#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

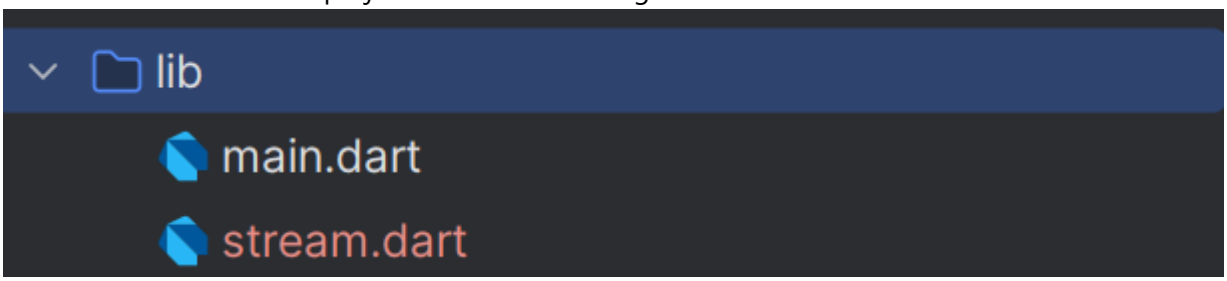
```
import 'package:flutter/material.dart';

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()

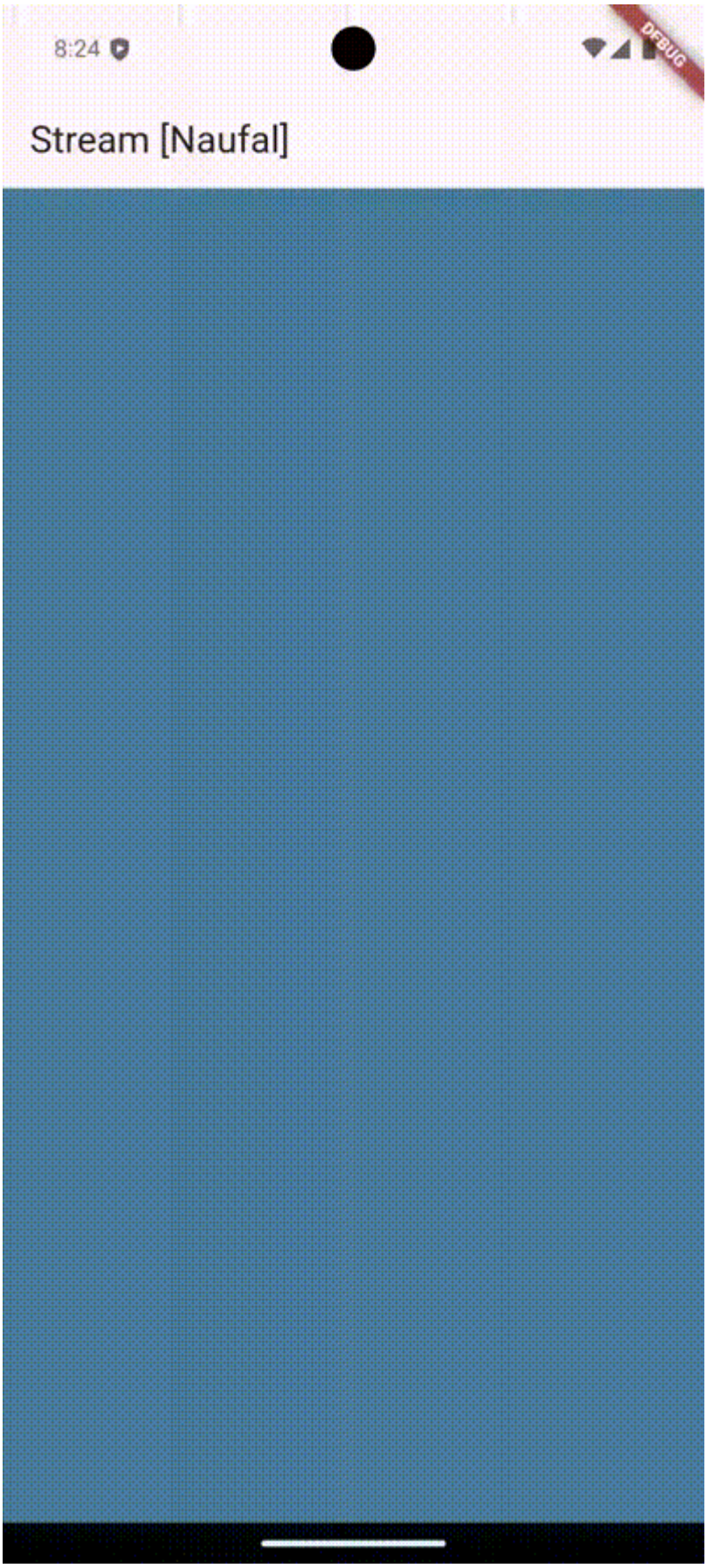
Sesuaikan kode seperti berikut.

```
@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()

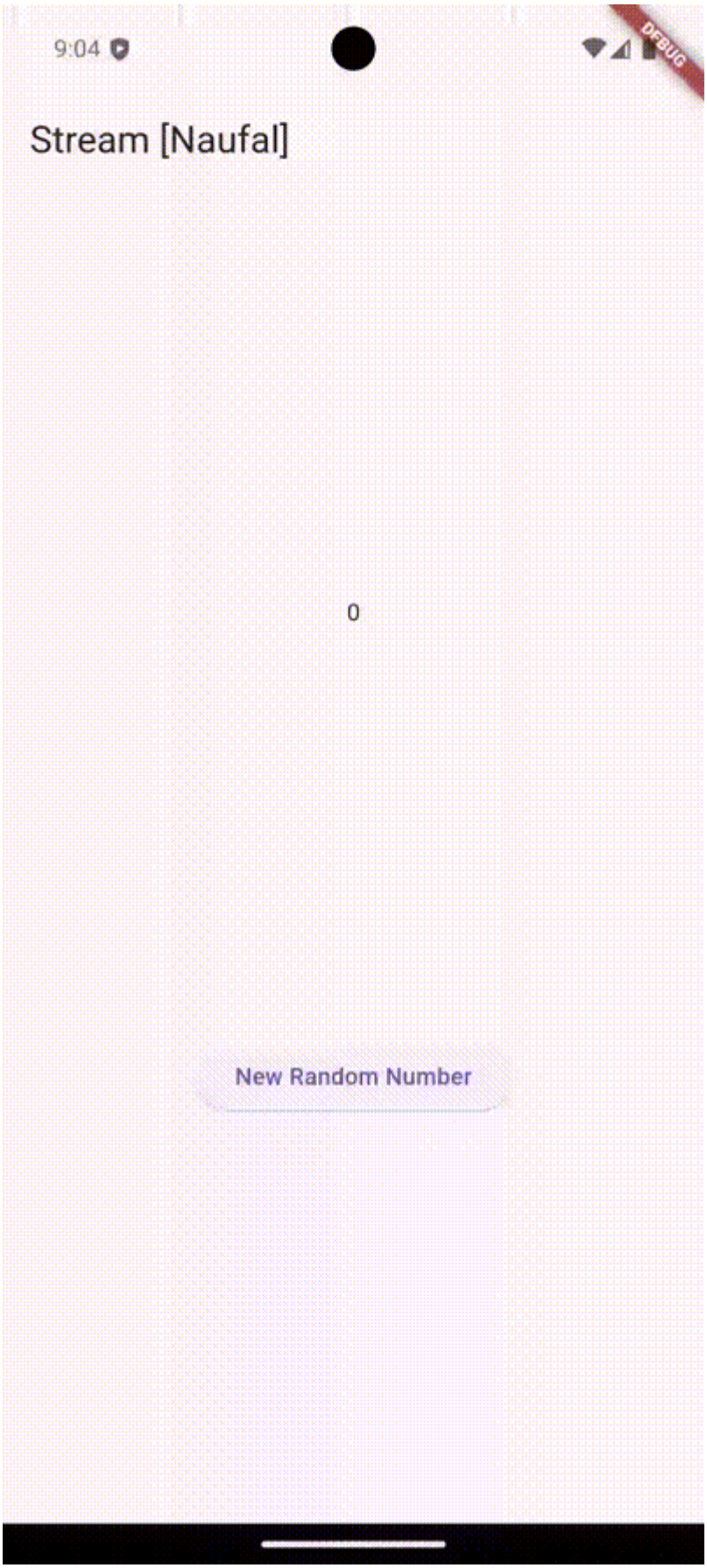
```
@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
        title: const Text("Stream [Naufal]"),
        ),
        body: SizedBox(
        width: double.infinity,
        child: Column(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            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.

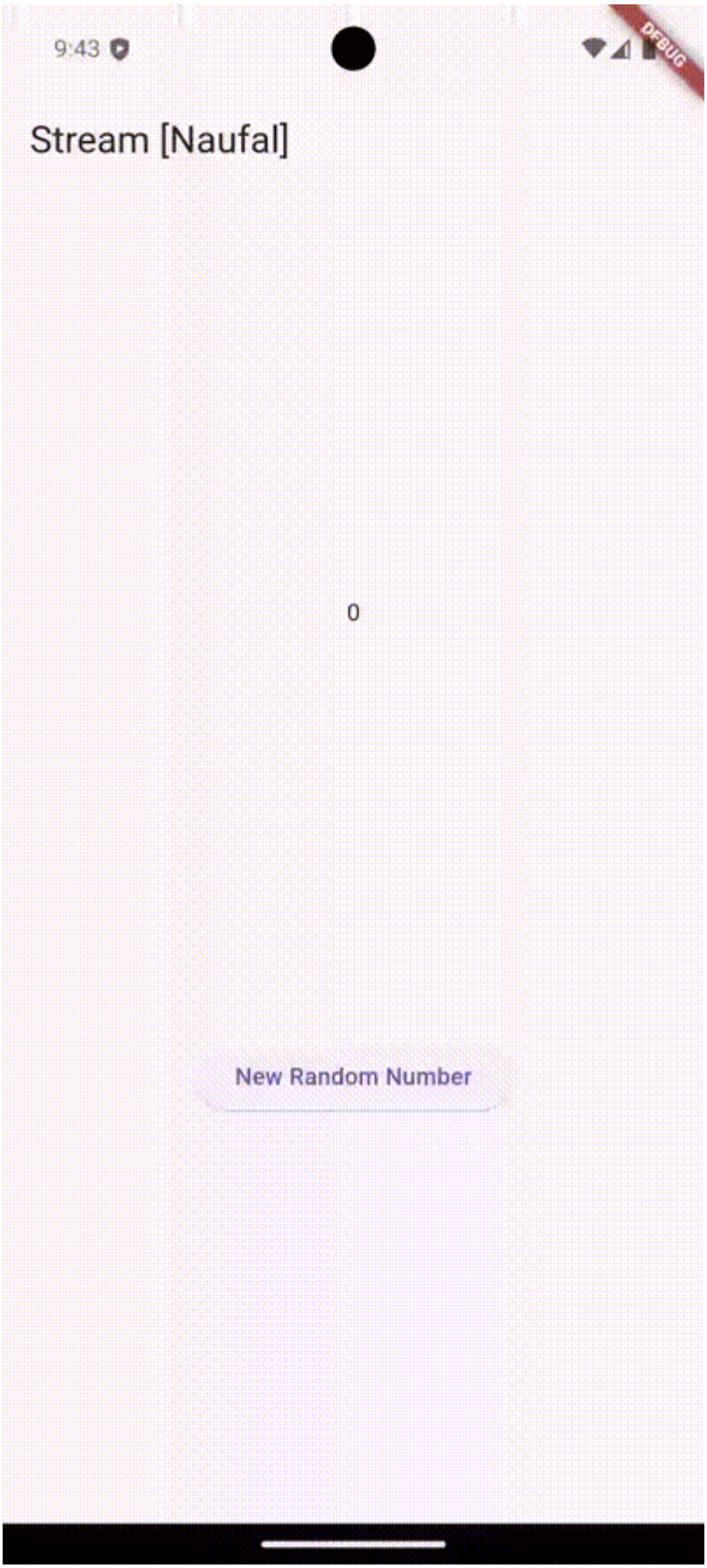
```
@override
void initState() {
    numberStream = NumberStream();
    numberStreamController = numberStream.controller;
    Stream stream = numberStreamController.stream;
    stream.transform(transformer).listen((event) {
        setState(() {
        lastNumber = event;
        });
    },).onError((error){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


9:43
DEBUG
Stream [Naufal]
0
New Random Number