BAB 5 PERANCANGAN DAN IMPLEMENTASI SISTEM

Pada bab ini memberikan penjelasan mengenai perancangan dan proses implementasi sistem yang meliputi perancangan sistem menggunakan diagram blok perangkat keras, skematik perangkat keras kemudian beberapa *state machine* yang ditanam pada perangkat sensor, aktuator maupun *gateway*. Implementasi sistem terdiri atas implementasi perangkat keras serta perangkat lunak.

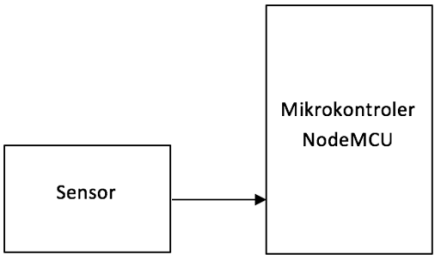
5.1 Perancangan Sistem

Pada bagian ini dijelaskan proses perancangan sistem yang dimulai dari perancangan perangkat keras meliputi diagram blok serta skematik perangkat keras dan perancangan perangkat lunak berupa *state machine* yang dijelaskan menggunakan *Finite State Machine*.

5.1.1 Perancangan Perangkat Keras

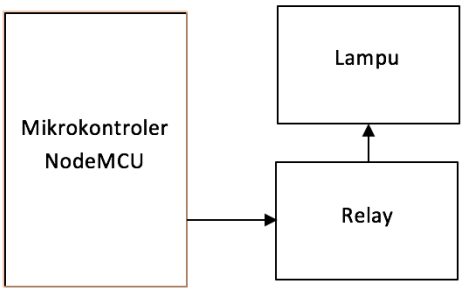
5.1.1.1 Diagram Blok

Perangkat keras pada sistem ini meliputi perangkat sensor serta perangkat aktuator. Kedua perangkat keras tersebut memiliki peran yang berbeda-beda, perangkat sensor bertugas untuk mengakuisisi data sensor serta mengirimkan data sensor ke aktuator dan perangkat aktuator bertugas untuk menerima data sensor yang telah dikirimkan oleh perangkat sensor serta melakukan suatu aksi terhadap objek aktuator menggunakan referensi data yang telah dikirimkan oleh perangkat sensor.



Gambar 5.1 Diagram blok perangkat sensor

Pada sistem ini memiliki 2 perangkat sensor pendukung yakni, perangkat dengan mikrokontroler serta sensor PIR untuk mendeteksi gerakan pada suatu ruangan dan perangkat dengan mikrokontroler serta sensor LDR untuk mengetahui kondisi cahaya pada suatu ruangan. Pada Gambar 5.1, Mikrokontroler yang digunakan untuk perangkat sensor yakni NodeMCU dikarenakan telah memiliki modul *wifi* yang sudah tertanam pada mikrokontroler tersebut serta setiap perangkat sensor memiliki sumber daya masing-masing. Setiap sensor pada perangkat sensor menghasilkan data-data yang berbeda, keseluruhan data tersebut memberikan dampak yang berbeda juga ketika diolah pada perangkat aktuator.



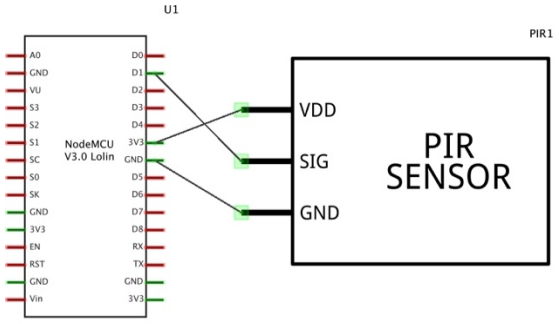
Gambar 5.2 Diagram blok perangkat aktuator

Pada Gambar 5.2, sama halnya pada perangkat sensor, pada perangkat aktuator juga menggunakan mikrokontroler yang sama yakni, NodeMCU. Jenis aktuator yang digunakan pada sistem ini adalah lampu yang di kontrol menggunakan *relay* berdasarkan data-data sensor yang masuk serta diolah pada perangkat aktuator. Jadi setelah data-data pada sensor diolah di mikrokontroler hingga menghasilkan keputusan dan kemudian hasil tersebut yang akan dijadakan dasar untuk melakukan aksi baik mematikan lampu ataupun menyalakan lampu. Pada perangkat aktuator juga didukung dengan sumber daya sehingga semua proses akan berjalan dengan baik.

Komponen sumberdaya yang digunakan pada kedua perangkat ini baik perangkat sensor serta aktuator menggunakan *powerbank* yang ditancapkan pada *micro-usb* pada komponen mikrokontroler NodeMCU. Selain fungsinya mengunggah *firmware* atau kode program kedalam mikrokontroler, *micro-usb* dapat digunakan oleh perangkat sumberdaya untuk memberikan sejumlah daya yang dibutuhkan oleh mikrokontroler NodeMCU.

5.1.1.2 Skematik

Pada tahap perancangan ini, penulis menggunakan perangkat lunak berupa Fritzing untuk melakukan pembuatan skematik. Skematik ini memberikan pendekatan pada perangkat keras untuk hubungan antara sumber daya serta sensor maupun aktuator yang digunakan dengan mikrokontroler sehingga dapat mempermudah dalam proses implementasi.

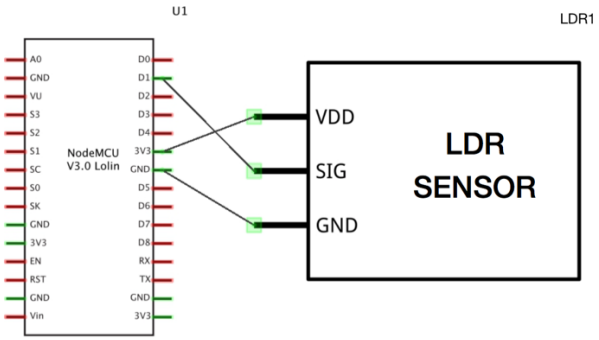


Gambar 5.3 Konfigurasi pin perangkat sensor menggunakan sensor pir

Pada Gambar 5.3 dijelaskan bahwa sensor PIR memiliki 3 pin, yakni 1 pin dibutuhkan untuk memberikan tegangan yang dihubungkan ke pin 3v3 pada mikrokontroler, 1 pin lainnya untuk memberikan *ground* yang dihubungkan

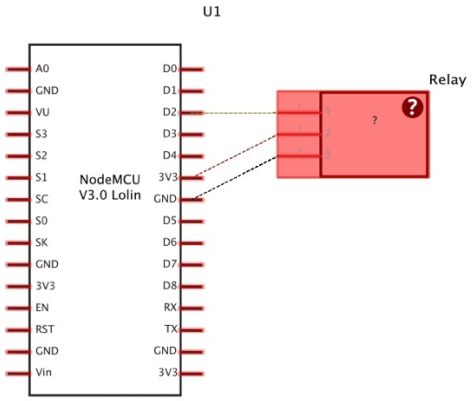
kepada pin *ground* pada mikrokontroler dan 1 pin untuk mengirimkan data sensor kepada mikrokontroler yang telah diakuisisi oleh sensor yang dihubungkan pada pin D0 pada mikrokontroler. Pin D1 pada mikrokontroler memiliki fungsi bahwa pin tersebut hanya menerima masukan berupa data digital saja sehingga tipe sensor yang dapat diproses hanyalah tipe sensor digital. Angka 0 pada pin hanya sebuah penanda nomor untuk identitas pin dan perlu didefiniskan pada *firmware*

Sensor PIR merupakan salah satu sensor yang menghasilkan data digital sehingga data yang diberikan hanyalah angka biner. Selain itu, sensor yang digunakan untuk perangkat sensor lainnya yakni, sensor LDR. Di pasaran, sensor ini memiliki 2 tipe yakni, LDR digital serta LDR analog dan sensor LDR yang digunakan pada sistem ini yakni LDR digital dikarenakan dengan tipe ini dapat menghasilkan data digital sehingga dapat mudah diproses serta mikrokontroler jenis ini sangat baik dalam memproses sinyal digital dibuktikan jumlah pin digital sangatlah banyak dibandingkan jumlah pin analog yang hanya memiliki 1.



Gambar 5.4 Konfigurasi pin perangkat sensor menggunakan sensor ldr

Pada Gambar 5.4, sensor yang digunakan yakni LDR digital sehingga kita tidak perlu menambahkan komponen elektronik lainnya untuk mendukung sensor tersebut, sensor ini telah memiliki modul yang didalam terangkai beberapa komponen elektronik yang saling terhubung sehingga sensor tersebut memiliki keluaran berupa data digital. Sensor ini memiliki sususan pin yang sama dengan sensor pir yakni, memiliki 3 pin yang menghubungkan pin vdd pada ldr dengan pin 3v3 pada mikrokontroler untuk memberikan tegangan kepada sensor, lalu menghubungkan pin gnd pada ldr dengan pin gnd pada mikrokontroler untuk memberikan *ground* serta menghubungkan pin sig dengan salah satu pin digital pada mikrokontroler ini.



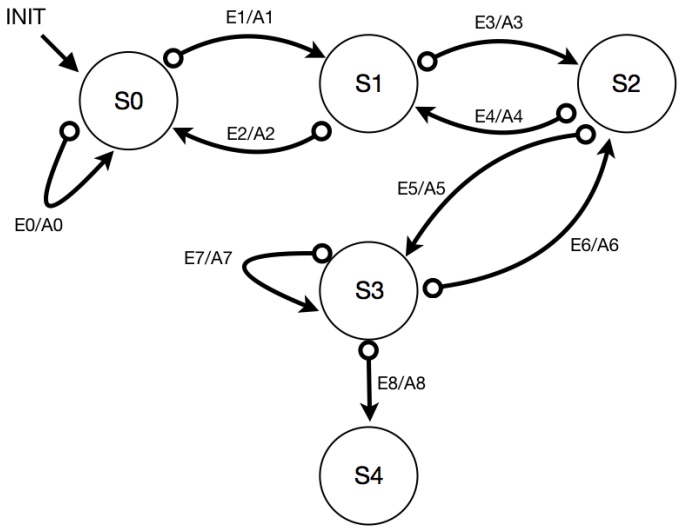
Gambar 5.5 Konfigurasi pin perangkat aktuator menggunakan *relay*

Pada Gambar 5.5 menjelaskan konfigurasi pin pada perangkat aktuator menggunakan *relay*, tidak memiliki banyak perbedaan dengan konfigurasi pin yang diterapkan pada perangkat sensor. Pada perangkat aktuator memiliki *relay* yang berfungsi untuk melakukan kendali terhadap objek aktuator yang digunakan pada sistem ini yakni, lampu. *Relay* merupakan komponen elektronik yang memiliki pemrosesan sinyal digital, oleh karena itu masukan pada *relay* ini haruslah data digital. Jadi mikrokontroler mengirimkan data digital kepada *relay* untuk selanjutnya diproses untuk memberikan aksi pada lampu.

Relay memiliki konfigurasi pin pada sensor lainnya, yakni memiliki 3 pin yang menghubungkan pin vcc pada *relay* dengan 3v3 pada mikrokontroler untuk memberikan tegangan kepada *relay*, menghubungkan pin gnd pada *relay* dengan pin gnd pada mikrokontroler untuk memberikan *ground* serta menghubungkan pin signal dengan salah satu pin digital pada mikrokontroler. Pada lampu cukup memutuskan salah satu kabel yang ada pada rangkaian lampu kemudian menghubungkan ke salah satu pin *relay* seperti rangkaian lampu dengan *switch* pada umumnya. *Relay* ini sebagai pengganti *switch* pada umumnya, namun dapat diproses dan dikendalikan menggunakan sinyal digital.

5.1.2 Perancangan Perangkat Lunak

5.1.2.1 *State machine* Perangkat Sensor



Gambar 5.6 *State machine* perangkat sensor

State machine perangkat sensor ditunjukkan menggunakan metode FSM (*Finite State Machine*) agar dapat memberikan gambaran umum sistem dalam sisi perangkat lunak untuk perangkat sensor namun tetap dapat melakukan pendekatan hubungan pada perangkat sensor antara keadaan satu dengan keadaan lainnya. Pada Gambar 5.6, dijelaskan *state machine* perangkat sensor memiliki 5 *state,* 9 *event* dan 9 *action.* S0 memiliki definisi kondisi awal sehingga proses yang terjadi pada perangkat sensor berawal dari S0 sehinggi ditandai dengan panah dengan nama *init.*

Tabel 5.1 Keterangan *state machine* perangkat sensor

State		Event		Action	
S0	Wifi Connected	E0	Wifi Reconnect	A0	Menghubungkan kembali antara sistem dengan wifi
S1	Broker Connected	E1	Broker Connect	A1	Menghubungkan sistem dengan broker
S2	Device Ready	E2	Broker Disconnect	A2	Sistem gagal terhubung dengan broker
S3	Device Operation	E3	Object Initialization	A3	Mikrokontroler melakukan inisialisasi file metadata
S4	Off	E4	Object Failed to Initialization	A4	Mikrokontroler gagal melakukan inisialisasi file metadata
		E5	Device Recognition	A5	Mikrokontroler mengirimkan pesan metadata ke gateway
		E6	Device Failed to Recognition	A6	Mikrokontroler menerima pesan metadata dari gateway
		E7	Device Action	A7	Mikrokontroler membaca dan mengirimkan data ke aktuator
		E8	Device Off	A8	Kesalahan pada sistem

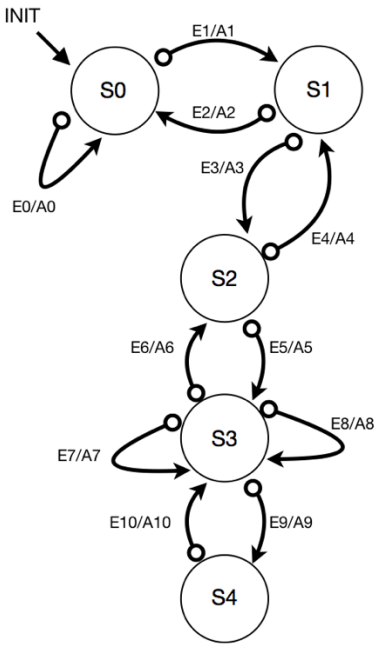
Pada Tabel 5.1, dijelaskan legenda untuk memberikan keterangan pada Gambar 5.6 sehingga dapat memberikan penjelasan dalam mengetahui

hubungan antara *state*, *event* dan *action* pada *state machine* perangkat sensor. Keseluruhan proses bermula ketika perangkat sensor dinyalakan maka perangkat akan melakukan percobaan untuk terhubung ke jaringan *wifi* sehingga akan berada pada keadaan S0 dan akan selalu mekanisme terhubung ulang secara terus menerus hingga perangkat sensor terhubung ke jaringan *wifi*. Selanjutnya, perangkat akan berpindah ke S1 ketika perangkat sudah terhubung ke *broker* dan mekanismenya seperti pada event sebelumnya yakni akan melakukan percobaan terhubung ulang secara terus menerus.

Keadaan selanjutnya terjadi ketika perangkat berhasil membuat objek perangkat sensor menggunakan file *metadata* dan mekanismenya terhubung ulangnya seperti pada keadaan sebelumnya. Untuk bertransisi ke S2 diperlukan mekanisme mengirim pesan kepada *gateway* agar dapat dikenali serta mendaftarkan layanan serta perangkat sensor baru sehingga sensor dapat saling terhubung dengan aktuator

Jika proses tersebut sukses, akan diteruskan ke S3 namun jika tidak sukses, maka akan dikembalikan ke keadaan sebelumnya dan akan terjadi proses perulangan terus menerus sampai proses sukses. Pada S3, perangkat sensor telah terhubung dengan *gateway* serta aktuator sehingga sudah dapat mengakuisisi data sensor serta mengirimkan data tersebut ke aktuator. Jika perangkat terjadi masalah teknis ataupun mati akan berpindah ke S4 dan perangkat akan mati dan tidak bisa melakukan operasi sensor kembali. *State machine* ini berlaku untuk kedua perangkat sensor baik perangkat yang menggunakan sensor LDR maupun PIR.

5.1.2.2 *State machine* Perangkat Aktuator



Gambar 5.7 *State machine* perangkat aktuator

State machine perangkat aktuator ditunjukkan pula menggunakan metode yang sama yang digunakan pada *state machine* untuk perangkat sensor yakni, FSM (*Finite State Machine*). Pada Gambar 5.7, dijelaskan *state machine* perangkat aktuator memiliki kuantitas setiap parameternya lebih banyak dibandingkan dengan perangkat sensor dikarenakan pada sistem ini banyak memiliki peranan penting pada perangkat aktuator sehingga akan banyak kondisi yang terjadi. *State machine* perangkat aktuator memiliki 5 *state*, 11 *event* dan 11 *action*. Semua proses pada *state machine* ini diawali pada S0.

Tabel 5.2 Keterangan *state machine* perangkat aktuator

State		Event		Action	
S0	Wifi Connected	E0	Wifi Reconnect	A0	Menghubungkan kembali antara sistem dengan wifi
S1	Broker Connected	E1	Broker Connect	A1	Menghubungkan sistem dengan broker
S2	Device Ready	E2	Broker Disconnect	A2	Sistem gagal terhubung dengan broker
S3	Device Operation	E3	Object Initialization	A3	Mikrokontroler melakukan inisialisasi file metadata
S4	Relation Connected	E4	Object Failed to Initialization	A4	Mikrokontroler gagal melakukan inisialisasi file metadata
		E5	Device Recognition	A5	Mikrokontroler mengirimkan pesan metadata ke gateway
		E6	Device Failed to Recognition	A6	Mikrokontroler menerima pesan metadata dari gateway
		E7	Device Action	A7	Mikrokontroler menerima pesan dari sensor dan memberikan aksi terhadap lampu
		E8	Keep Alive	A8	Mikrokontroler memeriksa ketersediaan perangkat sensor yang telah terintegrasi
		E9	Sensor Integration	A9	Mikrokontroler memeriksa perangkat sensor yang ingin terintegrasi
		E10	Sensor Failed to Integration	A10	Mikrokontroler menerima pesan berupa perangkat sensor dari gateway

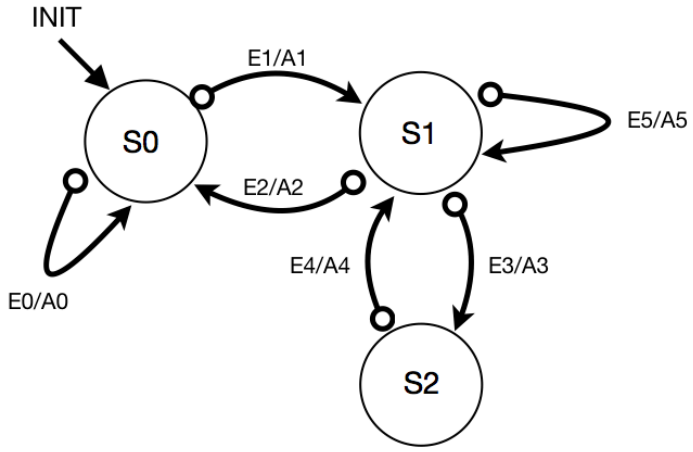
Pada Tabel 5.2, dipaparkan keterangan untuk *state machine* perangkat aktuator untuk menjelaskan hubungan antara setiap parameter pada *state machine*. Untuk proses awal hampir sama dengan perangkat sensor yang diawali dengan melakukan percobaan untuk berkoneksi dengan jaringan *wifi* serta *broker* hingga percobaan tersebut berhasil. Kemudian perangkat akan melakukan pembuatan objek perangkat aktuator yang dijalankan mikrokontroler untuk bertransisi ke S2 dan selanjutnya dilakukan mekanisme pengiriman pesan ke *gateway* agar dikenali serta mendaftarkan layanan dan perangkat aktuator baru sehingga perangkat aktuator dapat saling terhubung dengan sensor.

Jika mekanisme pengenalan berhasil maka dapat bertransisi ke S3, namun jika tidak berhasil akan bertransisi ke *state* sebelumnya dan akan terjadi proses perulangan terus menerus sampai mekanisme pengenalan berhasil. Jika perangkat aktuator telah berada di S3, maka perangkat aktuator tersebut sudah siap menerima data dari sensor serta mengirim status pada lampu ke *gateway*. Ketika ada interrupt dari *gateway* berupa permintaan akses untuk berelasi

antara perangkat aktuator tersebut dengan suatu perangkat sensor akan berpindah ke S4 yang kemudian akan mendapatkan pesan dari *gateway* apakah diijinkan perangkat sensor tersebut untuk berelasi.

Selain itu perangkat aktuator pada S3 akan selalu mengecek ketersediaan perangkat sensor yang telah berelasi sebelumnya terhadap perangkat aktuator sehingga perangkat aktuator dapat menghilangkan relasi apabila perangkat sensor tidak tersedia guna untuk menghemat proses komputasi.

5.1.2.3 *State machine* Perangkat *Gateway*



Gambar 5.8 *State machine* perangkat *gateway*

State machine perangkat *gateway* juga ditunjukkan menggunakan metode FSM (*Finite State Machine*) sehingga dapat memberikan gambaran umum sistem serta melakukan pendekatan hubungan pada perangkat *gateway* antara keadaan satu dengan keadaan lainnya. Pada Gambar 5.8, dijelaskan *state machine* untuk perangkat *gateway* sebagai perancangan dalam sisi perangkat lunak. *State machine* ini memiliki 3 *state*, 6 *event* dan 6 *action*. Kondisi awal pada *state machine* ini ditandai pada *state* S0.

Tabel 5.3 Keterangan *state machine* perangkat *gateway*

State		Event		Action	
S0	Broker Connected	E0	Broker Reconnect	A0	Menghubungkan kembali antara sistem dengan broker
S1	Device Operation	E1	Object Initialization	A1	Gateway melakukan inisialisasi objek
S2	New Device Connected	E2	Object Failed to Initialization	A2	Gateway gagal melakukan inisialisasi objek
		E3	Device Recognition	A3	Gateway menerima perangkat sensosr atau aktuator untuk saling terhubung
		E4	Device Failed to Recognition	A4	Gateway menolak perangkat sensor atau aktuator untuk saling terhubung
		E5	Device Action	A5	Gateway menerima data dari perangkat sensor atau aktuator

Pada Tabel 5.3, dijelaskan legenda untuk memberikan keterangan pada Gambar 5.8 sehingga dapat memberikan penjelasan dalam mengetahui hubungan pada semua parameter pada *state machine* perangkat *gateway*. Keseluruhan proses diawali ketika *gateway* telah terhubung ke *broker* dan akan selalu melakukan mekanisme terhubung ulang secara terus menerus hingga dapat terhubung dengan *broker*, selanjutnya sistem akan melakukan pembuatan objek *gateway* dengan mekanisme yang sama dengan keadaan sebelumnya.

Kemudian ketika sukses akan berpindah pada keadaan S1 dan *gateway* telah siap menerima *interrupt* baik dari perangkat sensor maupun aktuator. JIka terdapat *interrupt* dari perangkat aktuator berupa permintaan untuk terhubung ke *gateway* maka akan bertransisi ke S2 dan *gateway* akan memproses permintaan perangkat aktuator, jika diterima maka *gateway* akan menyimpan objek perangkat aktuator tersebut dan mengirimkan pesan balik ke perangkat aktuator untuk memberikan konfirmasi persetujuan permintaan untuk terhubung ke *gateway* dan jika tidak diterima maka *gateway* juga mengirimkan pesan balik untuk memberikan status permintaannya.

Jika terdapat *interrupt* dari perangkat sensor berupa permintaan untuk terhubung ke *gateway* serta aktuator pada lokasi yang sama maka akan bertransisi ke S2 dan *gateway* akan memproses permintaan perangkat sensor. *Gateway* akan memproses beberapa parameter yakni dengan mengecek ketersedian aktuator di lokasi tersebut, mengecek kuota perangkat yang diijinkan unuk berintegrasi dan mengecek jenis sensor yang diijinkan untuk berintegrasi.

Jika keseluruhan aspek tersebut dapat terpenuhi maka *gateway* akan menyimpan objek perangkat sensor tersebut serta mengirimkan pesan balik untuk memberikan konfirmasi persetujuan permintaan untuk terhubung ke *gateway* serta aktuator dan jika tidak maka akan memberikan konfirmasi jika perangkat sensor tersebut ditolak untuk berintegrasi. Selanjutnya ketika relasi sudah terbentuk, perangkat sensor akan selalu mengirimkan data sensor ke perangkat aktuator, namun perangkat *gateway* tetap bisa membaca komunikasi diantara keduanya dan menyimpan nilai terakhir yang dikirimkan dari perangkat sensor.

5.2 Implementasi Sistem

Pada bagian ini dijelaskan proses implementasi sistem yang dimulai dari implementasi perangkat keras baik perangkat aktuator maupun perangkat sensor dan implementasi perangkat lunak yang meliputi pengenalan perangkat dan layanan baru, pembuatan relasi antara sensor dengan aktuator, pembacaan dan pengiriman data sensor dan aktuator, pemberian aksi terhadap lampu oleh aktuator serta penghapusan relasi antara sensor dengan aktuator.

5.2.1 Implementasi Perangkat Keras

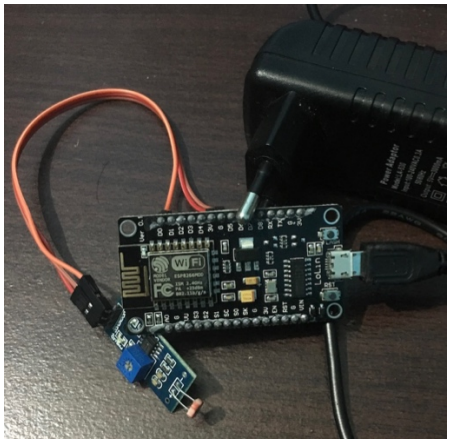
5.2.1.1 Implementasi Perangkat Sensor PIR

Sesuai dengan perancangan perangkat sensor menggunakan sensor PIR, implementasi perangkat sensor ini terdiri atas mikrokontroler NodeMCU sebagai pusat pemrosesan data serta sensor PIR untuk mendeteksi suatu gerakan pada suatu tempat uji. Konfigurasi pin pada perangkat tersebut sesuai dengan perancangan pada Gambar 5.3 dan masing-masing komponen dihubungkan menggunakan kabel jumper. Pada Gambar 5.9 menunjukkan hasil implementasi dari perancangan perangkat sensor menggunakan sensor PIR.

Gambar 5.9 Perangkat keras sensor menggunakan pir setelah dirakit

5.2.1.2 Implementasi Perangkat Sensor LDR

Sesuai dengan perancangan perangkat sensor menggunakan sensor LDR, implementasi perangkat sensor ini terdiri atas mikrokontroler NodeMCU sebagai pusat pemrosesan data, sensor ldr dan modulnya untuk mendeteksi tingkat intensitas cahaya pada suatu tempat. Konfigurasi pin pada perangkat tersebut sesuai dengan perancangan pada Gambar 5.4 dan dihubungkan menggunakan kabel jumper. Pada Gambar 5.10 menunjukkan hasil implementasi dari perancangan perangkat sensor menggunakan sensor LDR.



Gambar 5.10 Perangkat keras sensor menggunakan ldr setelah dirakit

5.2.1.3 Implementasi Perangkat Aktuator Lampu

Sesuai dengan perancangan perangkat aktuator, implementasi perangkat aktuator terdiri atas mikrokontroler NodeMCU sebagai pusat pemrosesan data, *relay* sebagai pusat kendali atas lampu serta lampu sebagi objek aktuator pada perangkat aktuator sistem ini. Konfigurasi pin pada perangkat tersebut sesuai dengan perancangan pada Gambar 5.5 dan dihubungkan menggunakan kabel jumper. Pada Gambar 5.11 menunjukkan hasil implementasi dari perancangan perangkat aktuator.

Gambar 5.11 Perangkat keras aktuator setelah dirakit

5.2.2 Implementasi Perangkat Lunak

5.2.2.1 Implementasi Persiapan Kerja Perangkat *Gateway*

Sesuai dengan rekayasa kebutuhan pada kode PKSA dari nomor 1000, koneksi *gateway* ke *broker* merupakan salah satu langkah awal yang harus dikerjakan agar masing-masing perangkat dapat terhubung. Selain itu perlu juga dimasukkan *Library* untuk mendukung komunikasi mqtt yakni Paho Mqtt Client. Implementasi kode program tersebut ada pada Tabel 5.4 yang diletakkan pada fungsi main pada program berbasiskan python.

Tabel 5.4 Koneksi *gateway* ke *broker*

No	Kode Program
1	try:
2	*gateway* = MQTTPervasiveSystem(
3	mqtt_host="ngehubx.online",
4	project_name="kardel",
5	mqtt_user_name="admintes",
6	mqtt_password="admin123"

```
7           )
8           gateway.connect()
9           gateway.username_pw_set()
10          gateway.client.loop_forever()
11      except KeyboardInterrupt:
12          print('\nSYSTEM HAS BEEN FORCED
13  CLOSE\n')
14          sys.exit()
```

5.2.2.2 Implementasi Mekanisme Persiapan Kerja dari Sensor dan Aktuator

Sesuai dengan rekayasa kebutuhan pada kode PKSA dari nomor 1100 sampai dengan 1102, koneksi keseluruhan perangkat baik perangkat aktuator maupun perangkat sensor ke jaringan *wifi* merupakan salah satu langkah awal yang harus dikerjakan agar masing-masing perangkat dapat terhubung. Selain harus memasukkan *Library* yang mendukung *wifi*, maka perlu untuk memasukkan *Library* lainnya seperti *Library* Arduino, mqtt serta json. Implementasi kode program tersebut ada pada Tabel 5.5.

Tabel 5.5 *Library* yang digunakan

No	Kode Program
1	#include <Arduino.h>
2	#include <ESP8266*Wifi*.h>
3	#include <MQTTClient.h>
4	#include <ArduinoJson.h>

Implementasi kode program untuk koneksi keseluruhan perangkat pada sistem ini ke jaringan *wifi* memiliki kode yang sama setiap perangkatnya. Kode program ini diletakkan pada fungsi setup() dikarenakan pada Arduino *framework,* fungsi ini akan dijalankan pertama kali dan hanya sekali dijalankan. Implementasi kode program tersebut ada pada Tabel 5.6.

Tabel 5.6 Koneksi perangkat ke jaringan *wifi*

No	Kode Program
1	*Wifi*.begin(*WIFI*_SSID, *WIFI*_PASSWORD);
2	while (!client.connected())
3	{
4	connect();
5	}

Pada Tabel 5.6 pada potongan kode di baris 1, program memanggil fungsi *Wifi*.begin() dimana didalamnya terdapat *WIFI*_SSID dan *WIFI*_PASSWORD. Kedua parameter tersebut nilainya berada pada file main.h yang berisi nama serta password jaringan *wifi*. Maka sebelumnya perlu dilakukan penambahan import header file main.h pada kode program agar kedua parameter tersebut nilainya dapat diambil pada *header file* tersebut.

Setelah keseluruhan perangkat terkoneksi ke jaringan *wifi*, maka langkah selanjutnya adalah menghubungkan keseluruhan perangkat baik perangkat

sensor maupun aktuator terhadap *broker*, langkah ini dilakukan agar keseluruhan perangkat dapat saling terkoneksi menggunakan protokol mqtt. Langkah ini sesuai dengan rekayasa kebutuhan pada kode PKSA dari nomor 1103 sampai dengan 1105. Implementasi kode program untuk kebutuhan tersebut memiliki kode yang sama serta diletakkan didalam fungsi setup untuk dilakukan pemanggilan pertama kali pada saatu perangkat dinyalakan serta potongan kode terdapat pada Tabel 5.7.

Tabel 5.7 Koneksi perangkat ke *broker*

No	Kode Program
1	`client.begin(MQTT_SERVER_NAME, net);`

Pada Tabel 5.7, program memanggil fungsi client.begin() dimana salah satu parameternya yakni MQTT_*SERVER*_NAME. Nilai dari parameter tersebut berada pada header file main.h yang berisi url/ip *server* dari mqtt *broker*. Langkah selanjutnya yakni menginisialisasi objek setiap perangkatnya. Langkah ini sesuai dengan rekayasa kebutuhan pada kode PKSA dari nomor 1106 sampai dengan 1108. Implementasi kode program terdapat pada Tabel 5.8.

Tabel 5.8 Inisialisasi objek perangkat

No	Kode Program
1	`setupDevice(DEVICE_MODE);`
2	`    for (int i = 0; i <`
3	`DEVICE_SERVICE_TOTAL; i++)`
4	`    {`
5	`        setupService(i);`
6	`    }`

Pada Tabel 5.8 dijelaskan program memanggil fungsi setupDevice untuk menginisialisasi objek perangkat serta setupService untuk menginisialisasi objek layanan yang tersedia pada perangkat tersebut. Pada fungsi setupDevice membutuhkan suatu parameter DEVICE_MODE, dimana parameter ini berisi untuk menginformasikan perangkat apa yang ingin diinisialisasi. Pada fungsi setupService jg membutuhkan suatu parameter DEVICE_SERVICE_TOTAL, dimana parameter ini berisi untuk menginformasikan total layanan yang tersedia. Kedua fungsi tersebut menghubungkan dengan header file metadata.h. Header file ini berisi keseluruhan identitas dari perangkat baik perangkat sensor maupun aktuator, oleh karena itu perlu dimasukkan terlebih dahulu header file tersebut kedalam kode program sehingga fungsi ini akan berjalan sesuai dengan kebutuhan.

5.2.2.3 Implementasi Mekanisme Pengenalan Perangkat dan Layanan Baru

Sesuai dengan rekayasa kebutuhan pada kode PPLB nomor 1200, pengiriman pesan *broadcast* oleh aktuator merupakan langkah awal dalam mekanisme pengenalan perangkat dan layanan baru dan hanya perangkat aktuator yang dapat mengawali langkah ini karena pada suatu lokasi harus terdapat perangkat aktuator terlebih dahulu. Implementasi kode program tersebut ada pada Tabel 5.9.

Tabel 5.9 Pengiriman pesan ke *gateway*

No	Kode Program
1	String payloadBC = "{\"deviceName\":" +
2	device.getDeviceName() + ","; payloadBC += "\"category\":" +
3	device.getCategory() + ","; payloadBC += "\"deviceType\":" +
4	device.getDeviceType() + ","; payloadBC += "\"ackTopic\":" +
5	device.getAckTopic() + ","; payloadBC += "\"location\":" +
6	device.getLocation() + ","; payloadBC += "\"service\": {"; for (int i = 0; i <
7	DEVICE_SERVICE_TOTAL; i++) { payloadBC += "\"" +
8	DEVICE_INTEGRATION_CATEGORY[i] + "\": {"; payloadBC += "\"name\":" +
9	service[i].getServiceName() + ","; payloadBC += "\"unit\":" +
10	service[i].getServiceData() + ","; payloadBC += "\"data\":" +

11	service[i].getServiceData() + "}";
	if (i + 1 < DEVICE_SERVICE_TOTAL)
	{

12	payloadBC += ","; } }

13	payloadBC += "}";
	if (mode == 0)
	{

14	// Additional of Actuator Mode
	// Root=Integration
	payloadBC += ",";
	payloadBC += "\"integration\":

15	{\"max\"" +
16	String(device.getMaximumIntegration()) +
17	","; payloadBC += "\"category\":" +
18	device.getCategoryIntegration(0) +
19	device.getCategoryIntegration(1) + "}"; } payloadBC += "}"; client.publish(PROJECT_*BROADCAST*_TOPIC,
20	payloadBC); }

Pada Tabel 5.9 dijelaskan potongan kode untuk pengiriman pesan oleh aktuator ke *gateway*, dimana pesan tersebut akan dibungkus pada format json sehingga perangkat *gateway* dapat menseleksi informasi yang dikirim dari aktuator. Fungsi ini dipanggil pada fungsi setup() dengan memberikan parameter DEVICE_MODE dari *metadata*. Langkah selanjutnya pesan akan diterima oleh *gateway* dan selanjutnya akan dilakukan mekanisme untuk mendaftarkan objek perangkat aktuator kedalam *gateway*. Langkah ini sesuai dengan rekayasa kebutuhan kode PPLB nomor 1202. Implementasi dari kode program tersebut ada pada Tabel 5.10.

Tabel 5.10 Pendaftaran aktuator/sensor oleh *gateway*

No	Kode Program
1	def deviceConnect(self, msg):
2	data = json.loads(msg)
3	replytopic = self.topic_pervasive +
4	data[“location”] + “/” + data[“deviceType”]
5	+ “/” + data[“deviceName”]
6	isCompatibility = False
7	if (self.checkingDevice(msg)):
8	if
9	(str(data[“deviceType”])==”sensor”):
10	print(‘masuk sensor’)
11	isCompatibility =
12	self.checkingRelation(msg)
13	else:
14	
15	self.relationCompatibility(msg)
16	
17	self.device[str(data[“deviceType”])][str(da
18	ta[“deviceName”])] = ast.literal_eval(msg)
19	device_connect_msg = {
20	“statuscode”: 200,
21	“replytopic_data”:
22	str(replytopic) + “/data”
23	}
24	if (isCompatibility):
25	
26	self.device[str(data[“deviceType”])][str(da
27	ta[“deviceName”])] = ast.literal_eval(msg)
28	os.system(‘clear’)
29	
30	pprint.pprint(json.dumps(self.device),
31	indent=1)
32	else:
33	device_connect_msg = {
34	“statuscode”: 404
35	}

No	Kode Program
36	print('blocked!')

Pada Tabel 5.10 dijelaskan mekanisme mendaftarkan objek perangkat aktuator oleh *gateway*, setiap ada pesan permintaan untuk terkoneksi antara perangkat baik perangkat sensor maupun sensor dengan perangkat *gateway* selalu memanggil fungsi ini. Objek yang didaftarkan disimpan pada variabel *device* dengan struktur data *dictionary*. Langkah selanjutnya yakni mengirimkan hasil proses sebelumnya kepada aktuator pengirim, langkah ini sesuai dengan rekayasa kebutuhan dengan kode PPLB nomor 1203. Implementasi kode program tersebut berada pada Tabel 5.11.

Tabel 5.11 Pengiriman umpan balik oleh *gateway*

No	Kode Program
1	self.client.publish(data["ackTopic"],
2	str(device_connect_msg), self.mqtt_qos)

Pada Tabel 5.11 terdapat potongan kode untuk melakukan mekanisme pengiriman umpan balik ke perangkat aktuator, umpan balik tersebut didapatkan dari hasil proses sebelumnya yang menyatakan sukses atau gagal dari proses tersebut. Pesan umpan balik ini berfungsi sebagai bentuk pesan konfirmasi terhadap permintaan yang telah diajukan dari perangkat aktuator. Potongan kode tersebut berada pada fungsi yang sama dari proses sebelumnya. Langkah selanjutnya aktuator mendapatkan informasi umpan balik tersebut sebagai tindak lanjut untuk proses selanjutnya. Implementasi kode program tersebut ada pada Tabel 5.12.

Tabel 5.12 Sensor/aktuator mendapatkan umpan balik

No	Kode Program
1	if (topic == device.getAckTopic())
2	{
3	DynamicJsonBuffer jsonBuffer;
4	Json*Object* &msgJson =
5	jsonBuffer.parse*Object*(payload);
6	// Check message response
7	if (msgJson[String("statuscode")]
8	== 200)
9	{
10	
11	device.setDeviceConnected(true);
12	
13	device.setPervasiveTopic(msgJson[String("re
14	plytopic_data")]);
15	}
16	else
17	{

```
18
19  device.setDeviceConnected(false);
20          }
21      }
```

Pada Tabel 5.12 dijelaskan aktuator mendapatkan pesan umpan balik yang dikirim oleh *gateway* sebagai bentuk pesan konfirmasi atas permintaan dari proses sebelumnya. Potongan kode ini berada pada fungsi messageReceived dikarenakan pada fungsi ini akan selalu terpanggil ketika ada pesan masuk ke dalam suatu perangkat yang menggunakan protokol komunikasi mqtt. Sebelumnya aktuator juga perlu melakukan mekanisme *subscribe* untuk bisa mendapatkan informasi umpan balik. Pesan umpan balik yang dikirimkan dari *gateway* dibungkus pada format json, oleh karena itu perangkat aktuator harus menyeleksi data terlebih dahulu untuk dapat mengetahui informasi umpan balik tersebut.

5.2.2.4 Implementasi Mekanisme Pembuatan Relasi antara Sensor dengan Aktuator

Sesuai dengan rekayasa kebutuhan dengan kode PRSA nomor 1300, pembentukan relasi diawali dari adanya permintaan sensor untuk terhubung oleh aktuator dalam suatu lokasi. Pada Tabel 5.9 dijelaskan melalui potongan kode bagaimana kode tersebut dapat mengirimkan suatu pesan *broadcast* terhadap *gateway*. Gambar tersebut tidak hanya berlaku untuk perangkat aktuator saja, namun dapat diimplementasikan pada perangkat sensor dan kemudian akan terjadi mekanisme pengecekan terhadap aktuator yang akan menjadi target perangkat sensor tersebut.

Mekanisme yang pertama ialah pengecekan aktuator yang berfungsi untuk mengetahui ketersediaan perangkat aktuator pada suatu target lokasi, mekanisme ini sesuai dengan rekayasa kebutuhan dengan kode PRSA nomor 1301. Pada Tabel 5.13 perangkat *gateway* akan mengecek *metadata* dari perangkat akutator, jika telah tersedia perangkat aktuator pada lokasi tersebut maka perangkat sensor dapat melanjutkan ke mekanisme selanjutnya, namun jika tidak maka perangkat sensor akan ditolak permintaannya dan *gateway* akan mengirimkan pesan konfirmasi terkait penolakan permintaan tersebut.

Tabel 5.13 Pengecekan ketersedian aktuator

No	Kode Program
1	`data = json.loads(msg)`
2	`        resultName = ""`
3	`        statusRelationalFlag = False`
4	`        currentItemInLocation = 0`
5	
6	`        # 1. searching actuator data in`
7	`location & msg=sensor`
8	`        try:`
9	`            targetLocationActuator =`
10	`self.device["locations"][str(data["location`

```
"])]
            for listobj in
targetLocationActuator:
                for item in
listobj.items():
                    if
(item[0]=="deviceName"):

currentItemInLocation =
len(targetLocationActuator)-1
                        resultName =
item[1]
                        break
                break
        except KeyError:
            statusRelationalFlag = False
            print('Actuator not available
in this area!')
        else:
            statusRelationalFlag = True
```

Ketersediaan aktuator pada suatu lokasi dianggap memiliki peran penting, dikarenakan apabila tidak memiliki aktuator terlebih dahulu maka perangkat sensor akan bingung mau integrasi ke perangkat aktuator yang mana. Mekanisme selanjutnya yakni pengecekan maksimal integrasi yang berfungsi untuk mengetahui batas maksimal untuk perangkat sensor yang dapat diijinkan terhubung oleh suatu perangkat aktuator serta pengecekan kategori sensor yang berfungsi untuk mengetahui kategori-kategori perangkat sensor yang dapat diijinkan terhubung ke suatu perangkat aktuator.

Tabel 5.14 Pengecekan maksimal integrasi dan kategori sensor

No	Kode Program
1	`if (statusRelationalFlag):`
2	`            try:`
3	`                print('masuk')`
4	`                targetIntegrationActuator =`
5	`self.device["actuator"][str(resultName)]["i`
6	`ntegration"]`
7	`                categoryCompability =`
8	`targetIntegrationActuator["category"]`
9	`                maximumIntegration =`
10	`int(targetIntegrationActuator["max"])`
11	`            except KeyError:`
12	`                statusRelationalFlag =`
13	`False`
14	`            else:`
15	`                statusRelationalFlag = True`

Pada Tabel 5.14 menunjukan 2 mekanisme yang sesuai dengan rekayasa kebutuhan dengan kode PRSA nomor 1302 serta nomor 1303. Mekanisme ini dilakukan oleh *gateway* dengan cara mengecek *metadata* dari perangkat aktuator yang telah disimpan objek aktuator tersebut pada mekanisme sebelumnya. Selanjutnya ketika 2 mekanisme telah dilakukan dan telah terverifikasi oleh *gateway* maka akan dilakukan penyimpanan *metadata* perangkat sensor tersebut dan membuat relasi antar keduanya yang sesuai dengan rekayasa kebutuhan dengan kode PRSA nomor 1304.

Mekanisme ini diimplementasikan pada Tabel 5.10 yang akan menyimpan *metadata* perangkat sensor setelah melalui 3 mekanisme sebelumnya dan langkah selanjutnya *gateway* akan memberikan pesan balik terhadap status permintaannya baik ditolak maupun diterima, langkah ini sesuai pada Tabel 5.11 serta sesuai dengan rekayasa kebutuhan dengan kode PRSA nomor 1305. Kemudian, ketika *gateway* mengirimkan pesan balik maka perangkat sensor akan mendapatkan pesan umpan balik tersebut. Pesan umpan balik tersebut akan diolah informasinya untuk mengetahui hasil dari permintaan tersebut.

Mekanisme ini diimplementasikan pada Tabel 5.12 dijelaskan mekanisme perangkat sensor dalam membuka informasi pesan balik yang dikirimkan oleh *gateway* menggunakan *Library* json untuk dapat mengetahui informasinya. Selain *gateway* mengirimkan pesan kepada perangkat sensor mengenai status dari proses permintaan relasi, *gateway* juga mengirimkan pesan kepada perangkat aktuator yang telah terhubung dengan informasi perangkat sensor yang telah terhubung oleh aktuator. Hal ini berfungsi untuk perangkat aktuator guna mendapatkan data sensor dari perangkat sensor yang baru saja terhubung. Mekanisme ini sesuai dengan rekayasa kebutuhan dengan kode PRSA nomor 1307.

Tabel 5.15 Pengiriman pesan *update* oleh *gateway*

No	Kode Program
1	`data = json.loads(msg)`
2	`        replytopic = self.topic_pervasive +`
3	`data["location"] + "/" + data["deviceType"]`
4	`+ "/" + data["deviceName"]`
5	`        print('relationCompatibility')`
6	
7	`        try:`
8	
9	`self.device["locations"][str(data["location`
10	`"])].append(`
11	`                {`
12	`                    "deviceName":`
13	`str(data["deviceName"]),`
14	`                    "deviceType":`
15	`str(data["deviceType"]),`
16	`                    "topic_pervasive":`

No	Kode Program
17	str(replytopic) + "/data"
18	}
19	)
20	except KeyError:
21	
22	self.device["locations"][str(data["location
23	"])] = [
24	{
25	"deviceName":
26	str(data["deviceName"]),
27	"deviceType":
28	str(data["deviceType"]),
29	"topic_pervasive":
30	str(replytopic) + "/data"
31	}
32	]
33	
34	if (bool(target_topic)):
35	*update*_relational = {
36	"statuscode": 200,
37	"deviceName":
38	str(data["deviceName"]),
39	"*update*_topic_sensor":
40	str(replytopic) + "/data"
41	}
42	print(target_topic)
43	
44	self.client.publish(target_topic +
45	"/*update*", str(*update*_relational),
46	self.mqtt_qos)

Pada Tabel 5.15 dijelaskan *gateway* melakukan mekanisme pengiriman pesan *update* yang berisi *metadata* yang penting untuk *gateway* dalam mendapatkan data sensor. Selanjutnya pada Tabel 5.16 dijelaskan perangkat aktuator telah mendapatkan pesan yang dikirimkan oleh *gateway* dan kemudian perangkat aktuator ini menyimpan pada objek relasi dan selanjutnya akan mendapatkan data sensor yang berada didalam relasi yang sama. Mekanisme ini sesuai dengan rekayasa kebutuhan dengan kode PRSA nomor 1308.

Tabel 5.16 Aktuator mendapatkan pesan *update*

No	Kode Program
1	DynamicJsonBuffer jsonBuffer;
2	Json*Object* &msgJson =
3	jsonBuffer.parse*Object*(payload);
4	
5	if (msgJson[String("statuscode")] ==
6	200)

No	Kode Program
7	{ {
8	for (int i = 0; i <
9	device.getMaximumIntegration(); i++)
10	{
11	if
12	(device.getTopicIntegration(i) == "")
13	{
14	
15	device.appendTopicIntegration(msgJson[Strin
16	g("*update*_topic_sensor")], i);
17	
18	device.appendDeviceNameIntegration(msgJson[
19	String("deviceName")], i);
20	break;
21	}
22	}
23	}
24	}

5.2.2.5 Implementasi Mekanisme Kerja dari Sensor

Sesuai dengan rekayasa kebutuhan dengan kode KS nomor 1400 dan 1401 melakukan proses akuisisi data sensor baik sensor PIR maupun sensor LDR yang dilakukan oleh mikrokontroler. Dalam hal ini karena sensor yang digunakan adalah dengan tipe digital maka mikrokontroler akan menggunakan fungsi digitalRead untuk membaca sensor dengan jenis digital. Implementasi kode dari proses tersebut terdapat pada Tabel 5.17 baik itu perangkat dengan sensor PIR maupun sensor LDR

Tabel 5.17 Pembacaan data sensor

No	Kode Program
1	int data = digitalRead(pin)
2	return String(data)

Setelah proses akuisisi data sensor telah berhasil dilakukan oleh mikrokontroler, maka selanjutnya adalah proses mengirimkan data sensor tersebut menggunakan protokol mqtt serta format teks json. Proses ini sesuai dengan rekayasa kebutuhan dengan kode KS nomor 1402-1403. Proses ini mula-mula membuat variabel dengan tipe data string dan kemudian dibungkus dengan format teks json yang kemudian dikirimkan menggunakan topik yang sudah ditentukan oleh *gateway* pada saat proses pembentuk relasi. Implementasi proses tersebut terdapat pada Tabel 5.18.

Tabel 5.18 Pengiriman data sensor/aktuator

No	Kode Program
1	String payloadPublish = "{\"deviceName\":"
2	+ device.getDeviceName() + ","; payloadPublish += "\"deviceType\":" +

```
No  Kode Program
4   device.getDeviceType() + ",";
5
6       // Root=Service
7       payloadPublish += "\"service\": {";
8       for (int i = 0; i <
9   DEVICE_SERVICE_TOTAL; i++)
10      {
11          payloadPublish += "\"" +
12  service[i].getServiceName() + "\": {";
13          payloadPublish += "\"name\":" +
14  service[i].getServiceName() + ",";
15          payloadPublish += "\"unit\":" +
16  service[i].getServiceData() + ",";
17          payloadPublish += "\"data\":" +
18  service[i].getServiceData() + "}";
19          if (i + 1 < DEVICE_SERVICE_TOTAL)
20          {
21              payloadPublish += ",";
22          }
23      }
24      payloadPublish += "}";
25
26      payloadPublish += "}";
27
28
29  client.publish(device.getPervasiveTopic(),
30  payloadPublish);
31  }
```

Pada dasarnya ketika sensor mengirimkan data, aktuator akan menangkap data sensor tersebut. Namun pada hal ini, *gateway* juga akan menangkap setiap data sensor yang sudah memiliki relasi dan kemudian menyimpan data tersebut didalam objek perangkatnya. Proses ini bertujuan untuk mengetahui data terakhir yang dikirimkan oleh perangkat sensor, sehingga *gateway* pun dapat mengetahui kondisi yang sekarang yang terjadi pada perangkat sensor maupun aktuator. Proses ini sesuai dengan rekayasa kebutuhan dengan kode KS nomor 1404 serta implementasi dari proses ini terdapat pada Tabel 5.19. Proses ini terjadi pada fungsi on_message, dimana fungsi ini yang bertanggungjawab untuk mengurusi setiap pesan yang masuk kedalam *gateway* dan kemudian dipilah-pilih sesuai dengan kebutuhannya dan dibungkus dalam format teks json.

Tabel 5.19 *Gateway* menangkap data sensor/aktuator

```
No  Kode Program
1   if (str(topic).find("remove")==-1):
2               data = json.loads(msg)
3               print("masuk update fix")
4
```

No	Kode Program
5	#Checking Device Name
6	obj =
7	self.device[str(data["deviceType"])].keys()
8	try:
9	
10	obj.index(str(data["deviceName"]))
11	except ValueError:
12	print('device has not been
13	registered')
14	else:
15	
16	self.device[str(data["deviceType"])][str(da
17	ta["deviceName"])]["service"] =
18	data["service"]
19	os.system('clear')
20	
21	pprint.pprint(json.dumps(self.device),
22	indent=1)

5.2.2.6 Implementasi Mekanisme Kerja dari Aktuator

Sesuai dengan rekayasa kebutuhan dengan kode KA nomor 1500-1501 akan melakukan proses penerimaan data sensor yang telah dikirimkan dari perangkat sensor baik perangkat sensor yang menggunakan sensor PIR dan LDR. Proses ini terjadi didalam fungsi messageReceived, dimana fungsi ini bertanggungjawab untuk mengurusi setiap pesan yang masuk ke dalam perangkat aktuator dan kemudian proses ini akan melakukan ektraksi informasi yang telah dikirimkan perangkat sensor menggunakan *Library* json. Implementasi kode dari proses tersebut terdapat pada Tabel 5.20.

Tabel 5.20 Aktuator menerima data sensor

No	Kode Program
1	DynamicJsonBuffer jsonBuffer;
2	Json*Object* &msgJson =
3	jsonBuffer.parse*Object*(payload);
4	String deviceService =
5	getServicesFromDeviceName(msgJson[String("d
6	eviceName")]);
7	
8	for (int i = 0; i <
9	device.getMaximumIntegration(); i++)
10	{
11	if (topic.substring(17) ==
12	device.getLocation() + "/" +
13	device.getDeviceNameIntegration(i) +
14	"/data")

No	Kode Program

15	{
	if
	(msgJson[String("deviceName")] ==
	device.getDeviceNameIntegration(i))
	{
	
	device.appendDeviceDataIntegration(msgJson[
	String("service")][deviceService][String("d
	ata")], i);
	break;
	}
	}
	}
	}

Setelah perangkat aktuator mendapatkan data dari perangkat sensor yang berguna sebagai bahan untuk mengolah informasi sehingga menghasilkan keputusan yang akan diteruskan kepada objek aktuatornya yakni, menyalakan lampu atau mematikan lampu. Proses ini sesuai dengan rekayasa kebutuhan dengan kode KA nomor 1502-1503. Awal proses ini akan mengecek terlebih dahulu jumlah sensor yang masuk, ketika data sensor yang masuk adalah sensor LDR ataupun sensor PIR dan nilainya adalah 0 maka lampu akan mati, selain itu maka lampu akan menyala, namun ketika data sensor yang masuk terdapat 2 data sensor yakni sensor LDR dan PIR, maka jika kedua nilainya memiliki nilai yang sama maka lampu akan mati, selain itu maka lampu akan menyala. Implementasi kode dari proses tersebut terdapat pada Tabel 5.21.

Tabel 5.21 Aktuator memberikan aksi kepada lampu

No	Kode Program

1	int actionFor*Relay* = 0;
	
	for (int i = 0; i <
	device.getMaximumIntegration(); i++)
	{
	if (i == 0)
	{
	if
	(device.getDeviceDataIntegration(i).toInt()
	< DEVICE_THRESHOLD_INTEGRATION_LDR)
	{
	actionFor*Relay* = 0;
	}
	else
	{
	actionFor*Relay* = 1;
	}
	}

```
No | Kode Program
19          else if (i == 1)
20          {
21              if (actionForRelay == 0)
22              {
23                  //case pagi
24                  if
25  (device.getDeviceDataIntegration(1).toInt()
26  == DEVICE_THRESHOLD_INTEGRATION_PIR)
27                  {
28                      actionForRelay = 1;
29                  }
30                  else
31                  {
32                      actionForRelay = 0;
33                  }
34              }
35              else
36              {
37                  if
38  (device.getDeviceDataIntegration(1).toInt()
39  == DEVICE_THRESHOLD_INTEGRATION_PIR)
40                  {
41                      actionForRelay = 0;
42                  }
43                  else
44                  {
45                      actionForRelay = 1;
46                  }
47              }
48          }
49      }
50
51
52  service[0].setServiceData(String(actionForR
53  elay));
54      return true;
55  }
56
57  digitalWrite(pin,
58  transformDataRelay(service[0].getServiceDat
59  a().toInt()));
```

Setelah perangkat aktuator memberikan suatu aksi terhadap lampu tersebut baik menyalakan maupun mematikan lampu yang berdasarkan data sensor yang masuk ke perangkat aktuator maka selanjutnya yakni aktuator mengirimkan data terhadap *gateway*. Data yang dikirimkan adalah kondisi lampu terakhir menggunakan format teks json. Implementasi dilakukan sama halnya dengan

mengirimkan data sensor oleh perangkat sensor yang terdapat pada Tabel 5.18. Proses ini sesuai dengan rekayasa kebutuhan dengan kode KA nomor 1504.

Kemudian *gateway* akan menangkap data aktuator berupa kondisi terakhir dari lampu dan menyimpannya kedalam objek perangkat aktuator tersebut agar *gateway* dapat mengetahui kondisi terakhir dari lampu tersebut. Proses ini sesuai dengan rekayasa kebutuhan dengan kode KA nomor 1505. Implementasi kode untuk proses tersebut sama halnya dalam *gateway* menangkap data sensor dan terdapat pada Tabel 5.19.

5.2.2.7 Implementasi Penghapusan Relasi Sensor

Proses penghapusan relasi sensor akan terjadi apabila perangkat sensor memiliki masalah teknis atau mati sehingga perangkat aktuator tidak mendapatkan data sensornya kembali alhasil perangkat aktuator tidak bisa memberikan suatu keputusan untuk menyalakan atau mematikan lampu. Mekanisme yang pertama ialah perangkat aktuator melakukan identifikasi menggunakan fitur milis pada mikrokontroler sebagai *timer*. Mekanisme ini sesuai dengan rekayasa kebutuhan dengan kode PRS nomor 1600 serta implementasi dari mekanisme ini terdapat pada Tabel 5.22.

Tabel 5.22 Aktuator mengecek data dari sensor

No	Kode Program

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
	for (int i = 0; i < device.getMaximumIntegration(); i++)
	    {
	        currentMillisKeepAlive = millis();
	
	        if (previousMillisKeepAlive[i] == 0)
	        {
	            previousMillisKeepAlive[i] = currentMillisKeepAlive;
	        }
	
	        //
	Serial.println(currentMillisKeepAlive + ", " + previousMillisKeepAlive[i]);
	
	        if (currentMillisKeepAlive - previousMillisKeepAlive[i] >= intervalTimeout)
	        {
	            previousMillisKeepAlive[i] = currentMillisKeepAlive;
	            String deviceName = device.getDeviceNameIntegration(i);
	            // Serial.println(String(i) + ", " + device.getDeviceNameIntegration(0) +

No	Kode Program
26	"-" + device.getDeviceNameIntegration(1));
27	
28	int whileCounter = 0;
29	int indexPath = 0;
30	while(deviceName == "")
31	{
32	if (whileCounter == 0)
33	{
34	indexPath = i-1;
35	deviceName =
36	device.getDeviceNameIntegration(indexPath);
37	}
38	else if (whileCounter < 2)
39	{
40	indexPath = i+1;
41	deviceName =
42	device.getDeviceNameIntegration(indexPath);
43	}
44	else
45	{
46	Serial.println("null
47	variable");
48	}
49	whileCounter++;
50	}
51	
52	Serial.println("*Object* deleted
53	- " + deviceName + " - " +
54	previousMillisKeepAlive[i]);
55	
56	if (deviceName == "")
57	{
58	
59	device.removeIntegration(indexPath);
60	}
61	else
62	{
63	
64	device.removeIntegration(i);
65	}
66	
67	
68	*update*Integrationto*Gateway*(deviceName);
69	faultCounter++;
70	Serial.println("faultCounter: "
71	+ String(faultCounter) + ", integrate: " +
	String(deviceTotalIntegrate));

No	Kode Program
72	
73	if (faultCounter ==
74	deviceTotalIntegrate)
75	{
76	previousMillisKeepAlive[i]
77	= 0;
78	currentMillisKeepAlive = 0;
79	integrateReady = false;
80	dataReady = false;
81	faultCounter = 0;
82	deviceTotalIntegrate = 0;
83	Serial.println("all device
84	integration has disconnected");
85	}
86	
87	break;
88	}
89	}
90	

Apabila dalam suatu waktu aktuator tidak mendapatkan data sensor maka tindak lanjut yang akan dilakukan perangkat aktuator yakni menghapus objek sensor didalam relasi tersebut. Dengan hal ini, perangkat aktuator tidak perlu berlangganan data sensor itu kembali. Proses ini sesuai dengan rekayasa kebutuhan dengan kode PRS nomor 1601. Implementasi kode dari proses ini terdapat pada Tabel 5.23.

Tabel 5.23 Aktuator menghapus objek sensor

No	Kode Program
1	topicIntegration[indexPath] = "";
2	deviceNameIntegration[indexPath] = "";
3	deviceDataIntegration[indexPath] = "";

Setelah menghapus objek sensor pada perangkat aktuator, maka perangkat aktuator perlu untuk memberikan pesan kepada *gateway* untuk ikut serta menghapus sensor tersebut baik didalam objek sensor maupun objek relasi pada *gateway*. Sehingga *gateway* tidak perlu lagi mendapatkan data dari perangkat sensor tersebut. Aktuator mengirimkan data ke *gateway* menggunakan format teks json. Proses ini sesuai dengan rekayasa kebutuhan dengan kode PRS nomor 1602 serta terdapat pada Tabel 5.24.

Tabel 5.24 Aktuator mengirimkan pesan *update* relasi ke *gateway*

No	Kode Program
1	String payloadBC = "{\"deviceName\": \"" +
2	deviceName + "\"" + ",";
3	payloadBC += "\"location\": \"" +
4	device.getLocation() + "\"";
5	payloadBC += "}";

```
6
7   client.publish(device.getPervasiveTopic() +
8   "/remove", payloadBC);
9   Serial.println("success");
```

Kemudian *gateway* akan menangkap pesan yang dikirimkan aktuator untuk diproses oleh *gateway*. Sama halnya dengan aktuator menghapus objek sensor, *gateway* pun akan menghapus sensor tersebut didalam objek sensor maupun objek relasi. Sehingga perangkat-perangkat yang tidak terhubung kembali tidak perlu disimpan lagi. Proses tersebut sesuai dengan rekayasa kebutuhan dengan kode PRS nomor 1603 serta implementasi kode dari proses tersebut terdapat pada Tabel 5.25.

Tabel 5.25 *Gateway* menghapus objek sensor

```
No  Kode Program
1   data = json.loads(msg)
2           targetIndex = 0
3           counter = 0
4           self.targetLocation =
5   str(data["location"])
6
7           try:
8               #remove from sensor Object
9
10  self.device["sensor"].pop(str(data["deviceN
11  ame"]))
12
13              #first, search index the target
14  deviceName in Object location
15              for item in
16  self.device["locations"][self.targetLocatio
17  n]:
18                  if (item["deviceName"] ==
19  data["deviceName"]):
20                      targetIndex = counter
21                      break
22
23                  counter = counter + 1
24
25              #remove deviceName in Object
26  location
27
28  self.device["locations"][self.targetLocatio
29  n].pop(targetIndex)
30              print "success!"
31              os.system('clear')
32
33  pprint.pprint(json.dumps(self.device),
```

No	Kode Program
34	indent=1)
35	except KeyError:
36	print "error: KeyError - " +
37	str(data["location"]) + ", " +
38	str(data["deviceName"])