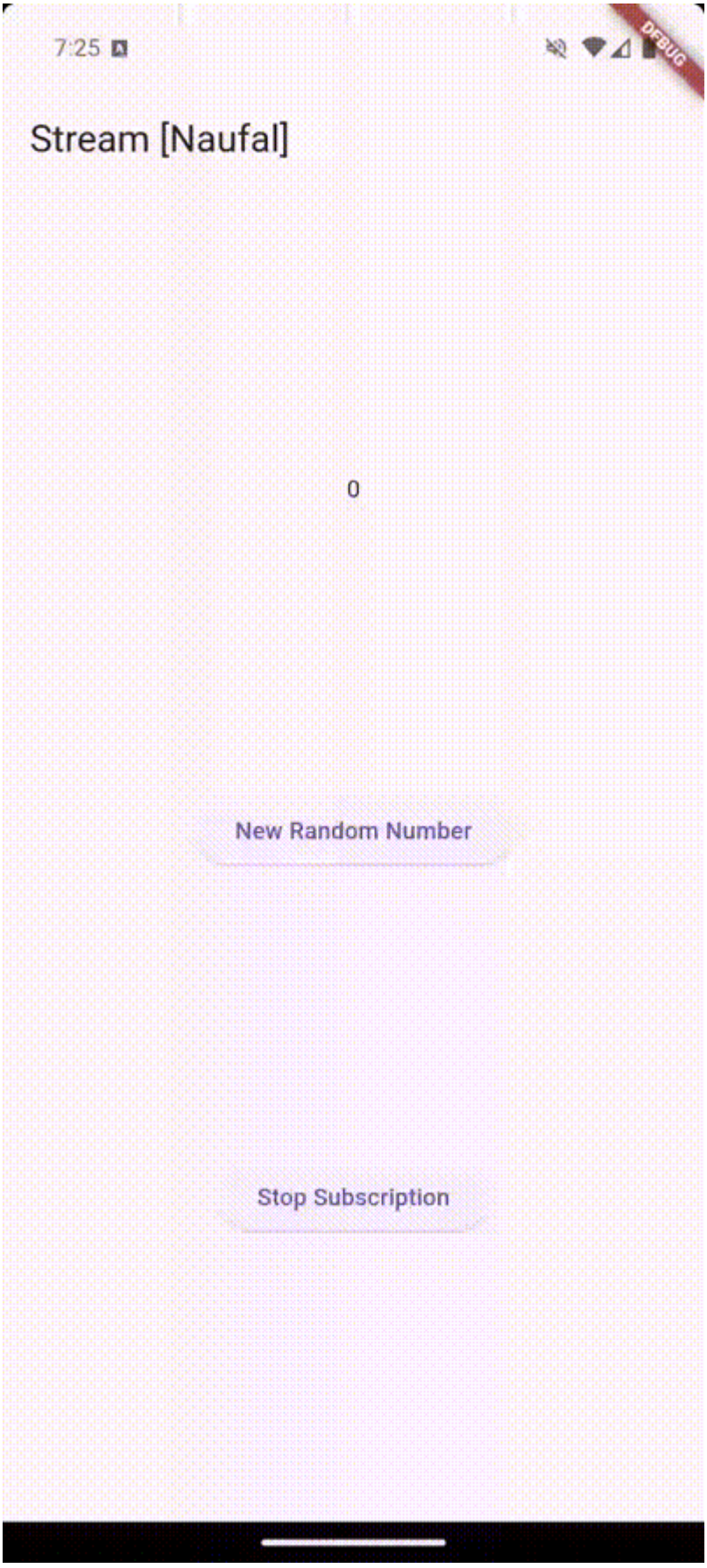
Langkah 8: Edit method `addRandomNumber()`

Edit kode seperti berikut ini.

```
void addRandomNumber() {
    Random random = Random();
    int myNum = random.nextInt(10);
    if (!numberStreamController.isClosed) {
        numberStream.addNumberToSink(myNum);
    } else {
        setState(() {
        lastNumber = -1;
        });
    }
        }
```

Langkah 9: Run

Anda akan melihat dua button seperti gambar berikut.



Langkah 10: Tekan button `Stop Subscription`

Anda akan melihat pesan di Debug Console seperti berikut.

```
D/EGL_emulation(13342): app_time_stats: avg=73.98ms min=7.22ms max=1067.27ms count=22
I/flutter (13342): OnDone was called
```

> Soal 9
>
> - Jelaskan maksud kode langkah 2, 6 dan 8 tersebut!
>   - Langkah 2 (initState): Mendengarkan data dari stream menggunakan subscription.listen, memperbarui nilai lastNumber secara real-time saat stream mengirimkan data baru.
>   - Langkah 6 (dispose): Membersihkan sumber daya dengan menutup numberStreamController dan membatalkan subscription untuk mencegah kebocoran memori.
>   - Langkah 8 (addRandomNumber): Menambahkan angka acak ke stream hanya jika stream belum ditutup; jika stream sudah ditutup, menampilkan indikator -1 sebagai status.
> - Capture hasil praktikum Anda berupa GIF dan lampirkan di README.
> - Lalu lakukan commit dengan pesan "W12: Jawaban Soal 9".

Praktikum 5: Multiple stream subscriptions

Langkah 1: Buka file main.dart

Ketik variabel berikut di class _StreamHomePageState

```
late StreamSubscription subscription2;
String values = '';
```

Langkah 2: Edit initState()

Ketik kode seperti berikut.

```
subscription = stream.listen(
  (event) {
    setState(() {
      values += '$event -';
    });
  },
);
subscription2 = stream.listen(
      (event) {
    setState(() {
      values += '$event -';
    });
  },
);
```

Langkah 3: Run

Lakukan run maka akan tampil error seperti gambar berikut.

> Soal 10
> Jelaskan mengapa error itu bisa terjadi ?
> **Jawab**:
> Karena stream dilakukan listen 2 kali

Langkah 4: Set broadcast stream

Ketik kode seperti berikut di method initState()

```
@override
void initState() {
  numberStream = NumberStream();
  numberStreamController = numberStream.controller;
  Stream stream = numberStreamController.stream.asBroadcastStream();
```

Langkah 5: Edit method build()

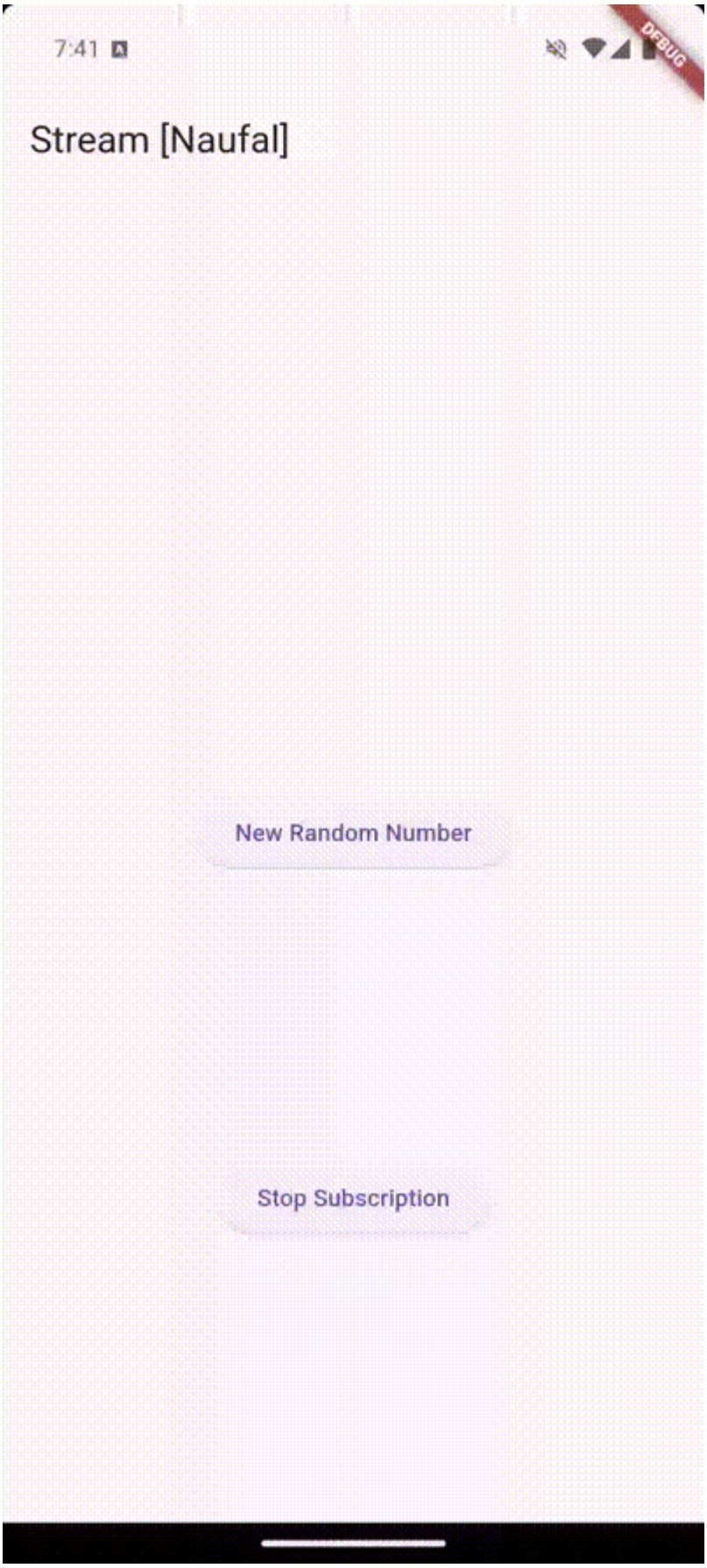
Tambahkan text seperti berikut

```
@override
Widget build(BuildContext context) {
```

```
    return Scaffold(
      appBar: AppBar(
        title: const Text("Stream [Naufal]"),
      ),
      body: SizedBox(
        width: double.infinity,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            Text(values),
            ElevatedButton(
                onPressed: () => addRandomNumber(),
                child: const Text('New Random Number')),
            ElevatedButton(
                onPressed: () => stopStream(),
                child: const Text('Stop Subscription')),
          ],
        ),
      ),
    );
  }
```

Langkah 6: Run

Tekan button `New Random Number` beberapa kali, maka akan tampil teks angka terus bertambah sebanyak dua kali.



Soal 11

- Jelaskan mengapa hal itu bisa terjadi ?
  Terjadi karena perubahan pada line `Stream stream =`

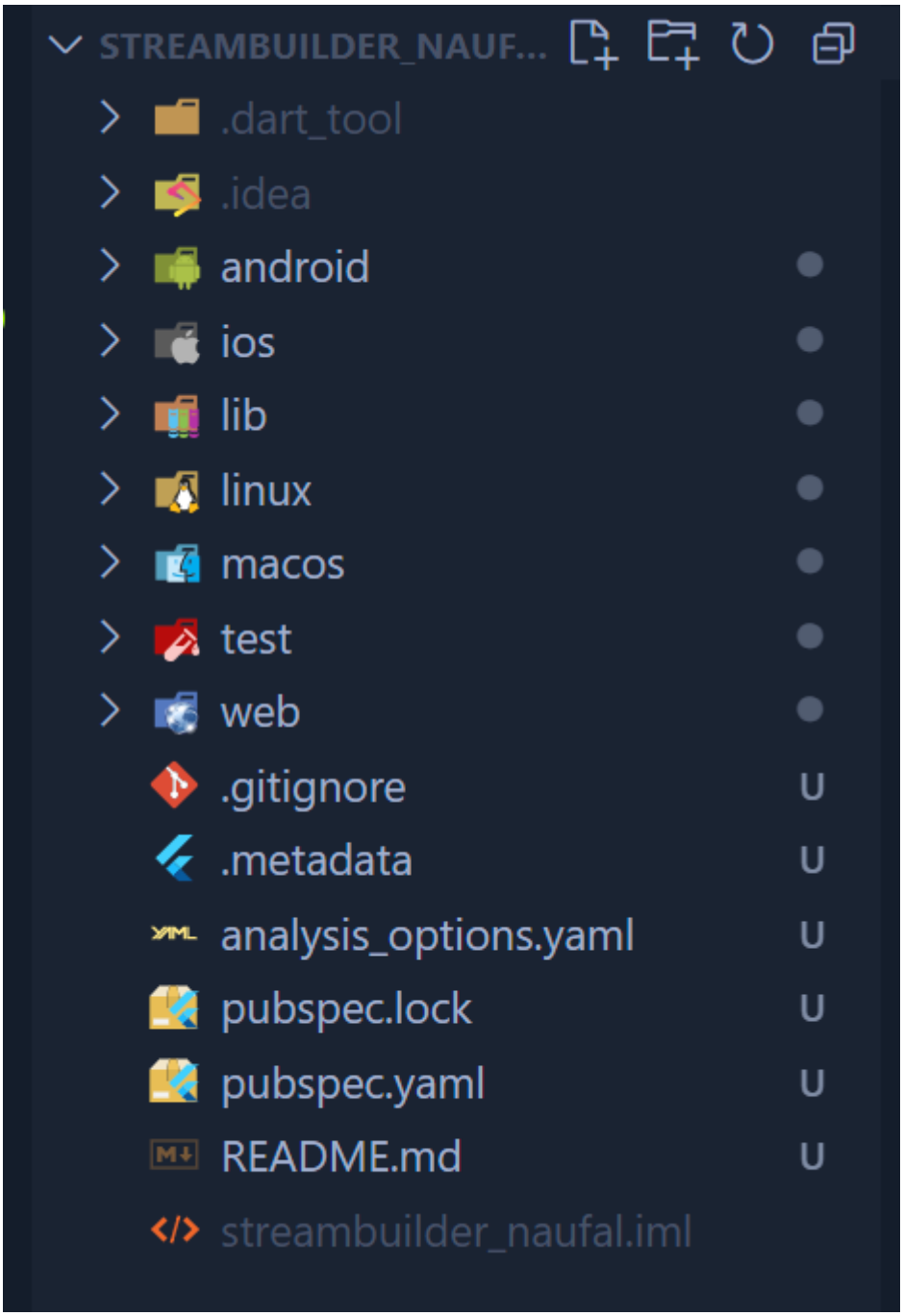
```
numberStreamController.stream.asBroadcastStream();.
```

- Capture hasil praktikum Anda berupa GIF dan lampirkan di README.
- Lalu lakukan commit dengan pesan "W12: Jawaban Soal 10,11".

Praktikum 6: StreamBuilder

Langkah 1: Buat Project Baru

Buatlah sebuah project flutter baru dengan nama streambuilder_nama (beri nama panggilan Anda) di folder week-12/src/ repository GitHub Anda.



Langkah 2: Buat file baru stream.dart

```
class NumberStream {
```

```
}
```

Langkah 3: Tetap di file stream.dart

```
import 'dart:math';

class NumberStream {
  Stream<int> getNumbers() async* {
    yield* Stream.periodic(
      const Duration(seconds: 1),
      (int t) {
        Random random = Random();
        int myNum = random.nextInt(10);
        return myNum;
      },
    );
  }
}
```

Langkah 4: Edit main.dart

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Stream',
      theme: ThemeData(primaryColor: Colors.deepPurple),
      home: const StreamHomePage(),
    );
  }
}

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

  @override
  State<StreamHomePage> createState() => _StreamHomePageState();
}

class _StreamHomePageState extends State<StreamHomePage> {
```

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Stream'),
      ),
      body: Container(

      ),
    );
  }
}
```

Langkah 5: Tambah variabel

Di dalam class _StreamHomePageState, ketika variabel ini.

```
late Stream<int> numberStream;
```

Langkah 6: Edit initState()

Ketik kode seperti berikut.

```
@override
void initState() {
  numberStream = NumberStream().getNumbers();
  super.initState();
}
```

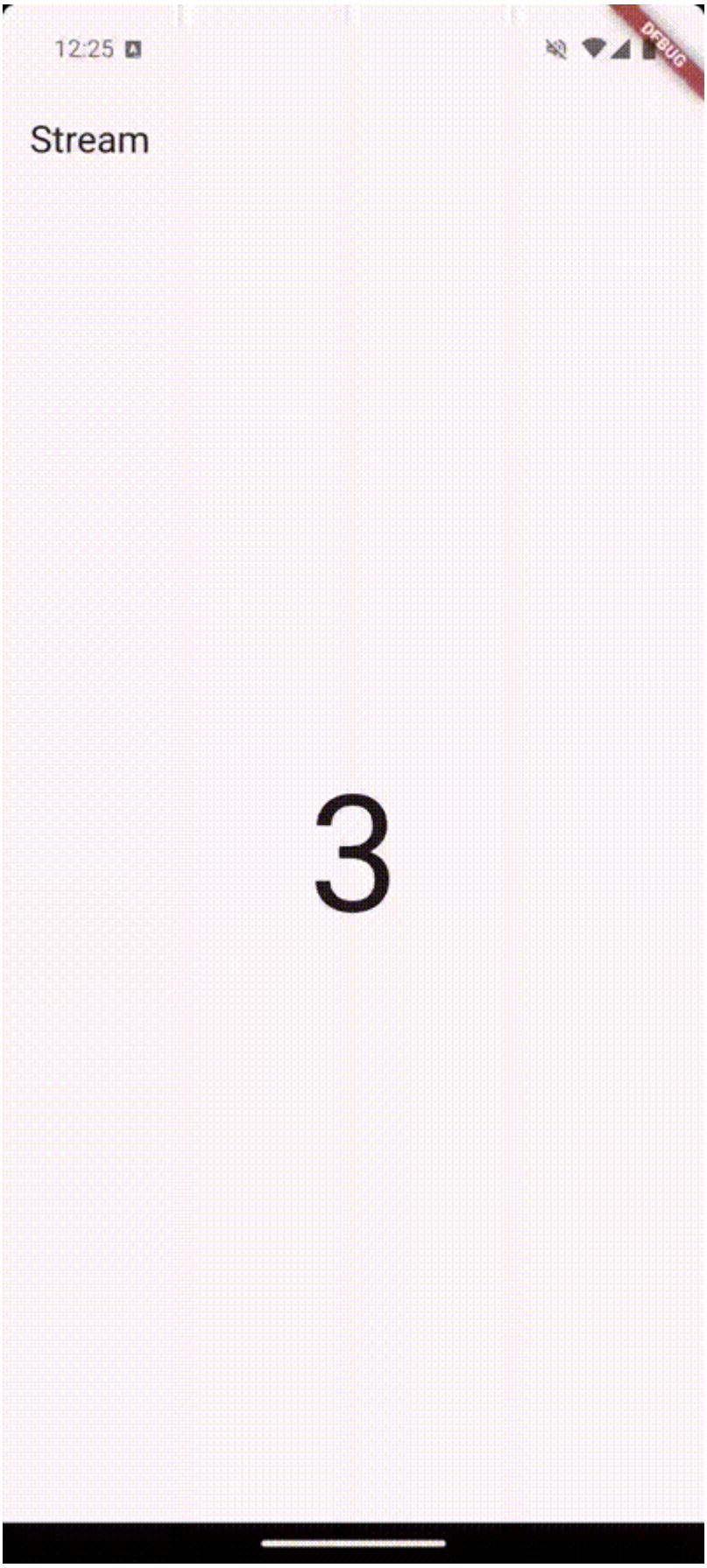
Langkah 7: Edit method build()

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Stream'),
    ),
    body: StreamBuilder(
      stream: numberStream,
      initialData: 0,
      builder: (context, snapshot) {
        if (snapshot.hasError) {
          print('Error!');
        }
        if (snapshot.hasData) {
          return Center(
```

```
              child: Text(
                snapshot.data.toString(),
                style: const TextStyle(fontSize: 96),
              ),
            );
          } else {
            return const SizedBox.shrink();
          }
        },
      ),
    );
  }
```

Langkah 8: Run

Hasilnya, setiap detik akan tampil angka baru seperti berikut.



Soal 12

- Jelaskan maksud kode pada langkah 3 dan 7 !
  Jawab:

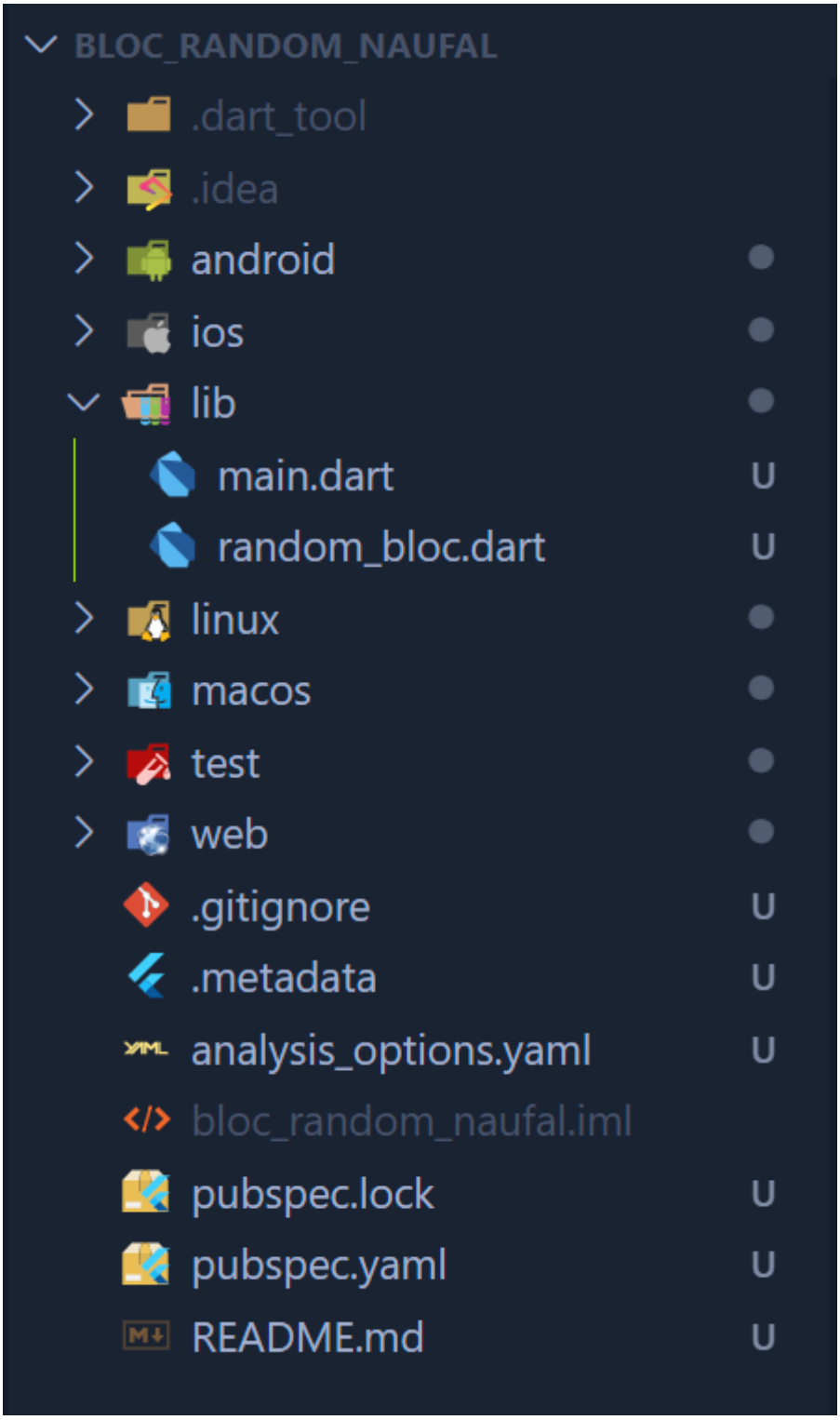
    - Langkah 3: Membuat stream data menggunakan Stream.periodic yang menghasilkan angka random (0–9) setiap detik menggunakan Random. Angka-angka ini akan ditampilkan secara bertahap melalui stream.
    - Langkah 7: Menggunakan StreamBuilder untuk menampilkan angka dari stream secara real-time di UI. StreamBuilder memperbarui tampilan setiap kali data baru dipancarkan.
- Capture hasil praktikum Anda berupa GIF dan lampirkan di README.
- Lalu lakukan commit dengan pesan "W12: Jawaban Soal 12".

Praktikum 7: BLoC Pattern

Langkah 1: Buat Project baru

Buatlah sebuah project flutter baru dengan nama bloc_random_nama (beri nama panggilan Anda) di folder week-12/src/ repository GitHub Anda. Lalu buat file baru di folder lib dengan nama

random_bloc.dart



Langkah 2: Isi kode random_bloc.dart

Ketik kode impor berikut ini.

```
import 'dart:async';
import 'dart:math';
```

Langkah 3: Buat class RandomNumberBloc()

```dart
import 'dart:async';
import 'dart:math';

class RandomNumberBloc {}
```

Langkah 4: Buat variabel StreamController

Di dalam class RandomNumberBloc() ketik variabel berikut ini

```dart
import 'dart:async';
import 'dart:math';

class RandomNumberBloc {
  final _generateRandomController = StreamController<void>();
  final _randomNumberController = StreamController<int>();

  Sink<void> get generateRandom => _generateRandomController.sink;
  Stream<int> get randomRumber => _randomNumberController.stream;
}
```

Langkah 5: Buat constructor

```dart
import 'dart:async';
import 'dart:math';

class RandomNumberBloc {
  final _generateRandomController = StreamController<void>();
  final _randomNumberController = StreamController<int>();

  Sink<void> get generateRandom => _generateRandomController.sink;
  Stream<int> get randomRumber => _randomNumberController.stream;

  RandomNumberBloc() {
    _generateRandomController.stream.listen(
      (_) {
        final random = Random().nextInt(10);
        _randomNumberController.sink.add(random);
      },
    );
  }
}
```

Langkah 6: Buat method dispose()

```dart

```

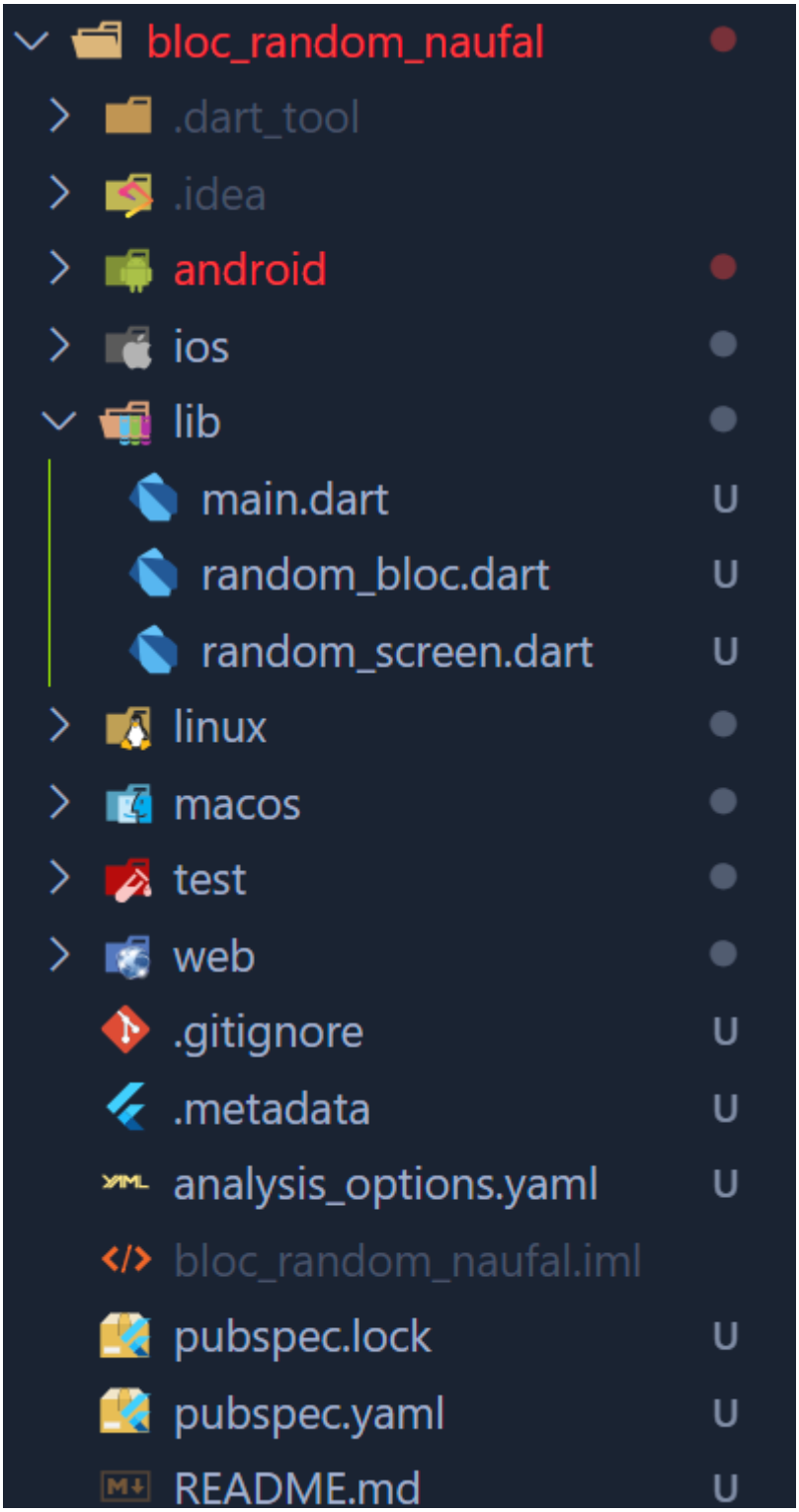
```dart
import 'dart:async';
import 'dart:math';

class RandomNumberBloc {
  final _generateRandomController = StreamController<void>();
  final _randomNumberController = StreamController<int>();

  Sink<void> get generateRandom => _generateRandomController.sink;
  Stream<int> get randomRumber => _randomNumberController.stream;

  RandomNumberBloc() {
    _generateRandomController.stream.listen(
      (_) {
        final random = Random().nextInt(10);
        _randomNumberController.sink.add(random);
      },
    );
  }

  void dispose() {
    _generateRandomController.close();
    _randomNumberController.close();
  }
}
```

Langkah 7: Edit main.dart

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue
      ),
      home: const RandomScreen(),
    );
  }
}
```

Langkah 8: Buat file baru random_screen.dart

Di dalam folder lib project Anda, buatlah file baru ini.



Langkah 9: Lakukan impor material dan random_bloc.dart

Ketik kode ini di file baru `random_screen.dart`

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

Langkah 10: Buat StatefulWidget RandomScreen

Buatlah di dalam file `random_screen.dart`

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

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

  @override
  State<RandomScreen> createState() => _RandomScreenState();
}

class _RandomScreenState extends State<RandomScreen> {
  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

Langkah 11: Buat variabel

Ketik kode ini di dalam `class _RandomScreenState`

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

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

  @override
  State<RandomScreen> createState() => _RandomScreenState();
}

class _RandomScreenState extends State<RandomScreen> {
  final _bloc = RandomNumberBloc();

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

Langkah 12: Buat method dispose()

Ketik kode ini di dalam `class _StreamHomePageState`

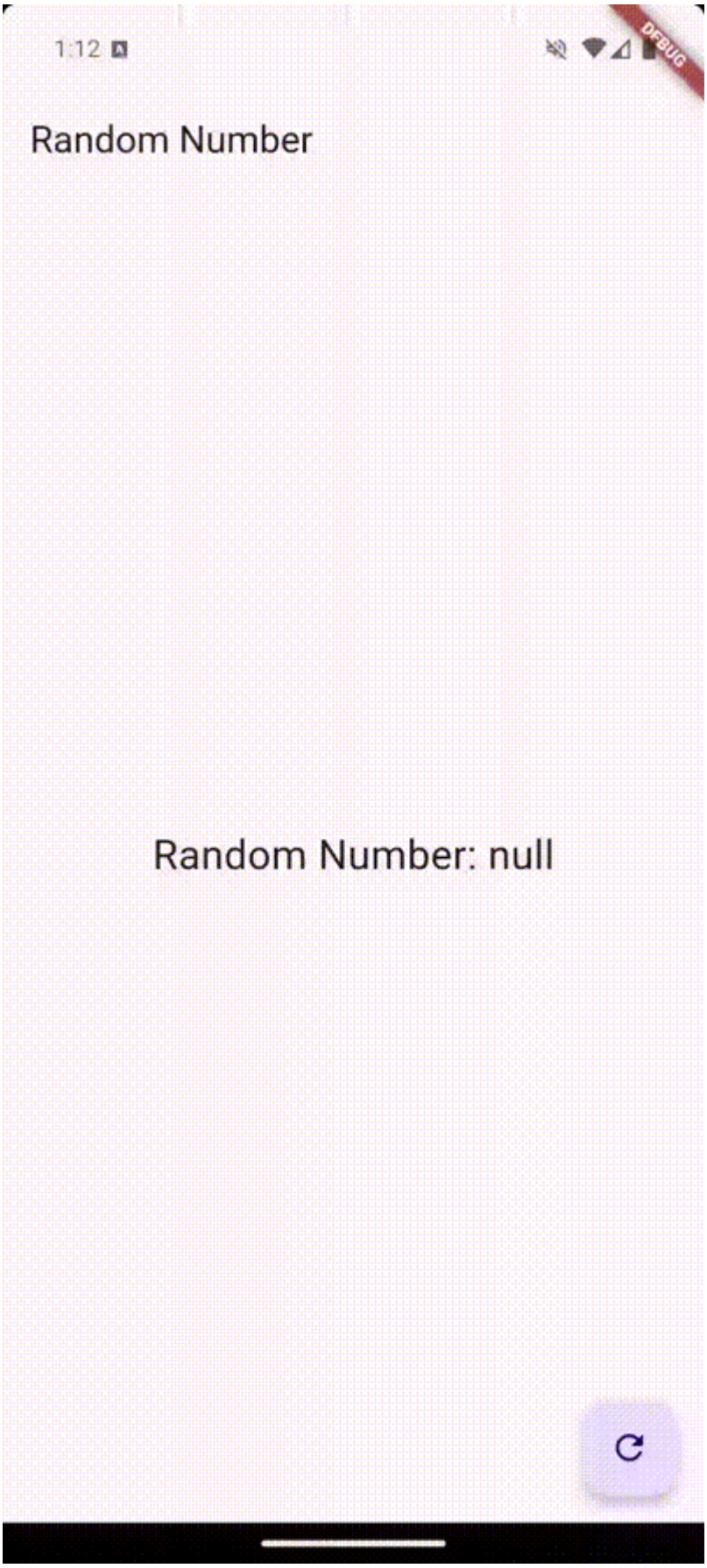
```dart
import 'package:flutter/material.dart';
import 'random_bloc.dart';

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

  @override
  State<RandomScreen> createState() => _RandomScreenState();
}

class _RandomScreenState extends State<RandomScreen> {
  final _bloc = RandomNumberBloc();

  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }

  @override
  void dispose() {
    _bloc.dispose();
    super.dispose();
  }
}
```

Langkah 13: Edit method build()

Ketik kode ini di dalam `class _StreamHomePageState`

```
```

Run aplikasi, maka Anda akan melihat angka acak antara angka 0 sampai 9 setiap kali menekan tombol FloactingActionButton.



Soal 13

- Jelaskan maksud praktikum ini ! Dimanakah letak konsep pola BLoC-nya ?
  Jawab:

Konsep pola BLoC terletak pada pemisahan antara logika bisnis (di BLoC) dan UI (di RandomScreen). UI hanya berfokus pada penyajian data, sementara BLoC mengelola aliran data dan logika penghitungan angka acak menggunakan streams dan stream controllers. Pola ini meningkatkan modularitas, skalabilitas, dan kemudahan pengujian aplikasi Flutter.

- Capture hasil praktikum Anda berupa GIF dan lampirkan di README.
- Lalu lakukan commit dengan pesan "W12: Jawaban Soal 13".