LAMPIRAN A KODE PROGRAM

A.1 KODE PROGRAM SENSOR / AKTUATOR

Main.cpp

```
1   //
2   //  main.cpp
3   //  PervasiveSystemProject
4   //
5   //  Created by Hedy Pamungkas on 9/8/17.
6   //  Copyright © 2017 Hedy Pamungkas. All rights
7   reserved.
8   //
9
10  #include <Arduino.h>
11  #include <ESP8266Wifi.h>
12  #include <MQTTClient.h>
13  #include <ArduinoJson.h>
14  #include "main.h"
15  #include "metadata.h"
16  #include "MQTTPervasiveSystem.h"
17
18  PervasiveDevice device;
19  PervasiveService service[2];
20
21  WifiClient net;
22  MQTTClient client(512);
23
24  void setup()
25  {
26      // put your setup code here, to run once:
27      Serial.begin(9600);
28      Wifi.begin(WIFI_SSID, WIFI_PASSWORD);
29      client.begin(MQTT_SERVER_NAME, net);
30      client.onMessage(messageReceived);
31
32      // connect();
33      while (!client.connected())
34      {
35          connect();
36      }
37
38      // SETUP DEVICE
39      // 0 = mode actuator, 1 = mode sensor
40      setupDevice(DEVICE_MODE);
41      for (int i = 0; i < DEVICE_SERVICE_TOTAL;
42  i++)
```

```
43      {
44          setupService(i);
45      }
46      delay(2000);
47      validateDataModel();
48
49      if (device.getDeviceType() == KEY_ACTUATOR)
50      {
51          pinMode(pin, OUTPUT);
52      }
53      else
54      {
55          pinMode(pin, INPUT);
56          // pinMode(pin, INPUT);
57      }
58
59      while (!device.deviceConnected())
60      {
61          // SETUP BROADCAST
62          delay(2000); //fix
63          setupBroadcast(DEVICE_MODE);
64
65  client.subscribe(String(device.getAckTopic()));
66      }
67      digitalWrite(pin, transformDataRelay(0));
68  }
69
70  void validateDataModel()
71  {
72      Serial.println("Model of Device Object");
73
74      Serial.println("device Name\t\t: " +
75  device.getDeviceName());
76      Serial.println("device Category\t\t: " +
77  device.getCategory());
78      Serial.println("device Type\t\t: " +
79  device.getDeviceType());
80      Serial.println("device Ack Topic\t: " +
81  device.getAckTopic());
82      Serial.println("device Location\t\t: " +
83  device.getLocation());
84
85      Serial.println("Model of Service Object");
86
87      Serial.println("service Name\t:" +
88  service[0].getServiceName());
89      Serial.println("service Unit\t:" +
```

```
service[0].getServiceUnit());
    Serial.println("service Data\t:" +
service[0].getServiceData());

    Serial.println("service Name\t:" +
service[1].getServiceName());
    Serial.println("service Unit\t:" +
service[1].getServiceUnit());
    Serial.println("service Data\t:" +
service[1].getServiceData());
}

void loop()
{
    // put your main code here, to run
repeatedly:
    client.loop();
    delay(10); // <- fixes some issues with Wifi
stability

    if (!client.connected())
    {
        connect();
    }

    if (device.getDeviceType() == KEY_ACTUATOR)
    {


client.subscribe(device.getPervasiveTopicWildcard
());
        delay(2000);

        if (integrateReady)
        {
            keepAliveState();

            if (dataReady && actionActuator())
            {
                // action actuator + publish
state
                Serial.println("action" +
service[0].getServiceData());
                digitalWrite(pin,
transformData_Relay_(service[0].getServiceData().to
Int()));
            }
```

```
        }

    }
    else
    {
        publishData();
    }

    delay(1000);
}

void setupDevice(int mode)
{
    device.setDeviceName(DEVICE_NAME);
    device.setCategory(DEVICE_CATEGORY);
    device.setDeviceType(DEVICE_TYPE);
    device.setAckTopic(DEVICE_ACKTOPIC);
    device.setLocation(DEVICE_LOCATION);

    if (mode == 0)
    {

device.setMaximumIntegration(DEVICE_INTEGRATION_M
AX);

        for (int i = 0; i < 2; i++)
        {

device.appendCategoryIntegration(DEVICE_INTEGRATI
ON_CATEGORY[i], i);
        }
    }
}

void setupService(int index)
{

service[index].setServiceName(DEVICE_SERVICE_NAME
[index]);

service[index].setServiceUnit(DEVICE_SERVICE_UNIT
[index]);

service[index].setServiceData(DEVICE_SERVICE_DATA
[index]);
}

```

```
184 void setupBroadcast(int mode)
185 {
186     String payloadBC = "{\"deviceName\": \"" +
187 device.getDeviceName() + "\"" + ",";
188     payloadBC += "\"category\": \"" +
189 device.getCategory() + "\"" + ",";
190     payloadBC += "\"deviceType\": \"" +
191 device.getDeviceType() + "\"" + ",";
192     payloadBC += "\"ackTopic\": \"" +
193 device.getAckTopic() + "\"" + ",";
194     payloadBC += "\"location\": \"" +
195 device.getLocation() + "\"" + ",";
196 
197     // Root=Service
198     payloadBC += "\"service\": {";
199     for (int i = 0; i < DEVICE_SERVICE_TOTAL;
200 i++)
201     {
202         payloadBC += "\"" +
203 DEVICE_SERVICE_NAME[i] + "\": {";
204         payloadBC += "\"name\": \"" +
205 service[i].getServiceName() + "\"" + ",";
206         payloadBC += "\"unit\": \"" +
207 service[i].getServiceData() + "\"" + ",";
208         payloadBC += "\"data\": \"" +
209 service[i].getServiceData() + "\"" + "}";
210         if (i + 1 < DEVICE_SERVICE_TOTAL)
211         {
212             payloadBC += ",";
213         }
214     }
215     payloadBC += "}";
216 
217     if (mode == 0)
218     {
219         // Additional of Actuator Mode
220         // Root=Integration
221         payloadBC += ",";
222         payloadBC += "\"integration\": {\"max\":"
223 + String(device.getMaximumIntegration()) + ",";
224         payloadBC += "\"category\": [\"" +
225 device.getCategoryIntegration(0) + "\"" + ", \""
226 + device.getCategoryIntegration(1) + "\"" + "]}";
227     }
228     payloadBC += "}";
229 
230     String test = "{\"deviceName\":\"BULB-
```

```
001\",\"category\":\"BULB\",\"deviceType\":\"actu
ator\",\"ackTopic\":\"kardel/ack/actuator/BULB-
001\",\"location\":\"kamar\",\"service\":{\"lamp\
":{\"name\":\"lamp\",\"unit\":\"\",\"data\":\"Tru
e\"},\"level\":{\"name\":\"level\",\"unit\":\"
%\",\"data\":0}},\"integration\":{\"max\":2,\"cat
egory\":[\"LDR\",\"PIR\"]}}";
    client.publish(PROJECT_BROADCAST_TOPIC,
payloadBC);
    Serial.println(payloadBC);
    Serial.println(PROJECT_BROADCAST_TOPIC);
    delay(2000);
}

void connect()
{
    // Serial.print("checking wifi...");
    while (Wifi.status() != WL_CONNECTED)
    {
        Serial.print(".");
        delay(1000);
    }

    Serial.print("connecting...");
    while (!client.connect("arduino",
MQTT_USERNAME, MQTT_PASSWORD))
    {
        Serial.print(".");
        delay(1000);
    }

    Serial.print(" ok!\n");
}

void keepAliveState()
{

    for (int i = 0; i <
device.getMaximumIntegration(); i++)
    {
        currentMillisKeepAlive = millis();

        if (previousMillisKeepAlive[i] == 0)
        {
            previousMillisKeepAlive[i] =
currentMillisKeepAlive;
        }
```

```
278
279         // Serial.println(currentMillisKeepAlive
280 + ", " + previousMillisKeepAlive[i]);
281
282         if (currentMillisKeepAlive -
283 previousMillisKeepAlive[i] >= intervalTimeout)
284         {
285             previousMillisKeepAlive[i] =
286 currentMillisKeepAlive;
287             String deviceName =
288 device.getDeviceNameIntegration(i);
289             // Serial.println(String(i) + ", " +
290 device.getDeviceNameIntegration(0) + "-" +
291 device.getDeviceNameIntegration(1));
292
293             int whileCounter = 0;
294             int indexPath = 0;
295             while(deviceName == "")
296             {
297                 if (whileCounter == 0)
298                 {
299                     indexPath = i-1;
300                     deviceName =
301 device.getDeviceNameIntegration(indexPath);
302                 }
303                 else if (whileCounter < 2)
304                 {
305                     indexPath = i+1;
306                     deviceName =
307 device.getDeviceNameIntegration(indexPath);
308                 }
309                 else
310                 {
311                     Serial.println("null
312 variable");
313                 }
314
315                 whileCounter++;
316             }
317
318             Serial.println("Object deleted - " +
319 deviceName + " - " + previousMillisKeepAlive[i]);
320
321             if (deviceName == "")
322             {
323
324 device.removeIntegration(indexPath);
```

```
            }
            else
            {
                device.removeIntegration(i);
            }


updateIntegrationtoGateway(deviceName);
            faultCounter++;
            Serial.println("faultCounter: " +
String(faultCounter) + ", integrate: " +
String(deviceTotalIntegrate));

            if (faultCounter ==
deviceTotalIntegrate)
            {
                previousMillisKeepAlive[i] = 0;
                currentMillisKeepAlive = 0;
                integrateReady = false;
                dataReady = false;
                faultCounter = 0;
                deviceTotalIntegrate = 0;
                Serial.println("all device
integration has disconnected");
            }

            break;
        }
    }
}

void updateIntegrationtoGateway(String
deviceName)
{
    String payloadBC = "{\"deviceName\": \"" +
deviceName + "\"" + ",";
    payloadBC += "\"location\": \"" +
device.getLocation() + "\"";
    payloadBC += "}";

    client.publish(device.getPervasiveTopic() +
"/remove", payloadBC);
    Serial.println("success");
}

void messageReceived(String &topic, String
&payload)
```

```
{
    // Serial.println("\nincoming: " + topic + "
- " + payload);

    if (device.deviceConnected() &&
(device.getDeviceType() == KEY_ACTUATOR))
    //save relational Object of sensor, if device
registered!
    {
        if (topic ==
device.getPervasiveTopicUpdateRelational())
        {
            // Serial.println("received topic in
update relational");
            updateTopicRelational(payload);
            delay(2000);
        }
        else if (topic !=
(device.getPervasiveTopic() + "/remove"))
        {
            // Serial.println("received payload
in topic relational");
            getDataFromTopicRelational(topic,
payload);
        }
    }

    if (topic == device.getAckTopic())
    {

        DynamicJsonBuffer jsonBuffer;
        JsonObject &msgJson =
jsonBuffer.parseObject(payload);
        // Check message response
        if (msgJson["statuscode"] == 200)
        {
            Serial.println("Registration Done");
            device.setDeviceConnected(true);

device.setPervasiveTopic(msgJson[String("replytop
ic_data")]);
        }
        else
        {
            Serial.println("Failed");
            device.setDeviceConnected(false);
        }
```

```
419     }
420
421     delay(3000);
422 }
423
424 void updateTopicRelational(String payload)
425 {
426     //append topic from Object relational
427     DynamicJsonBuffer jsonBuffer;
428     JsonObject &msgJson =
429 jsonBuffer.parseObject(payload);
430
431     if (msgJson["statuscode"] == 200)
432     {
433         for (int i = 0; i <
434 device.getMaximumIntegration(); i++)
435         {
436             if (device.getTopicIntegration(i) ==
437 "")
438             {
439                 previousMillisKeepAlive[i] =
440 millis();
441                 Serial.println("slot-" +
442 String(i) + " available");
443
444 device.appendTopicIntegration(msgJson["update_top
445 ic_sensor"], i);
446
447 device.appendDeviceNameIntegration(msgJson["devic
448 eName"], i);
449
450 Serial.println(device.getTopicIntegration(i) + "
451 - " + device.getDeviceNameIntegration(i) + " - "
452 + previousMillisKeepAlive[i]);
453                 deviceTotalIntegrate++;
454                 Serial.println("integrate:
455 "+String(deviceTotalIntegrate));
456                 integrateReady = true;
457                 break;
458             }
459         }
460     }
461     else
462     {
463         Serial.println("Failed");
464     }
465 }
466
```

```
467
468 void getDataFromTopicRelational(String topic,
469 String payload)
470 {
471     // subscribe all triggered + insert data
472 sensor
473     DynamicJsonBuffer jsonBuffer;
474     JsonObject &msgJson =
475 jsonBuffer.parseObject(payload);
476     String deviceNameFromRelational =
477 msgJson["deviceName"];
478     String deviceService =
479 getServicesFromDeviceName(deviceNameFromRelationa
480 l.substring(0, 3));
481     String data =
482 msgJson["service"][deviceService]["data"];
483
484     for (int i = 0; i <
485 device.getMaximumIntegration(); i++)
486     {
487         if (deviceNameFromRelational ==
488 device.getDeviceNameIntegration(i))
489         {
490
491 device.appendDeviceDataIntegration(data, i);
492             previousMillisKeepAlive[i] =
493 millis();
494             dataReady = true;
495             break;
496         }
497     }
498     dataCounter++;
499     checkDeviceDataIntegration();
500 }
501
502 void checkDeviceDataIntegration()
503 {
504     for (int i = 0; i <
505 device.getMaximumIntegration(); i++)
506     {
507
508 Serial.println(device.getDeviceNameIntegration(i)
509 + " => " + device.getDeviceDataIntegration(i) + "
510 - " + previousMillisKeepAlive[i]);
511     }
512 }
513
514
515
```

```
String getServicesFromDeviceName(String name)
{
    String result = "";
    if (name == "LDR")
    {
        return "light";
    }
    else if (name == "PIR")
    {
        return "motion";
    }
    else if (name == "BUZ")
    {
        return "vibration";
    }
    else
    {
        return "ERROR";
    }
}

bool actionActuator()
{
    int actionFor*Relay* = 0;
    bool swipeCondition = false;
    int deviceTotal =
calculateCurrentDeviceIntegrate();

    for (int i = 0; i <
device.getMaximumIntegration(); i++)
    {
        if (i == 0)
        {
            if (!isLDR(i))
            {
                swipeCondition = true;
                actionFor*Relay* =
actionActuatorWithSwipeCondition(actionFor*Relay*);
                break;
            }

            if
(device.getDeviceDataIntegration(i).toInt() ==
DEVICE_THRESHOLD_INTEGRATION_LDR)
            {
                actionFor*Relay* = 0;
            }

```

```
            else
            {
                actionFor*Relay* = 1;
            }

            if (deviceTotal > 1)
            {
                actionFor*Relay* =
actionActuatorWithoutSwipeCondition(actionFor*Rela
y*, 1);
            }
            else
            {
                break;
            }
        }
    }

service[0].setServiceData(String(actionFor*Relay*))
;
    return true;
}

int calculateCurrentDeviceIntegrate()
{
    return deviceTotalIntegrate;
}

bool isLDR(int i)
{
    if
(device.getDeviceNameIntegration(i).substring(0,
3) == "LDR")
    {
        return true;
    }
    return false;
}

int actionActuatorWithSwipeCondition(int
actionFor*Relay*)
{
    int actionFor*Relay*IntegrateLDR = 0;
    int deviceTotal =
calculateCurrentDeviceIntegrate();

    for (int i = 0; i <
```

```
device.getMaximumIntegration(); i++)
    {
        if (i == 0)
        {
            if
(device.getDeviceDataIntegration(i).toInt() ==
DEVICE_THRESHOLD_INTEGRATION_PIR)
            {
                actionFor*Relay* = 1;
            }
            else
            {
                actionFor*Relay* = 0;
            }

            if (deviceTotal == 1)
            {
                break;
            }
        }
        else if (i == 1)
        {
            if
(device.getDeviceDataIntegration(i).toInt() ==
DEVICE_THRESHOLD_INTEGRATION_LDR)
            {
                actionFor*Relay*IntegrateLDR = 0;
            }
            else
            {
                actionFor*Relay*IntegrateLDR = 1;
            }
        }
    }

    if (deviceTotal > 1)
    {
        actionFor*Relay* =
actionActuatorWithoutSwipeCondition(actionFor*Rela
y*IntegrateLDR, 0);
    }

    return actionFor*Relay*;
}

int actionActuatorWithoutSwipeCondition(int
actionFor*Relay*, int i)

```

```
{
    if (actionForRelay == 0)
    {
        if
(device.getDeviceDataIntegration(i).toInt() ==
DEVICE_THRESHOLD_INTEGRATION_PIR)
        {
            actionForRelay = 1;
        }
        else
        {
            actionForRelay = 0;
        }
    }
    else
    {
        if
(device.getDeviceDataIntegration(i).toInt() ==
DEVICE_THRESHOLD_INTEGRATION_PIR)
        {
            actionForRelay = 0;
        }
        else
        {
            actionForRelay = 1;
        }
    }
    return actionForRelay;
}

int transformDataRelay(int data)
{
    if (data == 1)
    {
        return 0;
    }
    return 1;
}

void publishData()
{
    // if (device.getDeviceType() == KEY_SENSOR)
{
    //     String dataSensor =
getDataFromSensor();
    //     service[0].setServiceData(dataSensor);
    // }
```

```
709
710     String payloadPublish = "{\"deviceName\": \""
711 + device.getDeviceName() + "\"" + ",";
712     payloadPublish += "\"deviceType\": \"" +
713 device.getDeviceType() + "\"" + ",";
714
715     // Root=Service
716     payloadPublish += "\"service\": {";
717     for (int i = 0; i < DEVICE_SERVICE_TOTAL;
718 i++)
719     {
720         payloadPublish += "\"" +
721 service[i].getServiceName() + "\": {";
722         payloadPublish += "\"name\": \"" +
723 service[i].getServiceName() + "\"" + ",";
724         payloadPublish += "\"unit\": \"" +
725 service[i].getServiceUnit() + "\"" + ",";
726         payloadPublish += "\"data\": \"" +
727 service[i].getServiceData() + "\"" + "}";
728         if (i + 1 < DEVICE_SERVICE_TOTAL)
729         {
730             payloadPublish += ",";
731         }
732     }
733     payloadPublish += "}";
734
735     payloadPublish += "}";
736
737     Serial.println(payloadPublish);
738     client.publish(device.getPervasiveTopic(),
739 payloadPublish);
740 }
741
742 String getDataFromSensor()
743 {
744     int data = digitalRead(pin);
745     return String(data);
746 }
```

Main.h

```
1   //
2   //  main.h
3   //  PervasiveSystemProject
4   //
5   //  Created by Hedy Pamungkas on 9/8/17.
6   //  Copyright © 2017 Hedy Pamungkas. All rights
```

```
7   reserved.
8   //
9
10  #ifndef main_h
11  #define main_h
12
13  #include <Arduino.h>
14
15  //setup wifi and mqtt
16  const char *WIFI_SSID = "Theex-HQ";
17  const char *WIFI_PASSWORD = "JuraganPeceLyeye";
18  // const char *WIFI_SSID = "Hedy's iPhone";
19  // const char *WIFI_PASSWORD =
20  "cumangbuatcobacoba";
21  const char *MQTT_SERVER_NAME = "ngehubx.online";
22  const char *MQTT_USERNAME = "admintes";
23  const char *MQTT_PASSWORD = "admin123";
24
25  //setup pin
26  int dataCounter = 0;
27  int currentValidCounter = 1;
28  int faultCounter = 0;
29  int deviceTotalIntegrate = 0;
30  int maximumFaultTolerance = 5;
31  const long intervalTimeout = 60000;
32  unsigned long currentMillisKeepAlive = 0;
33  unsigned long previousMillisKeepAlive[] = {0, 0};
34  bool integrateReady = false;
35  bool dataReady = false;
36  bool updateIntegrationSuccess = false;
37  #define pin D0
38
39  //indexing method
40  void setupDevice(int);
41  void setupService(int);
42  void messageReceived(String&, String&);
43  void setup();
44  void loop();
45  void validateDataModel();
46  void validateDataModelIntegration();
47  void checkDeviceDataIntegration();
48  void setupBroadcast(int);
49  void connect();
50  void keepAliveState();
51  void updateTopicRelational(String);
52  void updateIntegrationtoGateway(String);
53  void getDataFromTopicRelational(String, String);
```

```
54  String getServicesFromDeviceName(String);
55  bool actionActuator();
56  int calculateCurrentDeviceIntegrate();
57  bool isLDR(int);
58  int actionActuatorWithSwipeCondition(int);
59  int actionActuatorWithoutSwipeCondition(int,
60  int);
61  int transformData_Relay_(int);
62  void publishData();
63  String getDataFromSensor();
64
65  #endif /* main_h */
```

A.2 KODE PROGRAM *METADATA* SENSOR PIR

***Metadata*.h**

```
1   //
2   //  metadata.h
3   //  PervasiveSystemProject
4   //
5   //  Created by Hedy Pamungkas on 9/8/17.
6   //  Copyright © 2017 Hedy Pamungkas. All rights
7   reserved.
8   //
9
10  #ifndef metadata_h
11  #define metadata_h
12
13  #include <Arduino.h>
14
15  //device
16  String PROJECT_NAME = "kardel";
17  String PROJECT_ACKTOPIC = PROJECT_NAME + "/ack/";
18  String PROJECT_BROADCAST_TOPIC = PROJECT_NAME +
19  "/broadcast";
20  String DEVICE_NAME = "PIR-001";
21  String DEVICE_CATEGORY = "PIR";
22  String DEVICE_TYPE = "sensor";
23  String DEVICE_ACKTOPIC = PROJECT_ACKTOPIC +
24  DEVICE_TYPE + "/" + DEVICE_NAME;
25  String DEVICE_LOCATION = "kamar";
26
27  //additional for actuator
28  int DEVICE_INTEGRATION_MAX = 0;
29  String DEVICE_INTEGRATION_CATEGORY[] = {"LDR",
30  "PIR"};
```

```
int DEVICE_THRESHOLD_INTEGRATION_LDR = 0;
int DEVICE_THRESHOLD_INTEGRATION_PIR = 1;

//service
int DEVICE_SERVICE_TOTAL = 1;
String DEVICE_SERVICE_NAME[] = {"motion"};
String DEVICE_SERVICE_UNIT[] = {" "};
String DEVICE_SERVICE_DATA[] = {"0"};

//other
String KEY_ACTUATOR = "actuator";
String KEY_SENSOR = "sensor";
String KEY_LIST_OF_SERVICES[5][5] = {{"light",
"LDR"}, {"motion", "PIR"}, {"vibration", "BUZ"}};
const int DEVICE_MODE = 1;

#endif /* metadata_h */
```

A.3 KODE PROGRAM *METADATA* SENSOR LDR

***Metadata*.h**

```
//
//  metadata.h
//  PervasiveSystemProject
//
//  Created by Hedy Pamungkas on 9/8/17.
//  Copyright © 2017 Hedy Pamungkas. All rights
reserved.
//

#ifndef metadata_h
#define metadata_h

#include <Arduino.h>

//device
String PROJECT_NAME = "kardel";
String PROJECT_ACKTOPIC = PROJECT_NAME + "/ack/";
String PROJECT_BROADCAST_TOPIC = PROJECT_NAME +
"/broadcast";
String DEVICE_NAME = "LDR-001";
String DEVICE_CATEGORY = "LDR";
String DEVICE_TYPE = "sensor";
String DEVICE_ACKTOPIC = PROJECT_ACKTOPIC +
DEVICE_TYPE + "/" + DEVICE_NAME;
String DEVICE_LOCATION = "kamar";
```

```
26
27  //additional for actuator
28  int DEVICE_INTEGRATION_MAX = 0;
29  String DEVICE_INTEGRATION_CATEGORY[] = {"LDR",
30  "PIR"};
31  int DEVICE_THRESHOLD_INTEGRATION_LDR = 0;
32  int DEVICE_THRESHOLD_INTEGRATION_PIR = 1;
33
34  //service
35  int DEVICE_SERVICE_TOTAL = 1;
36  String DEVICE_SERVICE_NAME[] = {"light"};
37  String DEVICE_SERVICE_UNIT[] = {" "};
38  String DEVICE_SERVICE_DATA[] = {"0"};
39
40  //other
41  String KEY_ACTUATOR = "actuator";
42  String KEY_SENSOR = "sensor";
43  String KEY_LIST_OF_SERVICES[5][5] = {{"light",
44  "LDR"}, {"motion", "PIR"}, {"vibration", "BUZ"}};
45  const int DEVICE_MODE = 1;
46
47  #endif /* metadata_h */
```

A.4 KODE PROGRAM *METADATA* AKTUATOR

***Metadata*.h**

```
1   //
2   //  metadata.h
3   //  PervasiveSystemProject
4   //
5   //  Created by Hedy Pamungkas on 9/8/17.
6   //  Copyright © 2017 Hedy Pamungkas. All rights
7   reserved.
8   //
9
10  #ifndef metadata_h
11  #define metadata_h
12
13  #include <Arduino.h>
14
15  //device
16  String PROJECT_NAME = "kardel";
17  String PROJECT_ACKTOPIC = PROJECT_NAME + "/ack/";
18  String PROJECT_BROADCAST_TOPIC = PROJECT_NAME +
19  "/broadcast";
20  String DEVICE_NAME = "BULB-001";
```

```
21  String DEVICE_CATEGORY = "BULB";
22  String DEVICE_TYPE = "actuator";
23  const int DEVICE_MODE = 0;
24
25  // String DEVICE_NAME = "LDR-001";
26  // String DEVICE_CATEGORY = "LDR";
27  // String DEVICE_NAME = "PIR-001";
28  // String DEVICE_CATEGORY = "PIR";
29  // String DEVICE_TYPE = "sensor";
30  // const int DEVICE_MODE = 1;
31
32  String DEVICE_ACKTOPIC = PROJECT_ACKTOPIC +
33  DEVICE_TYPE + "/" + DEVICE_NAME;
34  String DEVICE_LOCATION = "kamar";
35
36  //additional for actuator
37  int DEVICE_INTEGRATION_MAX = 2;
38  String DEVICE_INTEGRATION_CATEGORY[] = {"LDR",
39  "PIR"};
40  int DEVICE_THRESHOLD_INTEGRATION_LDR = 0;
41  int DEVICE_THRESHOLD_INTEGRATION_PIR = 1;
42
43  //service
44  int DEVICE_SERVICE_TOTAL = 2;
45  String DEVICE_SERVICE_NAME[] = {"lamp", "level"};
46  String DEVICE_SERVICE_UNIT[] = {" ", " %"};
47  String DEVICE_SERVICE_DATA[] = {"0", "0"};
48
49  //other
50  String KEY_ACTUATOR = "actuator";
51  String KEY_SENSOR = "sensor";
52  String KEY_LIST_OF_SERVICES[5][5] = { {"light",
53  "LDR"},  {"motion", "PIR"}, {"vibration", "BUZ"}
54  };
55
56  #endif /* metadata_h */
```

A.5 KODE PROGRAM OBJEK PERVASIF

MQTTPervasiveSystem.cpp

```
1  //
2  //  MQTTPervasiveSystem.cpp
3  //  PervasiveSystemProject
4  //
5  //  Created by Hedy Pamungkas on 9/8/17.
6  //  Copyright © 2017 Hedy Pamungkas. All rights
```

```
7   reserved.
8   //
9
10  #include <Arduino.h>
11  #include "MQTTPervasiveSystem.h"
12
13  void PervasiveDevice::setDeviceName(String
14  newDeviceNew)
15  {
16      deviceName = newDeviceNew;
17  }
18
19  void PervasiveDevice::setCategory(String
20  newCategory)
21  {
22      category = newCategory;
23  }
24
25  void PervasiveDevice::setDeviceType(String
26  newDeviceType)
27  {
28      deviceType = newDeviceType;
29  }
30
31  void PervasiveDevice::setAckTopic(String
32  newAckTopic)
33  {
34      ackTopic = newAckTopic;
35  }
36
37  void PervasiveDevice::setLocation(String
38  newLocation)
39  {
40      location = newLocation;
41  }
42
43  void PervasiveDevice::setMaximumIntegration(int
44  newMaximumIntegration)
45  {
46      maximumIntegration = newMaximumIntegration;
47  }
48
49  void
50  PervasiveDevice::appendCategoryIntegration(String
51  newCategoryIntegration, int indexPath)
52  {
53      categoryIntegration[indexPath] =
```

```
54  newCategoryIntegration;
55  }
56
57  void
58  PervasiveDevice::appendTopicIntegration(String
59  newTopicIntegration, int indexPath)
60  {
61      topicIntegration[indexPath] =
62  newTopicIntegration;
63  }
64
65  void
66  PervasiveDevice::appendDeviceNameIntegration(Stri
67  ng newDeviceNameIntegration, int indexPath)
68  {
69      deviceNameIntegration[indexPath] =
70  newDeviceNameIntegration;
71  }
72
73  void
74  PervasiveDevice::appendDeviceDataIntegration(Stri
75  ng newDeviceDataIntegration, int indexPath)
76  {
77      deviceDataIntegration[indexPath] =
78  newDeviceDataIntegration;
79  }
80
81  void PervasiveDevice::removeIntegration(int
82  indexPath)
83  {
84      topicIntegration[indexPath] = "";
85      deviceNameIntegration[indexPath] = "";
86      deviceDataIntegration[indexPath] = "";
87  }
88
89  void PervasiveDevice::setDeviceConnected(bool
90  newState)
91  {
92      connected = newState;
93  }
94
95  void PervasiveDevice::setPervasiveTopic(String
96  newTopicPervasive)
97  {
98      pervasiveTopic = newTopicPervasive;
99  }
100
```

```
101 void PervasiveService::setServiceName(String
102 newServiceName)
103 {
104     serviceName = newServiceName;
105 }
106
107 void PervasiveService::setServiceUnit(String
108 newServiceUnit)
109 {
110     serviceUnit = newServiceUnit;
111 }
112
113 void PervasiveService::setServiceData(String
114 newServiceData)
115 {
116     serviceData = newServiceData;
117 }
118
119 String PervasiveDevice::getDeviceName() const
120 {
121     return deviceName;
122 }
123
124 String PervasiveDevice::getCategory() const
125 {
126     return category;
127 }
128
129 String PervasiveDevice::getDeviceType() const
130 {
131     return deviceType;
132 }
133
134 String PervasiveDevice::getAckTopic() const
135 {
136     return ackTopic;
137 }
138
139 String PervasiveDevice::getLocation() const
140 {
141     return location;
142 }
143
144 int PervasiveDevice::getMaximumIntegration()
145 const
146 {
147     return maximumIntegration;
```

```
148 }
149
150 String
151 PervasiveDevice::getCategoryIntegration(int
152 indexPath)
153 {
154     return categoryIntegration[indexPath];
155 }
156
157 String PervasiveDevice::getTopicIntegration(int
158 indexPath)
159 {
160     return topicIntegration[indexPath];
161 }
162
163 String
164 PervasiveDevice::getDeviceNameIntegration(int
165 indexPath)
166 {
167     return deviceNameIntegration[indexPath];
168 }
169
170 String
171 PervasiveDevice::getDeviceDataIntegration(int
172 indexPath)
173 {
174     return deviceDataIntegration[indexPath];
175 }
176
177 bool PervasiveDevice::deviceConnected()
178 {
179     return connected;
180 }
181
182 String PervasiveDevice::getPervasiveTopic() const
183 {
184     return pervasiveTopic;
185 }
186
187 String
188 PervasiveDevice::getPervasiveTopicUpdateRelationa
189 l() const
190 {
191     return pervasiveTopic + "/update";
192 }
193
194 String
```

```
195 PervasiveDevice::getPervasiveTopicWildcard()
196 const
197 {
198     String newTopic = ackTopic.substring(0,6);
199     return newTopic + "/pervasive/" + location +
200 "/#";
201 }
202
203 String PervasiveService::getServiceName() const
204 {
205     return serviceName;
206 }
207
208 String PervasiveService::getServiceUnit() const
209 {
210     return serviceUnit;
211 }
212
213 String PervasiveService::getServiceData() const
214 {
215     return serviceData;
216 }
```

MQTTPervasiveSystem.h

```
1   //
2   //  MQTTPervasiveSystem.h
3   //  PervasiveSystemProject
4   //
5   //  Created by Hedy Pamungkas on 9/8/17.
6   //  Copyright © 2017 Hedy Pamungkas. All rights
7   reserved.
8   //
9
10  #ifndef MQTTPervasiveSystem_h
11  #define MQTTPervasiveSystem_h
12
13  #include <Arduino.h>
14
15  class PervasiveDevice
16  {
17    public:
18      void setDeviceName(String);
19      void setCategory(String);
20      void setDeviceType(String);
21      void setAckTopic(String);
22      void setLocation(String);
```

```
23      void setMaximumIntegration(int);
24      void appendCategoryIntegration(String, int);
25      void appendTopicIntegration(String, int);
26      void appendDeviceNameIntegration(String,
27  int);
28      void appendDeviceDataIntegration(String,
29  int);
30      void removeIntegration(int);
31      void setDeviceConnected(bool);
32      void setPervasiveTopic(String);
33      String getDeviceName() const;
34      String getCategory() const;
35      String getDeviceType() const;
36      String getAckTopic() const;
37      String getLocation() const;
38      int getMaximumIntegration() const;
39      String getCategoryIntegration(int);
40      String getTopicIntegration(int);
41      String getDeviceNameIntegration(int);
42      String getDeviceDataIntegration(int);
43      bool deviceConnected();
44      String getPervasiveTopic() const;
45      String getPervasiveTopicUpdateRelational()
46  const;
47      String getPervasiveTopicWildcard() const;
48
49    private:
50      String deviceName;
51      String category;
52      String deviceType;
53      String ackTopic;
54      String location;
55      int maximumIntegration;
56      String categoryIntegration[5];
57      String topicIntegration[5] = {"", "", "", "",
58  ""};
59      String deviceNameIntegration[5] = {"", "",
60  "", "", ""};
61      String deviceDataIntegration[5] = {"", "",
62  "", "", ""};
63      bool connected = false;
64      String pervasiveTopic;
65  };
66
67  class PervasiveService
68  {
69    public:
```

```
70      void setServiceName(String);
71      void setServiceUnit(String);
72      void setServiceData(String);
73      String getServiceName() const;
74      String getServiceUnit() const;
75      String getServiceData() const;
76
77    private:
78      String serviceName;
79      String serviceUnit;
80      String serviceData;
81  };
82
83  #endif /* MQTTPervasiveSystem_h */
```

A.6 KODE PROGRAM *GATEWAY*

***Gateway*.py**

```
1   import paho.mqtt.client as paho
2   import datetime, json, time, sys, ast, pprint, os
3
4   class MQTTPervasiveSystem(Object):
5
6       def __init__(self, mqtt_host, project_name,
7   mqtt_user_name, mqtt_password, mqtt_port=1883,
8                        mqtt_keepalive=60,
9   mqtt_qos=0):
10          self.mqtt_host = mqtt_host
11          self.mqtt_port = mqtt_port
12          self.mqtt_user_name = mqtt_user_name
13          self.mqtt_password = mqtt_password
14          self.mqtt_keepalive = mqtt_keepalive
15          self.mqtt_qos = mqtt_qos
16          self.project_name = project_name
17          self.mqtt_topic_name = project_name +
18  "/#"
19          self.topic_broadcast = project_name +
20  "/broadcast"
21          self.topic_pervasive = project_name +
22  "/pervasive/"
23          self.targetLocation = ""
24          self.client = paho.Client()
25          self.counterOfConnect = 0
26          self.client.on_message = self.on_message
27          self.client.on_connect = self.on_connect
28          # self.client.on_log = self.on_log
```

```
29          self.device = {
30              "sensor": {
31
32              },
33              "actuator": {
34
35              },
36              "locations": {
37
38              }
39          }
40
41      def checkingDevice(self, msg):
42          data = json.loads(msg)
43          flag = False
44
45          #Checking Device Name
46          obj =
47  self.device[str(data["deviceType"])].keys()
48          try:
49              obj.index(data["deviceName"])
50          except ValueError:
51              flag = True
52          else:
53              flag = False
54              return flag
55
56          if (flag==True):
57              #Checking Actuator First
58
59  if(str(data["deviceType"])=="sensor"):
60                 return
61  bool(self.device["actuator"])
62              else:
63                  return True
64
65      def checkingRelation(self, msg):
66          data = json.loads(msg)
67          resultName = ""
68          statusRelationalFlag = False
69          currentItemInLocation = 0
70
71          # 1. searching actuator data in location
72  & msg=sensor
73          try:
74              targetLocationActuator =
75  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


        # 2. get data max and category
compatibility in actuator data
        if (statusRelationalFlag):
            try:
                print('masuk')
                targetIntegrationActuator =
self.device["actuator"][str(resultName)]["integra
tion"]
                categoryCompability =
targetIntegrationActuator["category"]
                maximumIntegration =
int(targetIntegrationActuator["max"])
            except KeyError:
                statusRelationalFlag = False
            else:
                statusRelationalFlag = True

            # 3. check condition location
            if
(currentItemInLocation<maximumIntegration) and
(statusRelationalFlag):
                try:

categoryCompability.index(str(data["category"]))
                except ValueError:
                    print('Incompability')
                else:
                    print('Compability')

self.relationCompatibility(msg,
```

```
targetLocationActuator[0]["topic_pervasive"])
            else:
                print('Oops, already max!')

        return statusRelationalFlag

    def relationCompatibility(self, msg,
target_topic=''):
        data = json.loads(msg)
        replytopic = self.topic_pervasive +
data["location"] + "/" + data["deviceType"] + "/"
+ data["deviceName"]
        print('relationCompatibility')

        try:

self.device["locations"][str(data["location"])].a
ppend(
                {
                    "deviceName":
str(data["deviceName"]),
                    "deviceType":
str(data["deviceType"]),
                    "topic_pervasive":
str(replytopic) + "/data"
                }
            )
        except KeyError:

self.device["locations"][str(data["location"])] =
[
                {
                    "deviceName":
str(data["deviceName"]),
                    "deviceType":
str(data["deviceType"]),
                    "topic_pervasive":
str(replytopic) + "/data"
                }
            ]

        if (bool(target_topic)):
            update_relational = {
                "statuscode": 200,
                "deviceName":
str(data["deviceName"]),
                "update_topic_sensor":
```

```
170 str(replytopic) + "/data"
171             }
172             print(target_topic)
173             self.client.publish(target_topic +
174 "/update", str(update_relational), self.mqtt_qos)
175 
176     def deviceConnect(self, msg):
177         data = json.loads(msg)
178         replytopic = self.topic_pervasive +
179 data["location"] + "/" + data["deviceType"] + "/"
180 + data["deviceName"]
181         isCompatibility = False
182 
183         if (self.counterOfConnect != 0):
184 
185             if (self.checkingDevice(msg)):
186 
187                 if
188 (str(data["deviceType"])=="sensor"):
189                     print('masuk sensor')
190                     isCompatibility =
191 self.checkingRelation(msg)
192                 else:
193 
194 self.relationCompatibility(msg)
195 
196 self.device[str(data["deviceType"])][str(data["de
197 viceName"])] = ast.literal_eval(msg)
198 
199                 device_connect_msg = {
200                     "statuscode": 200,
201                     "replytopic_data":
202 str(replytopic) + "/data"
203                 }
204 
205                 if (isCompatibility):
206 
207 self.device[str(data["deviceType"])][str(data["de
208 viceName"])] = ast.literal_eval(msg)
209 
210                 os.system('clear')
211 
212 pprint.pprint(json.dumps(self.device), indent=1)
213             else:
214                 device_connect_msg = {
215                     "statuscode": 404
216                 }
```

```
                print('blocked!')

            self.client.publish(data["ackTopic"],
str(device_connect_msg), self.mqtt_qos)
            self.counterOfConnect = 0

        else:
            self.counterOfConnect = 1

    def removeIntegration(self, topic, msg):
        # print topic, msg
        data = json.loads(msg)
        targetIndex = 0
        counter = 0
        self.targetLocation =
str(data["location"])

        try:
            #remove from sensor Object

self.device["sensor"].pop(str(data["deviceName"])
)

            #first, search index the target
deviceName in Object location
            for item in
self.device["locations"][self.targetLocation]:
                if (item["deviceName"] ==
data["deviceName"]):
                    targetIndex = counter
                    break

                counter = counter + 1

            #remove deviceName in Object location

self.device["locations"][self.targetLocation].pop
(targetIndex)
            print "success!"
            os.system('clear')

pprint.pprint(json.dumps(self.device), indent=1)
        except KeyError:
            print "error: KeyError - " +
str(data["location"]) + ", " +
str(data["deviceName"])

```

```
264     def pervasivePersistance(self, topic, msg):
265         if (str(topic).find("remove")==-1):
266             data = json.loads(msg)
267             print("masuk update fix")
268 
269             #Checking Device Name
270             obj =
271 self.device[str(data["deviceType"])].keys()
272             try:
273 
274 obj.index(str(data["deviceName"]))
275             except ValueError:
276                 print('device has not been
277 registered')
278             else:
279 
280 self.device[str(data["deviceType"])][str(data["de
281 viceName"])]["service"] = data["service"]
282                 os.system('clear')
283 
284 pprint.pprint(json.dumps(self.device), indent=1)
285 
286     def connect(self):
287         print("Starting MQTT Stream")
288         self.client.connect_async(
289             host=self.mqtt_host,
290             port=self.mqtt_port,
291             keepalive=self.mqtt_keepalive
292         )
293 
294     def username_pw_set(self):
295         self.client.username_pw_set(
296             self.mqtt_user_name,
297             password=self.mqtt_password
298         )
299 
300     def on_message(self, mqttc, userdata, msg):
301         # print('on_message topic:
302 {0}\tmsg.payload: {1}\ttime: {2}'.format(
303         #     msg.topic,
304         #     msg.payload,
305         #     datetime.datetime.now())
306         # )
307         # print(msg.payload)
308         # insert data_sensor
309         if (msg.topic==self.topic_broadcast):
310             self.deviceConnect(msg.payload)
```

```
        elif
(str(msg.topic).find(self.topic_pervasive)!=-1):
            if (str(msg.topic).find("remove")!=-
1):
                #remove integration
                print("masuk remove")
                self.removeIntegration(msg.topic,
msg.payload)
            else:
                if
(str(msg.topic).find("update")==-1):
                    print("masuk update")

self.pervasivePersistance(msg.topic, msg.payload)



    def on_log(self, client, userdata, level,
buf):
        print("log: ",buf)
        self.client.on_log=self.on_log #client
logging

    def on_connect(self, mqttc, userdata, flags,
msg):
        rc =
mqttc.subscribe(self.mqtt_topic_name, 0)
        # print('Subscribe topic: {0} RC: {1}
time: {2}'.format(self.mqtt_topic_name, rc,
datetime.datetime.now()))

if __name__ == '__main__':

    try:
        gateway = MQTTPervasiveSystem(
            mqtt_host="ngehubx.online",
            project_name="kardel",
            mqtt_user_name="admintes",
            mqtt_password="admin123"
        )
        gateway.connect()
        gateway.username_pw_set()
        gateway.client.loop_forever()
    except KeyboardInterrupt:
        print('\nSYSTEM HAS BEEN FORCED CLOSE\n')
        sys.exit()
```