BAB VIII
CARD READER & SERVO CONTROLLER

Pada praktikum MCS bab 8, praktikan akan membangun sebuah aplikasi yang dapat mengontrol servo dan melihat data id kartu yang masuk ke *database* melalui *sensor Radio Frequency Identification* (RFID). Agar dapat mengontrol servo dan membaca id kartu yang masuk, kita akan mengonsumsi data API yang telah dibuat pada pertemuan praktikum bab 6 dan bab 7.

8.1 Tujuan Praktikum

Tujuan	Penjelasan
Mengetahui cara mengontrol servo menggunakan kartu RFID	Pada bab ini, praktikan akan dijelaskan mengenai bagaimana cara untuk menggerakan servo menggunakan kartu RFID
Memahami cara memasangkan id kartu dengan database	Bab ini akan menggunakan kartu RFID yang di daftarkan ke *database* untuk memantau perubahan kondisi servo

8.2 Persyaratan Praktikum

Disarankan praktikan menggunakan *hardware* dan *software* sesuai pada dokumentasi ini. Apabila terdapat versi yang lumayan lampau dari versi yang direkomendasikan atau *hardware* yang lawas maka sebaiknya bertanya kepada Asisten Mengajar Shift.

***HARDWARE* YANG DIBUTUHKAN PRAKTIKUM**	**JENIS**
PC / Laptop CPU	**≥ 4 Cores**
PC / Laptop RAM	**≥ 8 GB**
PC / Laptop *Storage*	**≥ 10 GB**

SOFTWARE YANG DIBUTUHKAN PRAKTIKUM

Android *Studio* / *Visual Studio Code*

8.3 Materi Praktikum

Pada pertemuan sebelumnya, kita telah membuat 2 *database* dengan beberapa *route endpoint*. *Database* dan *endpoint* yang dibuat pada bab 6 merupakan *endpoint* untuk menangani proses pembacaan data kartu yang masuk ke *database* melalui RFID. Sedangkan, *database* dan *endpoint* pada bab 7 digunakan untuk mengontrol servo.

- *Endpoint* yang dibangun pada bab 6:

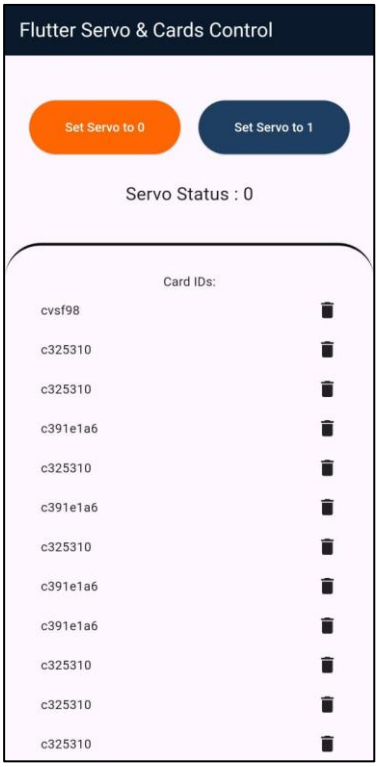
/cards (*Route* untuk membaca kartu dengan metode GET) ***/card/input/:id*** (*Route* untuk meng*input* id kartu ke *database* melalui parameter :id dengan menggunakan metode POST) ***/card/delete/:id*** (*Route* untuk menghapus id kartu dari *database* melalui parameter :id dengan menggunakan metode DELETE)

- *Endpoint* yang dibangun pada bab 7:

/servo/init-proj (*Route* untuk menginisialisasi nilai id menjadi 1 dan nilai status servo menjadi 0 dengan menggunakan metode POST) **/servo/status** (*Route* untuk membaca status servo dengan menggunakan metode GET) ***/servo/update/:srv_status*** (*Route* untuk mengubah status servo dengan parameter :srv_status dengan menggunakan metode PUT)

8.4 Prosedur Praktikum

8.4.1 Tampilan Aplikasi



Gambar 8.1 Tampilan Halaman Aplikasi yang Akan diimplementasikan

Penjelasan terkait bagaimana cara aplikasi bekerja akan diterangkan oleh asisten yang mengajar.

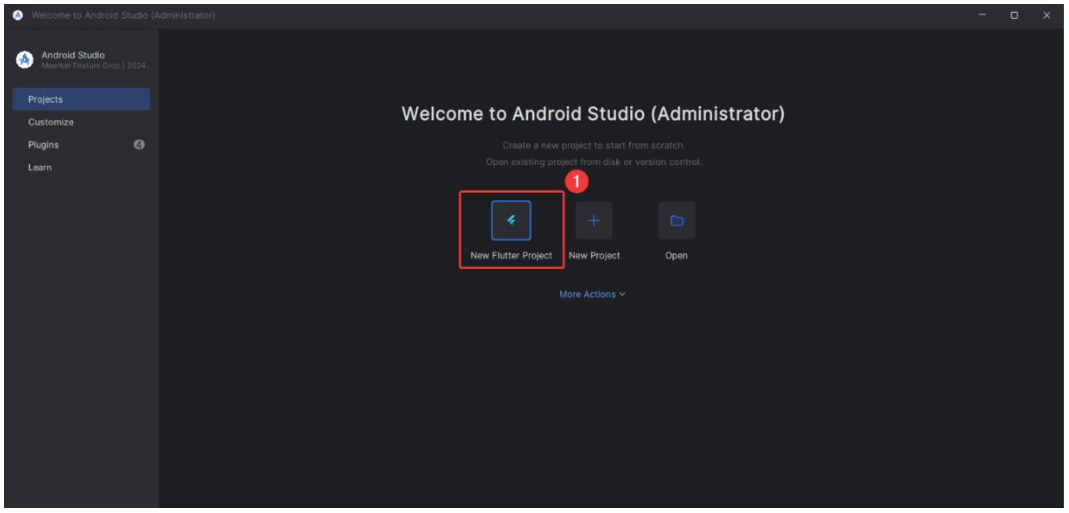
8.4.2 Implementasi Aplikasi

Implementasi aplikasi dilakukan dalam 2 lingkup berbeda, yakni lingkup pembangunan tampilan aplikasi menggunakan Flutter dan lingkup konfigurasi terhadap sensor yang digunakan. Sensor yang digunakan pada praktikum ini, antara lain servo dan RFID.

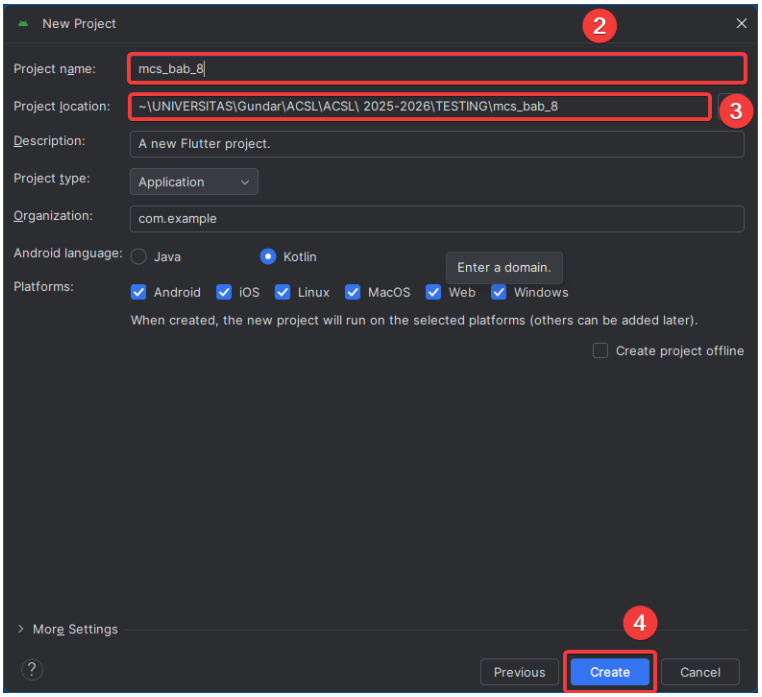
8.4.2.1 Pembuatan Aplikasi

Dalam mengimplementasikan tampilan dari desain aplikasi di atas, terdapat beberapa langkah yang harus dilewati terlebih dahulu agar proses praktikum dapat berjalan dengan lancar dan terselesaikan sesuai dengan apa yang

dituju. Langkah-langlah dalam pembuatan *project* baru sama seperti yang telah dilakukan pada praktikum pertemuan sebelumnya.



Gambar 8.2 Tampilan Awal *Software* Android *Studio*



Gambar 8.3 Proses Pembuatan *Project* Baru

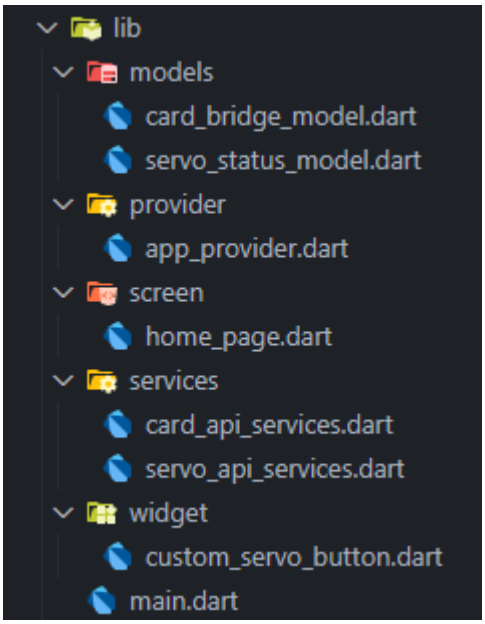
Setelah *project* berhasil terbentuk masuklah ke dalam file **pubspec.yaml** dan tambahkan *package* **provider** dan **dio**. Ketika sudah menambahkan kedua *package* tersebut jangan lupa untuk di **pub get** agar *package* yang ditambahkan dapat digunakan.

```
dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.8
  provider: ^6.1.5+1
  dio: ^5.9.0
```

Gambar 8.4 *Package* yang digunakan

Setelah melakukan pub get, buatlah struktur *tree project*, seperti yang terlihat pada Gambar 8.5.



Gambar 8.5 Struktur Tree Project

Berikutnya ketika struktur *tree project* sudah tersusun seperti pada gambar, masuklah ke dalam file **card_bridge_model.dart** dan isikan kode program berikut:

```
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,
  };
}
```

Setelah file tersebut terisikan dengan kode program yang membangun model untuk mendapatkan data API yang dikirimkan melalui RFID, langkah

berikutnya adalah membangun model yang akan digunakan untuk mengambil data yang dikirimkan oleh servo. Kode program tersebut dibentuk pada file **servo_status_model.dart**.

```
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,
  };
}
```

Setelah membuat kedua model *object* dart berdasarkan *response* API yang diberikan, langkah berikutnya adalah melakukan penulisan kode untuk file **card_api_service.dart** dan **servo_status_service.dart**. Berikut merupakan kode yang digunakan untuk membangun *service* API dari card_bridge_model.dart

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

class CardApiService {
  Dio dio = Dio();
  String cardBridgeUrl = "https://<ip_address>:<PORT>";

  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;
    }
  }
}
```

Kode program tersebut digunakan untuk meng*handle* bagian RFID. Pada bagian awal *class* tersebut, didefinisikan variabel **cardBridgeUrl** yang berisikan url yang tersusun dari **ip address** dan **port** yang telah ditentukan. Kemudian, terdapat 2 fungsi yang didefinisikan, yakni fungsi **getUid()** yang digunakan untuk membaca data id kartu yang tersimpan di dalam *database* dan fungsi **deleteCard()** yang digunakan untuk menghapus data id kartu dari *database*.

Selanjutnya masuklah ke dalam file **servo_api_status.dart** dan tuliskan kode program berikut:

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

class ServoApiService {
  Dio dio = Dio();
  String servoControllerUrl = "https://<ip_address>:<PORT>";

  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;
    }
  }
}
```

Kode program tersebut digunakan untuk meng*handle* bagian servo. Pada bagian awal *class* tersebut, didefinisikan variabel **servoControllerUrl** yang berisikan url yang tersusun dari **ip address** dan **port** yang telah ditentukan. Kemudian, terdapat 2 fungsi yang didefinisikan, yakni fungsi **getServoStatus()** yang digunakan untuk membaca status dari servo dan fungsi **writeServoStatus()** yang digunakan untuk menggerakan servo.

Kemudian masuklah ke dalam file **app_provider.dart** dan tuliskan kode program berikut ke file tersebut:

```
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();
  }
}
```

Kode program tersebut akan menginisialisasi seluruh atribut, seperti variabel dan fungs yang diperlukan ke dlaam 1 file. Sehingga, kita dapat menggunakannya secara berulang tanpa harus mendefinisikan dari awal. Berikutnya masuklah ke dalam file **main.dart** dan isikan dengan kode program berikut:

```
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(),
      ),
    );
  }
}
```

Pada file main.dart terlihat bahwa terdapat pemanggilan terhadap *class* AppProvider() yang bertujuan agar seluruh variabel dan fungsi yang telah didefinisikan pada provider dapat langsung dijalankan bersamaan pada saat aplikasi dijalankan. Selanjutnya, kita akan membuat sebuah *class* yang di dalamnya berisikan kode program yang akan membangun suatu tombol yang akan digunakan untuk mengontrol servo. Masuklah ke dlaam file **custom_servo_button.dart** dan masukkan kode program berikut:

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

class CustomServoButton extends StatelessWidget {
  String textLeftButton;
  String textRightButton;
  Color colorLeftButton;
  Color colorRightButton;
```

```
  Function() onTapLeftButton;
  Function() onTapRightButton;

  CustomServoButton({
    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 CustomServoButton() merupakan sebuah *class* yang akan membentuk tampilan *button* yang akan mengotrol servo. Di dalam *class* tersebut terdapat *constructor* untuk kebutuhan tampilan *button*nya ataupun proses bisnisnya. *Constructor* ini akan diisi ketika class ServoButton() dipanggil.

Setelah proses pembuatan *button* selesai dilakukan, bukalah file **home_page.dart** dan masukkan kode program berikut:

```
import 'package:flutter/material.dart';
import 'package:mcs_bab_8/provider/app_provider.dart';
import 'package:mcs_bab_8/widget/custom_servo_button.dart';
import 'package:provider/provider.dart';


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

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

```
class _HomePageState extends State<HomePage> {
  @override
  void 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),

              CustomServoButton(
                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),
```

```
              Row(
                children: [
                  Text("Servo   Status   :   ",   style:   const
TextStyle(fontSize: 20)),
                  StreamBuilder(
                    stream: appProvider.getServoStatus(),
                    builder: (context, snapshot) {
                      if       (snapshot.connectionState       ==
ConnectionState.waiting) {
                        return Text("-");
                      } else if (snapshot.hasError) {
                        return Center(
                          child:          Text("Error          =>
${snapshot.error}"),
                        );
                      } else {
                        appProvider.servoStatus = appProvider
                            .servoStatusModel!
                            .result[0]
                            .srvStatus
                            .toString();
                        return Text(appProvider.servoStatus);
                      }
                    },
                  ),
                ],
              ),

              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 ID
Register:")],
                ),
              ),

              Expanded(
                child: StreamBuilder(
                  stream: appProvider.getUid(),
                  builder: (context, snapshot) {
                    if        (snapshot.connectionState        ==
ConnectionState.waiting) {
                      return                        Center(child:
CircularProgressIndicator());
                    } else if (snapshot.hasError) {
                      return Center(
                        child:    Text("Error    to    get    ID:
${snapshot.error}"),
                      );
                    }  else   if   (snapshot.data   ==   null   ||
!snapshot.hasData) {
                      return const Center(child: Text("No data to
display"));
                    } else {
                      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,
                                      );
                                    },
                                  ),
                                ],
                              ),
                            );
                          },
                        );
                      }},
                    ),),
                ],
              ),);
        },);
    }
}
```

Pada kode program tersebut, terdapat widget **StreamBuilder()** yang digunakan. Widget tersebut umumnya digunakan ketika kita ingin membuat sebuah aplikasi yang menampilkan data secara *real time*. Penggunaan StreamBuilder() digunakan dalam 2 kondisi, yakni kondisi untuk meng*handle* status servo dan kondisi untuk meng*handle* data id kartu.

```
return Consumer<AppProvider>(
  builder: (context, appProvider, child) {
    return Scaffold(
      appBar: AppBar(
        // ...
      ),
      body: Column(
        children: [

          // ...

          Row(
            children: [
              Text("Servo    Status    :    ",    style:    const
TextStyle(fontSize: 20)),
              StreamBuilder(
                stream: appProvider.getServoStatus(),
                builder: (context, snapshot) {
                  if         (snapshot.connectionState         ==
ConnectionState.waiting) {
                    return Text("-");
                  } else if (snapshot.hasError) {
                    return Center(
                      child: Text("Error => ${snapshot.error}"),
                    );
                  } else {
                    appProvider.servoStatus = appProvider
                        .servoStatusModel!
                        .result[0]
                        .srvStatus
                        .toString();
```

```
                    return Text(appProvider.servoStatus);
                  }
                },
              ),
            ],
          ),

          // ...

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

Kode di atas merupakan kode yang akan meng*handle* status dari servo. Widget StreamBuilder() pada kode tersebut akan melakukan ***stream*** atau pemantauan secara langsung terhadap fungsi getStatusServo() yang telah didefinisikan pada provider. Terdapat beberapa kondisi yang akan ditampilkan, bergantung kepada proses apa yang sedang dijalankan. Jika proses pengambilan data masih berlangsung, maka aplikasi akan menampilkan *text* "-". Jika setelah pengambilan data ditemukan error, maka aplikasi akan menampilkan pesan error. Namun, jika aplikasi berhasil mengambil data, maka aplikasi akan menampilkan data berupa status dari servo tersebut.

```
return Consumer<AppProvider>(
  builder: (context, appProvider, child) {
    return Scaffold(
      appBar: AppBar(
        // ...
      ),
      body: Column(
        children: [
          // ...

          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 ID
Register:")],
            ),
          ),

          Expanded(
            child: StreamBuilder(
              stream: appProvider.getUid(),
              builder: (context, snapshot) {
                if (snapshot.connectionState ==
ConnectionState.waiting) {
                  return Center(child:
CircularProgressIndicator());
                } else if (snapshot.hasError) {
                  return Center(
                    child: Text("Error to get ID:
${snapshot.error}"),
                  );
                } else if (snapshot.data == null ||
!snapshot.hasData) {
                  return const Center(child: Text("No data to
display"));
                } else {
                  return ListView.builder(
                    shrinkWrap: true,
```

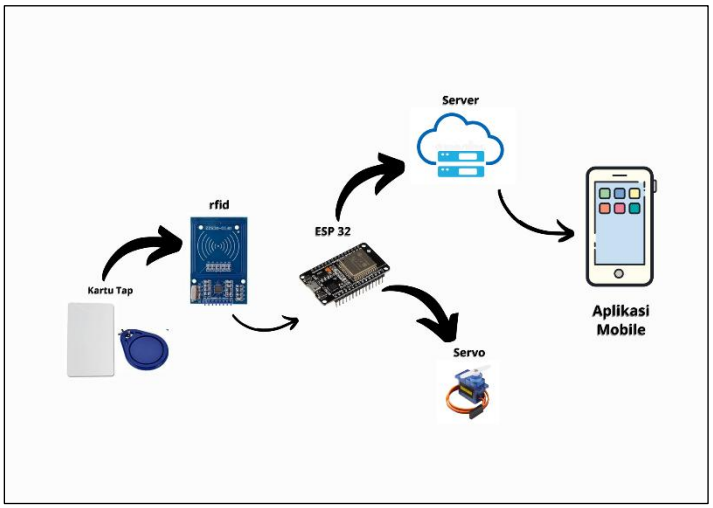
```
                    physics:                                    const
AlwaysScrollableScrollPhysics(),
                    itemCount:
appProvider.cardBridgeModel!.result.length,
                    itemBuilder: (context, index) {
                      return Container(
                        margin: 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,
                                );
                              },),
                          ],
                        ),
                      );
                    },);
                }},
            ),),
        ],
      ),);
  },);
```

Kode di atas merupakan kode yang akan meng*handle* data id kartu. Sama seperti kode program sebelumnya, kode program ini akan melakukan ***stream*** terhadap fungsi getUid() yang telah didefinisikan dan akan menghasilkan tampilan yang berbeda-beda berdasarkan kondisi yang sedang dilewati. Jika proses pengambilan data masih berlangsung, maka aplikasi akan animasi *loading* yang berada di bagian tengah. Jika setelah pengambilan data ditemukan error, maka aplikasi akan menampilkan pesan error. Jika, data tersebut beesifat null atau data *response* kosong, maka aplikasi akan menampilkan pesan bahwa data id tidak tersedia. Namun, jika aplikasi berhasil mengambil data, maka aplikasi akan menampilkan seluruh id *card* yang terdaftar.

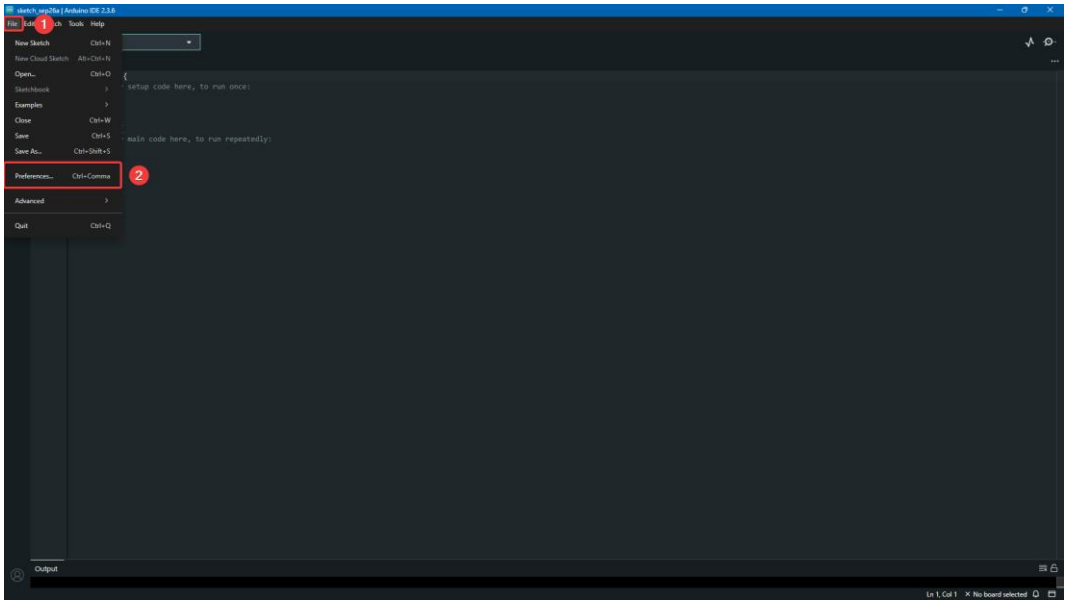
8.4.2.2 Konfigurasi Alat

Setelah proses pembangunan aplikasi selesai dilakukan, maka kita dapat berpindah pada proses konfigurasi sensor-sensornya. Praktikum memanfaatkan ESP32, RFID-RC522, dan Servo Motor untuk membaca tag RFID dan mengontrol servo berdasarkan instruksi dari server melalui HTTP *request*. Dengan memanfaatkan koneksi Wi-Fi, sistem ini memungkinkan komunikasi *realtime* dengan server, yang dapat digunakan untuk aplikasi seperti kontrol akses atau otomatisasi berbasis RFID.

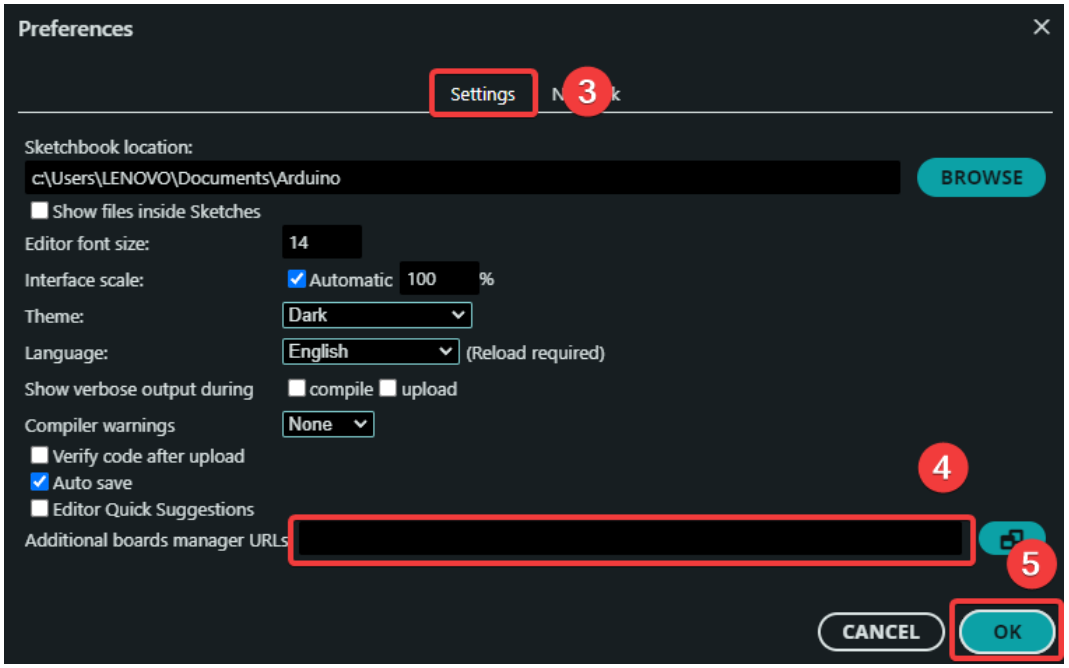


Gambar 8.6 Ilustrasi Arsitektur *Internet of Things*

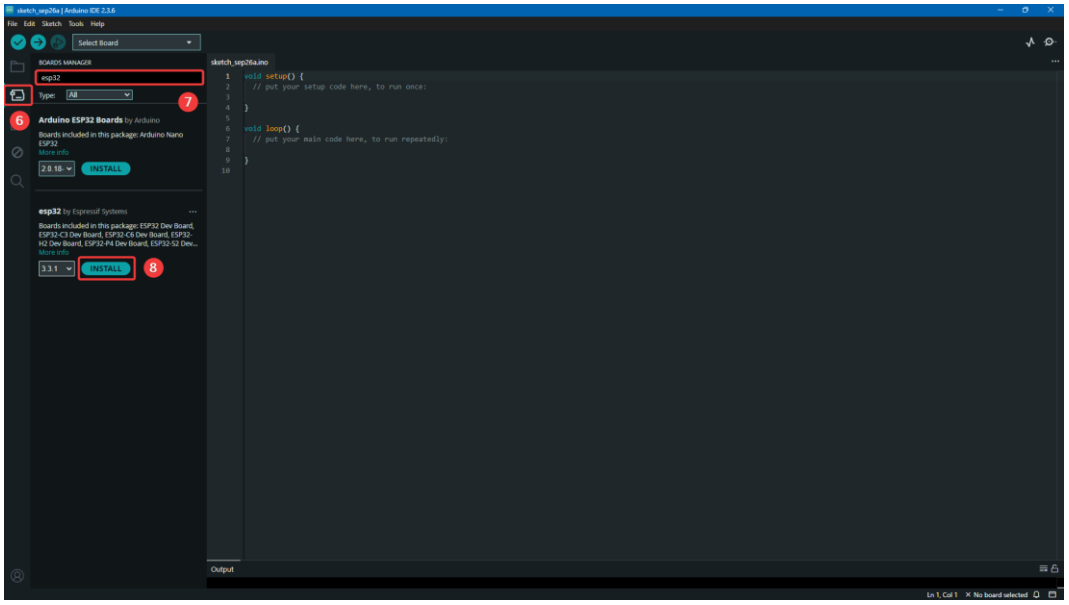
Bukalah *software* Arduino IDE yang telah terinstall pada *platform* anda dan lakukanlah instalasi terhadap *package* ESP-32 dengan mengikut langkah-langkah berikut.



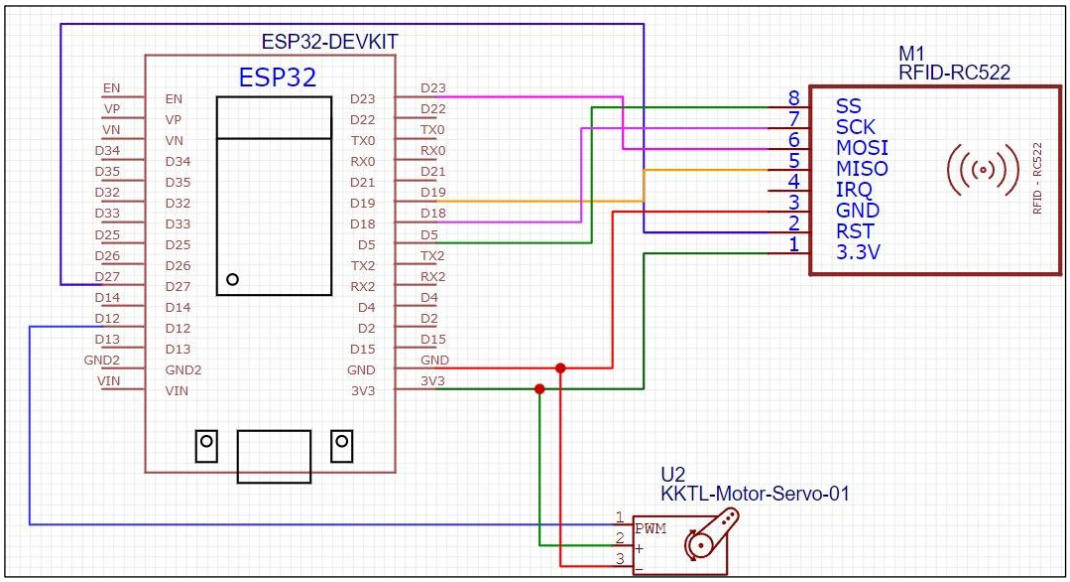
Gambar 8.7 Proses Instalasi *Package* ESP-32



Gambar 8.8 Proses Instalasi *Package* ESP-32



Gambar 8.9 Proses Instalasi Modul ESP-32



Gambar 8.10 Skematik Rangkaian

Komponen	Nama Pin	Pin ESP32	Keterangan
Servo	Signal	GPIO 12	Pin untuk sinyal kontrol servo
RFID	SS (SDA)	GPIO 5	Pin untuk Slave Select (SS)
RFID	RST	GPIO 27	Pin untuk Reset RFID
RFID	MOSI	GPIO 23	Pin untuk Master Out Slave In
RFID	MISO	GPIO 19	Pin untuk Master In Slave Out
RFID	SCK	GPIO 18	Pin untuk Serial Clock
RFID	GND	GND	Ground
RFID	3.3V	3.3V	Tegangan 3.3V

Gambar 8.11 Pin Servo dan ESP-32

Setelah proses *wiring* selesai dilakukan, kembalilah ke *software* Arduino IDE dan masukan kode program berikut:

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <SPI.h>
#include <MFRC522.h>
#include <ESP32Servo.h>

#define SS_PIN  5    // ESP32 pin GPIO5
#define RST_PIN 27   // ESP32 pin GPIO27

const char* ssid = "LAB LANJUT 121";        // SESUAIKAN DENGAN
SSID Wi-Fi YANG TERHUBUNG
const char* password = "12345678"; // SESUAIKAN DENGAN PASSWORD
Wi-Fi YANG TERHUBUNG

const               char*               serverURL               =
"http://192.168.121.185:8081/servo/status";

MFRC522 rfid(SS_PIN, RST_PIN);
Servo myServo;

void setup() {
  Serial.begin(115200);
  myServo.attach(12);
```

```
  WiFi.begin(ssid, password); // CONNECT TO Wi-Fi
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("Connected to WiFi");

  SPI.begin();           // INITIALIZE SPI BUD
  rfid.PCD_Init();       // INITIALIZE MFRC522

  Serial.println("Tap an RFID/NFC tag on the RFID-RC522 reader");
}

void loop() {
  // Check for RFID tag
  if (rfid.PICC_IsNewCardPresent()) {
    if (rfid.PICC_ReadCardSerial()) {
      MFRC522::PICC_Type               piccType               =
rfid.PICC_GetType(rfid.uid.sak);
      Serial.print("RFID/NFC Tag Type: ");
      Serial.println(rfid.PICC_GetTypeName(piccType));

      // PRINT UID IN SERIAL MONITOR IN HEX FORMAT
      String uidStr = "";
      Serial.print("UID:");
      for (int i = 0; i < rfid.uid.size; i++) {
        Serial.print(rfid.uid.uidByte[i] < 0x10 ? " 0" : " ");
        Serial.print(rfid.uid.uidByte[i], HEX);
        uidStr += String(rfid.uid.uidByte[i], HEX);
      }
      Serial.println();

      // SEND UID TO SERVER
      sendUIDToServer(uidStr);

      rfid.PICC_HaltA();        // HALT PICC
```

```
      rfid.PCD_StopCrypto1();   // STOP ENCRYPTION ON PCD
    }
  }
  checkServoStatus();
  delay(200);
}

void sendUIDToServer(String uid) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    String url = "http://192.168.121.185:8080/card/input/" + uid;
// SESUAIKAN DENGAN IP DAN PORT
    http.begin(url);

    int httpResponseCode = http.POST("");

    if (httpResponseCode > 0) {
      String response = http.getString();
      Serial.println("Server Response: " + response);
    } else {
      Serial.println("Error    on    sending    POST:    "    +
String(httpResponseCode));
    }

    http.end();
  } else {
    Serial.println("WiFi not connected");
  }
}

// FUNCTION UNTUK STATUS SERVO
void checkServoStatus() {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    http.begin(serverURL);

    int httpResponseCode = http.GET();
```

```
    if (httpResponseCode > 0) {
      String payload = http.getString();
      Serial.println("HTTP Response: " + payload);

     // Parse JSON response
      if (payload.indexOf("\"srv_status\":1") != -1) {
        // JIKA srv_status BERNILAI 1, SERVO AKAN BERGERAK KE CCW
        Serial.println("Servo moving to CCW");
        myServo.write(0);
      } else if (payload.indexOf("\"srv_status\":0") != -1) {
        // JIKA srv_status BERNILAI 0, SERVO AKAN BERGERAK KE CW
        Serial.println("Servo moving to CW");
        myServo.write(180);
      } else {
        Serial.println("Unknown status received");
      }
    } else {
      Serial.println("Error on HTTP request");
    }
    http.end();
  } else {
    Serial.println("WiFi Disconnected");
  }
}
```