PRAKTIKUM MCS BAB 8

Card reader and Servo controller

PENDAHULUAN

Pada praktikum MCS bab 8 akan membangun aplikasi untuk mengontrol servo dan melihat id kartu yang masuk ke database melalui sensor Radio Frequency Identification (RFID). Agar dapat mengontrol servo dan membaca kartu yang masuk kita akan mengkonsumsi API yang sudah dibuat pada bab 6 dan bab 7. API dikonsumsi menggunakan oleh flutter menggunakan package dio.

PRAKTIKUM BAB 7

Sebelumnya kita sudah membuat 2 endpoint beserta routenya pada bab 6 dan 7. Endpoint yang dibuat pada bab 6 adalah endpoint untuk membaca kartu yang masuk ke database melalui RFID sedangkan endpoint bab 7 adalah endpoint untuk mengotrol servo.

Endpoint untuk bab 6 memiliki base url sebagai berikut :

https://card-bridge-railway-production.up.railway.app

route dari base url endpoint bab 6 :

/cards (untuk membaca kartu dengan metode get)

/card/input/:id (untuk memasukkan id kartu ke database melalui parameter :id)

/card/delete/:id (untuk menghapus id kartu ke database melalui parameter :id)

Endpoint untuk bab 7 memiliki base url sebagai berikut :

https://srv-cntrllr-railway-production.up.railway.app

route dari base url endpoint bab 7 :

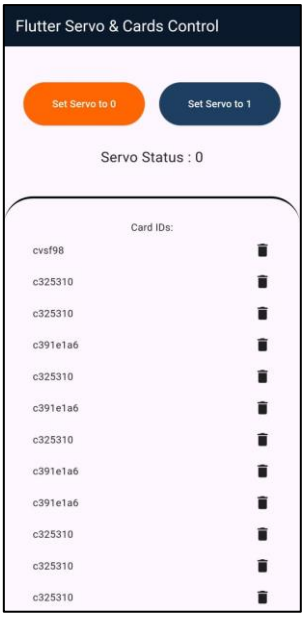
/servo/init-proj (untuk menginisialisasi nilai id menjadi 1 dan nilai status servo menjadi 0)

/servo/status (untuk membaca status servo)

/servo/update/:id (untuk mengubah status servo dengan parameter :id)

2 Endpoint di atas merupakan endpoint yang sudah dilakukan deployment ke railway.app, apabila tidak dilakukan deployment maka bisa menjalankan di desktop biasa kemudian diakses melalui ip address dan menyesuaikan portnya, contoh 192.168.x.x:8080/servo/status.

Tampilan aplikasi yang dibangun

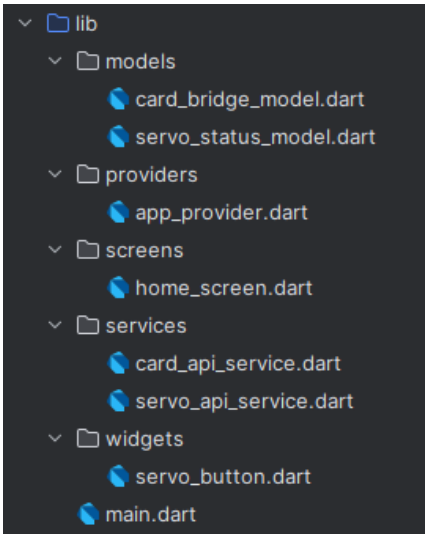


Tampilan Aplikasi Bab 8

Pada halaman aplikasi dapat dilihat bagian atas terdapat 2 button untuk mengontrol servo. Kemudian bagian bawahnya terdapat bentuk list yang merupakan id kartu yang sudah ada di database.

Saat mengubah status servo ataupun membaca dan menghapus id kartu aplikasi mengkonsumsi restful API menggunakan dio. Aplikasi melakukan *trigger* terhadap endpoint yang sudah dibuat menggunakan package dio. Aplikasi juga di*manage* menggunakan state management provider. Tambahkan package dio dan provider ke dalam flutter dan dapatkan versi terbarunya dari pub.dev.

Buatlah struktur project dengan folder dan file di dalam folder lib seperti berikut :



Struktur project

Isi dari file card_bridge_model.dart :

```dart
// To parse this JSON data, do
//
//     final cardBridgeModel = cardBridgeModelFromJson(jsonString);

import 'dart:convert';

CardBridgeModel cardBridgeModelFromJson(String str) =>
CardBridgeModel.fromJson(json.decode(str));

String cardBridgeModelToJson(CardBridgeModel data) =>
json.encode(data.toJson());

class CardBridgeModel {
  List<Result> result;

  CardBridgeModel({
    required this.result,
  });

  factory CardBridgeModel.fromJson(Map<String, dynamic> json) =>
CardBridgeModel(
    result: List<Result>.from(json["result"].map((x) =>
Result.fromJson(x))),
  );

  Map<String, dynamic> toJson() => {
    "result": List<dynamic>.from(result.map((x) => x.toJson())),
  };
}

class Result {
  String id;

  Result({
    required this.id,
  });
```

```
  factory Result.fromJson(Map<String, dynamic> json) => Result(
    id: json["id"],
  );

  Map<String, dynamic> toJson() => {
    "id": id,
  };
}
```

Isi dari file servo_status_model.dart :

```
// To parse this JSON data, do
//
//     final servoStatusModel = servoStatusModelFromJson(jsonString);

import 'dart:convert';

ServoStatusModel servoStatusModelFromJson(String str) =>
ServoStatusModel.fromJson(json.decode(str));

String servoStatusModelToJson(ServoStatusModel data) =>
json.encode(data.toJson());

class ServoStatusModel {
  List<Result> result;

  ServoStatusModel({
    required this.result,
  });

  factory ServoStatusModel.fromJson(Map<String, dynamic> json) =>
ServoStatusModel(
    result: List<Result>.from(json["result"].map((x) =>
Result.fromJson(x))),
  );

  Map<String, dynamic> toJson() => {
    "result": List<dynamic>.from(result.map((x) => x.toJson())),
  };
}

class Result {
  int id;
  int srvStatus;

  Result({
    required this.id,
    required this.srvStatus,
  });

  factory Result.fromJson(Map<String, dynamic> json) => Result(
    id: json["id"],
    srvStatus: json["srv_status"],
  );

  Map<String, dynamic> toJson() => {
    "id": id,
    "srv_status": srvStatus,
  };
}
```

Isi dari file card_api_service.dart :

```
import 'package:dio/dio.dart';
import 'package:mcs_bab_8/models/card_bridge_model.dart';

class CardApiService {
  Dio dio = Dio();
  String cardBridgeUrl = "https:// https://card-bridge-railway-
production.up.railway.app";

  Future<CardBridgeModel> getUid() async {
    try{
      final response = await dio.get("$cardBridgeUrl/cards");
      return CardBridgeModel.fromJson(response.data);
    }catch(e){
      rethrow;
    }
  }

  Future deleteCard({required String idCard}) async{
    try{
      final response = await
dio.delete("$cardBridgeUrl/card/delete/$idCard");
      return response.data;
    } catch(e){
      rethrow;
    }
  }
}
```

Base url disimpan di dalam variabel cardBridgeUrl yang nantinya akan diinterpolasi dengan variable tambahan saat dilakukan *trigger*. Fungsi getUid() digunakan untuk membaca id kartu yang ada di database, seedangkan deleteCard() digunakan untuk menghapus id kartu.

Isi dari file servo_api_service.dart :

```
import 'package:dio/dio.dart';
import 'package:mcs_bab_8/models/servo_status_model.dart';

class ServoApiService {
  Dio dio = Dio();
  String servoControllerUrl = "https://srv-cntrllr-railway-
production.up.railway.app";

  Future<ServoStatusModel> getServoStatus() async{
    try{
      final response = await dio.get("$servoControllerUrl/servo/status");
      return ServoStatusModel.fromJson(response.data);
    }catch(e){
      rethrow;
    }
  }

  writeServoStatus({required String status}) async {
    try{
```

```
      final response =  await
dio.put("$servoControllerUrl/servo/update/$status");
      return response.data;
    }catch(e){
      rethrow;
    }
  }
}
```

Base url disimpan di variabel servoControllerUrl yang nantinya akan diinterpolasi dengan variabel lain saat dilakukan *trigger*. Fungsi getServoStatus() digunakan untuk membaca status servo sedangkan fungsi writeServoStatus() digunakan untuk mengubah servo status.

Isi dari file app_provider.dart :

```
import 'package:flutter/cupertino.dart';
import 'package:mcs_bab_8/models/card_bridge_model.dart';
import 'package:mcs_bab_8/models/servo_status_model.dart';
import 'package:mcs_bab_8/services/card_api_service.dart';
import 'package:mcs_bab_8/services/servo_api_service.dart';

class AppProvider extends ChangeNotifier{
  ServoStatusModel? servoStatusModel;
  CardBridgeModel? cardBridgeModel;
  String servoStatus = "";
  String textLeftButton = "Set Servo to 0";
  String textRightButton= "Set Servo to 1";
  Color colorLeftButton = const Color(0xffFF6500);
  Color colorRightButton = const Color(0xff1E3E62);

  Stream getServoStatus() async*{
    while(true){
      yield servoStatusModel = await ServoApiService().getServoStatus();
      await Future.delayed(const Duration(seconds: 1));
      notifyListeners();
    }
  }

  Future changeServoStatus({required String status}) async{
    await ServoApiService().writeServoStatus(status: status);
    notifyListeners();
  }

  Stream getUid() async*{
    while(true){
      yield cardBridgeModel = await CardApiService().getUid();
      await Future.delayed(const Duration(seconds: 2));
      notifyListeners();
    }
  }

  Future deleteUid({required String uid}) async{
    await CardApiService().deleteCard(idCard: uid);
    notifyListeners();
  }
}
```

Persiapkan semua atribut keperlukan aplikasi baik berupa variabel ataupun fungsi yang nantinya akan di*provide* saat aplikasi berjalan.

Isi dari file main.dart :

```
import 'package:flutter/material.dart';
import 'package:mcs_bab_8/providers/app_provider.dart';
import 'package:mcs_bab_8/screens/home_screen.dart';
import 'package:provider/provider.dart';

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

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

  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (_) => AppProvider()),
      ],
      child: MaterialApp(
        title: 'mcs bab 8',
        theme: ThemeData(
          primarySwatch: Colors.blue,
          useMaterial3: true
        ),
        debugShowCheckedModeBanner: false,
        home: HomeScreen(),
      ),
    );
  }
}
```

Perisapkan provider dan tambahkan provider yang sudah dibuat agar state management provider dapat digunakan.

Sebelum membuat tampilan halaman atau *screen* pada aplikasi buatlah terlebih dahulu widget untuk dijadikan button saat mengontol servo. Button yang akan mengendalikan servo dibuat di dalam file yang bernama servo_button.dart.

Isi dari file servo_button.dart :

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

class ServoButton extends StatelessWidget {
  String textLeftButton;
  String textRightButton;
  Color colorLeftButton;
  Color colorRightButton;
  Function() onTapLeftButton;
```

```
  Function() onTapRightButton;

  ServoButton({
    super.key,
    required this.textLeftButton,
    required this.textRightButton,
    required this.colorLeftButton,
    required this.colorRightButton,
    required this.onTapLeftButton,
    required this.onTapRightButton,
  });

  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        ElevatedButton(
          style: ElevatedButton.styleFrom(
            padding: const EdgeInsets.symmetric(horizontal: 40, vertical:
20),
            backgroundColor: colorLeftButton,
          ),
          child: Text(
            textLeftButton,
            style: const TextStyle(color: Colors.white),
          ),
          onPressed: () {onTapLeftButton();},
        ),

        const SizedBox(width: 20),

        ElevatedButton(
          style: ElevatedButton.styleFrom(
            padding: const EdgeInsets.symmetric(horizontal: 40, vertical:
20),
            backgroundColor: colorRightButton,
          ),
          child: Text(
            textRightButton,
            style: const TextStyle(color: Colors.white),
          ),
          onPressed: () {onTapRightButton();},
        ),
      ],
    );
  }
}
```

Class ServoButton() adalah class yang akan membentuk button untuk mengotrol. Di dalam class tersebut terdapat *constructor* untuk kebutuhan tampilan *button*nya ataupun proses bisnisnya. *Constructor* ini akan diisi ketika class ServoButton() dipanggil. Class ServoButton() dipanggil saat membentuk halaman aplikasi di dalam file yang bernama home_screen.dart.

Isi dari file home_screen.dart :

```dart
import 'package:flutter/material.dart';
import 'package:mcs_bab_8/providers/app_provider.dart';
import 'package:mcs_bab_8/widgets/servo_button.dart';
import 'package:provider/provider.dart';

class HomeScreen extends StatefulWidget {
  HomeScreen({super.key});

  @override
  State<HomeScreen> createState() => _HomePageState();
}

class _HomePageState extends State<HomeScreen> {
  @override

  void initState() {
    // TODO: implement initState
    Provider.of<AppProvider>(context, listen: false).getUid();
    Provider.of<AppProvider>(context, listen: false).getServoStatus();
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Consumer<AppProvider>(
      builder: (context, appProvider, child) {
        return Scaffold(
          appBar: AppBar(
            title: const Text('Flutter Servo & Cards Control', style:
TextStyle(color: Colors.white),),
            backgroundColor: Color(0xff0B192C),
          ),
          body: Column(
            children: [
              const SizedBox(height: 50,),

              ServoButton(
                textLeftButton: appProvider.textLeftButton,
                textRightButton: appProvider.textRightButton,
                colorLeftButton: appProvider.colorLeftButton,
                colorRightButton: appProvider.colorRightButton,
                onTapLeftButton: () =>
appProvider.changeServoStatus(status: "0"),
                onTapRightButton: () =>
appProvider.changeServoStatus(status: "1"),
              ),

              const SizedBox(height: 30,),

              StreamBuilder(
                stream: appProvider.getServoStatus(),
                builder: (context, snapshot) {
                  if (snapshot.hasData){
                    appProvider.servoStatus =
appProvider.servoStatusModel!.result[0].srvStatus.toString();
                    return Container();
                  } else{
                    return Container();
                  }
                },
              ),
```

```
              Text(
                "Servo Status : ${appProvider.servoStatus}"
                , style: const TextStyle(fontSize: 20),
              ),

              const SizedBox(height: 40,),

              Container(
                width: double.infinity,
                decoration: const BoxDecoration(
                    border: Border(top: BorderSide(width: 3,color:
Colors.black)),
                    borderRadius: BorderRadius.only(
                      topLeft: Radius.circular(40), topRight:
Radius.circular(40),
                    )
                ),
                child: const Column(
                  children: [
                    SizedBox(height: 30,),
                    Text("Card IDs:"),
                  ],
                ),
              ),

              Expanded(
                child: StreamBuilder(
                  stream: appProvider.getUid(),
                  builder: (context, snapshot) {
                    if(snapshot.hasData){
                      return ListView.builder(
                        shrinkWrap: true,
                        physics: const AlwaysScrollableScrollPhysics(),
                        itemCount:
appProvider.cardBridgeModel!.result.length,
                        itemBuilder: (context, index) {
                          return Container(
                            margin: const EdgeInsets.symmetric(vertical:
10, horizontal: 40),
                            child: Row(
                              mainAxisAlignment:
MainAxisAlignment.spaceBetween,
                              children: [

Text(appProvider.cardBridgeModel!.result[index].id),
                                GestureDetector(
                                  child: const Icon(Icons.delete),
                                  onTap: () {
                                    appProvider.deleteUid(uid:
appProvider.cardBridgeModel!.result[index].id);
                                  },
                                )
                              ],
                            ),
                          );
                        },
                      );
                    } else{
                      return const Center(child: Text("no data to
display"));
```

```
                    }
                  },
                ),
              ),
            ],
          ),
        );
      },
    );
  }
}
```

Di dalam class ini membentuk halaman aplikasi sebagaimana gambar di atas. Fungsi getUid() dan fungsi getServoStatus() yang ada pada provider dipanggil di dalam initState() agar langsung menjalankan fungsi tersebut saat halaman aplikasi dibuka.2 fungsi tersebut dijalankan agar langsung membacar id kartu dan status servo yang ada pada database.

Class ServoButton() dipanggil untuk membentuk *button* pada aplikasi yang mengendalikan servo. Class ServoButton() memiliki *constructor* sehingga ketika class ServoButton() dipanggil maka akan diminta pengisian atribut yang ada pada class ServoButton().

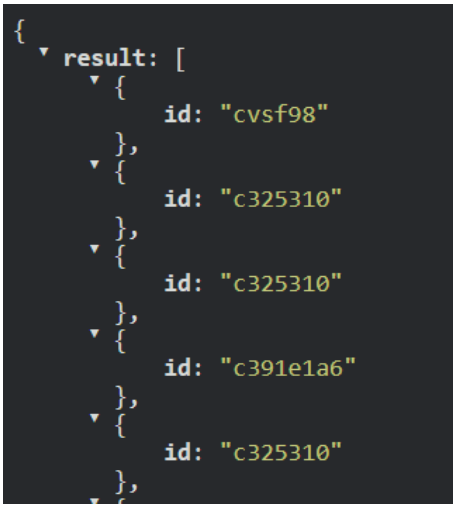
Streambuilder pada umumnya adalah widget pada flutter yang digunakan untuk mengambil data dari api secara realtime, namun di sini kita menggunakannya hanya untuk mengisi variabel servoStatusyang ada pada provider. Oleh karena itu return pada StreamBuilder di sini hanya berupa Container kosong saja. Adapun status dari servo ditampilkan di widget Text

```
Text(
  "Servo Status : ${appProvider.servoStatus}"
  , style: const TextStyle(fontSize: 20),
),
```

Kemudian lihat lagi pada StreamBuilder berikutnya yang digunakan untuk menampilkan id kartu. StreamBuilder akan *stream* ke fungsi getUid() yang ada pada provider. Di dalam fungsi getUid() kita bisa melihat terdapat perulangan untuk mendapatkan data id kartu. Perulangan dibuat agar mendapatkan data id kartu terbaru setiap 2 detik.

```
Stream getUid() async*{
  while(true){
    yield cardBridgeModel = await CardApiService().getUid();
    await Future.delayed(const Duration(seconds: 2));
    notifyListeners();
  }
}
```

Lalu jika saat *stream* terdapat data/data ada di database melalui API maka akan mengembalikan widget dalam bentuk ListView.builder(). ListView.builder digunakan untuk menampilkan id kartu dalam bentuk teks karena data id kartu yang ada pada API saat dibaca berbentuk list/array.



Contoh data id kartu dalam bentuk array di dalam API response

Lalu di bagian icon delete digunakan untuk menghapus id kartu yang dipilih

```
GestureDetector(
  child: const Icon(Icons.delete),
  onTap: () {
    appProvider.deleteUid(uid:
appProvider.cardBridgeModel!.result[index].id);
  },
)
```

Kemudian untuk bagian code untuk IOT beserta penjelasan silahkan kunjungi link https://github.com/Rokel15/GUNADARMA-ASCL-MCS di submenu yang bernama embeddedSystemComponents.

LAPORAN PENDAHULUAN (LP)

1. Apa nama package yang digunakan flutter untuk berkomunikasi dengan API?
2. Berikan perbedaan StreamBuilder dan FutureBuilder!
2. Berikan pengertian request body pada API!
3. Berikan pengertian response body pada API!

LAPORAN AKHIR (LA)

1. Berikan kesimpulan pada bab 8!