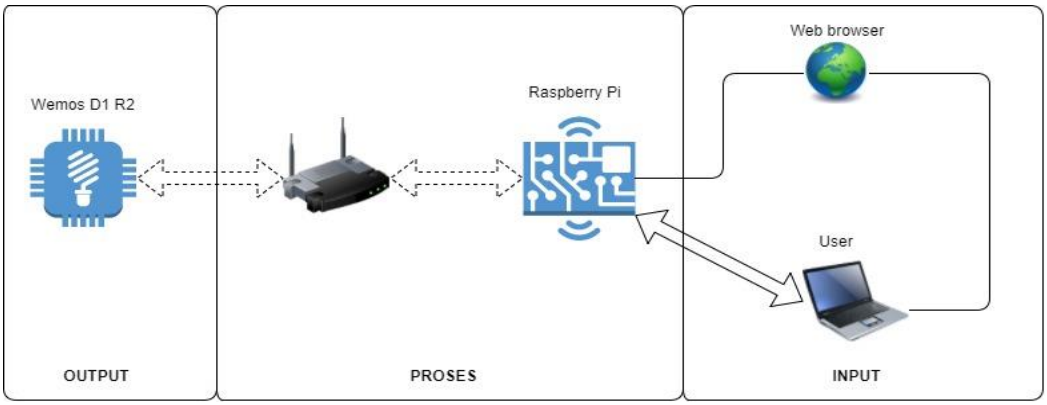
BAB 5 PERANCANGAN DAN IMPLEMENTASI

5.1 Perancangan Sistem

Proses perancangan sistem dibagi menjadi beberapa tahapan yang dimulai dengan diagram sistem dan dilanjutkan dengan diagram alir kerja sistem serta penjelasannya. Setelah itu dilakukan perancangan jaringan komunikasi yang digunakan pada sistem. Pada perancangan jaringan komunikasi yaitu dengan menggunakan koneksi wifi. Setelah perancangan jaringan selesai, dilanjutkan dengan perancangan server *websocket* dan *main server*. Kemudian dilakukan perancangan terhadap basis data yang digunakan pada sistem. Terakhir dilakukan perancangan terhadap program php yang akan menampilkan dan menangani antarmuka pada web browser.

5.1.1 Diagram Sistem

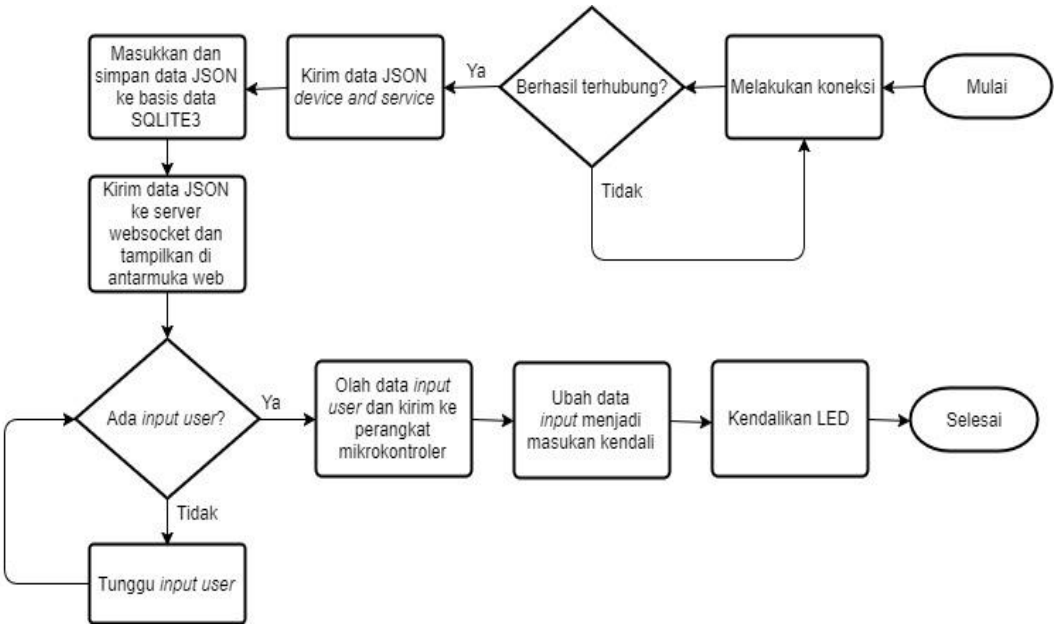
Pengembangan sistem digambarkan dengan diagram seperti yang ditunjukkan pada gambar berikut ini:



Gambar 5.1 Diagram Sistem

Berdasarkan Gambar 5.1, sistem terdiri dari *user* yang berupa laptop atau PC dan sebuah raspberry pi yang di dalamnya terdapat web browser. Pengguna bisa mengakses antarmuka pada web browser langsung pada raspberry pi atau bisa juga melalui laptop atau PC eksternal yang terhubung ke raspberry pi. Wemos D1 R2 terhubung ke raspberry pi menggunakan media komunikasi *wireless* (nirkabel). Koneksi wemos dengan raspberry menggunakan komunikasi socket UDP (*User Datagram Protocol*). Wemos terhubung ke raspberry melalui akses poin dimana raspberry juga terhubung ke akses poin tersebut. Wemos dan raspberry harus berada dalam satu subnet jaringan yang sama. Wemos nantinya dapat mencari keberadaan raspberry di dalam jaringan dengan melakukan *scanning* port pada raspberry yang digunakan dalam aplikasi pada skripsi ini.

5.1.2 Diagram Alir Kerja Sistem



Gambar 5.2 Diagram Alir Kerja Sistem

Berdasarkan Gambar 5.2, digambarkan sebuah diagram alir kerja sistem dimana sistem akan memulai koneksi mikrokontroler ke raspberry pi (wemos D1 R2). Setelah itu apakah berhasil terhubung, ketika masih terdapat *error* pada saat koneksi, maka akan kembali lagi ke melakukan koneksi dan dicek *error* apa yang terjadi. Yang dicek ketika terjadi *error* adalah di bagian konfigurasi alamat IP dan port, apakah sudah sesuai dan apakah sudah pada satu jaringan antara raspberry dan wemos.

Saat sudah berhasil terhubung, maka data JSON *device* dan *service* dikirim dan dimasukkan ke dalam file basis data yang bernama serverDatabase.db. Setelah itu, data JSON diambil dari basis data dan diteruskan dengan mengirim data JSON tersebut ke server *websocket* untuk kemudian ditampilkan pada web browser. Setelah tampil pada web browser, sistem menunggu adanya *input* yang dimasukkan. Ketika masih belum terdapat *input*, sistem akan terus menunggu kecuali sampai adanya interrupt eksternal. Ketika sistem telah mendeteksi adanya *input* yang dimasukkan, sistem akan mengolah data *input* tersebut menjadi suatu perintah yang akan dikirimkan ke mikrokontroler yaitu wemos, yang nantinya oleh wemos akan diubah menjadi data masukan yang berupa nilai yang digunakan untuk mengendalikan LED. Fungsi yang digunakan untuk mengubah data *input* menjadi masukan nilai kendali adalah `toInt` yang akan mengubah data *input* yang awalnya berupa valid string menjadi integer. Nilai berupa integer tersebut yang menjadi nilai masukan yang digunakan untuk mengendalikan LED.

5.1.3 Perancangan Jaringan Komunikasi

Pada perancangan jaringan komunikasi akan dibahas mengenai rancangan komunikasi yang digunakan pada sistem. Yaitu menggunakan media koneksi komunikasi wifi. Berikut adalah mekanismenya:

Gambar 5.3 Komunikasi via Wifi

Komunikasi via wifi pada sistem ini dilakukan pada jaringan yang sama. Keduanya antara wemos dan raspberry harus terkoneksi ke suatu jaringan tertentu, yang setelahnya baru dapat saling berkomunikasi melalui *host* pada alamat IP dan PORT yang telah dikonfigurasi pada sisi raspberry.

5.1.4 Perancangan Server *Websocket*

Server *websocket* dirancang untuk menangani koneksi klien atau *user* atau web browser yang digunakan untuk antarmuka pengendalian LED. Server *websocket* juga digunakan untuk menangani setiap perangkat yang terhubung. Hai ini dirancang untuk dapat menangani dan berkomunikasi dengan web browser dan perangkat dimana setiap yang terhubung akan disimpan pada variabel `clients` yang sudah didefinisikan dengan alamat port 8888. Server *websocket* juga dirancang untuk mengetahui data JSON yang berupa *device* dan *service* dari perangkat wemos sehingga pengguna atau *user* dapat mengetahui tentang perangkat yang akan dikendalikan dan servis apa saja yang tersedia. Server *websocket* dibuat menggunakan web *framework* dan *library* jaringan asinkron yaitu tornado.

5.1.5 Perancangan *Main Server*

Main server dirancang untuk berinteraksi dengan perangkat yang terhubung. Pada *main server* terdapat bagian yang akan mengolah data string dari *input* user melalui antarmuka web, data tersebut diolah menjadi urutan perintah yang kemudian dikirim ke wemos.

5.1.6 Perancangan Basis Data

Basis data dirancang sesuai tabel berikut ini:

Tabel 5.1 Rancangan Basis Data

Nama Tabel	Kolom	Deskripsi
Device	device_id	varchar(32) primary key

	device_name	varchar(32)
	Port	varchar(32)
	device_con	varchar(10)
Service	service_id	varchar(32) primary key
	service_name	varchar(32)
	service_type	varchar(32)
	service_data	varchar(32)
	service_value	varchar(32)
device_service	device_id	varchar(32) primary key
	service_id	varchar(32)

Berdasarkan Tabel 5.1, basis data yang dibuat terdiri dari tiga jenis tabel yakni *device*, *service* dan *device_service* yang masing-masing mempunyai kolom yang akan dijelaskan berikut ini:

- Device
    - device_id : digunakan untuk menyimpan ID dari perangkat yang terhubung. Didefinisikan dengan tipe data varchar(32) yang dapat menampung data sebanyak 32 karakter dan diatur menjadi *primary key*.
    - device_name : digunakan untuk menyimpan nama dari perangkat yang terhubung. Didefinisikan dengan tipe data varchar(32) yang dapat menampung data sebanyak 32 karakter.
    - port : pada port berapa perangkat terhubung. Didefinisikan dengan tipe data varchar(32) yang dapat menampung data sebanyak 32 karakter.
    - device_con : perangkat terhubung melalui media usb atau wifi. Didefinisikan degan tipe data varchar(10) yang dapat menampung data sebanyak 10 karakter.
- Service
    - service_id : digunakan untuk menyimpan ID dari *service* yang ditawarkan perangkat yang terhubung. Diatur sebagai *primary key* dan tipe data yang digunakan adalah varchar(32).
    - service_name : digunakan untuk menyimpan nama dari *service* yang ditawarkan perangkat yang terhubung. Tipe data yang digunakan adalah varchar(32).

- service_type : digunakan untuk menyimpan tipe dari *service* yang ditawarkan perangkat yang terhubung. Tipe data yang digunakan adalah varchar(32).
- service_data : digunakan untuk menyimpan data dari *service* yang ditawarkan perangkat yang terhubung. Tipe data yang digunakan adalah varchar(32).
- service_value : digunakan untuk menyimpan nilai (*value*) dari *service* yang ditawarkan perangkat yang terhubung. Tipe data yang digunakan adalah varchar(32).

- Device_service
  - device_id : digunakan untuk menyimpan ID dari perangkat. Diatur sebagai *primary key* dengan tipe data varchar(32).
  - service_id : digunakan untuk menyimpan ID dari *service* yang ditawarkan oleh perangkat. Tipe data yang digunakan adalah varchar(32).

5.1.7 Perancangan Antarmuka Web

Penataan antarmuka web dirancang menggunakan ekstensi program php. Dan script yang digunakan menggunakan javascript dan digunakan *library* jquery untuk penulisan kode javascript-nya. Setiap *input* yang diberikan melalui antarmuka web, diteruskan ke file ini (*tornado.php*) untuk selanjutnya dikirimkan ke server *websocket* untuk digunakan sebagai data *input* bagi wemos yang terhubung. Antarmuka ini dapat diakses dari *host* dan melalui jaringan manapun. Hal ini dirancang untuk fleksibilitas *user* agar ketika mereka sedang terhubung pada jaringan manapun, mereka (*user*) akan sangat mudah untuk mengakses sistem ini.

5.2 Implementasi Sistem

Proses implementasi sistem dilakukan saat semua perancangan telah selesai dilakukan. Proses implementasi terdiri dari implementasi jaringan komunikasi wifi, implementasi server *websocket*, implementasi *main server*, implementasi basis data, implementasi antarmuka web dan implementasi program wemos D1 R2.

5.2.1 Implementasi Jaringan Komunikasi Wifi

Perangkat wemos D1 R2 berkomunikasi menggunakan media komunikasi wifi. Wemos D1 R2 awalnya diimplementasikan sebagai klien yang akan melakukan proses *scanning* terhadap port raspberry dan setelah ditemukan maka wemos akan berubah peran sebagai server yang akan menerima setiap permintaan kontrol LED yang dikirimkan raspberry. Implementasi komunikasi ini dimulai dengan menjalankan program connectedWifi.py pada raspberry. Program connectedWifi.py juga berperan sebagai *main server* untuk mengirimkan data *input user* ke wemos. Inisialisasi port pada raspberry mengawali jalannya program. Wemos terhubung ke raspberry pada port yang telah didefinisikan menggunakan socket UDP.

Program 1: IdleClientState pada Wemos

```
1  void idleClientState(){
2     deviceData.printTo(sendDevice);
3     Udp.beginPacket("255.255.255.255", 8000);
4     Udp.print(sendDevice);
5     Udp.println("");
6     Udp.endPacket();
7     sendDevice.remove(0);
8     arduinoStateMachine.transitionTo(waitRequest);
9     arduinoStateMachine.update();
10 }
```

Berdasarkan Program 1, *state* yang akan berjalan pertama kali ketika wemos dinyalakan adalah idleClientState. Di dalam fungsi tersebut terdapat perintah untuk mencari port raspberry yaitu port 8888 dengan melakukan *broadcast* ke semua perangkat yang ada di jaringan. Setelah raspberry ditemukan, data JSON dikirim dengan menggunakan perintah `Udp.print(sendDevice);`. Setelah data JSON dikirim, *state* akan berubah menjadi serverWaitReq *state*.

Program 2: ServerWaitReq dan waitRes State pada Wemos

```
1  void serverWaitReq(){
2    int packetSize = Udp.parsePacket();
3    if(packetSize){
4      Serial.println();
5      Serial.print("Menerima Paket Dengan Ukuran: ");
6      Serial.println(packetSize);
7
8      int len = Udp.read(packetBuffer, packetSize);
9      if(len > 0){
10       packetBuffer[len] = 0;
11     }
12     Serial.println();
13     Serial.println("Konten ");
14     Serial.print(String(packetBuffer));
15     int index = String(packetBuffer).indexOf('_');
16     String serviceId = String(packetBuffer).substring(0,index);
17     String valueData = String(packetBuffer).substring((index+1));
18     for(int i = 0; i < totalService; i++){
19       if(serviceList[i][0] == serviceId){
20         serviceHandler[i](valueData);
21         serviceList[i][4] = valueData;
22         generateJSON();
23         delay(500);
24         deviceData.printTo(sendDevice);
25         Udp.beginPacket("255.255.255.255", 8000);
26         Udp.print(sendDevice);
27         Udp.println("");
28         Udp.endPacket();
29         sendDevice.remove(0);
30         arduinoStateMachine.transitionTo(waitResponse);
31         arduinoStateMachine.update();
32       }
33     }
34   }
35 }
36
37 void waitRes(){
38   arduinoStateMachine.transitionTo(waitRequest);
39   arduinoStateMachine.update();
40 }
```

Seperti pada Program 2, serverWaitReq *state* digunakan untuk menunggu adanya permintaan kontrol terhadap LED yang dikirimkan raspberry. Permintaan kontrol dikirim dalam bentuk valid string yang berupa *service_id* dan *service-_value*. Pesan akan di-*parsing* terlebih dahulu dan ditampung pada variabel packetSize untuk mengetahui panjang pesan yang diterima. Nilai inilah yang akan digunakan sebagai patokan untuk proses pembacaan pesan asli yang nantinya akan ditampung pada variabel packetBuffer. Setelah pesan diketahui maka akan dipisah terlebih dahulu menggunakan perintah `indexOf('_')` dan ditampung pada variabel index. *Service_id* diketahui dengan menggunakan perintah `substring(0,index)` sementara *service_value* dapat diketahui dengan menggunakan perintah `substring((index+1))`. *Service_id* dan *service_value* menjadi patokan yaitu servis mana yang akan dikontrol apakah on off, dimming atau rgb dan nilai kontrol yang diberikan tertampung pada variabel *service-value*. Sementara untuk waitRes *state* akan mengembalikan *state* ke serverWaitReq setelah respon untuk kontrol LED diberikan.

```
Program 3: UDP Socket
1   sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
2   start_time = time.time()
4   print("discovery start_time: %s" % start_time)
5   sock.bind((host, port))
6
7   data, addr = sock.recvfrom(1024)
8   print addr[0]
9   print addr[1]
10  print("")
11  dataRecv = json.loads(data)
12  print dataRecv
13  print ("")
14  deviceCon = "WIFI"
15  insert_to_database(dataRecv, addr[0] + ":" + str(addr[1]),
16  deviceCon)
17  deviceData = read_device_data()
18  websocket_send_once('dev' + deviceData)
19  end_time = time.time()
20  print("discovery end_time: %s" % end_time)
21  time_taken = end_time - start_time
22  print("Discovery time: %s" % time_taken)
23  print("")
```

Seperti pada Program 3, di sisi raspberry diimplementasikan program python yang bernama connectedWifi.py dimana program tersebut akan menghandel koneksi dari wemos menggunakan *socket* UDP. Koneksi dari wemos akan di-*bind* pada variabel host dan port, dimana host adalah alamat IP dari raspberry dan port adalah port yang digunakan yaitu port 8000. Setelah di-*bind*, data JSON yang dikirim wemos akan diterima dan ditampung pada variabel data. Untuk variabel addr menampung alamat (IP dan port) dari wemos. Data JSON tersebut akan di-*load* menjadi data tuple dengan menggunakan perintah `json.loads(data)`. Kemudian data tersebut dimasukkan ke dalam basis data dengan memanggil fungsi `insert_to_database`. Data yang dimasukkan adalah data yang berasal dari wemos yaitu data *service description* dan *device description*, pasangan IP dan port wemos serta deviceCon, yakni string "WIFI", yang menandakan bahwa koneksi menggunakan wifi. Sebelum data dikirim ke server websocket dengan memanggil fungsi `websocket_send_once`, data akan di-*read* terlebih dahulu menggunakan pemanggilan fungsi `read_device_data`.

5.2.2 Implementasi Server *Websocket*

Pada implementasi server *websocket* digunakan *library* `tornado.websocket`, `tornado.web`, `tornado.httpserver` dan `tornado.ioloop`.

```
Program 4: Variabel clients dan port
1  #declaring variables
2  clients = [] #variable reserved for connected client to
3  websocket
4  port = 8888 #websocket used port
```

Berdasarkan Program 4, setiap terdapat koneksi atau klien yang terhubung, maka akan disimpan pada variabel clients dalam bentuk list. Setiap ada klien baru

yang terhubung, maka akan ditambahkan pada list dengan menggunakan perintah `clients.append`. Port yang diinisialisasikan adalah alamat port 8000. Dalam implementasi server *websocket*, di dalamnya terdapat sebuah fungsi yang bernama *read* untuk membaca *device data* dari basis data serverDatabase.db.

Program 5: Fungsi read

```
1  #read device data from database
2  def read():
3      deviceData = read_device_data()
4      return deviceData
```

Program 6: Fungsi device_handler

```
1   #received device information handler
2   def device_handler(data):
3       data_device = "upd"
4
5       #read data device from received message
6       for line in data:
7           data_device += line
8
9       #send data device to all client, actually intended to web
10  browser
11      for c in clients:
12          c.write_message(data_device)
```

Berdasarkan Program 6, fungsi `device_handler` digunakan untuk menangani pengiriman data ke klien yang terhubung. Terdapat variabel `data_device` yang berisi "`upd`". Hal itu merupakan sebuah penanda untuk file *tornado.php* ketika menerima data dari perangkat yang terhubung melalui server websocket. Saat server *websocket* menerima data dari perangkat, data tersebut akan ditambahkan dengan penanda "`upd`" sebelum dikirimkan ke *tornado.php* menggunakan fungsi `c.write_message` untuk dijadikan sebagai tampilan antarmuka pada web browser.

Program 7: Websocket Handler

```
1   #websocket handler
2   class WSHandler(tornado.websocket.WebSocketHandler):
3       def check_origin(self, origin):
4           return True
5
6       #when connection has been made from clients
7       def open(self):
8           clients.append(self) #add new client to clients variable
9           deviceJson = read() #read device data from database
10          self.write_message("upd" + deviceJson) #send data device to
11  recently connected client
12
13      #when message received
14      def on_message(self, message):
15          print (message)
16
17          #when data device information is received
18          if message[0:3] == 'dev':
19              device_handler(message[3:])
20
21          #when message service request is received
22          elif message[0:3] == 'ser':
23              for c in clients:
24                  c.write_message(message)
25
26          #when message service respons has been received
27          elif message[0:3] == "res" :
28              message = message[3:]
29              print(message)
30
31      #when connection is closed by a client
32      def on_close(self):
33           #message = "putus"
34          clients.remove(self)
```

Websocket handler digunakan untuk menangani koneksi dan komunikasi *websocket* yang terjadi. Koneksi yang datang ditangani oleh fungsi `open` yang di dalamnya terdapat baris kode untuk melakukan penambahan klien ke variabel `clients`. Ketika *message* diterima, maka akan ditangani pada fungsi `on_message`. *Message* yang diterima dikelompokkan berdasarkan karakter ke 0 – 3. Jika karakter itu adalah dev, *message* akan dikirim menggunakan fungsi `device_handler`. *Message* yang dikirim adalah karakter setalah 3 atau data JSON *device* dan *service*. Saat *user* memasukkan nilai *input* pada antarmuka web, nilai tersebut akan ditangani oleh percabangan *message* 'ser'. Percabangan *message "*res" digunakan untuk menangani pesan respon dari perangkat yang terhubung. Di akhir bagian tersebut terdapat fungsi on_close yang berfungsi untuk menutup koneksi *websocket*.

5.2.3 Implementasi Data JSON

Implementasi data JSON dilakukan pada wemos. Berikut ini adalah implementasinya.

Program 8: Implementasi Data JSON

```
1  void generateJSON(){
2    deviceData["device_id"] = device_id;
3    deviceData["device_name"] = device_name;
4    JsonArray& service = deviceData.createNestedArray("service");
5    for(int i = 0; i < totalService; i++){
6      JsonObject& serviceData = jsonBuffer.createObject();
7      serviceData["service_id"] = serviceList[i][0];
8      serviceData["service_name"] = serviceList[i][1];
9      serviceData["service_type"] = serviceList[i][2];
10     serviceData["service_data"] = serviceList[i][3];
11     serviceData["service_value"] = serviceList[i][4];
12     service.add(serviceData);
13   }
14 }
```

Seperti pada Program 8, JSON diimplementasikan di dalam fungsi yang bernama generateJSON. Di dalamnya terdapat perintah untuk membuat pasangan nilai yaitu `device_id` dan isi dari variabel `device_id`, serta `device_name` dan isi dari variabel `device_name`. Informasi ID dari setiap perangkat ditandai dengan device_id dan nama dari setiap perangkat ditandai dengan device_name. Pada baris program terdapat tulisan `deviceData`, itu merupakan sebuah nama Json *Object* dimana nama tersebut digunakan untuk menampung seluruh data JSON yang dibuat. Terdapat juga perintah untuk membuat *nested array* yang merupakan *child* dari obyek utama. Data di dalamnya adalah informasi ID *service* dari masing-masing *service* yang dideskripsikan, nama *service*, tipe *service* (*input* atau *output*), *service* data (bool, int, dsb) dan *service value* yaitu nilai dari *service* yang diberikan oleh *user*.

5.2.4 Integrasi Data JSON dalam Javascript, HTML dan PHP

Untuk dapat ditampilkan pada antarmuka *web browser*, data JSON dari mikrokontroler harus diintegrasikan dengan javascript, html dan program PHP. Integrasi ini dilakukan pada sebuah file yang bernama tornado.php. File inilah yang akan menampilkan antarmuka *web browser* sehingga *user* dapat berinteraksi dengan sistem. Di dalam file tersebut berisi baris-baris program. Integrasi data JSON dimulai dengan melakukan pembacaan *input user* yang ditampung pada variabel `reqData` kemudian dikirim menggunakan perintah `ws.send` ke server *websocket* menggunakan penanda atau *tag* `ser`. Terdapat `a`, `b`, `c` dan `reqData` yang masing-masing menyimpan informasi dari jenis koneksi perangkat yang digunakan, yaitu USB dan/atau WIFI, port atau alamat IP yang digunakan, *service ID* dan *service value*.

Program 9: Pembacaan User Input

```
1
2   var tes;
3                   $(function(){
4                           var ws;
5                           tes = function tes(a,b,c,d) {
6                                   var reqData =
7   document.getElementById("status"+d).value
8                                   ws.send("ser" + a + "_" + b +
9   "_" +  c + "_" + reqData);
10                                  //  alert("ser" + a + "_" + b +
11  "_" +  c + "_" + reqData);
12                                  //  d = new Date();
13                                  //  alert(d.getTime());
14                          }
15
```

Program 10: Penanda upd

```
1   var logger = function(msg) {
2           if (msg.slice(0,3) == "upd"){
3                   d = new Date();
4                   //  alert(msg);
5                   msg = msg.slice(3);
6                   var str = msg.split("\n");
7                   var i, cnt = "<table><tbody>";
8                   cnt = cnt + "<tr><th class='hed' style='width:
9   25%;'>Port</th>" +
10                  "<th class='hed' style='width: 25%;'>Device</th>"
11  +
12                  "<th class='hed' style='width:
13  50%;'>Service</th></tr>";
```

Berdasarkan Program 10, upd merupakan penanda atau *tag* yang digunakan untuk menerima data JSON dari server *websocket*. Data tersebut diterima dan ditampilkan pada tabel Port, Device dan Service. Terdapat perulangan untuk menampilkan data JSON pada tabel Port dan Device. Sebelum dapat ditampilkan, data JSON harus dilakukan *parsing* terlebih dahulu menggunakan fungsi `JSON.parse`. Kemudian terdapat juga perulangan untuk menghandel data dari *service* yang tersedia, dimana di dalamnya terdapat perintah untuk membuat kolom *input* dan tombol (*button*). Ketika tombol (*button*) ditekan oleh *user*, program akan mengeksekusi fungsi tes dengan mengirimkan data *device connection*, port, *service ID* dan reqData yang merupakan nilai *input* dari *user*. Kemudian dikirim menggunakan perintah `ws.send` ke server websocket yang selanjutnya akan dijadikan sebagai data *input* untuk mikrokontroler.

5.2.5 Implementasi Obyek LED

Pada skripsi ini digunakan LED dan RGB LED sebagai output dari sistem yang dibuat. Implementasi LED dilakukan pada wemos. LED akan mendapatkan perintah kontrol dari *user* melalui antarmuka pada *web browser*. Nilai yang diberikan akan dikonversi menjadi `int` menggunakan perintah `toInt` dan nilai ini dihandel oleh sebuah `serviceHandler` dan diteruskan ke fungsi dari *service* yang dideskripsikan. Dalam hal ini adalah onOff, dimmer dan rgb.

Program 11: Fungsi onOff

```
1  void onOff(String value){
2    if(value == "1"){
3      digitalWrite(D6, HIGH);
4    }
5    if(value == "0"){
6      digitalWrite(D6, LOW);
7    }
8  }
```

Seperti pada Program 11, *value* (nilai) *input* dari *user* akan dibaca. Ketika nilainya adalah 1, LED akan menyala dan sebaliknya, ketika nilai yang dibaca adalah 0, LED akan mati. Tetapi saat nilai yang diberikan di luar dari 1 dan 0, maka sistem tidak akan memberikan pengaruh terhadap LED. Pin output yang digunakan untuk LED menyesuaikan sesuai yang digunakan pada wemos.

Program 12: Fungsi dimmer

```
1  void dimmer(String value){
2    if(value <= "255"){
3      analogWrite(D7, value.toInt());
4    }
5    else{
6      analogWrite(D7, LOW);
7    }
8  }
```

Fungsi dimmer digunakan untuk menghandel permintaan pengendalian PWM terhadap LED. Nilai maksimum dari PWM adalah 255. Ketika dan selama nilai *input* dari *user* kurang dari sama dengan 255, maka output yang diberikan ke LED menyesuaikan dengan *input* dari *user* dan tingkat cahaya LED yang menyala akan bervariatif. Tetapi ketika nilai *input* yang diberikan lebih dari 255, LED akan mati. Pin yang digunakan menyesuaikan wemos.

Program 13: Fungsi rgb

```
1  void rgb(String value){
2    if(value){
3      int indexRGB0 = value.indexOf("_");
4      int indexRGB1 = value.indexOf("_", indexRGB0+1);
5      String reqID0 = value.substring(0,indexRGB0);
6      String reqID1 = value.substring(indexRGB0+1, indexRGB1);
7      String reqID2 = value.substring((indexRGB1+1));
8      setColor(reqID0.toInt(), reqID1.toInt(), reqID2.toInt());
9    }
10 }
11
12 void setColor(int redValue, int greenValue, int blueValue) {
13   analogWrite(redPin, redValue);
14   analogWrite(greenPin, greenValue);
15   analogWrite(bluePin, blueValue);
16 }
```

Fungsi rgb digunakan untuk menghandel pemintaan pengendalian RGB LED. Sedangkan fungsi setColor digunakan untuk memberikan sinyal analog ke RGB LED yang terhubung. Pin tempat RGB LED terhubung menyesuaikan wemos. Di dalam fungsi rgb terdapat kondisi if else, dimana nilai *input* dari *user* akan dibaca dan dilihat, ketika nilai sama dengan 255, maka RGB LED akan menyala dengan warna

merah. Ketika nilai sama dengan 143, warna yang akan menyala adalah violet dan ketika nilai sama dengan 128, warna yang akan menyala adalah biru dongker.

5.2.6 Implementasi Basis Data SQLITE3

Basis data ini berfungsi untuk menyimpan data JSON dan nilai (*value*) dari *service* yang tersedia. Basis data diberi nama serverDatabase.db. Pembuatan basis data menggunakan perintah `sqlite3 serverDatabase.db`. Implementasi basis data dan tabel-tabel yang dibuat dilakukan sesuai perancangan. Basis data dihandel menggunakan file yang bernama database_handler.py. Di dalam file tersebut terdapat beberapa fungsi yang memiliki masing-masing perannya yaitu `database_connect`, `read_device_data`, `insert_to_database`, `delete_from_database`, `update_service_data`, `clear_database`.

Program 14: Fungsi database_connect

```
1  #connect to used database
2  def database_connect():
3      try:
4          conn = sqlite3.connect(database)
5      except:
6          pass
7      return conn, conn.cursor()
```

Pertama, dilakukan koneksi ke basis data terlebih dahulu menggunakan fungsi `database_connect`. Koneksi ke basis data dilakukan dengan perintah `sqlite3.connect(database)`. Setelah berhasil terkoneksi, program akan menjalankan fungsi berikutnya yaitu `read_device_data`. Fungsi ini dipanggil ketika ingin membaca data yang disimpan dalam basis data. Data yang diberikan dari fungsi ini adalah device_id pada *row* ke-0, device_name pada *row* ke-1, port pada *row* ke-2 dan device_con pada *row* ke-3. Kemudian data *service* yang tampilkan adalah service_id, service_name, service_type, service_data dan service_value.

Fungsi `insert_to_database` dipanggil ketika ingin memasukkan data ke dalam basis data yang telah dibuat. Fungsi ini memiliki tiga parameter yaitu data, port dan con. Yang dilakukan pertama kali adalah koneksi ke basis data terlebih dahulu, kemudian *query* `INSERT INTO device VALUES` dieksekusi untuk memasukkan data device_id, device_name, port dan con ke dalam basis data. *Query* `INSERT INTO service VALUES` dieksekusi untuk memasukkan data service_id, service_name, service_type, service_data dan service_value ke dalam basis data. Setelah data berhasil dimasukkan, koneksi dari basis data akan di-*commit* dan ditutup menggunakan perintah `conn.commit` dan `conn.close`.

Program 15: Fungsi delete_from_database

```
#delete target device data from database
def delete_from_database(port):
    conn, c = database_connect()
    query = ""

    #get device_id from the taget device
    query = "SELECT device_id FROM device WHERE port=\'"
+ port + "\'"
    c.execute(query)
    device_id = c.fetchone()

    #get all service_id from target device
    query = "SELECT service_id FROM device_service WHERE
device_id=\'" + str(device_id[0]) + "\'"
    c.execute(query)
    service_id = c.fetchall()

    #delete target device data from DEVICE_SERVICE table
    query = "DELETE FROM device_service WHERE
device_id=\'" + str(device_id[0]) + "\'"
    c.execute(query)

    #delete target device data from DEVICE table
    query = "DELETE FROM device WHERE device_id=\'" +
str(device_id[0]) + "\'"
    c.execute(query)

    #delete target device's all service from SERVICE
table
    for sid in service_id:
        query = "DELETE FROM service WHERE service_id=\'"
+ str(sid[0]) + "\'"
        c.execute(query)

    conn.commit()
    conn.close()
```

Fungsi delete_from_database, hapus data dari basis data dengan menggunakan parameter port sebagai referensi data mana yang ingin dihapus. Dalam fungsi tersebut, terdapat perintah untuk melakukan *fetching* data yaitu data device_id dan service_id. Hal ini berguna untuk menandai data yang akan dihapus. Setelah itu perintah *query* `DELETE FROM device_service WHERE device_id` akan mengahapus data dari tabel device_service. *Query* `DELETE FROM device WHERE device_id` akan menghapus data dari tabel device. Kemudian terdapat perintah untuk menghapus semua data *service* dari tabel service.

Program 16: Fungsi update_service_data

```
#update service data after service was done
def update_service_data(data):
    conn, c = database_connect()

    if data[1] == "00013":
        data1 = "00013"
        s = '_'
        data = (data[2], data[3], data[4])
        data_join = s.join( data )
        #data2 = "dunia"

        query = "UPDATE service SET service_value = \'" +
data_join + "\' WHERE service_id = \'" + data1 + "\'"
        c.execute(query)

    else:
        query = "UPDATE service SET service_value = " +
data[2] + " WHERE service_id = \'" + data[1] + "\'"
        c.execute(query)

    conn.commit()
    conn.close()
```

Fungsi `update_service_data` dipanggil saat ingin memperbarui isi dari basis data pada tabel service dan kolom service_value. Setiap terdapat perubahan nilai pada service_value ketika sedang melakukan kendali LED pada mikrokontroler, maka nilai tersebut disimpan dan diperbarui yang kemudian nilai akan kembali ditampilkan pada antarmuka *web browser* sebagai informasi bagi *user* yang menjalankannya. Terakhir terdapat fungsi `clear_database` yang akan menghapus semua data dari basis data yang digunakan. Berikut adalah tampilan fungsi `clear_database` yang dimaksud.

Program 17: Fungsi clear_database

```
#delete all database data
def clear_database():
    conn, c = database_connect()

    c.execute("DELETE FROM device")
    c.execute("DELETE FROM service")
    c.execute("DELETE FROM device_service")

    conn.commit()
    conn.close()
```

Terdapat tiga *query* yang digunakan yaitu `DELETE FROM device`, `DELETE FROM service` dan `DELETE FROM device_service`. Masing-masing akan menghapus data dari tabel device, service dan device_service.