Modul 1

Listening to a Stream of Data

Using a stream and asynchronous programming in Flutter.

Module Overview

Mengenal metode asinkron dalam penerapannya pada aplikasi, tipe metode asinkron pada flutter dan bagaimana cara menerapkannya.

Module Objectives

Setelah mempelajari dan mempraktikkan modul ini, mahasiswa diharapkan dapat:

- Mengetahui apa yang dimaksud dengan asinkron
- Mengenali tipe metode asinkron pada flutter
- Menerapkan metode asinkron sesuai dengan penggunaannya

Dalam membangun sebuah aplikasi tentu tidak dapat terlepas dari data yang menjadi konten daripada aplikasi tersebut. Pada umumnya data-data ini akan diload

ketika pengguna sedang melakukan sesuatu di dalam aplikasi. Hal ini dimungkinkan karena adanya metode asinkron yang membuat pengguna tidak perlu menunggu terlalu lama untuk mendapatkan data yang diinginkan sembari menggunakan aplikasi tersebut.

Metode asinkron memungkinkan program untuk mengerjakan suatu pekerjaan (task) lain selagi menunggu operasi lain selesai. Berikut merupakan beberapa contoh aktivitas dalam aplikasi yang dapat menggunakan metode asinkron:

1. Mengambil/mengunduh data dari sebuah jaringan
2. Menuliskan data ke database
3. Membaca data dari sebuah file

Dalam Flutter terdapat 2 tipe metode asinkron yang dapat dilakukan yaitu Future dan Stream. Future hanya dapat digunakan untuk melakukan pemanggilan data dalam sekali pemanggilan sedangkan Stream dapat dipergunakan untuk melakukan pemanggilan data secara berkelanjutan.

Asynchronous Programming: Future

Pada contoh ini kita akan mencoba mengambil data dan menampilkannya pada aplikasi Flutter dengan menggunakan Future.

Bagian Front-End

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

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

class MyHomePage extends StatefulWidget {
  const MyHomePage({Key? key, required this.title}) : super(key: key);

  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'Daftar Pengguna',
              style: Theme.of(context).textTheme.headline6,
            ),
            Padding(
              padding: EdgeInsets.all(15),
              child: Text(
                '$data',
              ),
            )
          ],
        ),
      ),
    );
  }
}
```

1. Di atas void main() deklarasikan sebuah variabel data baru. Nantinya variabel data ini kita gunakan untuk menyimpan data yang kita peroleh dari database.

```
1 var data = [];
```

2. Tambahkan sebuah fungsi getUserData() di bawah variabel data tadi. Fungsi getUserData() seharusnya berisi query untuk mengambil data ataupun query database. Pada contoh ini variabel data sebelumnya kita asumsikan diisi dengan data nama pengguna dari database.

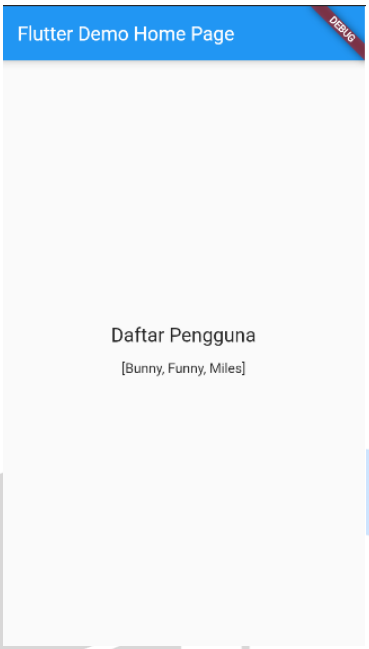
```
1 Future<void> getUserData() {
2   // Contoh proses mengambil data, query database
3   data = ['Bunny', 'Funny', 'Miles'];
4   return Future.delayed(const Duration(seconds: 3), () {
5     print("Downloaded ${data.length} data");
6   });
7 }
```

3. Pada void main(), tambahkan pemanggilan fungsi getUserData(); dan print('Getting user data...'); untuk melihat urutan dari perintah yang dijalankan.

```
1 void main() {
2   runApp(const MyApp());
3   getUserData();
4   print('Getting user data ... ');
5 }
```

4. Jalankan project dan berikut merupakan keluaran yang dihasilkan pada bagian console.

```
PROBLEMS 3   OUTPUT   DEBUG CONSOLE   TERMINAL   JUPYTER
Restarted application in 388ms.
Getting user data...
Downloaded 3 data
```



Pada contoh di atas fungsi getUserData() sebenarnya dijalankan lebih dahulu namun ketika fungsi getUserData() dijalankan. Fungsi ini mengembalikan fungsi Future.delayed yang mengakibatkan adanya waktu tunggu selama 3 detik (dari const Duration). Hal ini membuat perintah print pada fungsi void main() dijalankan terlebih dahulu selagi menunggu waktu tunggu 3 detik yang ada.

Penggunaan async dan await

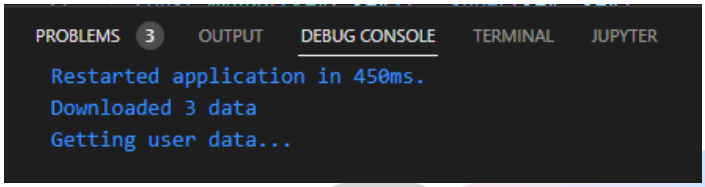
Keyword async dan await dapat dipergunakan untuk membuat sebuah fungsi asinkron. Berikut ketentuan dasar untuk penggunaan async dan await:

- Untuk membuat fungsi asinkron, tambahkan keyword async sebelum badan fungsi.
- Keyword await hanya dapat dipergunakan pada fungsi asinkron.

5. Ubah fungsi void main() menjadi fungsi asinkron dengan menambahkan keyword async sebelum badan fungsi dan await pada pemanggilan fungsi getUserData().

```dart
void main() async {
  runApp(const MyApp());
  await getUserData();
  print('Getting user data ... ');
}
```

6. Jalankan kembali project dan lihat kembali keluaran yang diberikan pada console.



Pada keluaran di atas dapat terlihat perbedaan di mana yang tadinya perintah print('Getting user data...') dijalankan terlebih dahulu selagi menunggu delay selama 3 detik pada fungsi getUserData() sedangkan kini perintah print tersebut dijalankan setelah delay 3 detik tersebut selesai karena adanya keyword await yang membuat program menunggu eksekusi fungsi getUserData() selesai terlebih dahulu.

Tanpa async-await	Menggunakan async-await
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER Restarted application in 388ms. Getting user data... Downloaded 3 data	PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER Restarted application in 450ms. Downloaded 3 data Getting user data...

Asynchronous Programming: Streams

Streams merupakan cara pengiriman data secara berkelanjutan dalam Flutter. Stream pada Flutter bersifat umum, artinya Stream dapat mengembalikan data dengan tipe apapun.

Pada contoh ini kita akan mencoba membuat sebuah indicator persen dengan memanfaatkan Stream flutter.

Bagian Front-End

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

class contoh2 extends StatefulWidget {
  const contoh2({Key? key}) : super(key: key);

  @override
  State<contoh2> createState() => _contoh2State();
}

class _contoh2State extends State<contoh2> {
  int percent = 100;
  int getSteam = 0;
  double circular = 1;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Stream")),
      body: Center(child: LayoutBuilder(builder: (context, constraints) {
        final double avaWidth = constraints.maxWidth;
        final double avaHeight = constraints.maxHeight;
        return Column(
            mainAxisAlignment: MainAxisAlignment.center,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              Expanded(
                  child: CircularPercentIndicator(
                radius: avaHeight / 5,
                lineWidth: 10,
                percent: circular,
                center: Text("$percent %"),
              )) // CircularPercentIndicator // Expanded
            ]); // Column
      })), // LayoutBuilder // Center
      floatingActionButton: FloatingActionButton(onPressed: () {}),
    ); // Scaffold
  }
}
```

Di sini kita akan memanfaatkan widget library Percent Indikator. Untuk itu lakukan penginstalan library dengan menambahkan percent_indicator: ^4.0.1 pada dependencies pubsec.yaml. Anda dapat menggunakan pubspec Assist yang sudah anda install pada Commont Pallette (Ctrl+Shift+P) pada Visual Code anda.

1. Pertama, kita akan membuat variable Stream final _myStream dengan periodik dalam braket berdurasi adalah 1, int count dan return count. Letakkan variable ini pada class contoh2State anda.

```
final Stream _myStream =
    Stream.periodic(const Duration(seconds: 1), (int count) {
  return count;
}); // Stream.periodic
```

2. Saat Anda menggunakan Stream, maka anda dapat menggunakan Stream.listen untuk mendapatkan data dari steam, dan objek StreamSubscription untuk mengakses stream anda. Anda akan mengakses stearm dari _myStream menggunakan StreamSubscription dengan cara membuat variable dari StreamSubscription.

```
class _contoh2State extends State<contoh2> {
  late StreamSubscription _sub;
  final Stream _myStream =
      Stream.periodic(const Duration(seconds: 1), (int count) {
    return count;
  }); // Stream.periodic
```

3. Setiap data yang terbaca dalam stream, akan kita simpan pada

```
int getSteam = 0;
```

4. Pada Event onPress di FAB, kita akan mendaftarkan Steam yang kita buat pada _sub.

```
floatingActionButton: FloatingActionButton(onPressed: () {
  _sub = _myStream.listen((event) {

  });
}), // FloatingActionButton
```

5. Anda dapat menerima nilai yang dikirimkan oleh stream yang anda buat, melalui event pada _mySteam.listen anda, sehingga ada dapat menampung

nilai tersebut pada getStream yang telah anda deklarasikan

```
floatingActionButton: FloatingActionButton(onPressed: () {
  _sub = _myStream.listen((event) {
    getSteam = int.parse(event.toString());
```

6. Untuk menglihat hasil Steam yang dikembalikan oleh count, anda dapat mencoba algorime berikut.

```
floatingActionButton: FloatingActionButton(onPressed
  _sub = _myStream.listen((event) {
    getSteam = int.parse(event.toString());
    setState(() {
      if (_Persen - getSteam <= 0) {
        _sub.cancel();
        _Persen = 0;
        circular = 0;
      } else {
        _Persen = _Persen - getSteam;
        circular = _Persen / 100;
      }
    });
  });
}), // FloatingActionButton
```

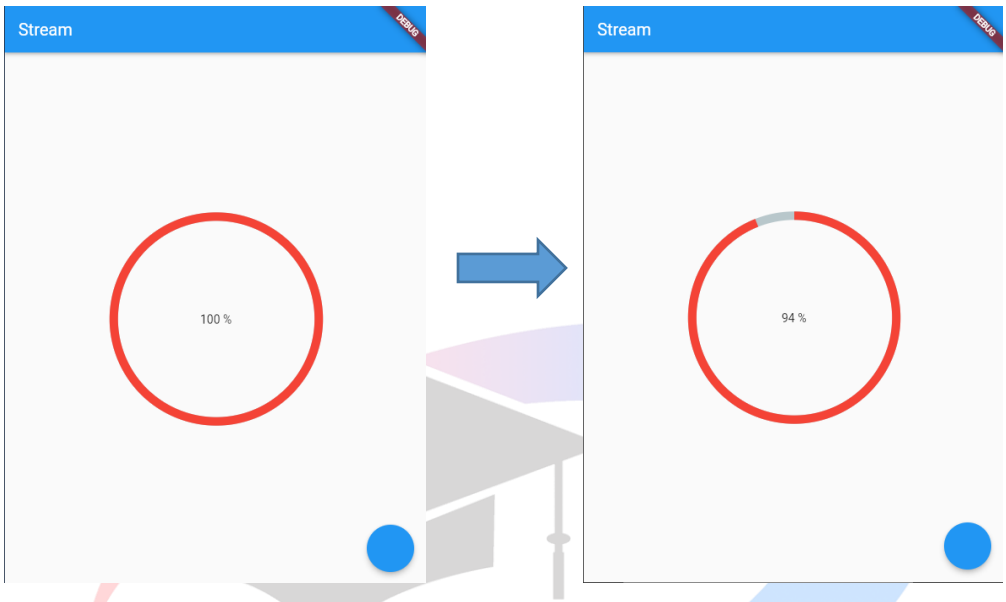
7. Adapun algoritme berikut, hanyalah pengecekan untuk menghentikan stream Ketika persentasi sudah mencapai 0 dengan memberikan fungsi cancel() pada variable StreamSubscription kita, dan menghentikan persentasi dan percent_indicator pada nilai 0

```
setState(() {
  if (_Persen - getSteam <= 0) {
    _sub.cancel();
    _Persen = 0;
    circular = 0;
```

8. Jika belum mencapai 0, anda dapat mengurangi nilai persen sesuai dengan nilai yang dikembalikan oleh steam.

```
  } else {
    _Persen = _Persen - getSteam;
    circular = _Persen / 100;
  }
```

7. Jalankan project dan berikut merupakan keluaran yang dihasilkan pada bagian console.



Latihan

1. Dari contoh Asynchronous Programming: Future, tambahkan data yang dibaca sehingga dapat muncul pada ListView (Minimal nama, dan nomor telepon)
2. Dari contoh Asynchronous Programming: Streams, tambahkan fungsi untuk melakukan,
   a. Nim Genap Reset, Play dan Stop
   b. Nim Ganjil Play, dan Menambahkan Indikator Text, Ketika kondisi penuh (100%) akan menampilkan text "Full", dan Ketika kondisi kosong (0%) akan menapilkan text "Empty". Selain dari 2 kondisi tersebut, text tidaka akan di munculkan.