Part 1 and 2 Explanation

Java project

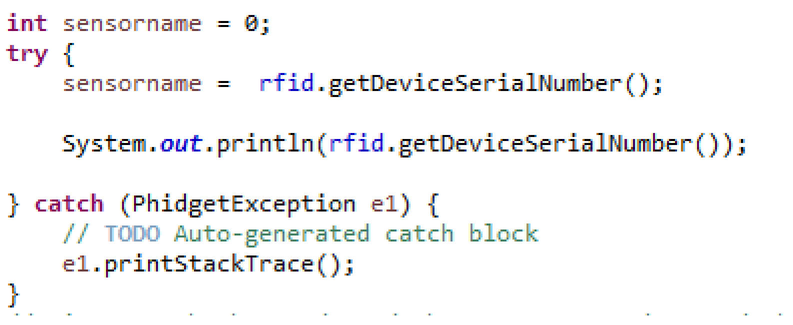
- database table is on a remote server to the actual lock/motor.
- A separate class has been developed to open the lock using an RFID tag, opening for authorized tags. Tags and doors are linked so that a tag only opens the appropriate door
- A log records the tag id used to attempt the open the lock, the time, and success/fail of the attempt.
- MQTT has been used to open door
- Data is sent in json format.
- Json is sent using HTTP
- Client will only open a given door and not all doors
- Log table is updated upon success and fail

Phone app

- user verification is used in the form of a phone IMEI number to ensure the correct phone opens the correct door.
- The app has a button to open the lock.
- Data is sent to server in json format
- Client will only open a given door and not all doors
- Log table is updated upon success and fail
- Sends push notification when door has been opened from phone or card scan.

Client-side Sensor Client

The Client starts by getting the serial number of the card reader, so the server can look up which door it is opening by the serial number of the scanner. This is done so that the door is seen as one unit. the serial number is also stored in the DB for the motor and is used to get back the door id. This is used to validate that a card is for one door and not many.



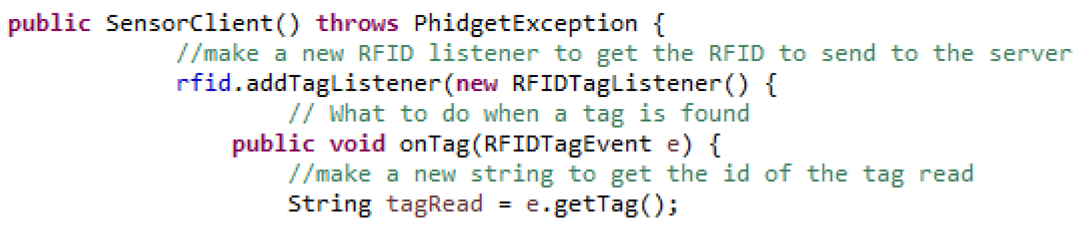
```
int sensorname = 0;
try {
    sensorname =  rfid.getDeviceSerialNumber();

    System.out.println(rfid.getDeviceSerialNumber());

} catch (PhidgetException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}
```

The Client also has a “clientType” which is used In the server to understand which type of client is sending data and what kind of activity to preform.

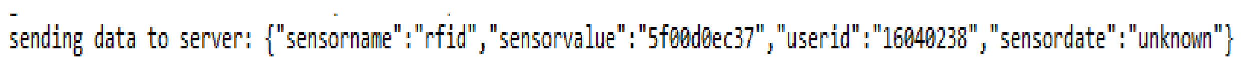
```
RFIDScanned.setclienttype("RFIDClient");
```

The client sends json data to the server when the key card is scanned. This is done by the sensor client class. there is an event listener which is listening for an RFID card to be scanned.

once a card has been scanned the OnTag method then calls the Send to server method and sends Json data to the server. The json data is created by a google library called gson. And the information is put together using a sensor data object class.

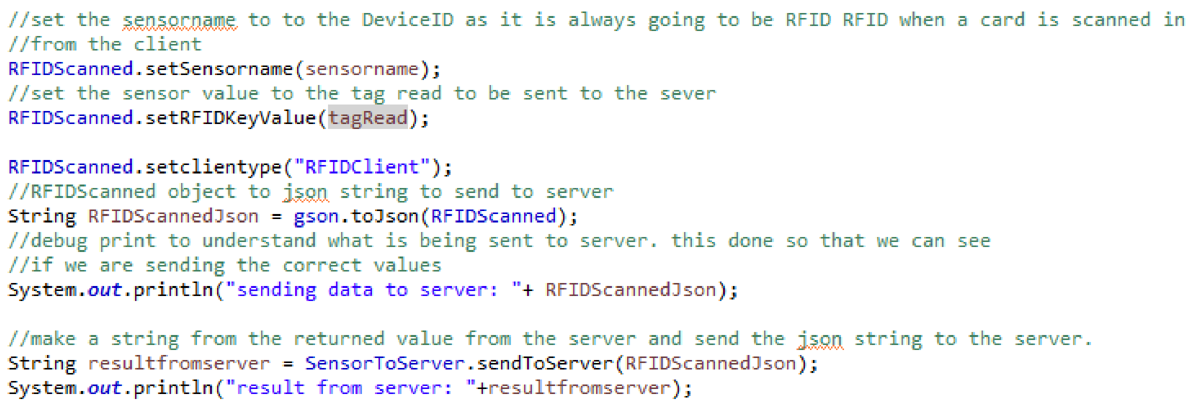


```
public SensorClient() throws PhidgetException {
            //make a new RFID listener to get the RFID to send to the server
            rfid.addTagListener(new RFIDTagListener() {
                    // What to do when a tag is found
                public void onTag(RFIDTagEvent e) {
                    //make a new string to get the id of the tag read
                    String tagRead = e.getTag();
```

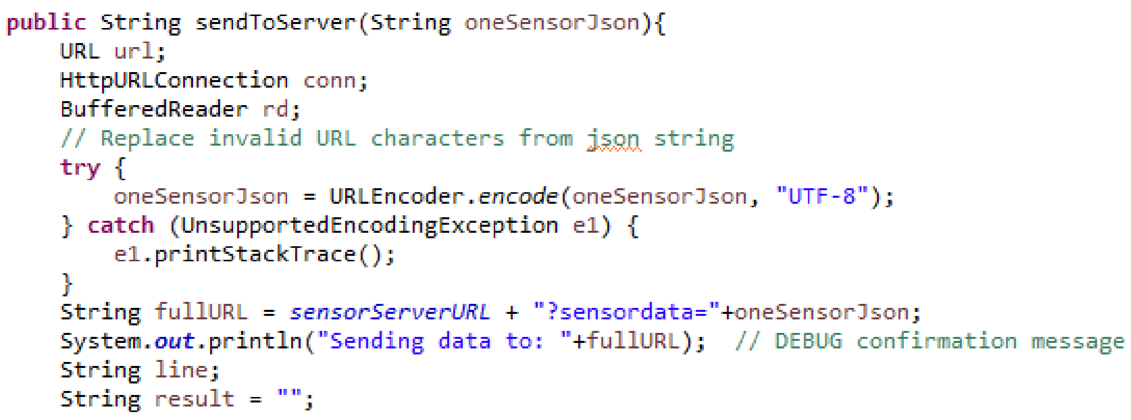


```
sending data to server: {"sensorname":"rfid","sensorvalue":"5f00d0ec37","userid":"16040238","sensordate":"unknown"}
```

The SensorToServer method takes one argument of a string which is converted to json in the SensorClient class. The Send to server method encodes the json string using UTF-8 encoding and then uses HTTP to send the json to the server.



```
//set the sensorname to to the DeviceID as it is always going to be RFID RFID when a card is scanned in
//from the client
RFIDScanned.setSensorname(sensorname);
//set the sensor value to the tag read to be sent to the sever
RFIDScanned.setRFIDKeyValue(tagRead);

RFIDScanned.setclienttype("RFIDClient");
//RFIDScanned object to json string to send to server
String RFIDScannedJson = gson.toJson(RFIDScanned);
//debug print to understand what is being sent to server. this done so that we can see
//if we are sending the correct values
System.out.println("sending data to server: "+ RFIDScannedJson);

//make a string from the returned value from the server and send the json string to the server.
String resultfromserver = SensorToServer.sendToServer(RFIDScannedJson);
System.out.println("result from server: "+resultfromserver);
```



```
public String sendToServer(String oneSensorJson){
    URL url;
    HttpURLConnection conn;
    BufferedReader rd;
    // Replace invalid URL characters from json string
    try {
        oneSensorJson = URLEncoder.encode(oneSensorJson, "UTF-8");
    } catch (UnsupportedEncodingException e1) {
        e1.printStackTrace();
    }
    String fullURL = sensorServerURL + "?sensordata="+oneSensorJson;
    System.out.println("Sending data to: "+fullURL);  // DEBUG confirmation message
    String line;
    String result = "";
```

```
try {
   url = new URL(fullURL);
   conn = (HttpURLConnection) url.openConnection();
   conn.setRequestMethod("GET");
   rd = new BufferedReader(new InputStreamReader(conn.getInputStream()));
```

```
Sending data to: http://localhost:8081/ServerSide/SensorServerDB?sensordata=%
```

Once the data is sent to the server the server processes the data and checks if the card is valid (explained in part two). The server then sends the sensor value scanned and the door id back to the client Once the data is sent back from the server after server has validated the card. The client the client then uses MQTT to publish out to subscribing motors.

```
Door id:1
Published data. Topic: 1/motor   Message: open door
```

This is done is a separate class to the Sensor Client but is called in the sensor client.

```
if(resultfromserver.contains(tagRead)) {
     publisher.start(doorid);
}
```

The Subscribing door will then open, given its door id. The subscriber will then move the motor to a given position, thus opening the door, it will then move the motor back again to close the door.

```
Door id:2
{"sensorname":0,"RFIDKeyValue":"11006047c4","doorid":"2","clientType":"unknown","PhoneID":"1603"}
Published data. Topic: 2/motor   Message: door opened from card scan
```

```
Message arrived. Topic: 1/motor  Message: open door
In singleton constructor
moving to 180.0
Waiting until motor at position 180
Position Changed: 50.16973684210526
```

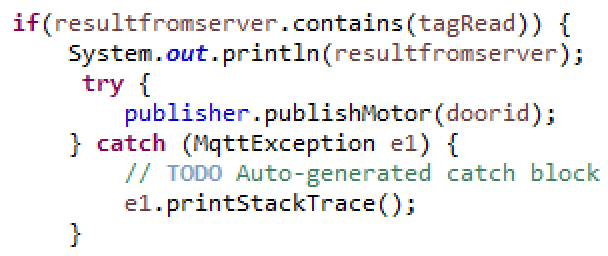
If the card is not correct for the door then the door will not open and the client will publish to an attempts MQTT channel. The attempts channel is given the scanner sensor id to publish

```
Door id:unknown

Published data. Topic: 549945/attempts   Message: door failed to open door from card scan
```

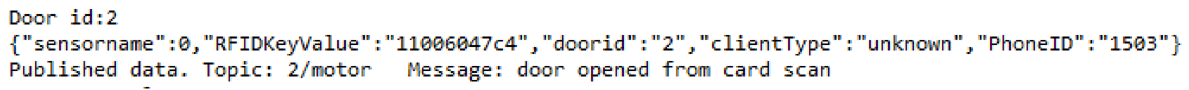
```
else {
   try {
       publisher.publishFailed(RFIDScanned.getSensorname());
   } catch (MqttException e1) {
       // TODO Auto-generated catch block
       e1.printStackTrace();
   }
```

Client-Side Publisher

An instance of the publisher class created when the RFID client starts, but the publisher only publishes when the door id has been sent back from the server. The publish motor method is given the same door id to publish to given door

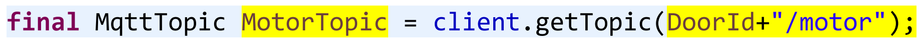


```
if(resultfromserver.contains(tagRead)) {
     System.out.println(resultfromserver);
      try {
         publisher.publishMotor(doorid);
     } catch (MqttException e1) {
         // TODO Auto-generated catch block
         e1.printStackTrace();
     }
```



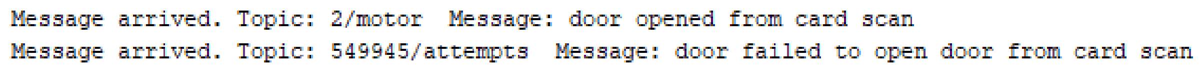
```
Door id:2
{"sensorname":0,"RFIDKeyValue":"11006047c4","doorid":"2","clientType":"unknown","PhoneID":"1503"}
Published data. Topic: 2/motor   Message: door opened from card scan
```

The publish motor method takes in one argument of type String, it uses this string door id to set the topic to publish out to. The topic is what the subscribing motor is listening to.

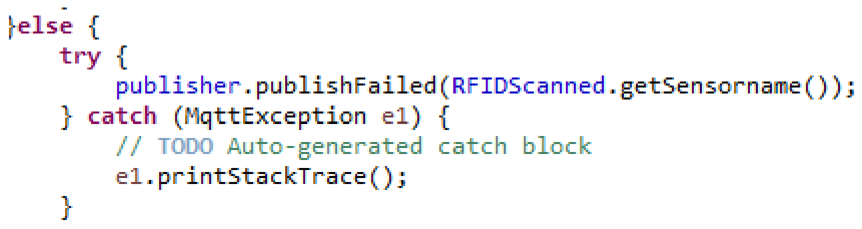


```
final MqttTopic MotorTopic = client.getTopic(DoorId+"/motor");
```

when an attempt is made on a door and the card is wrong for that door, the client will publish out over MQTT to tell anyone subscribing namely the phone that it is wrong.

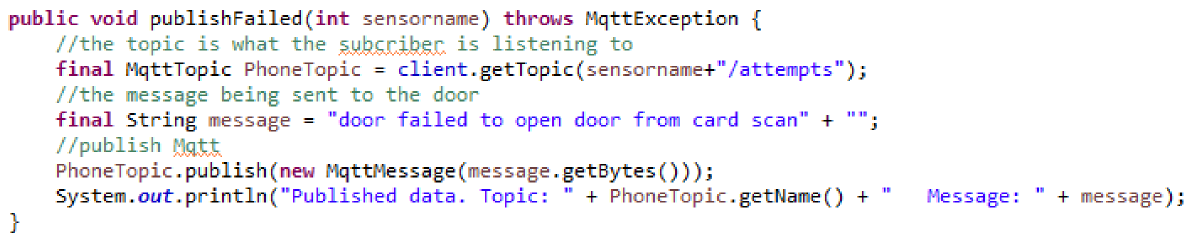


```
Message arrived. Topic: 2/motor  Message: door opened from card scan
Message arrived. Topic: 549945/attempts  Message: door failed to open door from card scan
```



```
}else {
    try {
        publisher.publishFailed(RFIDScanned.getSensorname());
    } catch (MqttException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
```

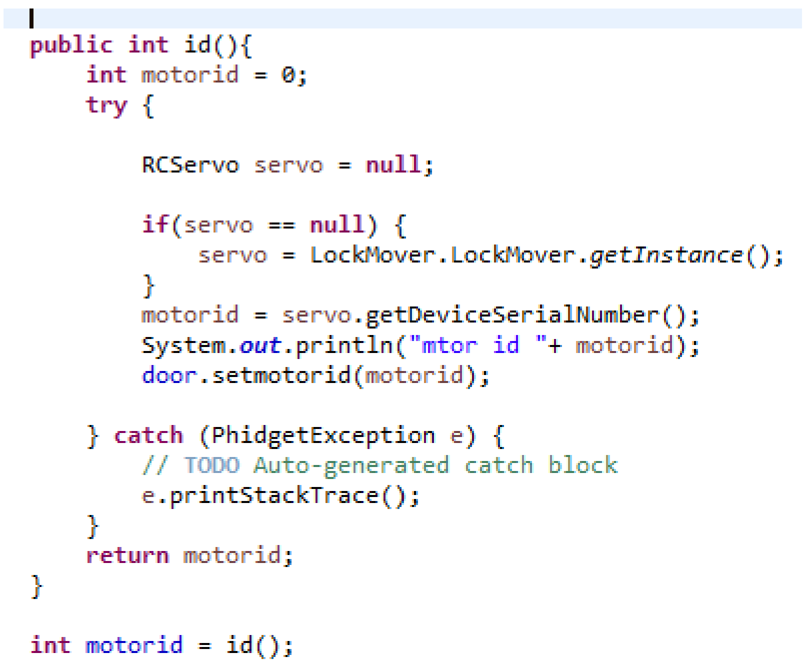
The publisher also has an attempts method which is called when a wrong card is scanned.



```
public void publishFailed(int sensorname) throws MqttException {
    //the topic is what the subcriber is listening to
    final MqttTopic PhoneTopic = client.getTopic(sensorname+"/attempts");
    //the message being sent to the door
    final String message = "door failed to open door from card scan" + "";
    //publish Mqtt
    PhoneTopic.publish(new MqttMessage(message.getBytes()));
    System.out.println("Published data. Topic: " + PhoneTopic.getName() + "   Message: " + message);
}
```

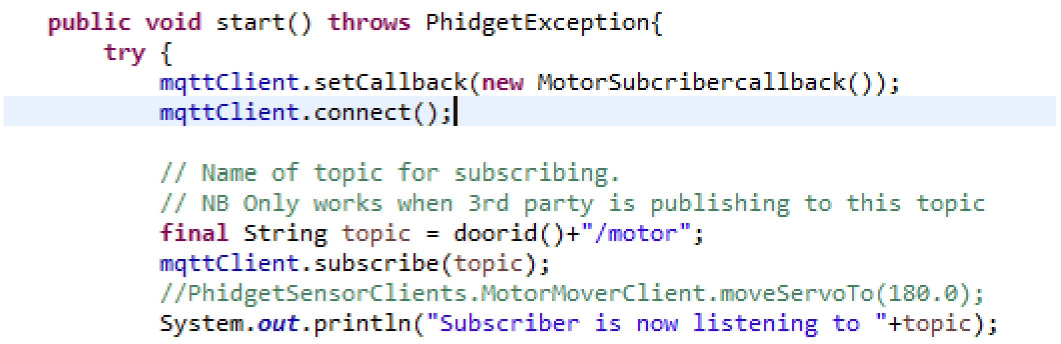
Client-Side Subscriber

the Client-side Subscriber, gets the Serial number of the motor, this motor id is then sent to the server.

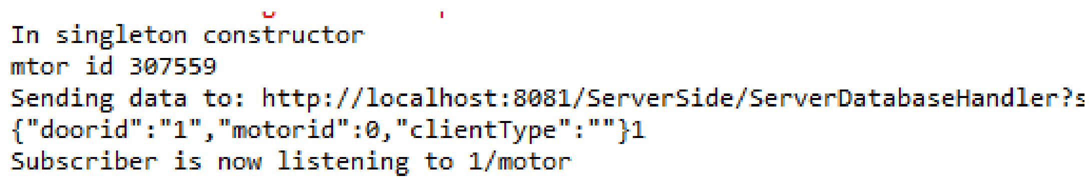


```
public int id(){
    int motorid = 0;
    try {

        RCServo servo = null;

        if(servo == null) {
            servo = LockMover.LockMover.getInstance();
        }
        motorid = servo.getDeviceSerialNumber();
        System.out.println("mtor id "+ motorid);
        door.setmotorid(motorid);

    } catch (PhidgetException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return motorid;
}

int motorid = id();
```

The server sends back the door id for its Serial number, which is stored in the DB, the door ID is sent back in JSON format, a converted to an object and the door id is given to the subscriber. This look up is only preformed when the subscriber first starts up.

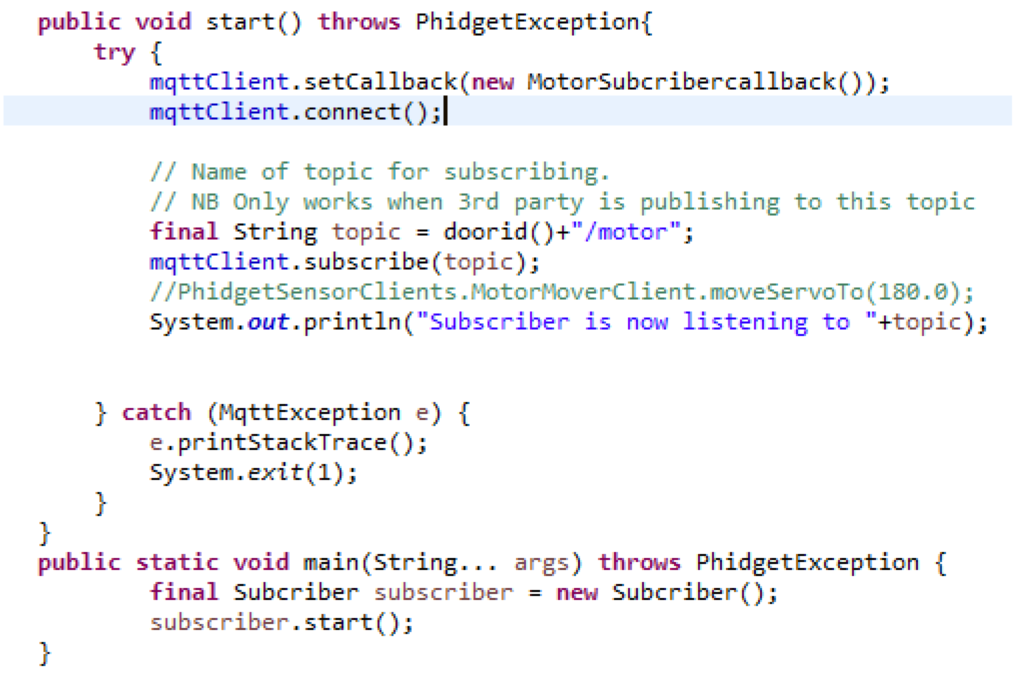


```
public void start() throws PhidgetException{
    try {
        mqttClient.setCallback(new MotorSubcribercallback());
        mqttClient.connect();

        // Name of topic for subscribing.
        // NB Only works when 3rd party is publishing to this topic
        final String topic = doorid()+"/motor";
        mqttClient.subscribe(topic);
        //PhidgetSensorClients.MotorMoverClient.moveServoTo(180.0);
        System.out.println("Subscriber is now listening to "+topic);
```



```
In singleton constructor
mtor id 307559
Sending data to: http://localhost:8081/ServerSide/ServerDatabaseHandler?s
{"doorid":"1","motorid":0,"clientType":""}1
Subscriber is now listening to 1/motor
```

The Client-Side Subscriber is always running and listening for a publisher to publish out to its topic, this is because the door should always be ready to open. The motor call back moves the motor when the Subscriber calls it

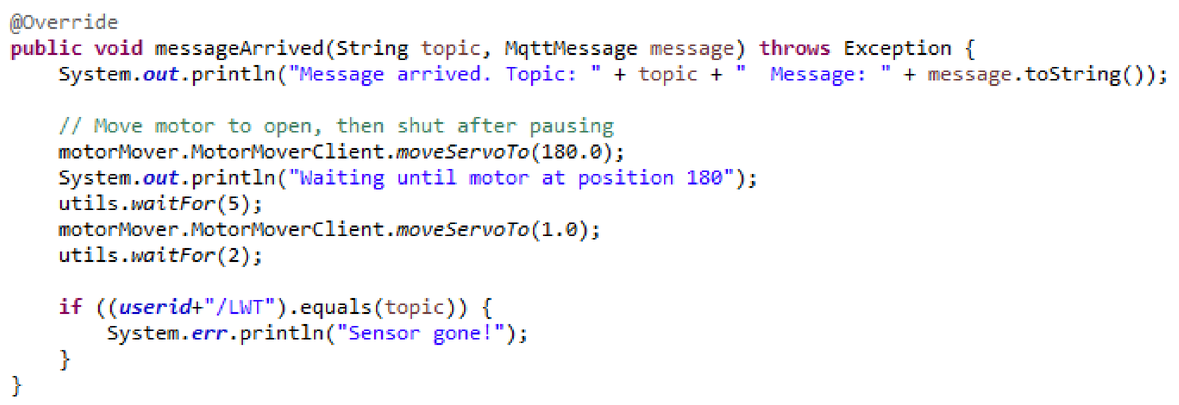


```java
public void start() throws PhidgetException{
    try {
        mqttClient.setCallback(new MotorSubcribercallback());
        mqttClient.connect();

        // Name of topic for subscribing.
        // NB Only works when 3rd party is publishing to this topic
        final String topic = doorid()+"/motor";
        mqttClient.subscribe(topic);
        //PhidgetSensorClients.MotorMoverClient.moveServoTo(180.0);
        System.out.println("Subscriber is now listening to "+topic);


    } catch (MqttException e) {
        e.printStackTrace();
        System.exit(1);
    }
}
public static void main(String... args) throws PhidgetException {
        final Subcriber subscriber = new Subcriber();
        subscriber.start();
}
```

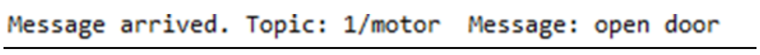
The call the call back method moves the motor and prints out the topic and message that is currently moving the motor for.

The call back method will also move the motor back to the closed pos, in our case this is 1.0

```java
motorMover.MotorMoverClient.moveServoTo(180.0);
```



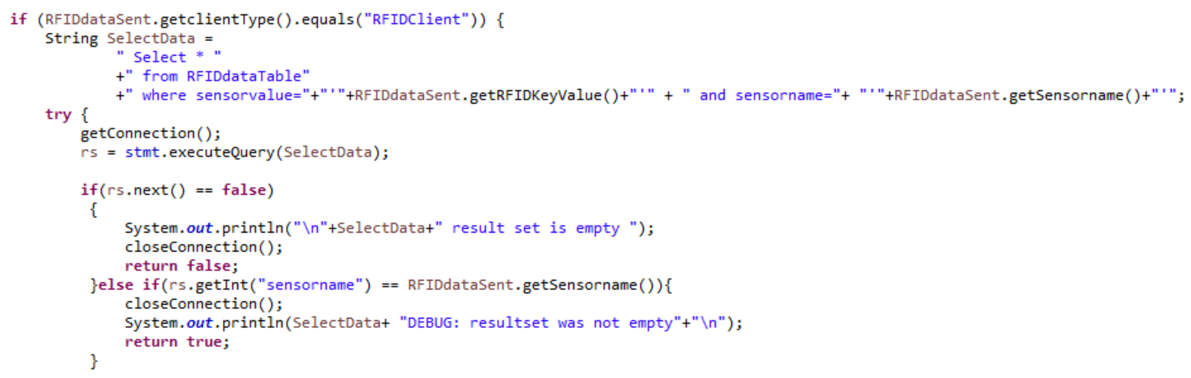
```java
@Override
public void messageArrived(String topic, MqttMessage message) throws Exception {
    System.out.println("Message arrived. Topic: " + topic + "  Message: " + message.toString());

    // Move motor to open, then shut after pausing
    motorMover.MotorMoverClient.moveServoTo(180.0);
    System.out.println("Waiting until motor at position 180");
    utils.waitFor(5);
    motorMover.MotorMoverClient.moveServoTo(1.0);
    utils.waitFor(2);

    if ((userid+"/LWT").equals(topic)) {
        System.err.println("Sensor gone!");
    }
}
```



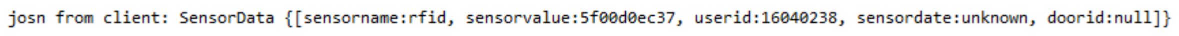
```
Message arrived. Topic: 1/motor  Message: open door
```

Server-side

the server gets json from the client and checks that key exits and is correct for the given door by checking the if the serial number of the scanner is the same one sent as in the DB. If it's not the IsValid method returns false and the server tells the client the key is wrong.



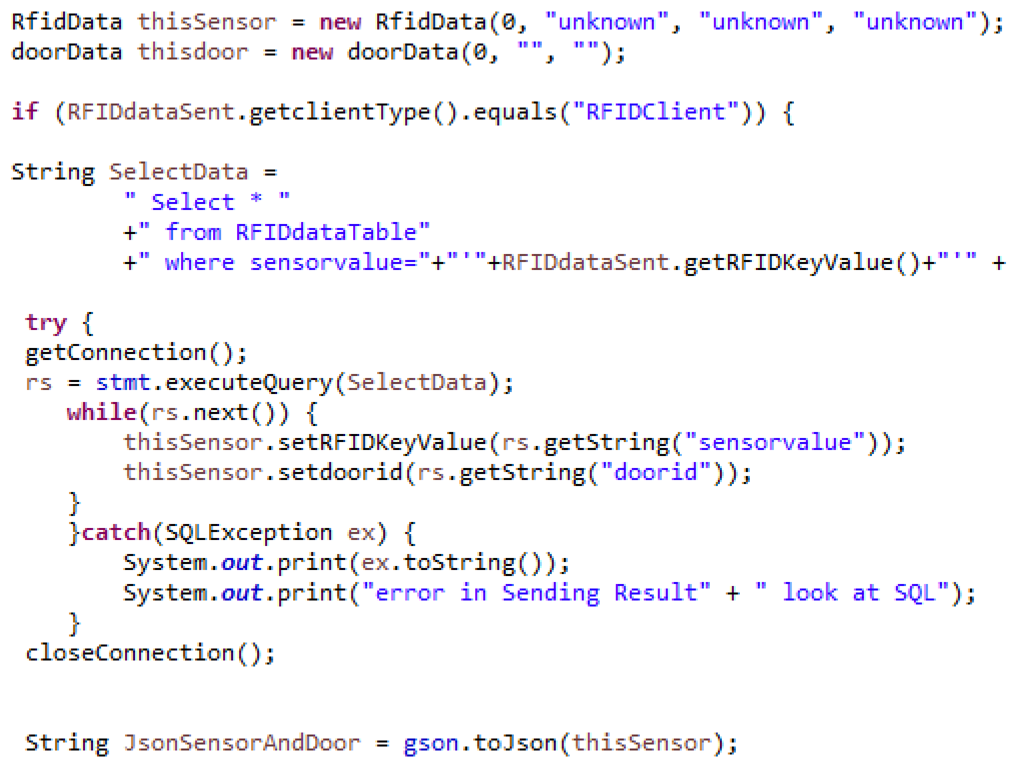
```
if (RFIDdataSent.getclientType().equals("RFIDClient")) {
    String SelectData =
            " Select * "
            +" from RFIDdataTable"
            +" where sensorvalue="+"'"+RFIDdataSent.getRFIDKeyValue()+"'" + " and sensorname="+ "'"+RFIDdataSent.getSensorname()+"'";
    try {
        getConnection();
        rs = stmt.executeQuery(SelectData);

        if(rs.next() == false)
         {
            System.out.println("\n"+SelectData+" result set is empty ");
            closeConnection();
            return false;
         }else if(rs.getInt("sensorname") == RFIDdataSent.getSensorname()){
            closeConnection();
            System.out.println(SelectData+ "DEBUG: resultset was not empty"+"\n");
            return true;
         }
```



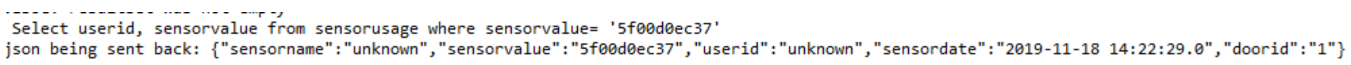
```
josn from client: SensorData {[sensorname:rfid, sensorvalue:5f00d0ec37, userid:16040238, sensordate:unknown, doorid:null]}
```

If the result set is not empty, then the method returns true.

If this method returns true, the subsequent result set is then called so that the reponse form the server can be packaged up into a json string by the send result method. This method updates a new RFIDdata object to be converted to json so that it can sent back to the client

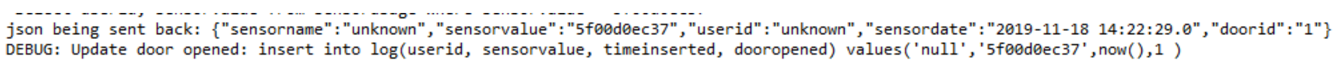


```
RfidData thisSensor = new RfidData(0, "unknown", "unknown", "unknown");
doorData thisdoor = new doorData(0, "", "");

if (RFIDdataSent.getclientType().equals("RFIDClient")) {

String SelectData =
        " Select * "
        +" from RFIDdataTable"
        +" where sensorvalue="+"'"+RFIDdataSent.getRFIDKeyValue()+"'" +

 try {
 getConnection();
 rs = stmt.executeQuery(SelectData);
    while(rs.next()) {
        thisSensor.setRFIDKeyValue(rs.getString("sensorvalue"));
        thisSensor.setdoorid(rs.getString("doorid"));
    }
    }catch(SQLException ex) {
        System.out.print(ex.toString());
        System.out.print("error in Sending Result" + " look at SQL");
    }
 closeConnection();


 String JsonSensorAndDoor = gson.toJson(thisSensor);
```



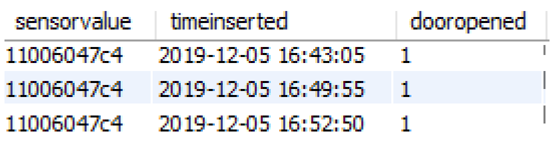
```
 Select userid, sensorvalue from sensorusage where sensorvalue= '5f00d0ec37'
json being sent back: {"sensorname":"unknown","sensorvalue":"5f00d0ec37","userid":"unknown","sensordate":"2019-11-18 14:22:29.0","doorid":"1"}
```

Once this done the server then sends back the new json so the client can open the door. The server also logs in the database if the door has been opened



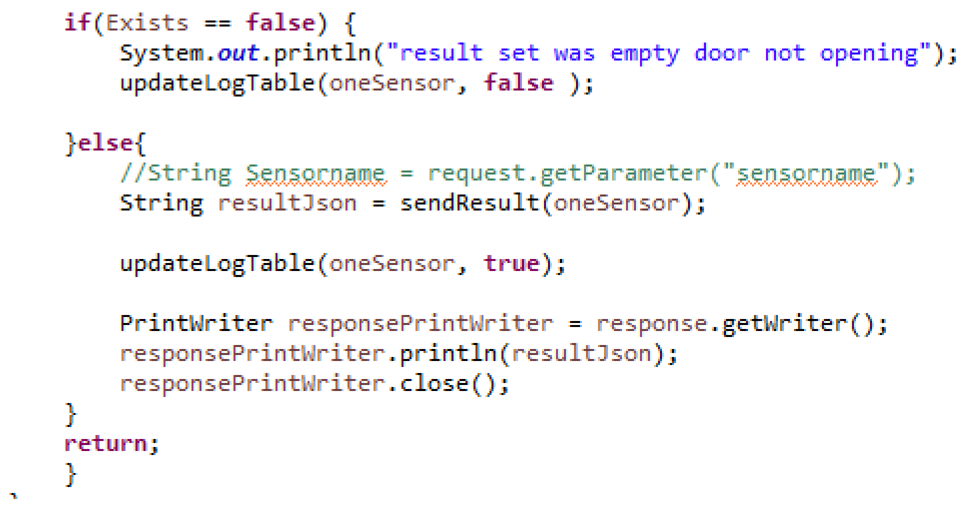
```
json being sent back: {"sensorname":"unknown","sensorvalue":"5f00d0ec37","userid":"unknown","sensordate":"2019-11-18 14:22:29.0","doorid":"1"}
DEBUG: Update door opened: insert into log(userid, sensorvalue, timeinserted, dooropened) values('null','5f00d0ec37',now(),1 )
```

This done by a method in the server that will only update the log table if a key from the server exists in the database.



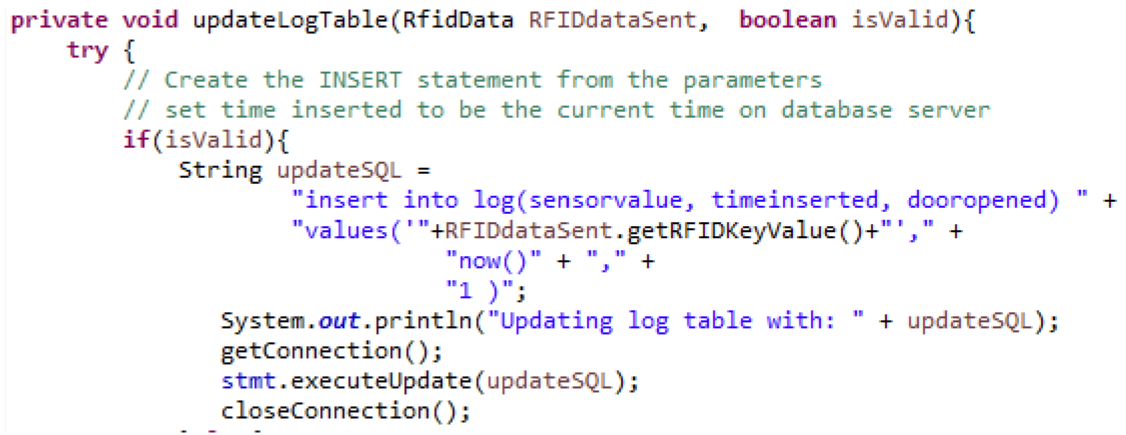
sensorvalue	timeinserted	dooropened
11006047c4	2019-12-05 16:43:05	1
11006047c4	2019-12-05 16:49:55	1
11006047c4	2019-12-05 16:52:50	1

The one in the last column donates If the attempt to open the door was successful or not.



```
    if(Exists == false) {
        System.out.println("result set was empty door not opening");
        updateLogTable(oneSensor, false );

    }else{
        //String Sensorname = request.getParameter("sensorname");
        String resultJson = sendResult(oneSensor);

        updateLogTable(oneSensor, true);

        PrintWriter responsePrintWriter = response.getWriter();
        responsePrintWriter.println(resultJson);
        responsePrintWriter.close();
    }
    return;
    }
```

this method as above takes two arguments and updates dates the table upon the Boolean value of “isValid”.



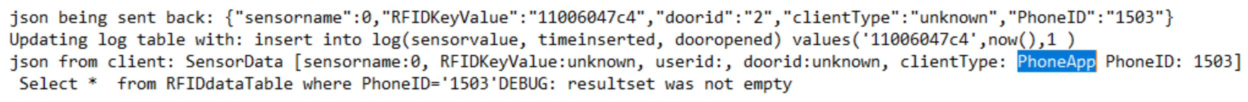
```
private void updateLogTable(RfidData RFIDdataSent,  boolean isValid){
    try {
        // Create the INSERT statement from the parameters
        // set time inserted to be the current time on database server
        if(isValid){
            String updateSQL =
                    "insert into log(sensorvalue, timeinserted, dooropened) " +
                    "values('"+RFIDdataSent.getRFIDKeyValue()+"'," +
                            "now()" + "," +
                            "1 )";
                System.out.println("Updating log table with: " + updateSQL);
                getConnection();
                stmt.executeUpdate(updateSQL);
                closeConnection();
```

<u>Phone app</u>

The IMEI number is in the app and this is sent to the server, the server then gets the data for this phone on create. The server sends back the doorid for the IMEI number so the door can be opened by the phone by pressing the open door button

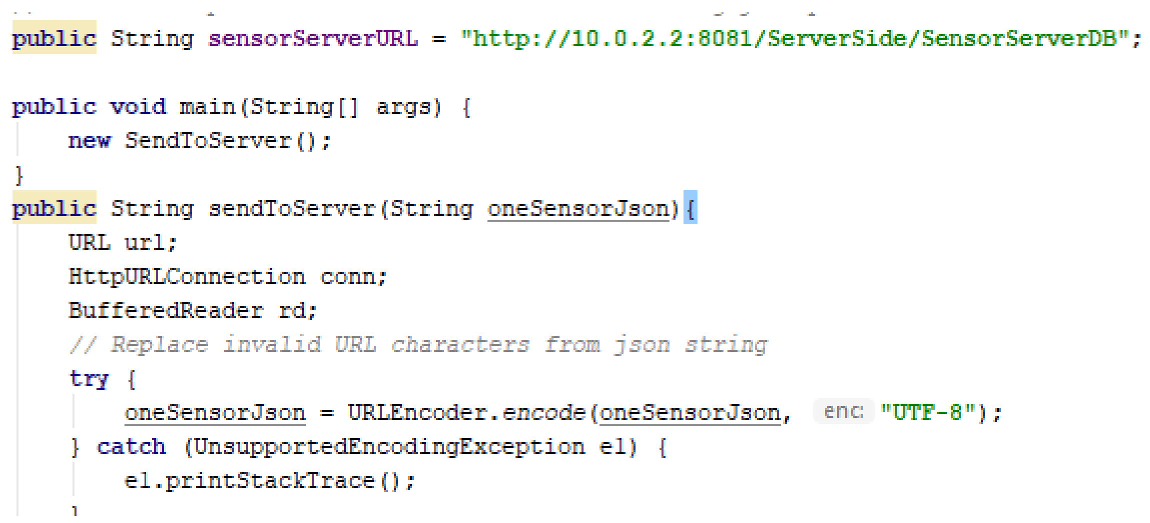
```
{"PhoneID":"1503","RFIDKeyValue":"unknown","clientType":"PhoneApp","doorid":"unknown","sensorname":0}doorid: 2
```

Server print out:



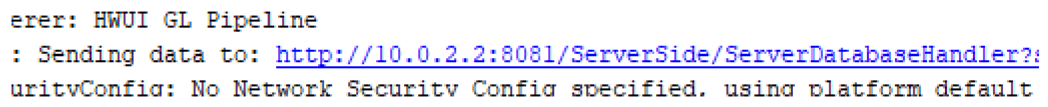
```
json being sent back: {"sensorname":0,"RFIDKeyValue":"11006047c4","doorid":"2","clientType":"unknown","PhoneID":"1503"}
Updating log table with: insert into log(sensorvalue, timeinserted, dooropened) values('11006047c4',now(),1 )
json from client: SensorData [sensorname:0, RFIDKeyValue:unknown, userid:, doorid:unknown, clientType: PhoneApp PhoneID: 1503]
 Select *  from RFIDdataTable where PhoneID='1503'DEBUG: resultset was not empty
```

Sending data the server is done by a separate class to the main class. The Toserver method takes one argument of a string which is converted to json in the main class. The Send to server method encodes the json string using UTF-8 encoding and then uses HTTP to send the json to the server. This done the same as the client side.



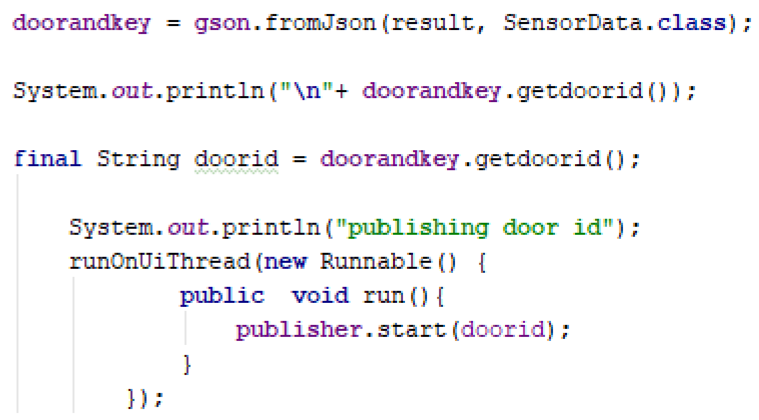
```
public String sensorServerURL = "http://10.0.2.2:8081/ServerSide/SensorServerDB";

public void main(String[] args) {
    new SendToServer();
}
public String sendToServer(String oneSensorJson){
    URL url;
    HttpURLConnection conn;
    BufferedReader rd;
    // Replace invalid URL characters from json string
    try {
        oneSensorJson = URLEncoder.encode(oneSensorJson,  enc: "UTF-8");
    } catch (UnsupportedEncodingException e1) {
        e1.printStackTrace();
```

In the print it shows the full address of the server that is sending the data to.



```
erer: HWUI GL Pipeline
: Sending data to: http://10.0.2.2:8081/ServerSide/ServerDatabaseHandler?
urityConfig: No Network Security Config specified, using platform default
```

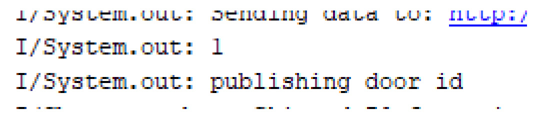
Once the data is sent back from the server after server has validated the IMEI number, the app gets a door id from the data that has been sent in the json



```
doorandkey = gson.fromJson(result, SensorData.class);

System.out.println("\n"+ doorandkey.getdoorid());

final String doorid = doorandkey.getdoorid();

    System.out.println("publishing door id");
    runOnUiThread(new Runnable() {
            public  void run(){
                publisher.start(doorid);
            }
        });
```

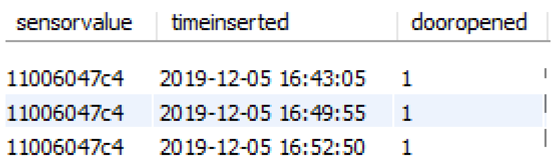
This runs on a separate thread to make sure there is not to much running on the main thread, this is done so that the app holds no other processes on the main thread such as UI updates.

Once the client gets back a json string from the server it publishes the door-id once to subscriber

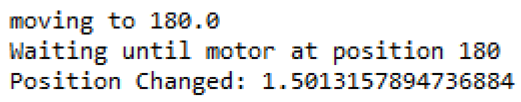


```
I/System.out: Sending data to: http:/
I/System.out: 1
I/System.out: publishing door id
```

The Subscribing door will then open, given its door id. The subscriber will then move the motor to a given position, thus opening the door, it will then move the motor back again to close the door. The attempt is also stored in log table in the DB



sensorvalue	timeinserted	dooropened
11006047c4	2019-12-05 16:43:05	1
11006047c4	2019-12-05 16:49:55	1
11006047c4	2019-12-05 16:52:50	1

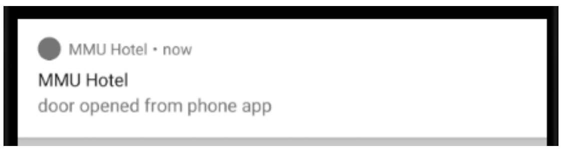


```
moving to 180.0
Waiting until motor at position 180
Position Changed: 1.5013157894736884
```

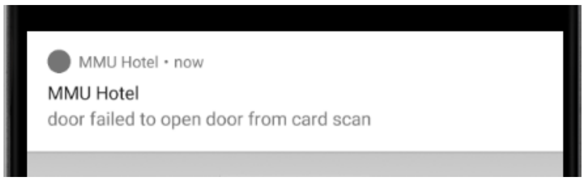
Notifications on phone

The phone gets a notification every time the motor opens, this is sent from the client side when the client gets back a json object from the server or if it gets back a failed message.

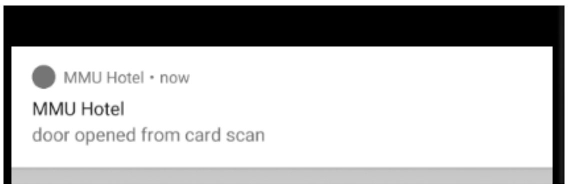
The message changes from the MQTT depending on if its failed door opening or not. Notifications are handled from the notification helper class. It also changes if the door opens on the client side from a card scan



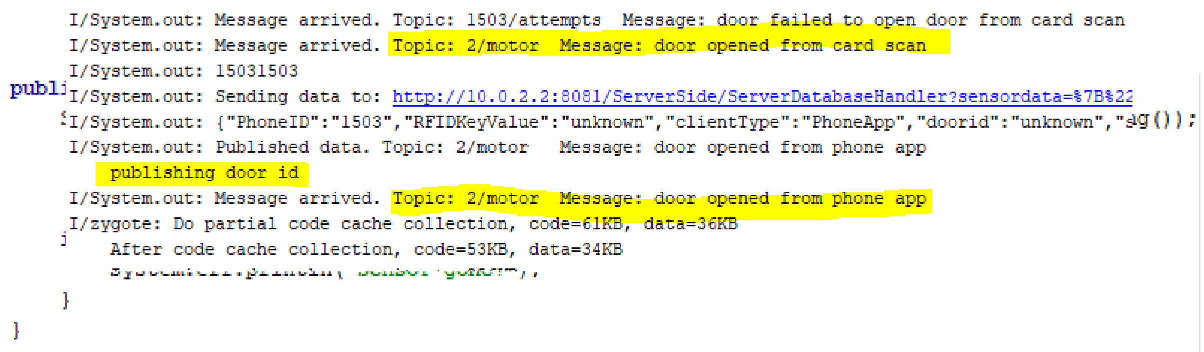
MMU Hotel • now
MMU Hotel
door opened from phone app



MMU Hotel • now
MMU Hotel
door failed to open door from card scan

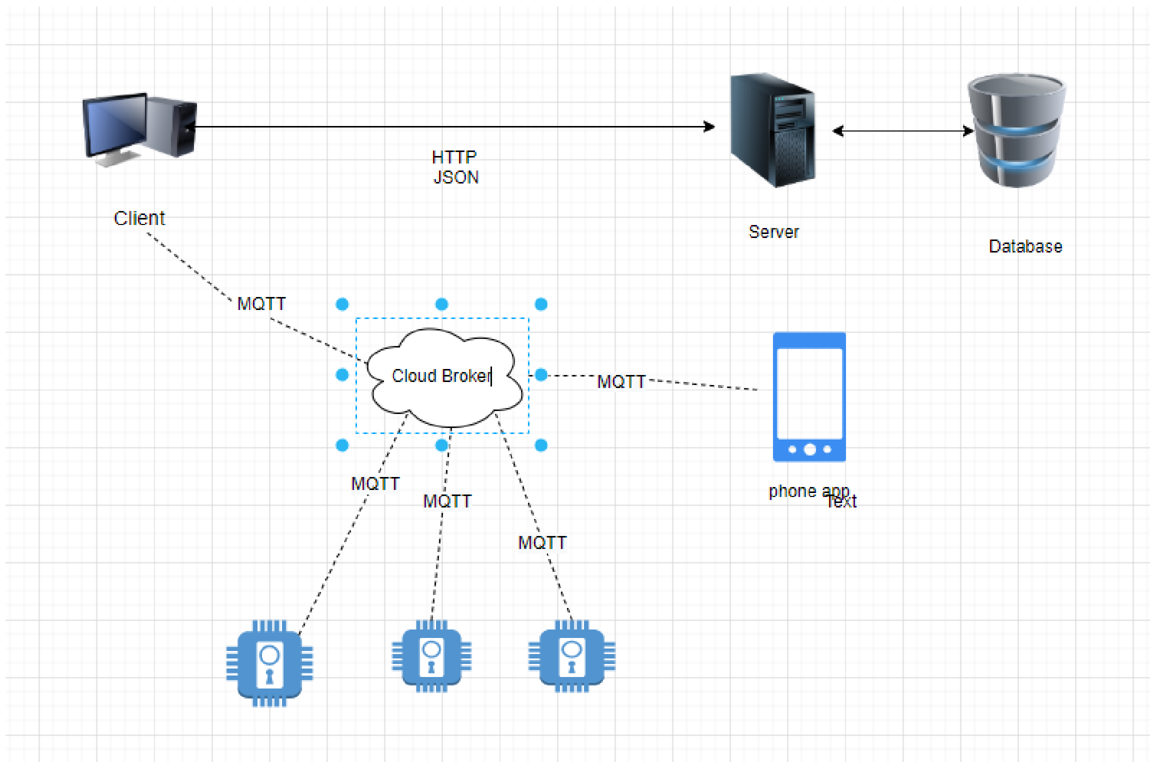


MMU Hotel • now
MMU Hotel
door opened from card scan

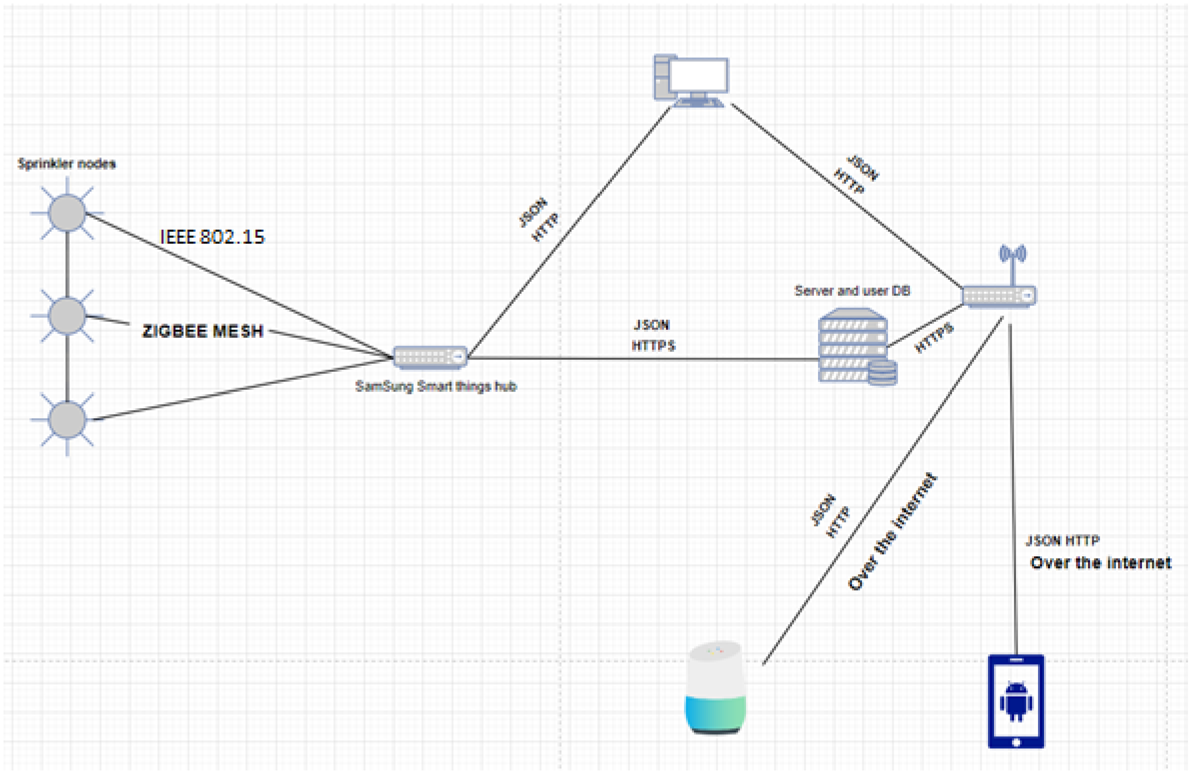


```
   I/System.out: Message arrived. Topic: 1503/attempts  Message: door failed to open door from card scan
   I/System.out: Message arrived. Topic: 2/motor  Message: door opened from card scan
   I/System.out: 15031503
publiI/System.out: Sending data to: http://10.0.2.2:8081/ServerSide/ServerDatabaseHandler?sensordata=%7B%22
   I/System.out: {"PhoneID":"1503","RFIDKeyValue":"unknown","clientType":"PhoneApp","doorid":"unknown","sg());
   I/System.out: Published data. Topic: 2/motor   Message: door opened from phone app
      publishing door id
   I/System.out: Message arrived. Topic: 2/motor  Message: door opened from phone app
   I/zygote: Do partial code cache collection, code=61KB, data=36KB
      After code cache collection, code=53KB, data=34KB

   }
}
```

Overview for part 1 and 2



Sprinkler System Part 3



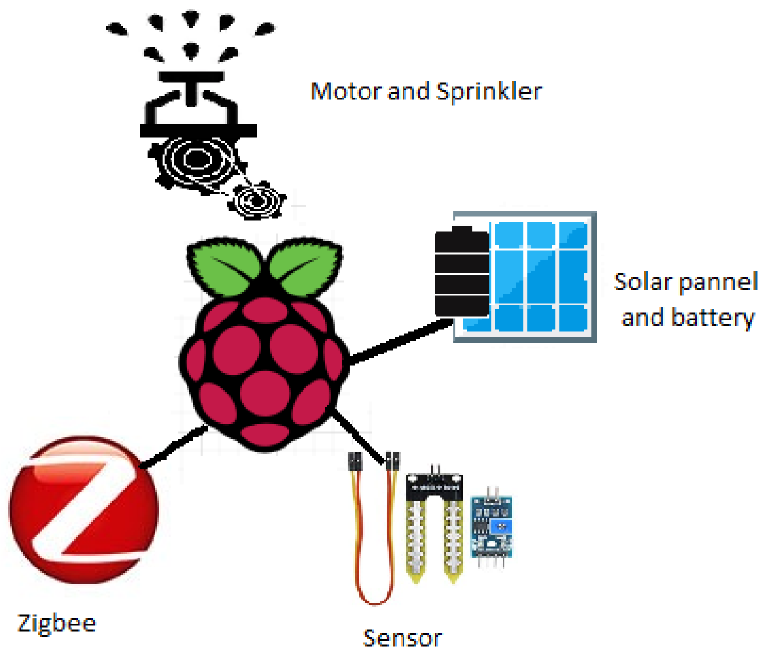
Sprinkler Node

There will be Raspberry pi as the centre of the sprinkler, this will act as the middle man between the soil moisture sensor, the Zigbee transceiver and the motor that controls the sprinkler. The sprinkler will be able to speed up and slow down and change direction, depending on the given instruction received from the user. The sprinkler will be able to start or stop watering plants depending on the time of day and the moisture level in the soil. This will be implemented via python installed and written onto the raspberry pi through a flash card, along with the Zigbee code supplied by Zigbee.

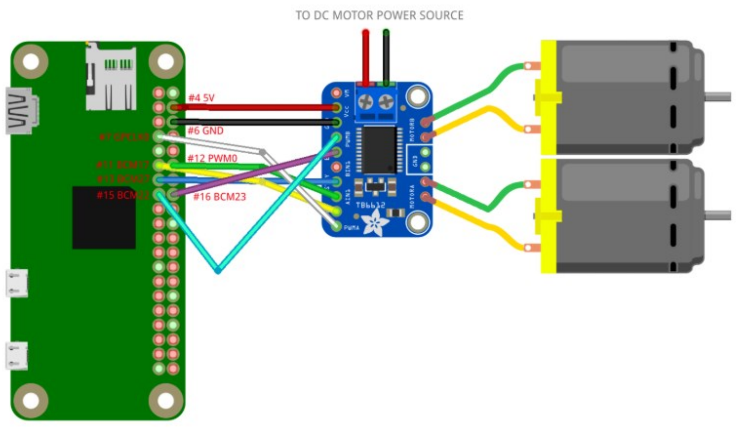
Zigbee is a low power and cost-effective way of communicating over a mesh network, this means that each sprinkler node will also be able to speak to the other nodes, sending information so they can all act as a unit, thus they will always do the same thing meaning they should never be out of sync. The Zigbee node will be connected to a "Samsung smart things hub" which then connects to a Wi-Fi router to transfer JSON data over the internet to a server to process the JSON data. The server will then send JSON data back to the router, which would be identified via its IP. The smart things hub will then take the JSON data and pass it to the Zigbee network, the Raspberry Pi's will then interpret the JSON data

The Raspberry Pi in each node will use a KeeYees 5 Pcs High Sensitivity Soil Moisture Sensor Module to monitor soil humidity, then use this information to determine when to water the plants. The Raspberry Pi's would also us the internal clock, giving the sprinklers a smarter element. For example, if the ground is dry and it's summer time and it's the middle of the day the sprinkler system should water the plants.

Brief overview of the node and motor circuit below:



Adafruit TB6612 1.2A DC/Stepper Motor Driver Breakout Board [ADA2448]



<u>How Sprinklers Work Over ZIGBEE</u>

Each of the sprinklers' motors will connect to a server through the mesh network the ZigBee's create. The server will be supplied by the company which will broadcast out to the given Sprinkler system. Depending on the instructions given by the user from a phone or computer the Sprinklers will be able to move, spray more or less water, or just be told to stop spraying.

Upon first time set up the user will be asked to connect to each Sprinklers motor that will have a Zigbee transceiver connecting to a Samsung smart things hub allowing them to connect to the local LAN the same as a wireless network router or a Wi-Fi repeater.

The Sprinklers would connect to their internet through the Samsung smart things up and will communicate to the router and each other. Each sprinkler will have its own IP address. This gives the system more safety makes the system more scalable in the future.

ZigBee is a good choice as it typically used in low data rate applications that require long battery life and secure networking. This allows to run the sprinklers for longer and gives us a more secure network. As we are using a greener approach using ZigBee will not drain the unit's battery when the battery is not being charged or does not have a lot of charge.

<u>Power Supply For Sprinklers, Motorized Solenoid Valve</u>

the system will use power supplies that run on solar energy panels on the top of each node and powering everything. This will work by having small battery packs connected to the unit (the sprinkler, valve, motor and transceiver). Each Sprinkler will have one solar panel.

The battery will charge whenever it can. It will allow the user to change the battery themselves, we will need to do this as the batteries will eventually break. We could just use normal rechargeable AAs. By using a solar panel not only is the system greener but it's also easier to recharge the batteries as It's done automatically by the solar panel.

It will use a water barrel that fill up with rain water from the gutter system connected to the house. This will work mostly by gravity but installing a pump to make sure each sprinkler gets water efficiently. We will also allow users to fill up this barrel with their own water if they need to.

A motorized solenoid valve for water supply will be attached from the water barrel which can be remotely controlled to shut off or open the water through a node within Sprinklers. the Solenoid valve will be controllable via google home interface, computer or phone app.

google home interface

for the google home setup a user will be able to download an app from the app store on their phone, the app that allow the user to pick their "house" giving them the same ID, so the server can only send to their sprinklers.

 the google home will give them the ability to ask things like "speed up sprinkler one", this voice data will then be sent to a server.

how to data will be sent and processed

the Json data from the google home or computer will be then sent over the internet using HTTPS to as server. The data from the google home will need to be processed in a server using the open source code for the google home operating system.

Security

We will need to make sure all user data coming in is coming from valid users to make sure no one is sending malicious data. We can do this by using attribute-based encryption. We also need to understand which user is sending us data and who they are. We could do this by getting a token from a user. This can be generated from there house code or their location.

When data is being sent over the internet HTTPS should be used to Encrypt the data along with end to end Encryption to the server can Decrypt JSON that has been sent. This does mean the company will have to pay for an SSL certificate.

The easiest way to validate each user as mentioned will be to have a username and password account system, this system will add a token to all traffic to the servers know which user is sending data, so the server sends instructions back to the right client. Computer software and set up software for controlling sprinklers will be available for users to download giving them access to controls to change the speed of all or just one sprinkler, the amount of water for one or all of sprinklers, through a GUI. This software should only send data over the internet that in encrypted. All Id's and relevant user information such as location data should hashed and salted in the database.

We will need to make sure when are receiving data and sending data to the internet that we are both not sending any data that could be used by hackers to gain accesses to databases, this means not sending clear text JSON and not including database column names. To encrypt the traffic, we could use and end to end encryption. All ZigBee networks use are secured by 128-bit symmetric encryption keys, this will mean we don't have to use our encryption on the mesh network.

Pricing

BACOENG 1" DN25 Brass BSP 2 Port Motorized Ball Valve – Amazon £36 each

Google home – google store - £49

ESP-WROOM-S2 Wi-Fi transceiver – Mouser electronics - £2.60 each

TP-Link AC1200 Wireless Dual Band VDSL/ADSL Modem Router for Phone Line Connections – amazon – £64

DOKIO 100w Solar Panel for 12v Battery kit – amazon – 77.77

Silverline 868552 Impulse Garden Sprinkler – amazon -£8.54

Samsung SmartThings hub amazon- 24.24

RaspBee premium - ZigBee addon for Raspberry Pi with Firmware -30.39

KeeYees 5 Pcs High Sensitivity Soil Moisture Sensor Module with Female to Female Jump Wires, Sensor Module Watering System Manager for raspberry pi – 6.99