

ZAIN AL-HASSAN 17106353

MOBILE APPLICATION DEVELOPMENT

INTERNET OF THINGS

NOTICE: SQL workbench code is below with valid pin number to run application

TASK1:

Server up and running:

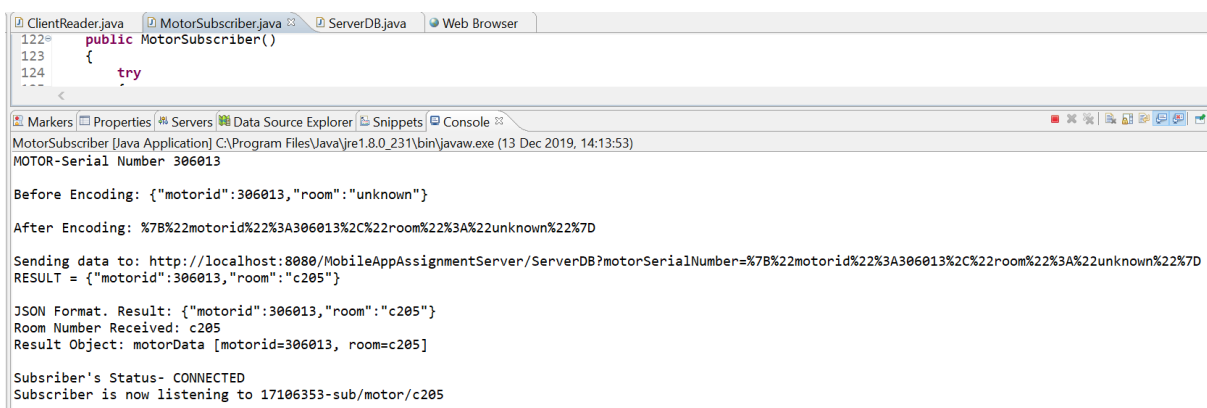
```
Server is up and running

Upload sensor data with http://localhost:8080/MobileAppAssignmentServer/ServerDB?rfidSerialNumber=somerfidvalue
View last sensor reading at http://localhost:8080/MobileAppAssignmentServer/ServerDB?getdata=true

DEBUG: lastRfid JSON: {"tagid":"unknown","readerid":0,"valid":false,"room":"unknown"}
DEBUG: lastRfid TEXT: rfidData [tagid=unknown, readerid=0, valid=false, room=unknown]
```

Picture above shows the console messages displayed when the server is running. The main purpose of the serverdatabase.java class is to have the ability perform send and get requests to get the valid room number, hence, making all the variables used dynamic. This has been done by linking the database to the server by making a connection through MYSQL workbench.

Running motorsubscriber :



```
122 public MotorSubscriber()
123 {
124     try
125     {
126         // ...
127     }
128 }

Markers | Properties | Servers | Data Source Explorer | Snippets | Console
MotorSubscriber [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (13 Dec 2019, 14:13:53)
MOTOR-Serial Number 306013

Before Encoding: {"motorid":306013,"room":"unknown"}

After Encoding: %7B%22motorid%22%3A306013%2C%22room%22%3A%22unknown%22%7D

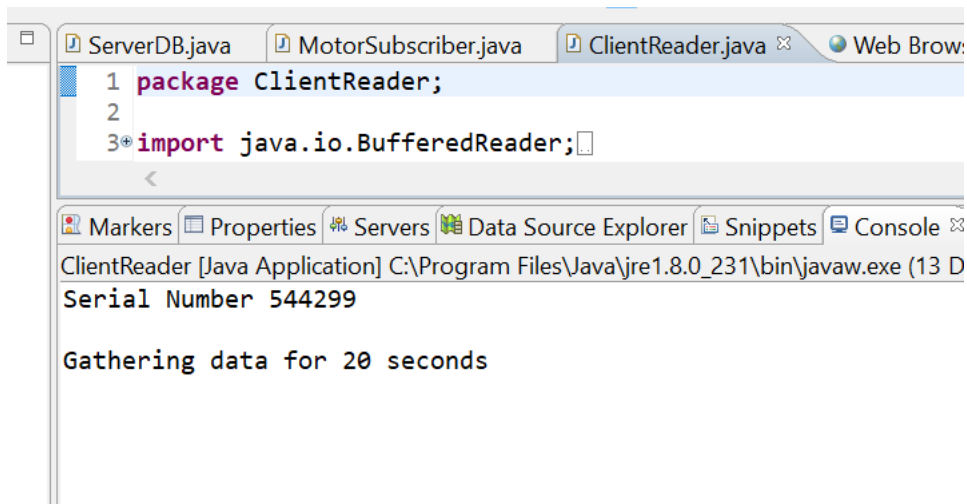
Sending data to: http://localhost:8080/MobileAppAssignmentServer/ServerDB?motorSerialNumber=%7B%22motorid%22%3A306013%2C%22room%22%3A%22unknown%22%7D
RESULT = {"motorid":306013,"room":"c205"}

JSON Format. Result: {"motorid":306013,"room":"c205"}
Room Number Received: c205
Result Object: motorData [motorid=306013, room=c205]

Subscriber's Status- CONNECTED
Subscriber is now listening to 17106353-sub/motor/c205
```

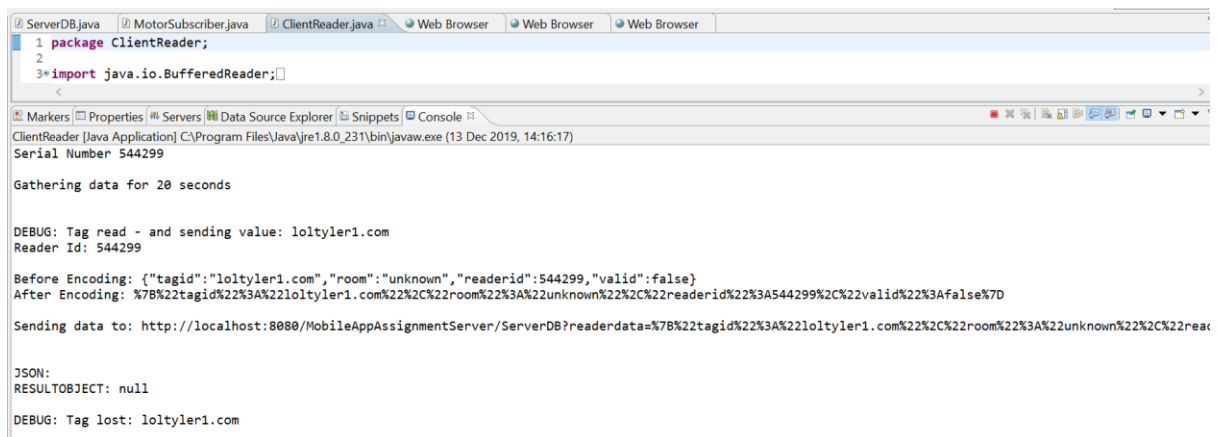
Upon running the motor subscriber, serial number as shown above is sent to the server to verify if the motor id shown in the console has any room names attached to them. If so, room = "unknown" is replaced with the room received by the server. The whole process of requesting and sending data is done using JSON Format.

Running Client Reader:



Serial number has been used in the same way as motor subscriber again to verify the tags. This allows to retrieve relevant information back from the sever after sending the rfid serial number to the method called “sendToServer”. The data retrieved in the client as a response is not just used for validation but is also being published to the relevant publishers such as Publish Notifications.

Wrong tag:



If a wrong tag is scanned, above messages are displayed. The whole process of validation is also terminated, hence, not allowing any queries to be successfully executed such as `checkForValid`.

Validated tag:

```
ServerDB.java MotorSubscriber.java ClientReader.java Web Browser Web Browser Web Browser Web Browser
1 package ClientReader;
2
3 import java.io.BufferedReader;

Markers Properties Servers Data Source Explorer Snippets Console
ClientReader [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (13 Dec 2019, 14:17:08)
Serial Number 544299

Gathering data for 20 seconds

DEBUG: Tag read - and sending value: 5f00dc0863
Reader Id: 544299

Before Encoding: {"tagid":"5f00dc0863","room":"unknown","readerid":544299,"valid":false}
After Encoding: %7B%22tagid%22%3A%225f00dc0863%22%2C%22room%22%3A%22unknown%22%2C%22readerid%22%3A544299%2C%22valid%22%3Afalse%7D

Sending data to: http://localhost:8080/MobileAppAssignmentServer/ServerDB?readerdata=%7B%22tagid%22%3A%225f00dc0863%22%2C%22room%22%3A%22unknown%22%2C%22
The result = {"tagid":"5f00dc0863","readerid":544299,"valid":true,"room":"c205"}

JSON: {"tagid":"5f00dc0863","readerid":544299,"valid":true,"room":"c205"}
RESULTOBJECT: rfidData [tagid=5f00dc0863, room=c205, readerid=544299, valid=true]

Published Data. Topic: 17106353/rfid Message: 5f00dc0863

Publishing message : c205 to topic: 17106353-sub/motor/c205
Published data. Topic: 17106353-sub/motor/c205 Message: c205TEST DOOR NUMBER

Tag Read: 5f00dc0863, ACCESS GRANTED, Interacting with the DOOR: c205
DEBUG: Tag lost: 5f00dc0863

Closed RFID Reader
```

The console above shows all the messages that are displayed once a tag gets validated after going through the process of validation. All data retrieved is at least shown once in the JSON format. Furthermore, correct methods publish the data which then allows the user to open the lock.

```
ServerDB.java MotorSubscriber.java ClientReader.java Web Browser Web Browser Web Browser Web Browser
1 package ClientReader;
2
3 import java.io.BufferedReader;

Markers Properties Servers Data Source Explorer Snippets Console
MotorSubscriber [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (13 Dec 2019, 14:17:01)
MOTOR-Serial Number 306013

Before Encoding: {"motorid":306013,"room":"unknown"}

After Encoding: %7B%22motorid%22%3A306013%2C%22room%22%3A%22unknown%22%7D

Sending data to: http://localhost:8080/MobileAppAssignmentServer/ServerDB?motorSerialNumber=%7B%22motorid%22%3A306013%2C%22room%22
RESULT = {"motorid":306013,"room":"c205"}

JSON Format. Result: {"motorid":306013,"room":"c205"}
Room Number Received: c205
Result Object: motorData [motorid=306013, room=c205]

Subscriber's Status- CONNECTED
Subscriber is now listening to 17106353-sub/motor/c205
Message arrived. Topic: 17106353-sub/motor/c205Message: c205TEST DOOR NUMBER

In singleton constructor
Constructing MotorMover
In singleton constructor
moving to 0.0

Motor initially positioned at: 0
moving to 145.0

In singleton constructor
moving to 0.0
```

This console displays the coordinates of the motor its been moved to, when the whole process off validation is successfully executed. The motor is also coded in a way where it closes the lock automatically after few seconds.

Along with the debug messages, the SQL database will now be updated and it will look like the following:

```
47 • SELECT * FROM attempts;
```

tagid	readerid	date	valid
5f00dc0863	544299	2019-12-13 15:45:42	1
4d00474622	544299	2019-12-13 15:45:47	0

0 is for invalid tags, 1 showing valid tag.

```
45  
46 • SELECT * FROM validtags;
```

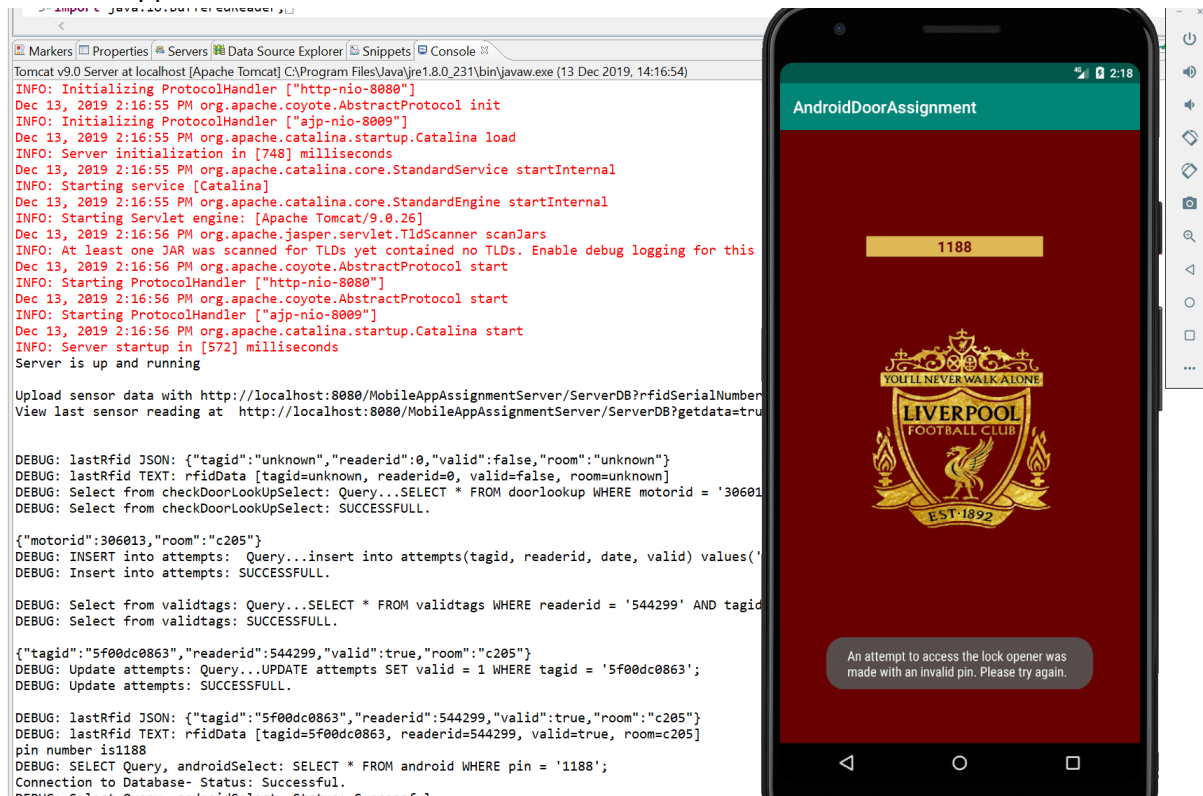
tagid	readerid	room
5f00dc0863	544299	c205
4d004a5cf6	387997	E127

Above are the only valid tags that will move the certain locks within the system. Their pin numbers are as followed.

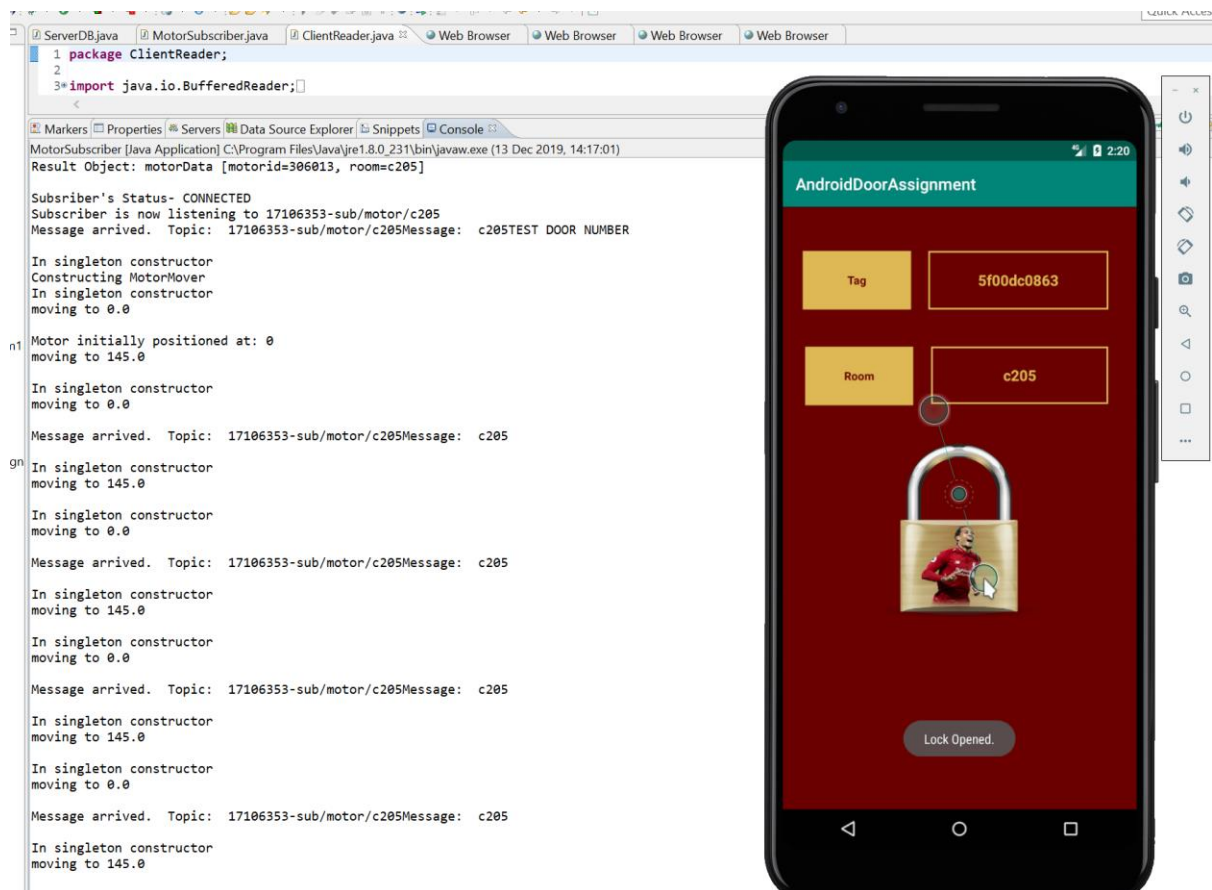
```
49 • SELECT * FROM android;
```

androidid	room	tagid	pin
1	c205	5f00dc0863	1122
2	F127	4d004a5cf6	1234

Android Application:



Any number other than the correct one provided will give the pop up notification above. Also, this is achieved using the Http Get request method which confirms the pin entered is not found in the database which is shown to user with an altered message.



When the button is pressed. A message is published to the subscriber which listens for the unique client id + motor. Once the message is accepted by the publisher, it requests the motor mover to perform its action to move the lock.

Task 3:

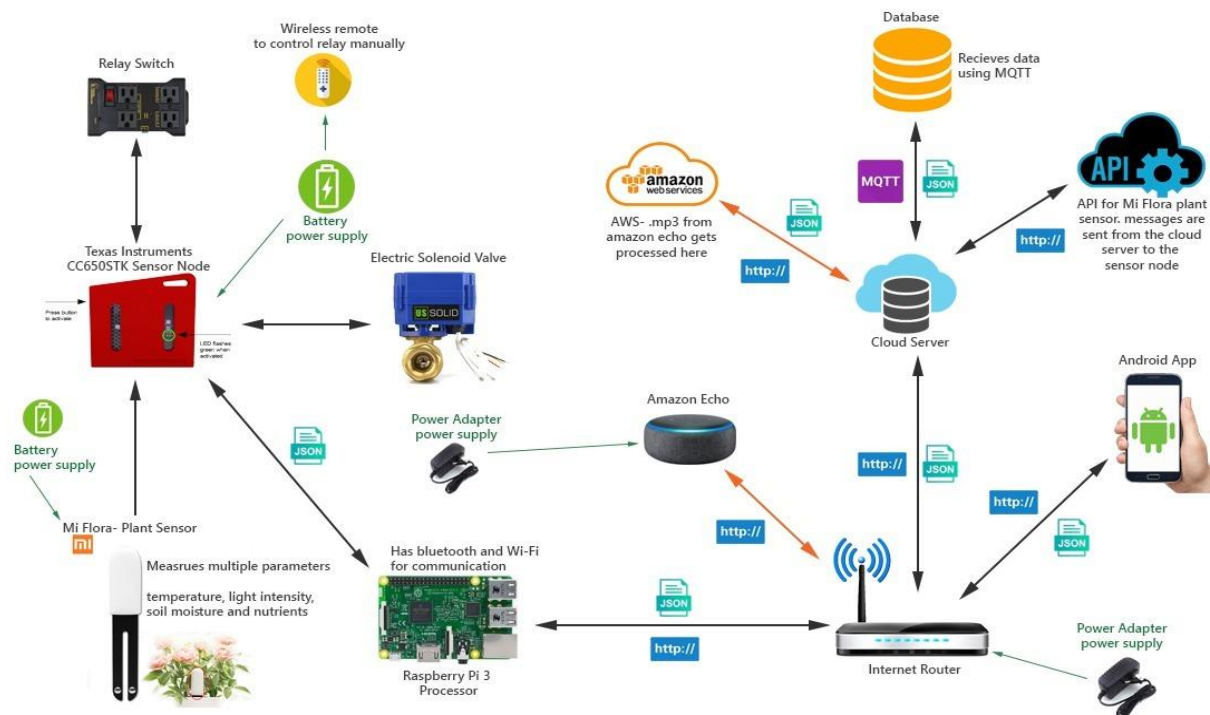


Diagram above shows how the smart sprinkler system will function, the communication protocols used such as the HTTP protocol and MQTT in data transfer and power supplies are also shown using green arrows. Following is the list of the hardware that are required in order for this system to function.

- Relay Switch (Actuator) which controls the electric solenoid valve. It will be connected to the sensor node which will be responsible for turning it on and off.
- Texas Instruments CC605STK Sensor Node (battery powered). It communicates using low power consumption Bluetooth. This will receive the plant sensors data as electrical signals and use the API from the cloud server to convert it into JSON string.
- Motorised Solenoid Valve- Controls the water supply. Switched in and off by the relay.
- Mi Flora(plant sensor) which uses Bluetooth to do a simple Bluetooth connection to the sensor node. It measures/detects 4 things(Light, temperature, Soil Moisture, nutrients). Free API available
- Raspberry PI 3 Processor- uses Bluetooth and Wi-Fi to connect to devices
- Internet Router gives access to all connected devices
- Amazon Echo- Makes this system accessible using speech commands.
- Wireless remote- allows the relay to be manually controlled in case the sensor node is unable to gain internet connection or when the plant sensor malfunctions.

This system functions when the plant sensor (Mi Flora) inputs 4 parameters being the temperature in Celsius, light intensity in LUX, soil nutrients in pH and soil moisture in %.

These parameters are outputted to the sensor node (Texas instrument CC605STK) as electrical signals using Bluetooth. This sensor node and the plant sensor both use Bluetooth to transfer data. Sensor node uses the API available on the cloud server using the processor raspberry pi 3 which is connected to the internet router in order to convert the electrical signals into Json string. The reason why this processor was chosen is the fact that it can communicate using both Bluetooth and Wi-Fi. The sensor node then sends the json string that contains plant sensor values as a Json string to the processor. The processor is responsible for receiving this Json string and sending to the cloud sever using the internet router for it to be stored in the SQL database. The SQL database receives this data using MQTT. The database also have set values to which plant sensors values are compared and if any of them is higher than the set values; The database publishes a message using MQTT to the sensor node, upon receiving this data the sensor node turns the relay switch off, turning the solenoid valve off.

In order to make this system more effective, I will ensure that the plant sensor is not continuously measuring the parameters, rather it uses logical intervals to measure this parameters to conserve battery.

Android Application will then receive JSON data of current valve condition from the database to processor via Wi-Fi. The way it will be displayed is in a table of which sensor has been activated. It will receive the following information: location, duration and details of plant monitor.

Android Application/Amazon Echo:

Amazon Echo works via voice activation, as the user will have to say (Alexa) to activate device. Alexa uses its own kit called (Alexa skill kit). When Alexa responds to users orders it will perform some programming logic, first the recording is sent as an mp3 file to amazon cloud using internet router. Amazon cloud server receives the mp3 file and its ASR (Automated Speech Recognition) converts the speech to text data.

Furthermore, the text now is sent as a JSON format which will then be sent to the cloud server where it is stored in the database. Processor Upon receiving data sends Boolean(true/false) to the sensor node which either turns the Relay Switch on/off.

Besides, amazon echo and android app both require an average speed internet connection to switch the relay on/off. In case of failure or slow responses, the use of a remote control has been implemented to manually control the Relay Switch.

Pricing to build the system :

<u>ITEM</u>	<u>PRICE</u>
-------------	--------------

OLLIVAN Plant Monitor, Original Xiaomi Soil Moisture Meter Tester Nutrient/Temperature/Sunlight/Moisture Sensor Detector for Flower Plant Care - Wireless Bluetooth Plant Sensor	£24.99
Raspberry Pi 4 Modell B 4GB ARM-Cortex-A72 4x 1,50GHz, 4GB RAM, WLAN-ac, Bluetooth 5, LAN, 4x USB, 2x Micro-HDMI	£56.50
Jun_Electronic 5V 3A USB C/Type-C Power Supply Adapter with ON/OFF switch for Raspberry Pi 4...	£6.99
Echo Dot (3rd Gen) - Smart speaker with Alexa - Heather Grey Fabric	£24.99
200 series DIN-Rail 433MHz FM wireless switching system(RELAY SWITCH)	£29.78
WiFi Internet Network Router Wireless	£16.14
1/2" Stainless Steel Electric Solenoid Valve 110V AC Normally Closed VITON	£39.99
TOTAL	199.38

Security side:

With each system comes the thought of security and privacy aspects, here are some thoughts and research I made regarding the security and privacy aspects:

- Using the relay is insurance if Wi-Fi communication gets lost, there will still be a method to control the system manually.
- Since am using a Wi-Fi router I will have to ensure its password protected to avoid exterior users.
- The android phone has to be password protected so no one can get access to the phone and control the relay switch.
- Used Wi-Fi and Bluetooth but not Zigbee because protocols can easily be attacked by unwanted users.
- Amazon Echo is advantageous simply because it is encrypted.

COULD DOS:

- Secure messaging is to be applied to the SQL database (MQTT)
- Use HTTPS to secure and avoid any data being sent to get lost, however HTTPS is slower and will increase the budget therefore I will stick to http

REFERENCES:

https://www.amazon.co.uk/OLLIVAN-Original-Moisture-Nutrient-Temperature/dp/B01LXOJSWA/ref=sr_1_2?keywords=mi+flora+plant+sensor&qid=1576076311&s=outdoors&sr=1-2

https://www.amazon.co.uk/Raspberry-Pi-ARM-Cortex-A72-Bluetooth-Micro-HDMI/dp/B07TC2BK1X/ref=sr_1_2?keywords=raspberry+pi+processor&qid=1576076392&s=outdoors&sr=8-2

https://www.amazon.co.uk/Echo-Dot-3rd-Gen-speaker/dp/B07PDHSQCJ/ref=sr_1_7?keywords=amazon+echo&qid=1576076594&sr=8-7

https://www.ebay.com/itm/WiFi-Internet-Network-Router-Wireless/223736103884?trkparms=ispr%3D1&hash=item3417b603cc:m:mW-Kh-Df2DmITU_aRQPRzlg&enc=AQAEAAACQBPxNw%2BVj6nta7CKEs3N0qV9bILWDKNxGg52xAh0wxncaJFr8vq9sJhRP1HvQiWiMGUTrvAM2hfOag1q1wIE5ss3o3OBSN7VYrZLExgHTCTY3GHp%2FkLqTs8W1gydcfzq%2FCR3g7SWx8zN3nffg%2FfZyoRGSDDCQ6IJAtZGFp9re9jqpT54GSuibwyc%2FEwdVucWcRcdm3pUmBiR1GgCw%2FPX9w99QSdz3ZzVXoXQXMgu6t51DXayuNZBYvBOJAE8hoA2OMltEalE67O26yKz28P2OaCgWckPZg5e3tj0X1kthZ0OJqklrr3zYKq7vaqDyHAqyBTU9vliJgcrpStJg4AqfwvSdrFq1dazG3Azg21Ne8Kt7W9U1hu69u%2Bxr3KJ67wxDquHK%2BACOi3kn7STFoemIJUTy6a7bscZDUqycZKysiRkSOlgviHxGe8LYXLI%2FXCUMrlBUlzHqGJvcVkA4mrD1%2Bow5sgDajkiXXt6ctgRyZmMFR76YyDIUo1dejNMSIsGsJbOuUPNTNPP1eyag7CA3VGT8mtbazBbmdAUIcY9%2BtfHRYkiBTauTYXwAFvi3bW0FJeGsLjq0TsCXv3Xek0WA2I1kRdwZbH1m64qxA2k4A%2Fz2hgpF3a14QIU6Cd4gAeswFnIRYxUVJ4EPBC8BtMTIpJ2nFDuoWHdugrh3w7VNmH2OK3FMTlqBSsZep8N%2FOfOdCG%2B3ONOC5S4UfuEdGwsNRyL2tVgln2JRJG0uclGRtNGAkIjzH6eTryANN69FNOYNA%3D%3D&checksum=2237361038843a7a65879c3941f68fbd7fbfc0a2177&enc=AQAEAAACQBPxNw%2BVj6nta7CKEs3N0qV9bILWDKNxGg52xAh0wxncaJFr8vq9sJhRP1HvQiWiMGUTrvAM2hfOag1q1wIE5ss3o3OBSN7VYrZLExgHTCTY3GHp%2FkLqTs8W1gydcfzq%2FCR3g7SWx8zN3nffg%2FfZyoRGSDDCQ6IJAtZGFp9re9jqpT54GSuibwyc%2FEwdVucWcRcdm3pUmBiR1GgCw%2FPX9w99QSdz3ZzVXoXQXMgu6t51DXayuNZBYvBOJAE8hoA2OMltEalE67O26yKz28P2OaCgWckPZg5e3tj0X1kthZ0OJqklrr3zYKq7vaqDyHAqyBTU9vliJgcrpStJg4AqfwvSdrFq1dazG3Azg21Ne8Kt7W9U1hu69u%2Bxr3KJ67wxDquHK%2BACOi3kn7STFoemIJUTy6a7bscZDUqycZKysiRkSOlgviHxGe8LYXLI%2FXCUMrlBUlzHqGJvcVkA4mrD1%2Bow5sgDajkiXXt6ctgRyZmMFR76YyDIUo1dejNMSIsGsJbOuUPNTNPP1eyag7CA3VGT8mtbazBbmdAUIcY9%2BtfHRYkiBTauTYXwAFvi3bW0FJeGsLjq0TsCXv3Xek0WA2I1kRdwZbH1m64qxA2k4A%2Fz2hgpF3a14QIU6Cd4gAeswFnIRYxUVJ4EPBC8BtMTIpJ2nFDuoWHdugrh3w7VNmH2OK3FMTlqBSsZep8N%2FOfOdCG%2B3ONOC5S4UfuEdGwsNRyL2tVgln2JRJG0uclGRtNGAkIjzH6eTryANN69FNOYNA%3D%3D&checksum=2237361038843a7a65879c3941f68fbd7fbfc0a2177

Appendix: SQL WORKBENCH CODE :

Drop TABLE IF EXISTS android;

Drop TABLE IF EXISTS doorlookup;

Drop TABLE IF EXISTS attempts;

Drop TABLE IF EXISTS validtags;

CREATE TABLE validtags

```
(  
    tagid VARCHAR(50),  
    readerid VARCHAR(50),  
    room VARCHAR(50)  
);
```

CREATE TABLE attempts

```
(  
    tagid VARCHAR(50),  
    readerid VARCHAR(50),  
    date TIMESTAMP,  
    valid BOOLEAN  
);
```

CREATE TABLE doorlookup

```
(  
    motorid VARCHAR(50),  
    room VARCHAR(50)  
);
```

CREATE TABLE android

```
(  
    androidid INT(11) NOT NULL AUTO_INCREMENT,  
    room VARCHAR(50),  
    tagid VARCHAR(50),  
    pin INT(4),
```

PRIMARY KEY(androidid)

);

INSERT INTO validtags VALUES('5f00dc0863', '544299', 'c205');

INSERT INTO validtags VALUES('4d004a5cf6', '387997', 'E127');

INSERT INTO doorlookup VALUES('306013', 'c205');

INSERT INTO doorlookup VALUES ('306613', 'E127');

INSERT INTO android (room, tagid, pin) VALUES("c205", "5f00dc0863", "1122");

INSERT INTO android(room,tagid,pin) VALUES("E127", "4d004a5cf6", "1234");

SELECT * FROM validtags;

SELECT * FROM attempts;

SELECT * FROM doorlookup;

SELECT * FROM android;