

## Laporan Praktikum

# Pertemuan 12

---

## Lanjutan State Management dengan Streams

---

### Data Mahasiswa

Nama : Fanesabhirawaning Sulistyio

NIM : 2241720027

Kelas : 3C

Prodi : D-IV Teknik Informatika

Jurusan : Teknologi Informasi

## Praktikum 1: Dart Streams

---

Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda. Jawablah di laporan praktikum Anda (ketik di `README.md`) pada setiap soal yang ada di beberapa langkah praktikum ini.

**Perhatian:** Diasumsikan Anda telah berhasil melakukan setup environment Flutter SDK, VS Code, Flutter Plugin, dan Android SDK pada pertemuan pertama.

### 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`

Ketiklah kode seperti berikut ini.

```
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(
        primarySwatch: Colors.deepPurple,
      ),
      home: const StreamHomePage(),
    );
  }
}

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

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

### Soal 1

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

**Jawab**

```
return MaterialApp(
  title: 'Fanesa',
  theme: ThemeData(
    primarySwatch: Colors.purple,
    visualDensity: VisualDensity.adaptivePlatformDensity,
  ),
  home: const StreamHomepage(),
);
```

- Gantilah warna tema aplikasi sesuai kesukaan Anda.

**Jawab**

```
return MaterialApp(
  title: 'Fanesa',
```

```
theme: ThemeData(  
  primarySwatch: Colors.purple,  
  visualDensity: VisualDensity.adaptivePlatformDensity,  
),  
home: const StreamHomepage(),  
);
```

- Lakukan commit hasil jawaban Soal 1 dengan pesan "W12: Jawaban Soal 1"

### Langkah 3: Buat file baru `stream.dart`

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

```
import 'package:flutter/material.dart';  
  
class ColorStream {  
  
}
```

### Langkah 4: Tambah variabel `colors`

Tambahkan variabel di dalam class `ColorStream` seperti berikut.

```
final List<Color> colors = [  
  Colors.blueGrey,  
  Colors.amber,  
  Colors.deepPurple,  
  Colors.lightBlue,  
  Colors.teal  
];
```

## Soal 2

- Tambahkan 5 warna lainnya sesuai keinginan Anda pada variabel `colors` tersebut.

### Jawab

```
class ColorStream {  
  final List<Color> colors = [  
    Colors.blueGrey,  
    Colors.amber,  
    Colors.deepPurple,  
    Colors.lightBlue,  
    Colors.teal,  
    Colors.pink,  
    Colors.red,  
    Colors.orange,  
    Colors.green,  
    Colors.blue.shade900,  
  ];  
}
```

- Lakukan commit hasil jawaban Soal 2 dengan pesan "W12: 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:

```
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!

#### Jawab

Fungsi keyword `yield*` pada kode tersebut adalah untuk menghasilkan nilai dari stream `Stream.periodic`. Keyword `yield*` memungkinkan kita untuk menghasilkan nilai dari stream lain atau iterable.

- Apa maksud isi perintah kode tersebut?

#### Jawab

Isi perintah kode tersebut adalah untuk menghasilkan stream warna yang berganti setiap satu detik. Stream ini dihasilkan dengan cara menghasilkan nilai dari stream `Stream.periodic` yang setiap detiknya menghasilkan nilai integer. Nilai integer tersebut kemudian digunakan untuk menentukan indeks warna yang akan dihasilkan.

- Lakukan commit hasil jawaban Soal 3 dengan pesan "W12: 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`

```
Color bgColor = Colors.blueGrey;  
late ColorStream colorStream;
```

## 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;  
    });  
  }  
}
```

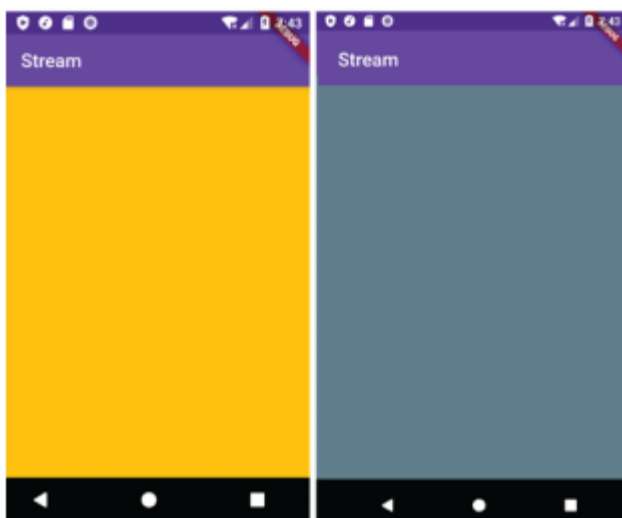
## Langkah 10: Lakukan override `initState()`

Ketik kode berikut:

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

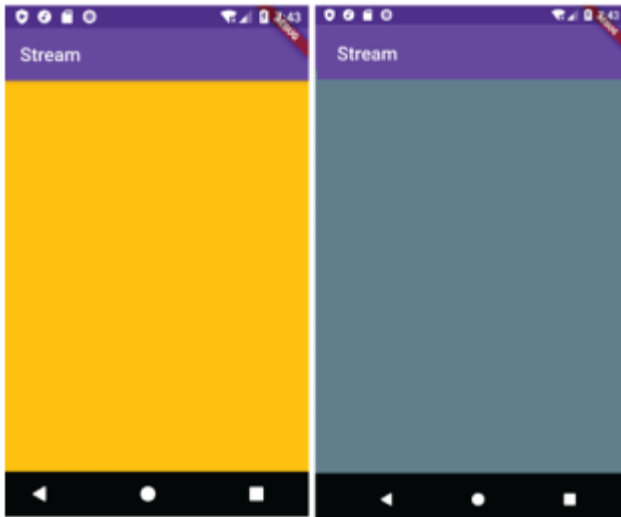
## Langkah 11: Ubah isi `Scaffold()`

Sesuaikan kode seperti berikut.



## Langkah 12: Run

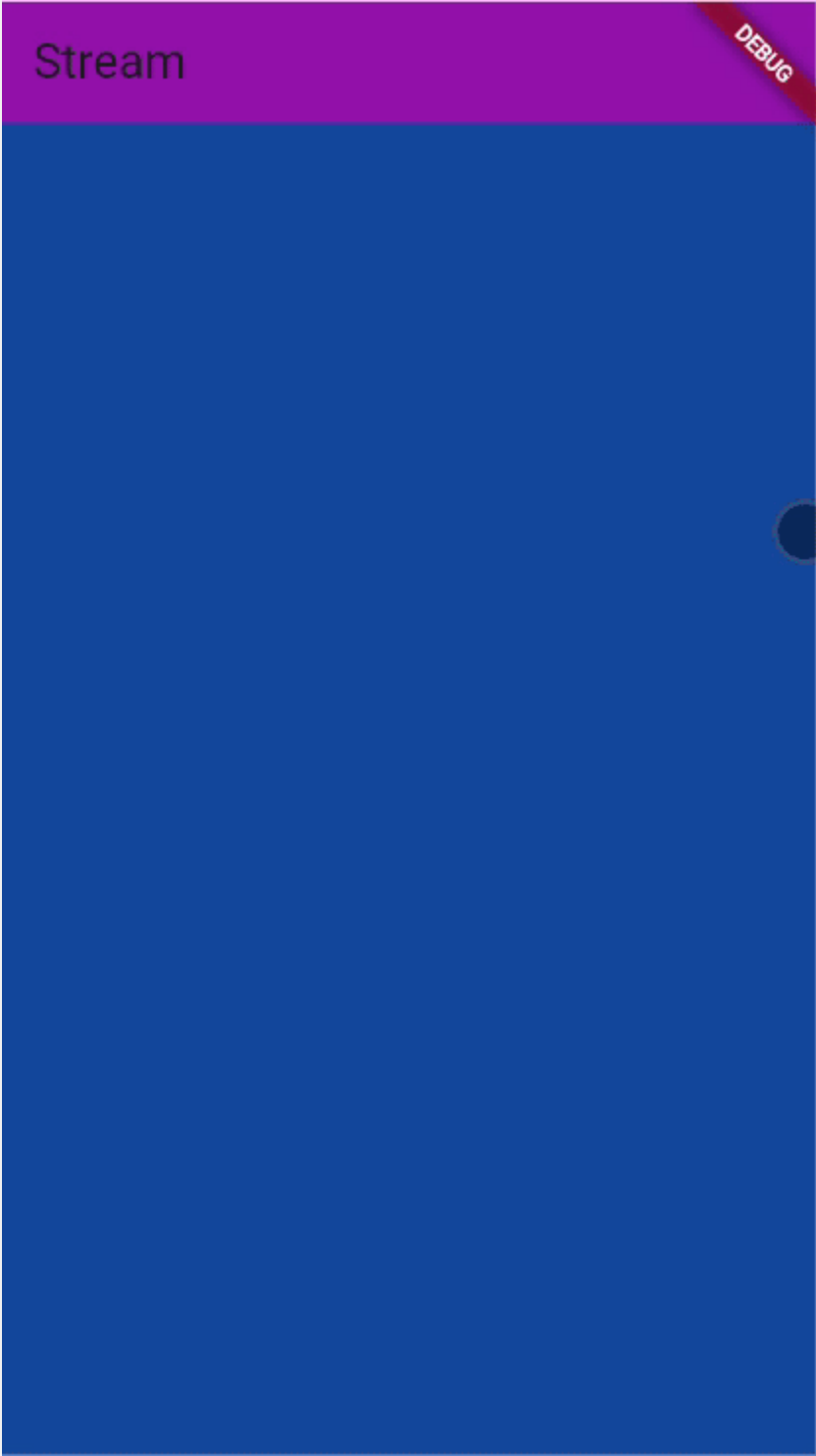
Lakukan running pada aplikasi Flutter Anda, maka akan terlihat berubah warna background setiap detik.



#### Soal 4

- Capture hasil praktikum Anda berupa GIF dan lampirkan di [README](#).

**Jawab**

The image shows a mobile application interface. At the top, there is a purple header bar with the word "Stream" in white. In the top right corner of the purple bar, there is a red ribbon-like label with the word "DEBUG" in white. Below the header, the main area of the app is a solid blue color. On the right side of this blue area, there is a small, dark blue circular button.

- Lakukan commit hasil jawaban Soal 4 dengan pesan "W12: Jawaban Soal 4"

Langkah 13: Ganti isi method `changeColor()`

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

```
colorStream.getColors().listen((eventColor) {  
  setState(() {  
    bgColor = eventColor;  
  });  
});
```

**Soal 5**

- Jelaskan perbedaan menggunakan `listen` dan `await for` (langkah 9)!

### Jawab

#### Listen:

1. Sifat Sinkronus-Asinkronus: Metode `listen` bersifat asinkronus, yang berarti bahwa eksekusi program dapat melanjutkan ke baris kode berikutnya tanpa menunggu pengiriman data ke stream selesai.
2. Fleksibilitas Handling: Anda dapat menggunakan metode `listen` untuk menentukan handler fungsi yang akan dijalankan setiap kali ada perubahan pada stream.
3. Non-blocking: Penggunaan `listen` memungkinkan eksekusi program untuk melanjutkan ke baris berikutnya tanpa harus menunggu setiap data diambil dari stream.

#### Await for:

4. Sifat Sinkronus: Metode `await for` bersifat sinkronus, yang berarti bahwa eksekusi program akan tetap menunggu hingga ada data yang tersedia di stream sebelum melanjutkan ke baris kode berikutnya.
5. Penggunaan Iterator: Penggunaan `await for` mirip dengan penggunaan iterator untuk mengonsumsi nilai dari stream secara satu per satu.
6. Blocking: Penggunaan `await for` akan memblokir eksekusi program sampai data tersedia di stream atau stream ditutup.

- Lakukan commit hasil jawaban Soal 5 dengan pesan "W12: Jawaban Soal 5"

**Catatan:** Stream di Flutter memiliki fitur yang powerful untuk menangani data secara async. Stream dapat dimanfaatkan pada skenario dunia nyata seperti real-time messaging, unggah dan unduh file, tracking lokasi user, bekerja dengan data sensor IoT, dan lain sebagainya.

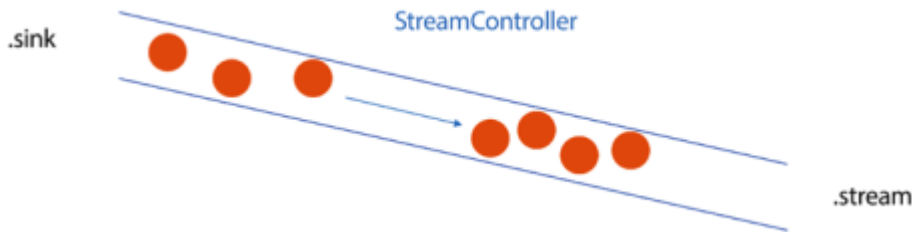
## Praktikum 2: Stream Controllers dan Sinks

`StreamControllers` akan membuat jembatan antara `Stream` dan `Sink`. `Stream` berisi data secara sekuensial yang dapat diterima oleh subscriber manapun, sedangkan `Sink` digunakan untuk mengisi (injeksi) data.

Secara sederhana, `StreamControllers` merupakan stream management. Ia akan otomatis membuat stream dan sink serta beberapa method untuk melakukan kontrol terhadap event dan fitur-fitur yang ada di dalamnya.

Anda dapat membayangkan stream sebagai pipa air yang mengalir searah, dari salah satu ujung Anda dapat mengisi data dan dari ujung lain data itu keluar. Anda dapat melihat konsep stream pada gambar diagram berikut ini.





Di Flutter, Anda dapat menggunakan `StreamController` untuk mengontrol aliran data `stream`. Sebuah `StreamController` memiliki sebuah properti bernama `sink` yang berguna untuk insert data, sedangkan properti `stream` berguna untuk menerima atau keluarnya data dari `StreamController` tersebut.

Setelah Anda menyelesaikan praktikum 1, Anda dapat melanjutkan praktikum 2 ini. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code), Android Studio, atau code editor lain kesukaan Anda. Jawablah di laporan praktikum Anda pada setiap soal yang ada di beberapa langkah praktikum ini.

**Perhatian:** Diasumsikan Anda telah berhasil menyelesaikan Praktikum 1.

Pada codelab ini, kita akan menambah kode dari aplikasi **stream** di praktikum sebelumnya.

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()`

Tambahkan kode berikut.

```
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()`

Tambahkan kode berikut.

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

Langkah 11: Edit method `build()`

```
body: SizedBox (  
  width: double.infinity,  
  child: Column(  
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
    crossAxisAlignment: CrossAxisAlignment.center,  
    children: [  
      Text(lastNumber.toString()),  
      ElevatedButton(  
        onPressed: () => addRandomNumber(),  
        child: 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!

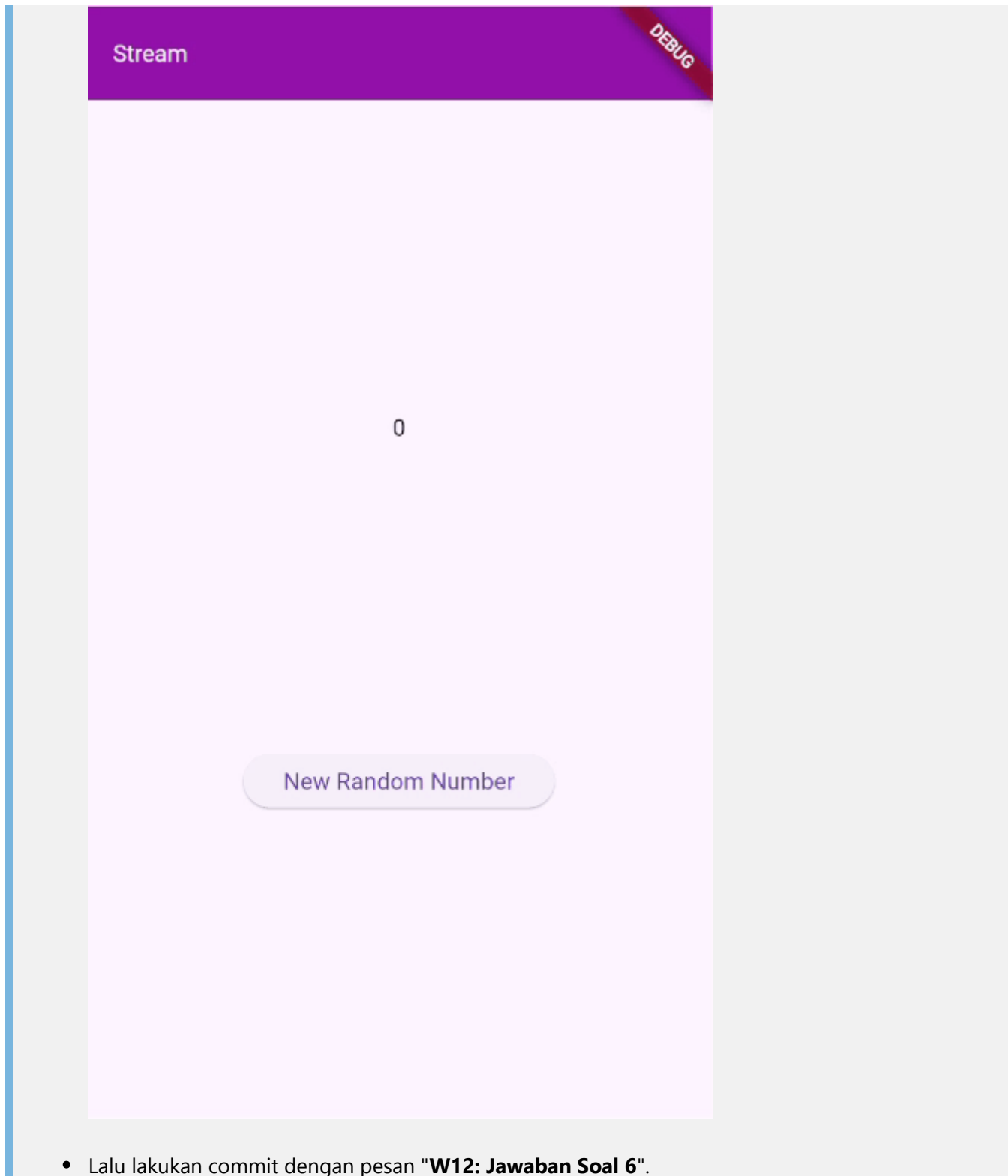
**Jawab**

Langkah 8: Edit initState() Metode initState() dipanggil ketika widget pertama kali dibuat. Dalam langkah ini, kode sedang membuat objek NumberStream dan objek StreamController. StreamController digunakan untuk mengontrol aliran data yang dikeluarkan oleh NumberStream. Aliran kemudian didengarkan, dan metode setState() dipanggil setiap kali peristiwa baru dikeluarkan. Ini memastikan bahwa widget diperbarui setiap kali nomor terbaru diterima.

Langkah 10: Tambah method addRandomNumber() Metode addRandomNumber() digunakan untuk menambahkan nomor acak ke aliran data. Metode ini pertama-tama membuat objek Random dan kemudian memanggil metode nextInt() untuk mendapatkan nomor acak antara 0 dan 9. Nomor acak kemudian ditambahkan ke aliran menggunakan metode addNumberToSink().

- Capture hasil praktikum Anda berupa GIF dan lampirkan di README.

**Jawab**



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.

```
stream.listen((event) {  
  setState(() {  
    lastNumber = event;  
  });  
}).onError((error) {  
  setState(() {  
    lastNumber = -1;  
  });  
});
```

### 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

Pada langkah 13, kita menambahkan method `addError()` ke kelas `Stream`. Method ini digunakan untuk menambahkan error ke stream.

Pada langkah 15, kita mengedit method `addRandomNumber()`. Kita mengomentari dua baris kode yang sebelumnya digunakan untuk menambahkan random number ke stream. Kemudian, kita menambahkan kode baru untuk menambahkan error ke stream.

- Kembalikan kode seperti semula pada Langkah 15, comment `addError()` agar Anda dapat melanjutkan ke praktikum 3 berikutnya.

##### Jawab

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

- Lalu lakukan commit dengan pesan "**W12: Jawaban Soal 7**".

## Praktikum 3: Injeksi Data ke Streams

---

Skenario yang umum dilakukan adalah melakukan manipulasi atau transformasi data stream sebelum sampai pada UI end user. Hal ini sangat berguna ketika Anda perlu melakukan filter data berdasarkan kondisi tertentu, memvalidasi data, memodifikasinya, atau menjalankan proses lain yang memicu beberapa output baru. Contohnya melakukan konversi angka ke string, melakukan perhitungan, atau menghilangkan data yang berulang.

Pada praktikum ini, Anda akan menggunakan `StreamTransformers` dalam `stream` untuk melakukan operasi `map` dan filter data.

Setelah Anda menyelesaikan praktikum 2, Anda dapat melanjutkan praktikum 3 ini. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code), Android Studio, atau editor lain kesukaan Anda. Jawablah di laporan praktikum Anda pada setiap soal yang ada di beberapa langkah praktikum ini.

**Perhatian:** Diasumsikan Anda telah berhasil menyelesaikan Praktikum 2.

Langkah 1: Buka `main.dart`

Tambahkan variabel baru di dalam `class _StreamHomePageState` seperti berikut.

```
late StreamTransformer transformer;
```

Langkah 2: Tambahkan kode ini di `initState`

Edit bagian `initState` dengan menambahkan kode berikut.

```
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 tambahan pada `initState` seperti kode berikut.

```
stream.transform(transformer).listen((event) {  
  setState(() {  
    lastNumber = event;  
  });  
}).onError((error) {  
  setState(() {  
    lastNumber = -1;  
  });  
});  
super.initState();
```

#### Langkah 4: Run

Terakhir, jalankan aplikasi dengan menekan **F5** atau **run**. Jika aplikasi sudah berjalan, lakukan **hot restart**. Maka hasilnya akan terlihat seperti gambar berikut ini dengan tampilan angka dari 0 hingga 90.



#### Soal 8

- Jelaskan maksud kode langkah 1-3 tersebut!

##### Jawab

##### Langkah 1

Pada langkah ini, kita menambahkan variabel baru bernama transformer di dalam class `_StreamHomePageState`. Variabel ini akan digunakan untuk menyimpan objek `StreamTransformer`.

##### Langkah 2



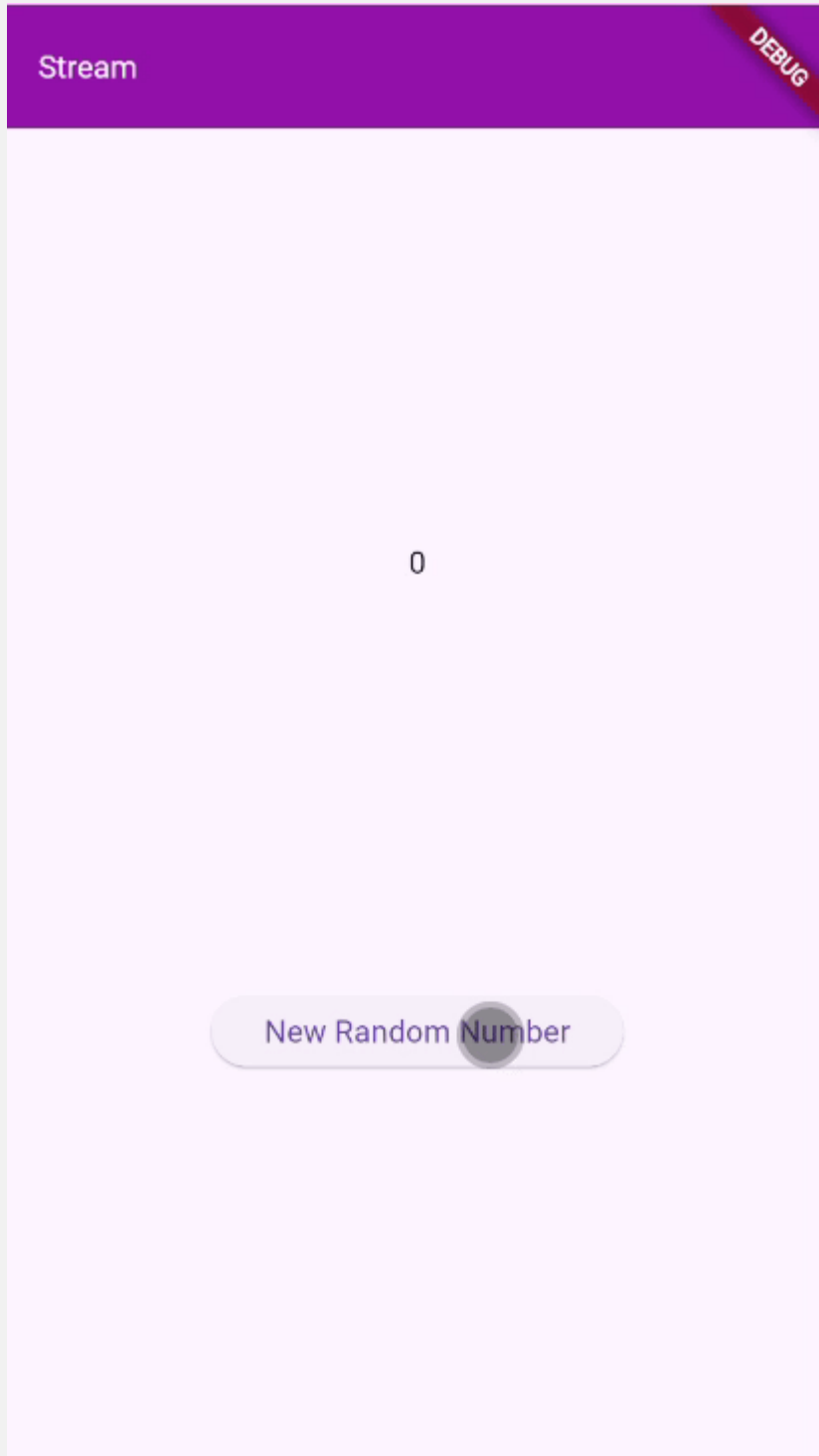
Pada langkah ini, kita menambahkan kode untuk membuat objek `StreamTransformer`. Objek ini akan digunakan untuk mengubah data yang diterima dari stream. Dalam kasus ini, kita akan mengubah data integer menjadi integer yang dikalikan dengan 10.

### Langkah 3

Pada langkah ini, kita melakukan edit kode di `initState()`. Kita menambahkan kode untuk menggunakan objek transformer untuk mengubah data yang diterima dari stream.

- Capture hasil praktikum Anda berupa GIF dan lampirkan di README.

### Jawab



- Lalu lakukan commit dengan pesan "**W12: Jawaban Soal 8**".

## Praktikum 4: Subscribe ke Stream Events

---

Dari praktikum sebelumnya, Anda telah menggunakan method `listen` mendapatkan nilai dari stream. Ini akan menghasilkan sebuah `Subscription`. `Subscription` berisi method yang dapat digunakan untuk melakukan `listen` pada suatu event dari stream secara terstruktur.

Pada praktikum 4 ini, kita akan gunakan `Subscription` untuk menangani event dan error dengan teknik praktik baik (best practice), dan menutup `Subscription` tersebut.

Setelah Anda menyelesaikan praktikum 3, Anda dapat melanjutkan praktikum 4 ini. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda. Jawablah di laporan praktikum Anda pada setiap soal yang ada di beberapa langkah praktikum ini.

**Perhatian:** Diasumsikan Anda telah menyelesaikan Praktikum 3.

### Langkah 1: Tambah Variabel

Tambahkan variabel berikut di dalam `class _StreamHomePageState`.

```
late StreamSubscription subscription;
```

### Langkah 2: Edit `initState()`

Edit bagian `initState` dengan menambahkan kode berikut.

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

### Langkah 3: Tetap di `initState()`

Tambahkan kode berikut di `initState`.

```
subscription.onError((error) {  
  setState(() {  
    lastNumber = -1;  
  });  
});
```

#### Langkah 4: Tambah Properti `onDone()`

Tambahkan kode berikut di bawah `onError`.

```
subscription.onDone(() {  
  print('OnDone was called');  
});
```

#### Langkah 5: Tambah Method Baru

Tambahkan method baru berikut di dalam `class _StreamHomePageState`.

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

#### Langkah 6: Pindah ke Method `dispose()`

Jika `dispose()` belum ada, buat method ini dan tambahkan kode berikut di dalamnya.

```
subscription.cancel();
```

#### Langkah 7: Pindah ke Method `build()`

Tambahkan button kedua di method `build()` dengan kode berikut.

```
ElevatedButton(  
  onPressed: () => stopStream(),  
  child: const Text('Stop Subscription'),  
)
```

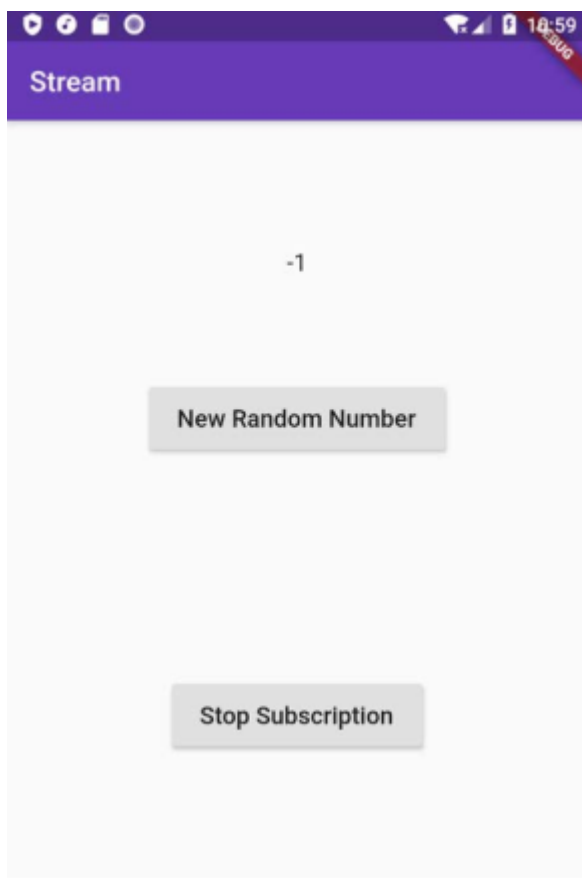
#### Langkah 8: Edit Method `addRandomNumber()`

Edit method `addRandomNumber()` seperti kode berikut.

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

## Langkah 9: Run

Jalankan aplikasi, Anda akan melihat dua button seperti gambar berikut.



## Langkah 10: Tekan Button 'Stop Subscription'

Saat Anda menekan button ini, akan muncul pesan di Debug Console seperti berikut.

```
PROBLEMS 3 OUTPUT TERMINAL DEBUG CONSOLE  
Restarted application in 1,009ms.  
I/flutter ( 5570): OnDone was called
```

### Soal 9

- Jelaskan maksud kode langkah 2, 6, dan 8 tersebut!

### Jawab

## Langkah 2

Pada langkah ini, kita menambahkan kode untuk membuat objek `NumberStream` dan `NumberStreamController`. Objek `NumberStream` akan digunakan untuk menghasilkan stream integer secara acak. Objek `NumberStreamController` akan digunakan untuk mengontrol stream tersebut.

## Langkah 6

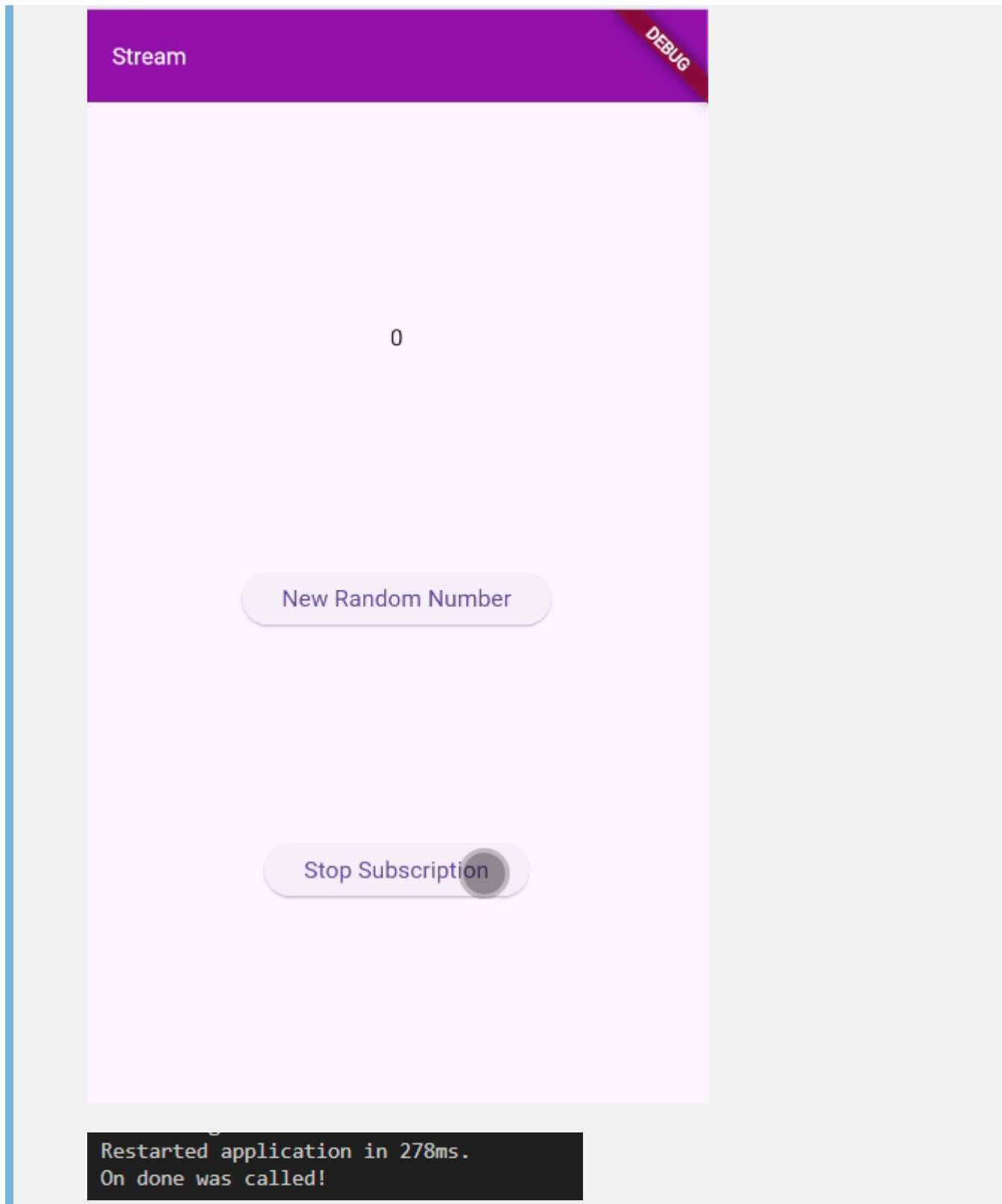
Pada langkah ini, kita menambahkan kode untuk membatalkan subscription di `dispose()`. Subscription ini harus dibatalkan untuk mencegah memory leak.

## Langkah 8

Pada langkah ini, kita menambahkan kode untuk memeriksa apakah stream telah ditutup sebelum menambahkan data ke stream. Jika stream telah ditutup, kita akan mengubah nilai variabel `lastNumber` dengan nilai `-1`.

- Capture hasil praktikum Anda berupa GIF dan lampirkan di README.

## Jawab



- Lalu lakukan commit dengan pesan "**W12: Jawaban Soal 9**".

## Praktikum 5: Multiple Stream Subscriptions

Secara default, stream hanya bisa digunakan untuk satu subscription. Jika Anda mencoba untuk melakukan subscription yang sama lebih dari satu, maka akan terjadi error. Untuk menangani hal itu, tersedia broadcast

stream yang dapat digunakan untuk multiple subscriptions. Pada praktikum ini, Anda akan mencoba untuk melakukan multiple stream subscriptions.

Setelah Anda menyelesaikan praktikum 4, Anda dapat melanjutkan praktikum 5 ini. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda. Jawablah di laporan praktikum Anda pada setiap soal yang ada di beberapa langkah praktikum ini.

**Perhatian:** Diasumsikan Anda telah menyelesaikan Praktikum 4.

Langkah 1: Buka file `main.dart`

Tambahkan variabel berikut di dalam `class _StreamHomePageState`.

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

Langkah 2: Edit `initState()`

Tambahkan kode berikut di `initState`.

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

Langkah 3: Run

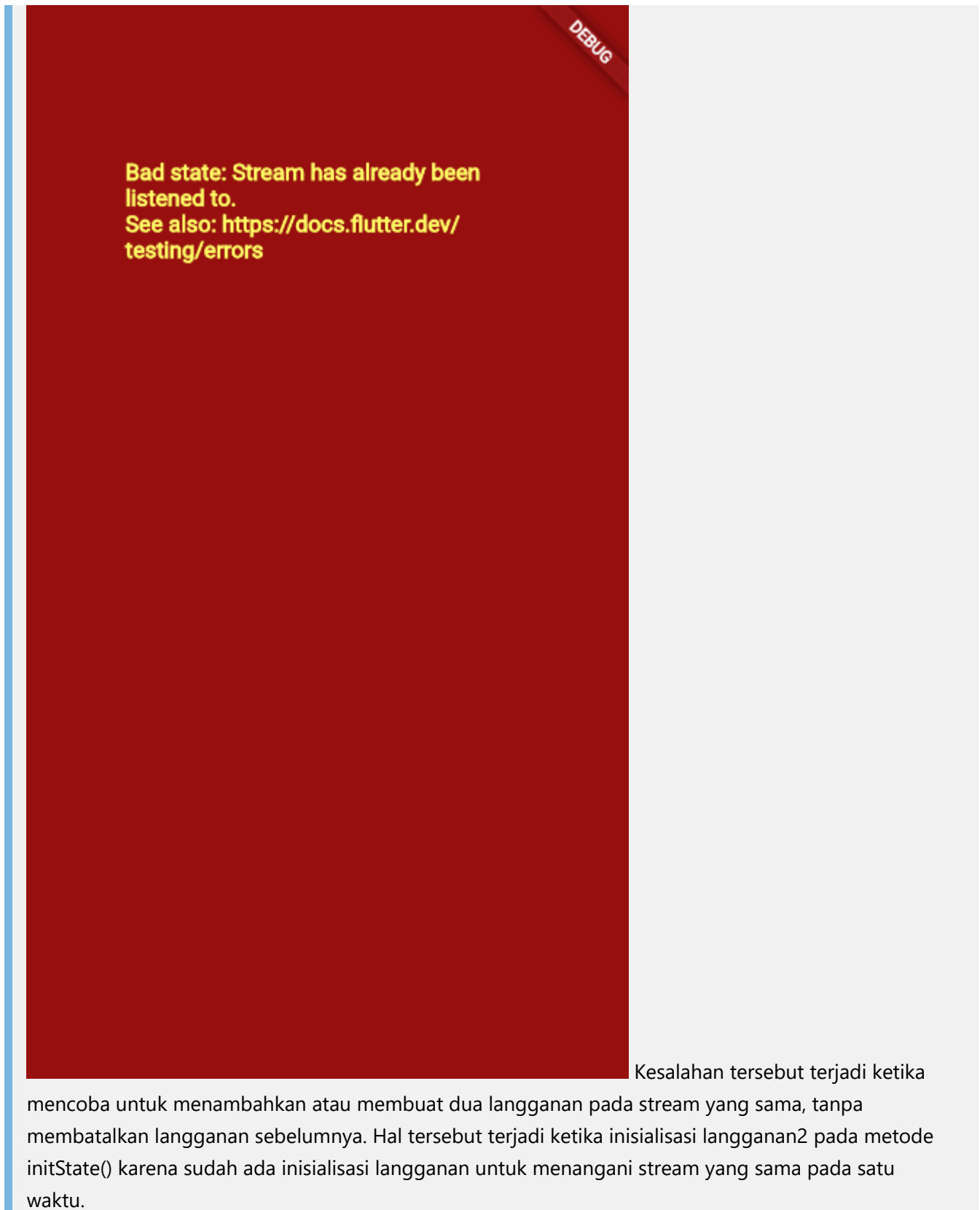
Jalankan aplikasi, dan akan muncul error seperti berikut.



**Soal 10:** Jelaskan mengapa error tersebut bisa terjadi?

**Jawab**





Kesalahan tersebut terjadi ketika mencoba untuk menambahkan atau membuat dua langganan pada stream yang sama, tanpa membatalkan langganan sebelumnya. Hal tersebut terjadi ketika inisialisasi langganan2 pada metode `initState()` karena sudah ada inisialisasi langganan untuk menangani stream yang sama pada satu waktu.

#### Langkah 4: Set Broadcast Stream

Untuk memungkinkan multiple subscriptions, edit `initState` agar stream menjadi broadcast seperti pada kode berikut.

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

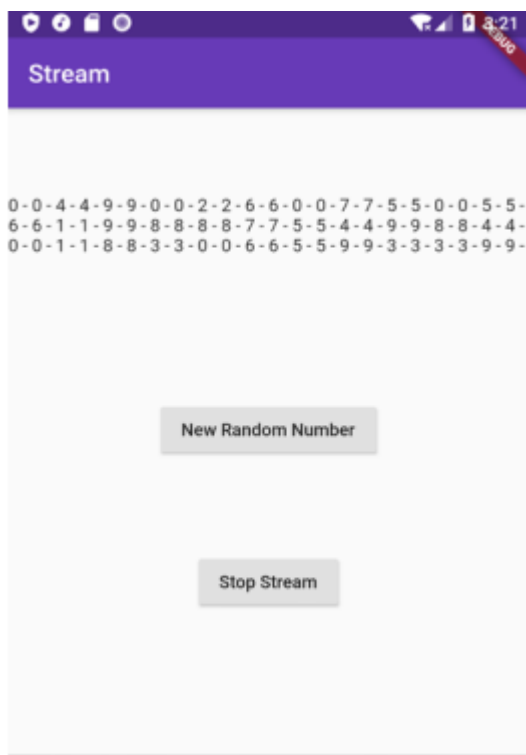
### Langkah 5: Edit Method `build()`

Tambahkan kode berikut pada method `build()` untuk menampilkan data dari kedua subscriptions.

```
child: Column(  
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
    crossAxisAlignment: CrossAxisAlignment.center,  
    children: [  
        Text(values),  
    ],  
)
```

### Langkah 6: Run

Tekan tombol **New Random Number** beberapa kali. Angka yang ditampilkan akan bertambah dua kali untuk setiap subscription, seperti pada gambar berikut.



### Soal 11

- Jelaskan mengapa angka terus bertambah dua kali untuk setiap subscription.

#### Jawab

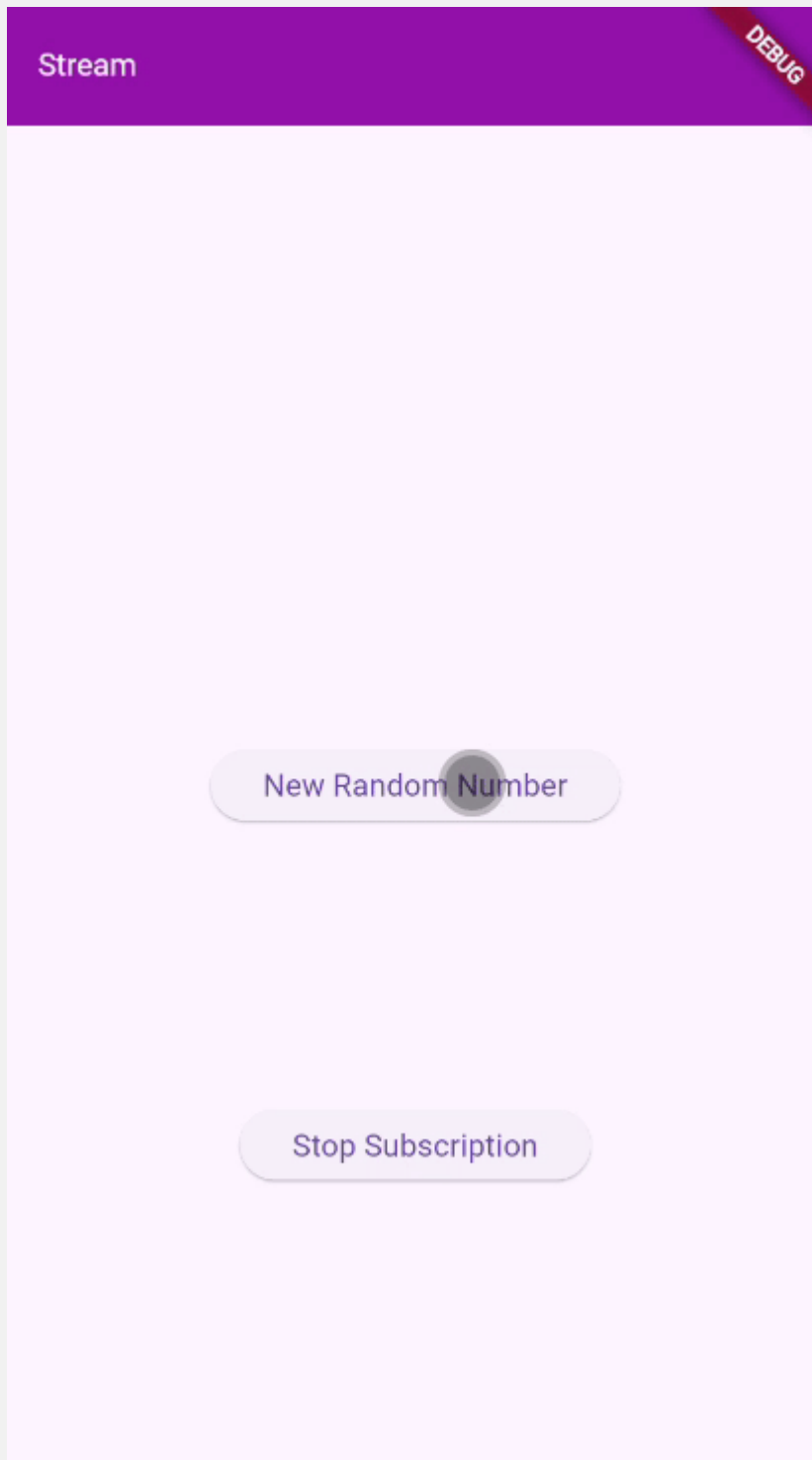
Saat tombol "New Random Number" ditekan, maka akan menghasilkan dua angka random yang sama. Angka-angka tersebut merupakan output dari stream yang dipanggil oleh objek

subscription dan subscription2. Stream tersebut akan mengembalikan nilai berupa event (angka random) yang dipisahkan dengan tanda "-".

Saat tombol "Stop Stream" ditekan, maka akan menghentikan langganan terhadap stream. Hal ini menyebabkan stream tidak lagi bisa mengeluarkan output, meskipun tombol "New Random Number" ditekan.

- Capture hasil praktikum Anda dalam bentuk GIF dan lampirkan di README.

### Jawab



```
Restarted application in 278ms.  
On done was called!
```

- Lalu lakukan commit dengan pesan "**W12: Jawaban Soal 10,11**".

## Praktikum 6: StreamBuilder

---

StreamBuilder adalah sebuah widget untuk melakukan listen terhadap event dari stream. Ketika sebuah event terkirim, maka akan dibangun ulang semua turunannya. Seperti halnya widget FutureBuilder pada pertemuan pekan lalu, StreamBuilder berguna untuk membangun UI secara reaktif yang diperbarui setiap data baru tersedia.

Setelah Anda menyelesaikan praktikum 5, Anda dapat melanjutkan praktikum 6 ini. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda. Jawablah di laporan praktikum Anda pada setiap soal yang ada di beberapa langkah praktikum ini.

**Perhatian:** Diasumsikan Anda telah menyelesaikan Praktikum 5.

### Langkah 1: Buat Project Baru

Buat project Flutter baru dengan nama `streambuilder_nama` (gunakan nama panggilan Anda) di folder `week-13/src/` pada repository GitHub Anda.

### Langkah 2: Buat file `stream.dart`

Tambahkan kode berikut di dalam file `stream.dart`.

```
class NumberStream {}
```

### Langkah 3: Tambah Stream Controller

Tambahkan kode berikut di `stream.dart` untuk membuat stream dan mengirim data secara periodik.

```
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`

Tambahkan kode berikut di `main.dart` untuk menggunakan `StreamBuilder` dalam UI aplikasi.



## Langkah 5: Tambah Variabel

Tambahkan variabel berikut di dalam `class _StreamHomePageState`.

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

## Langkah 6: Edit `initState()`

Tambahkan kode berikut di dalam `initState()` untuk menginisialisasi stream controller.

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

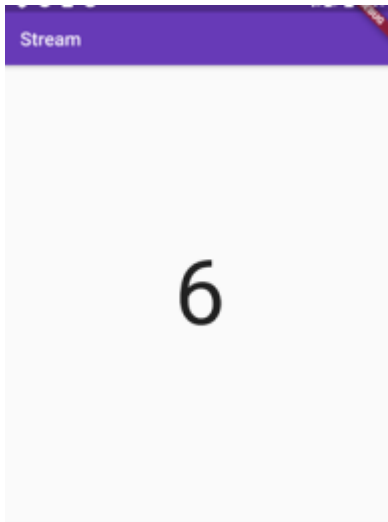
## Langkah 7: Edit Method `build()`

Tambahkan kode berikut pada method `build()` untuk menampilkan data dari stream menggunakan `StreamBuilder`.

```
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

Jalankan aplikasi, dan setiap detik akan muncul angka baru pada tampilan seperti berikut.

**Soal 12**

- Jelaskan maksud kode pada langkah 3 dan 7.

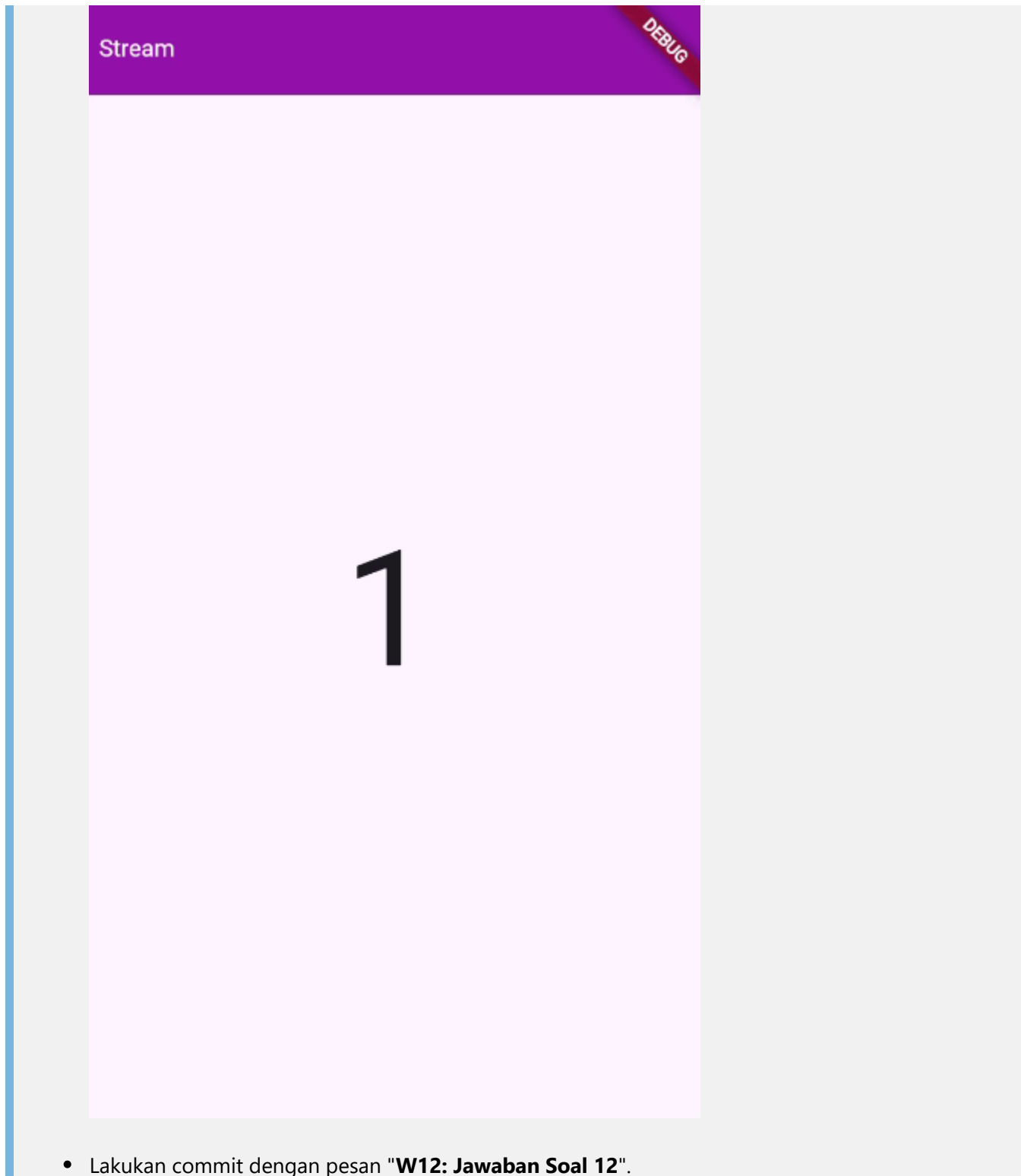
**Jawab**

Langkah 3 melibatkan pembuatan class `NumberStream()`, yang mencakup metode `getNumbers()` untuk menghasilkan stream yang berisi angka-angka acak. Stream ini diperbarui setiap 1 detik.

Sementara itu, Langkah 7 menunjukkan penggunaan kode untuk membuat antarmuka pengguna (UI) yang dapat menampilkan nilai dari stream secara real-time. Untuk mencapai ini, digunakan `StreamBuilder`, yang secara otomatis memperbarui antarmuka setiap kali ada perubahan dalam stream. Perubahan ini dapat berupa perubahan nilai atau munculnya error. Dalam kasus terjadinya error, pesan 'Error!' akan ditampilkan. Jika tidak ada error dan data diterima dari stream, angka acak akan ditampilkan dengan ukuran font setara dengan 96. Namun, jika tidak ada data yang diterima, antarmuka akan menampilkan widget kosong.

- Capture hasil praktikum Anda dalam bentuk GIF dan lampirkan di README.

**Jawab**

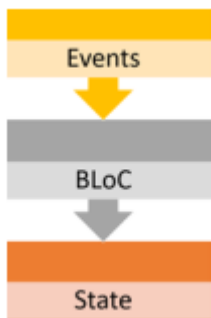


## Praktikum 7: BLoC Pattern

---

Ketika menggunakan pola BLoC, maka segalanya merupakan stream event. BLoC atau Business Logic Component adalah lapisan antara semua sumber data dan UI yang mengonsumsi data itu. Contohnya seperti sumber data dari HTTP layanan web servis atau JSON dari sebuah basis data.

Sebuah BLoC menerima stream data dari sumbernya, proses itu membutuhkan logika bisnis Anda, dan return stream data ke subscriber-nya. Perhatikan diagram pola kerja BLoC berikut ini.



Alasan utama menggunakan BLoC adalah memisahkan logika bisnis aplikasi dengan presentasi UI pada widget, terutama akan sangat berguna ketika aplikasi Anda mulai semakin kompleks dan membutuhkan akses state di berbagai tempat. Hal ini akan membuat semakin mudah dalam menggunakan kode Anda, pada beberapa bagian di aplikasi atau bahkan berbeda aplikasi. Selain itu, BLoC secara independen berdiri sendiri dengan UI, sehingga sangat mudah dilakukan isolasi dalam proses testing.

Pada praktikum codelab ini, Anda akan membuat aplikasi sederhana menggunakan BLoC, namun Anda dapat dengan mudah mengembangkannya untuk aplikasi yang lebih besar ruang lingkupnya.

Setelah Anda menyelesaikan praktikum 6, Anda dapat melanjutkan praktikum 7 ini. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda. Jawablah di laporan praktikum Anda pada setiap soal yang ada di beberapa langkah praktikum ini.

**Perhatian:** Diasumsikan Anda telah menyelesaikan Praktikum 6.

### Langkah 1: Buat Project Baru

Buat project Flutter baru dengan nama `bloc_random_nama` (gunakan nama panggilan Anda) di folder `week-13/src/` pada repository GitHub Anda. Lalu buat file baru di folder `lib` dengan nama `random_bloc.dart`.

### Langkah 2: Tambah Kode Import

Tambahkan kode berikut di dalam file `random_bloc.dart`.

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

### Langkah 3: Buat Class `RandomNumberBloc`

Buat class `RandomNumberBloc()` di dalam file `random_bloc.dart`.

```
class RandomNumberBloc {}
```

### Langkah 4: Buat Variabel `StreamController`

Di dalam class `RandomNumberBloc()`, tambahkan variabel `StreamController` berikut.



```
// StreamController for input events
final _generateRandomController = StreamController<void>();
// StreamController for output
final _randomNumberController = StreamController<int>();
// Input Sink
Sink<void> get generateRandom => _generateRandomController.sink;
// Output Stream.
Stream<int> get randomNumber => _randomNumberController.stream;
_secondsStreamController.sink;
```

## Langkah 5: Buat Constructor

Tambahkan constructor untuk menginisialisasi `StreamController`.

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

## Langkah 6: Buat Method `dispose()`

Tambahkan method `dispose()` untuk membersihkan `StreamController`.

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

## Langkah 7: Edit `main.dart`

Tambahkan kode berikut di `main.dart` untuk memanggil `RandomNumberBloc` di aplikasi.

```
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 `random_screen.dart`

Di dalam folder `lib`, buatlah file baru bernama `random_screen.dart`.

## Langkah 9: Tambah Kode Import di `random_screen.dart`

Tambahkan import material dan `random_bloc.dart`.

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

## Langkah 10: Buat `StatefulWidget` `RandomScreen`

Buat class `RandomScreen` di `random_screen.dart`.

## Langkah 11: Tambah Variabel

Tambahkan variabel berikut di dalam class `_RandomScreenState`.

```
final _bloc = RandomNumberBloc();
```

## Langkah 12: Buat Method `dispose()`

Tambahkan method `dispose()` untuk membersihkan controller.

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

## Langkah 13: Edit Method `build()`

Tambahkan kode ini di dalam class `_RandomScreenState`.

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: const Text('Random Number')),
    body: Center(
      child: StreamBuilder<int>(
        stream: _bloc.randomNumber,
        initialData: 0,
        builder: (context, snapshot) {
          return Text(
            'Random Number: ${snapshot.data}',
            style: const TextStyle(fontSize: 24),
          );
        },
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: () => _bloc.generateRandom.add(null),
      child: const Icon(Icons.refresh),
    ),
  );
},
```

#### Langkah 14: Run

Jalankan aplikasi. Setiap kali menekan tombol **FloatingActionButton**, angka acak antara 0 sampai 9 akan tampil.

#### Soal 13

- Jelaskan maksud praktikum ini dan letak konsep pola BLoC-nya.

##### Jawab

##### 1. Class RandomNumberBloc:

RandomNumberBloc adalah implementasi BLoC yang menggunakan dua StreamController: satu untuk mengontrol input events (\_generateRandomController), dan satu untuk mengontrol output (\_randomNumberController). \_generateRandomController digunakan untuk mengirim events yang akan memicu pembangkitan nomor acak, sedangkan \_randomNumberController mengontrol stream output yang berisi nomor acak yang dihasilkan.

##### 2. Kelas MyHomePage:

MyHomePage merupakan antarmuka pengguna sederhana yang tidak secara langsung terlibat dalam logika bisnis. Ini tidak menyertakan logika khusus terkait BLoC. Namun, dalam pengembangan aplikasi yang lebih kompleks, logika bisnis dapat dipindahkan ke dalam BLoC untuk menjaga kesatuan dan pemisahan tanggung jawab.

##### 3. Class RandomScreen:

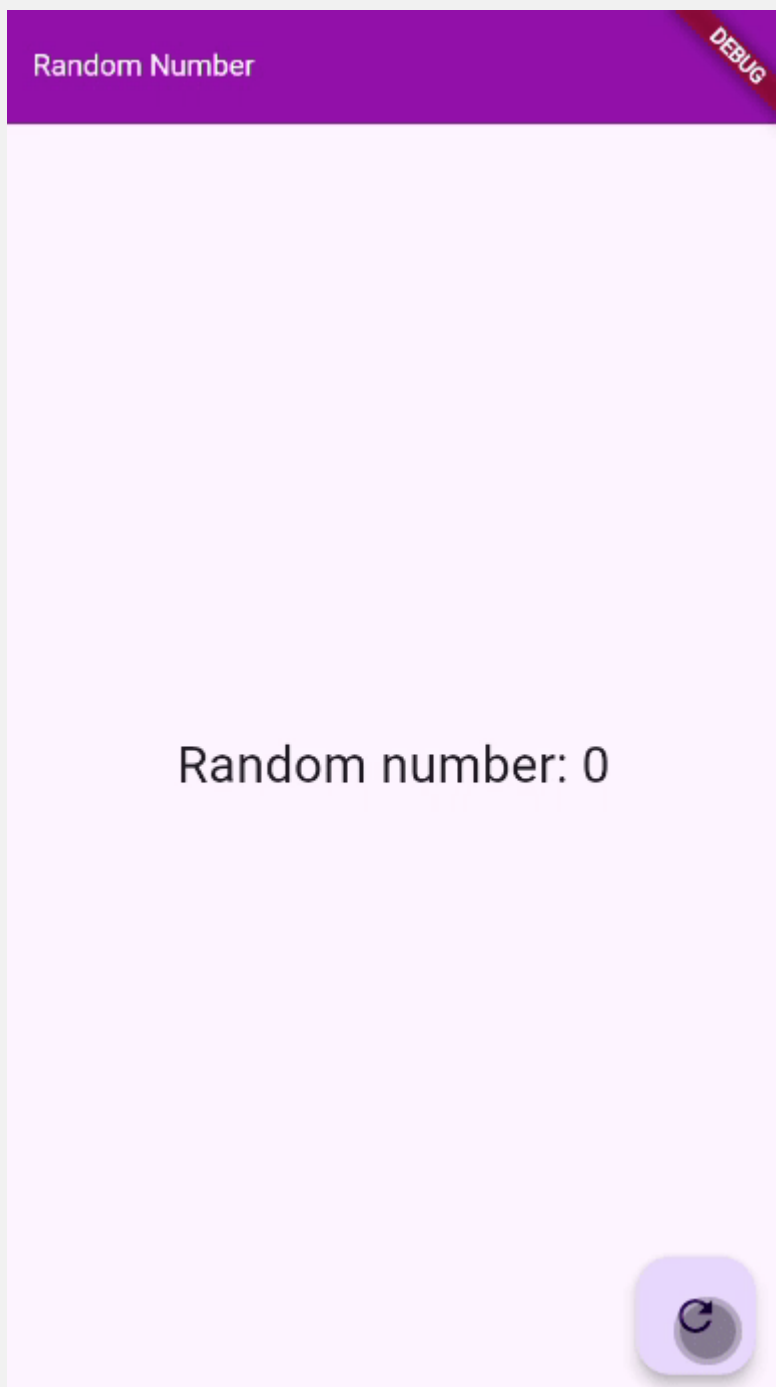
RandomScreen adalah antarmuka pengguna yang menggunakan RandomNumberBloc. State dari widget ini dikendalikan oleh stream yang dihasilkan oleh `_bloc.randomNumber`. Setiap kali event dikirim melalui `_bloc.generateRandom`, nomor acak baru dihasilkan dan diperbarui di UI. Dengan memisahkan logika bisnis ke dalam RandomNumberBloc, antarmuka pengguna dapat fokus pada tampilan dan merespons perubahan state.

#### 4. Pemanggilan BLoC di main.dart:

BLoC (RandomNumberBloc) diinisialisasi dan dimiliki oleh `_RandomScreenState`. Pemanggilan `_bloc.generateRandom.add(null)` pada tombol tindakan antarmuka pengguna memicu pembangkitan nomor acak melalui BLoC.

- Capture hasil praktikum Anda dalam bentuk GIF dan lampirkan di README.

#### Jawab



- Lakukan commit dengan pesan "**W12: Jawaban Soal 13**".