Build an IoT Platform from Scratch for use in a Smart Home Environment with Facial Recognition Security

1st Jayoung Hong
School of Computer Science and Engineering
Pusan National University
Pusan, Republic of Korea
hannahrillo@pusan.ac.kr

2nd Yuting Tan
Purdue Polytechnic Institute
Purdue University
West Lafayette, United State
yutingtan55@gmail.com

3rd Donghyeon Lee
School of Computer Science and Engineering
Pusan National University
Pusan, Republic of Korea
dlehdgus1675@naver.com

4th Eunjong Kim
School of Computer Science and Engineering
Pusan National University
Pusan, Republic of Korea
eunjong147@pusan.ac.kr

5th Soyeon Lee
School of Computer Science and Engineering
Pusan National University
Pusan, Republic of Korea
bp05040705@gmail.com

***Abstract*—As applicable areas of IoT platforms are significantly vast, from everyday life to industrial purposes, the needs for the IoT platform and services to fulfill the needs have also increased. The definition of IoT Platform is not defined clearly. There isn't a certain form of building it. It means that anyone can design one's own IoT platform that suits his/her purpose. However, using pre-existing services will not be the optimal choice for an individual to build one's platform in economical aspects. This paper presents the process to build a basic form of IoT Platform from scratch, which enables end-to-end(device-to-user and vice versa) communication IoT cloud platform using MQTT protocol via mosquitto, which is an open-source message broker. Furthermore, the IoT Platform will be implemented as a prototype with the theme 'Smart Home'. In this step, more functionalities such as device control, data collection, monitoring, and visualization by using a database, Node-RED, and vision learning libraries are added. This process will show the future possibilities of the implementation of a self-made IoT platform and guideline for the first step-users who wants to build their own IoT platform.**

***Index Terms*—Raspberry Pi, Node-Red, Message Queuing Telemetry Transport (MQTT), Internet of Things (IoT) Service Platform, Sensor, Open Source, Cloud Service**

I. Introduction

The 'Internet of Things' means the network of devices that physically exist in the real world. It could be 'things' such as smartphones, computers, or even streetlights. When these 'physical' things are combined with digital systems, it becomes an IoT platform [1]. So IoT Platform is like some kind of magic that enables the 'things' to communicate and to operate with digital systems. These platforms are used not only in industrial areas but also in our daily life. For example, you can control the air conditioner with your voice [2]. That is also a kind of IoT platform. As there are no set format or tools to fulfill the conditions to be an 'IoT platform', anything that could be imagined is possible.

The Cloud IoT services are growing and so are useful tools to create the specific IoT platform designed for special purposes. This means that the IoT market is getting larger: In 2014, about 6 billion devices have installed. This number grew to nearly 30 billion devices for the year 2020 [3]. As a result, the importance of IoT platforms like Amazon AWS IoT, Google Cloud IoT is also increasing. Users can connect devices and servers easier and manage data with those IoT platform services [4]. However, it can be sometimes expensive, and it is difficult to predict the cost [5]. Therefore, building one's own IoT platform could be a good option for a user in economical aspects too.

There are some other advantages other than just an economical aspect. One of them is that users can design its functionalities as they want. Also, it isn't really hard to build one nowadays, since there are many references, libraries, and services to make it super convenient to implement some functions. Moreover, in a real-life, there are lots of IoT platforms and devices that are related to the 'Smart Home'. Some of those functions can be easily developed from our work. Building a prototype of a smart home IoT platform could be a good implementation to show and to apply to the work done in the previous steps to build an IoT platform from Scratch by implementing end-to-end communications using MQTT protocol and the devices that are really used in a real-version of 'Smart Home' [6].

II. Methods

The initial and the most important goal of this project is to show the process to build our own IoT Platform. In this step, it will be focused on building a basic structure of the

IoT platform that enables end-to-end communications and test data/control flow using some basic applications and devices. MQTT and many devices were used in this step to try and experience the issues aroused in a different number of cases.

Below is the abstract diagram of the IoT platform that is going to be built in this step. This shows the direction of the communication flows.

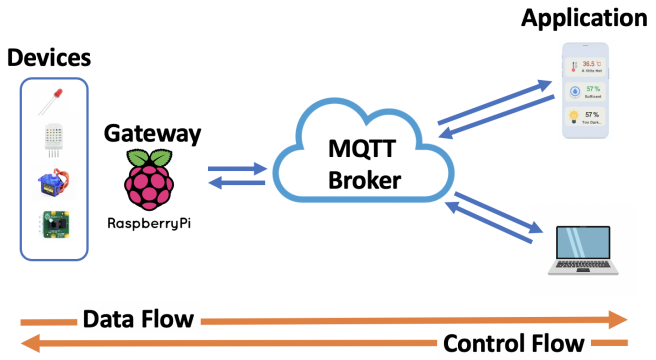

Fig. 1. an abstract diagram for the IoT platform

A. Devices

Devices are modules or sensors that will work as one end of the IoT Cloud platform that receives commands from the user-side and conduct a certain operation, or send data to a user-side. To communicate with the MQTT protocol, every code was set to be either a publisher or a subscriber. MQTT-paho python library [7], and the function with ca.crt file path were also used in the codes for the connection with the broker.

The following devices are used for the IoT platform. Python codes and libraries were used to control them. LED will be simply turned on when it gets a message "on", and turned off when "off". DHT 22 requires Adafruit DHT python library. Temperature and Humidity can be read from here. A servo motor can rotate the axis. The angle and the speed can be controlled by code. RPi.GPIO python library [8], is used to enable GPIO to the motor and control it. Servo or Servo. Angular could be used also, but for the PWM control and to remove jitters, RPi.GPIO would be a better option.

A camera module(Raspberry Pi camera module v2) is used for a real-time image capture system. It can take pictures and stream videos. It's significantly easy to connect and enable using Raspberry Pi OS.

B. Gateway

Gateway connects the multiple devices stated previously using GPIO pins and will function as an MQTT client. A gateway could be either a publisher or a subscriber, depends on the devices and functions that a user wants. Here it is used for both a publisher and a subscriber. The model used is Raspberry Pi 4 Model B. And the OS used is Raspberry Pi OS 32-bit. MQTT protocol was used.

1) Settings: An OS should be installed on Raspberry Pi. Raspberry Pi OS is officially recommended by the Raspberry Pi foundation. The Official version of the OS can be downloaded without any cost from the official website [9].

2) MQTT Client setting: To communicate using the MQTT protocol, the RPi should be set as an MQTT Client. Client setting can be simply done by executing the following commands in the shell. 'Paho-mqtt' is a Python library.

- sudo apt-get update
- sudo apt-get install python3.6
- pip install paho-mqtt

For communication with the MQTT broker, SSL/TLS certification is required. The ca.crt file that is provided from the broker must be configured in the Raspberry Pi. After that, the function stated below should be written in the client program to state the path the ca.crt file is located and to make it work.

- mqttc.tls_set('path for ca.crt')
- mqttc.tls_insecure_set(True)

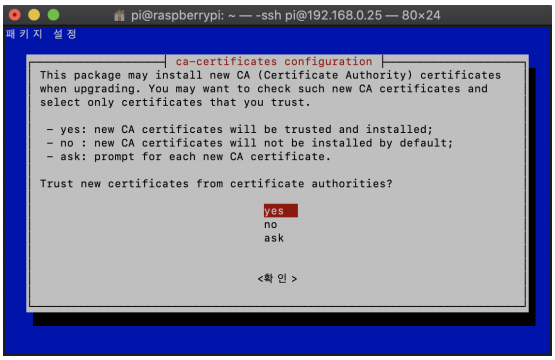

Fig. 2. ca-certificates configuration

C. Server

It is not easy for an individual to build a whole system of a local server. Every aspect should be precisely considered. Such as operating system environment or physical components of the local devices should be also considered. However, one of the trickiest issues is hardware overheating. If a server is built on a local environment, as the server for an IoT cloud platform should remain as it is turned on, high costs are required for the maintenance and research for alternatives to address these issues. A cooling system for the hardware would be needed, for example [10].

This can be easily solved by using a cloud server hosting service(Digital Ocean). Database or other settings can be added to it.

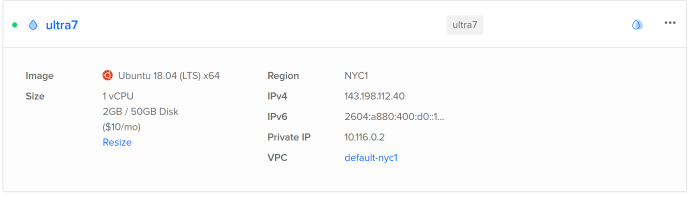

Fig. 3. The server is hosted by DigitalOcean.

D. MQTT Broker

The clients are required to communicate through an MQTT broker. When a client publishes their messages via specific topics, the broker delivers the messages to other clients which subscribe to the topic [11]. Mosquitto is an open-source message broker that implements MQTT. It is suitable for low-powered devices and microcontrollers.

When communicating with the MQTT protocol, It needs to choose which port to use. A port is a kind of path connecting between client and server. Default is the 1883 port. However, port 1883 is unencrypted and unauthenticated. Therefore, the 8883 port was used, which is encrypted. It reduces the risk of being tapped or intercepted during data transmission [12].

```
root@ultra7:~/openSSLFiles# ls
ca.crt  client.crt  server.crt
ca.key  client.csr  server.csr
ca.srl  client.key  server.key
```

Fig. 4. Files required for certification.

For implementing an environment to communicate with the 8883 port, Transport Layer Security(TLS), is required. TLS encrypts data that is sent over the internet to make it safe from the hackers or danger of data leakage [13]. OpenSSL was used for creating elements which used for certification.

```
# Default listener
port 8883

# Certificate based SSL/TLS support
cafile /etc/mosquitto/ca_certificates/ca.crt
certfile /etc/mosquitto/certs/server.crt
keyfile /etc/mosquitto/certs/server.key
```

Fig. 5. .conf file is modified.

After creating the element, it was required to modify the mosquitto.conf file that allows the .conf file to know what users need. To use the 8883 port, inserting “Port 8883” or “listener 8883” to the .conf file is required. Also, the .conf file has to know the path of the certification files and then the broker and the client can communicate through the 8883 port with a low risk of the data being intercepted.

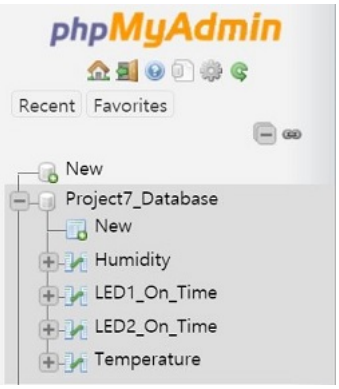

Fig. 6. Database and its tables.

To implement an IoT platform, it needs Databases that can store data. MySQL and phpMyAdmin were used. MySQL is an open-source database management system. phpMyAdmin is an open-source management tool for MySQL written in PHP. It can create, modify, delete databases, tables, fields, columns, manage users and permissions, and more. To store data, it is required to build databases and their tables, columns of the tables.

The tables have columns that can indicate various data. For example, a table named ‘Humidity’ can consist of the humidity of the home and the time when its data is received.

Finished consisting of tables and columns, the database is ready to receive data. Users should implement how to deliver data to the database. Node-RED can help this progress. Node-RED is a flow-based development tool for connecting devices, APIs, servers, and databases. Node-RED has various nodes with useful functions. Therefore, users can implement the flows in the desired direction easily.

After all the previous processes, users can see that the data sent from the device are stored in the database.

E. Web Application

Before creating a web application, the web application should be hosted. Because users had to be able to access the web app. The server(web app) can be accessible from the outside through port forwarding, and it is supported by Goorm IDE, a Docker-based cloud programming tool. In the “runserver” command in Django, the web application has opened port 80 and Goorm IDE connects port 80 to the outside.

```
python manage.py runserver 0.0.0.0:80
```

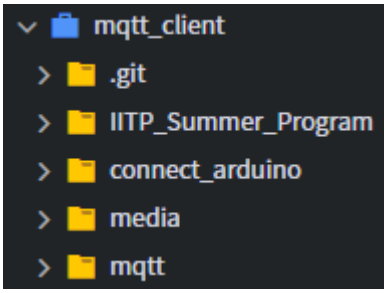

Fig. 7. File structure of the web application

The project name is “IITP_Summer_Program” and created apps “connect_arduino” and “MQTT” within the project. App “connect_arduino” was created to make HTTP communication with Raspberry Pi and save it on the database. And app “mqtt” allows users to interact directly with raspberry pi. Django uses urls.py files to map the app to URL. User can connect App “connect_arduino” when /api/ is attached after the default hosting address (https://ultraman.run.goorm.io), the app “mqtt” when /mqtt/ is attached, and admin page where we can check the database when /admin/ is attached.

App “connect_arduino” will be written as “rest api page” from now on. The purpose of “rest api page”

is to receive images from Raspberry pi using HTTP communication and store them in database. If the user sends an HTTP request using the post method to this URL (https://ultraman.run.goorm.io/api/image/) defined in the urls.py file, our web app saves it to the database. The body of the post method must have text and image, the text contains the name of the person detected in the image, and the image contains the picture when it was taken.

App "mqtt" will be written as "mqtt page" from now on. The latter was chosen between connecting the MQTT client from Django and using mqtt.js in the HTML file. Because the former method should be able to handle MQTT asynchronously, which was too difficult. We added the mqtt.js file to HTML using CDN. A content delivery network (CDN) refers to a geographically distributed group of servers that work together to provide fast delivery of Internet content. When the user connects to the address (https://ultraman.run.goorm.io/mqtt/pub/), the user will see a page where the user can interact with the Raspberry Pi. The page looks like the following picture.

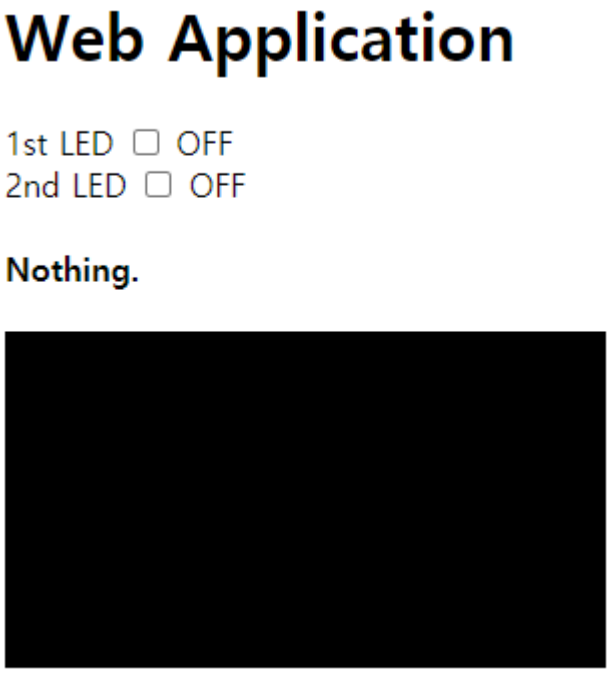

Fig. 8. web app page UI

The two buttons use the MQTT protocol to turn the LED on and off. When the LED is turned on, the letters change to "on", and when it is turned off, the letters change to "off". When the message arrives at the topic "face_req", the HTML page prints the picture stored in the database on the screen. Then, along with his name, the letter "he came to the door" is printed out.

F. Mobile Application

1) MQTT connection: Downloaded 'mosquitto broker' from the site (https://mosquitto.org/download/). After that, the broker was run through 'mosquitto.exe'. Use the Paho library as an MQTT client on Android. Therefore, added 'Client.mqttv3' and 'android.server' to the dependency (Gradle) of the Android project. Since users needed to use the network, registered the permission and service in the Android manifest. Create 'MqttClient' in 'MainActivity' and input server address and port as arguments. Create an MQTT token with the connect function and subscribe to one topic (String type) if the connection was successful. The application can send a message (String type) to the subscribed topic through the publish function.

```
IMqttToken token = mqttAndroidClient.connect(getMqttConnectionOption());
token.setActionCallback(new IMqttActionListener() {
    @Override
    public void onSuccess(IMqttToken asyncActionToken) {
        mqttAndroidClient.setBufferOpts(getDisconnectedBufferOptions());
        Log.e( tag: "Connect_success", msg: "Success");
        tv_output.setText("Connect_success");
        tv_output2.setText("Connect_success");
        try {
            mqttAndroidClient.subscribe( topic: "myled1", qos: 0 );
            Log.e( tag: "subscribe1", msg: "myled1");
            mqttAndroidClient.subscribe( topic: "myled2", qos: 0 );
            Log.e( tag: "subscribe2", msg: "myled2");
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }
```

Fig. 9. Make subscriber

2) HTTP connection: Use the 'HttpURLConnection' object when making a request to the web server and receiving a response, but the amount of code required for the request and response is large. Furthermore, should use threads. If developers didn't understand the handler, the app may end abnormally. Several libraries exist to solve this problem. Among them, HTTP connection was implemented using Volley, which is used the most. So added volley library to dependency(Gradle) and permission to 'AndroidManifest.xml' for internet use.

Create a request object with the 'newRequestQueue' function of the 'requestQueue' object and put this object in the request queue, 'requestQueue' automatically requested the web server, received a response, and call the designated function for users to use. The specified function used the 'StringRequest' object to access the web page or server in the GET method and retrieved the value. Also, it can send a message from Android to the server by overriding the Map function.

```
IMqttToken token = mqttAndroidClient.connect(getMqttConnectionOption());
token.setActionCallback(new IMqttActionListener() {
    @Override
    public void onSuccess(IMqttToken asyncActionToken) {
        mqttAndroidClient.setBufferOpts(getDisconnectedBufferOptions());
        Log.e( tag: "Connect_success", msg: "Success");
        tv_output.setText("Connect_success");
        tv_output2.setText("Connect_success");
        try {
            mqttAndroidClient.subscribe( topic: "myled1", qos: 0 );
            Log.e( tag: "subscribe1", msg: "myled1");
            mqttAndroidClient.subscribe( topic: "myled2", qos: 0 );
            Log.e( tag: "subscribe2", msg: "myled2");
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }
```

Fig. 10. Get the value by accessing the GET method

III. Building a Prototype

After most of the goals have been accomplished in the previous steps, a prototype of an IoT Platform with the topic 'Smart Home' was built. An IoT-based smart home enables various things such as lighting or a home security camera [14]. This is a good challenge to implement the works that have been done to build an IoT platform from scratch to a widely used form of IoT platform. In this step, more functionalities were added such as data analysis using a database, Node-Red, and vision learning python libraries.

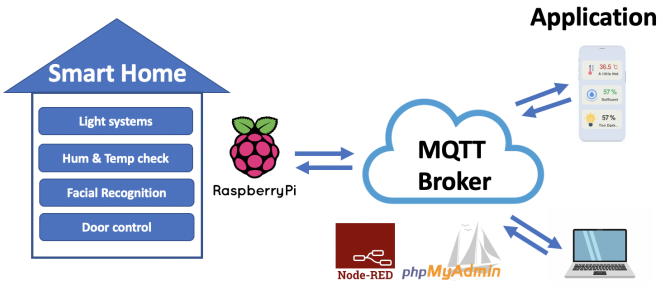

Fig. 11. a schematic for the smart home IoT platform

A. Settings

Since the attempts to try communication with the applications didn't go well, Node-RED Dashboard is used here. Data monitoring and control flow operation can be easily done here. All the detailed settings are available by Node-RED. It helps not only organizing the data but also visualizes it. It provides various 'palettes' that users can easily import.

B. Light systems

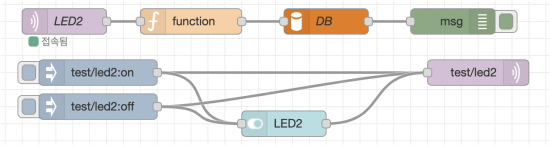

Fig. 12. Light system control flow using Node-RED

Lighting and off 2 LEDs independently was used to check the communications and status of the whole IoT system in the previous steps. These two LEDs are presented as a lamp and a light attaches to the wall. PhpMyAdmin is attached hereto write SQL and store data in the database. Every time the LEDs are turned on and off, the time information is stored in the database, and from that, the user can get a total length of time that the LEDs are remained on. This could be implemented into a system that users can monitor their monthly electricity consumption.

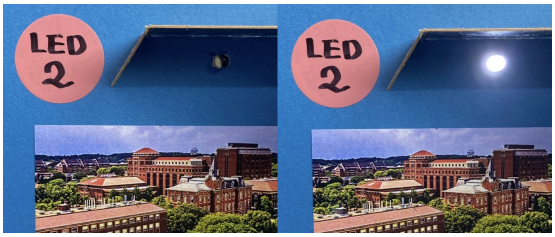

Fig. 13. smart home light system using LEDs

C. Humidity and Temperature check

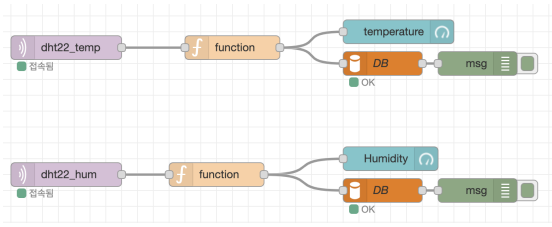

Fig. 14. Data flow of DHT22

+ Options

temp	humidity	logDate
24.7	25.7	2021-08-14 08:36:02
24.5	29	2021-08-14 08:36:12
25.9	45.2	2021-08-14 08:36:22
27.6	30.6	2021-08-14 08:36:32
22.1	55	2021-08-14 08:36:42

Fig. 15. Temperature and humidity data stored in the database

A DHT22 sensor and the Adafruit DHT python library were used. The DS1820 temperature sensor is used in IoT projects often too, but as lack of analog pin in the Raspberry Pi and additional devices needed, DHT22 sensor has been used here. It is not only easier to implement, but also can provide values of the Humidity. The program is to set data every 10 seconds and sends it to the broker via MQTT protocol and stored in the database. The figure below is the values that are visualized by Node-RED Dashboard.

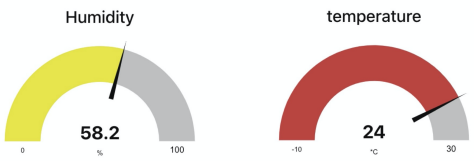

Fig. 16. Temperature and humidity data visualized on dashboard

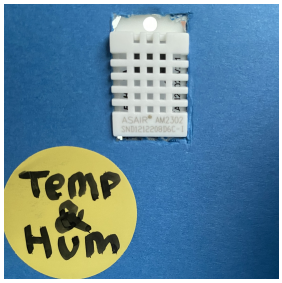

Fig. 17. smart home temperature and humidity check using DHT22

D. Facial Recognition

Using a camera, many functions can be developed, but sending images or videos through the real-time communication system would be most appropriate for 'Smart Home'.

The camera is used as a face recognition camera to be installed on the doorway to notify users who are the visitor when someone visits the house. For these functionalities, opencv-4.5.1 [15] and face-recognition 1.3.0 python libraries [16]; were used and example codes from them were used to train a model to recognize different faces. This could be a good smart doorway CCTV system already.

To make the function more suitable for the prototype, more codes were added to conduct the following actions. The camera takes a photo when a face is detected and recognized. Send the photo and the name of the visitor to the broker. Users can check the image and the name of the visitor through the Applications.

Fig. 18. smart home face camera using Rpi Camera

Sending an Image itself through MQTT could have been challenging, however, since the memory of an image file was too big, an alternative was used. Using the idea that an image can be accessed everywhere with its path, an additional server using HTTP was built to upload the image file and create a URL path for the image. In that way, it will work much faster and use fewer memories. The server was developed with Node.js, and hosted to 'Heroku(cloud server)'. Also, the time and the name history of the visitor are stored on the server.

The program takes a picture of when a face is detected and recognized. Then the additional server creates a path for the image and returns it. The path and the identified name of the visitor are sent to the broker in the form of a string via an MQTT message payload. Through the applications, the user

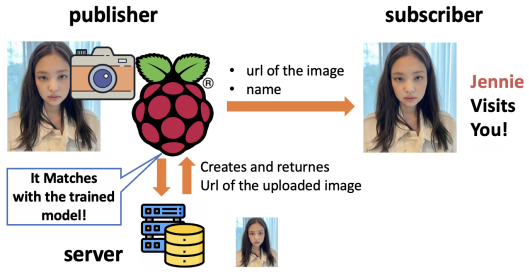

Fig. 19. This figure shows how the program works and describes the flow of the data

can check the image just like the real-time screen capture. This can be also checked in Node-red.

```
// Enable File Upload part
const upload = multer({
  storage: multer.diskStorage({
    destination: function (req, file, cb) {
      cb(null, "uploads");
    },
    filename: function (req, file, cb) {
      console.log(req, "is file");
      const NAME = req.body.data.split(" ")[0];
      cb(null, NAME + ".jpg");
    },
  }),
});
```

Fig. 20. server code to enable file upload

```
// Get Face Image => Save & return Uploaded Image URL
app.post("/messages", upload.array("file") async (req, res) => {
  const { name, data } = req.body;
  const SVAED_FILENAME = req.files[0].filename;
  return res.send(
    `https://iotserver-jy.herokuapp.com/public/${SVAED_FILENAME}`
  );
});
```

Fig. 21. When the server gets a image file it returns uploaded URL

E. Door Control

Now the user will know when someone arrives at the doorway. Then the user should be able to open the door automatically too. For the door control function, turning movement using a servo motor is used. RPi.GPIO python library is used to enable GPIO to the motor and control it. Servo or Servo. Angular could be used also, but for the PWM control and remove jitter, RPi.GPIO would be a better option.

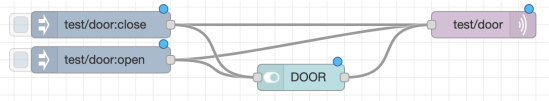

Fig. 24. Door Control flow using Node-RED

The functionality is simple. When it gets an open/close signal from a user, it opens or closes itself. The same control is available on Node-Red Dashboard by implementing the switch button.

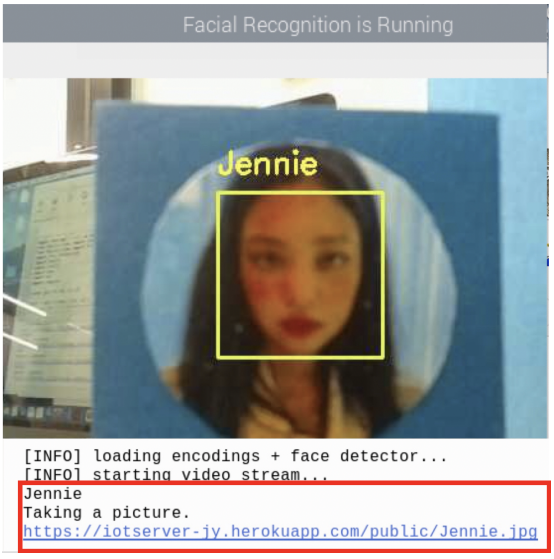

Fig. 22. URL returned to the program

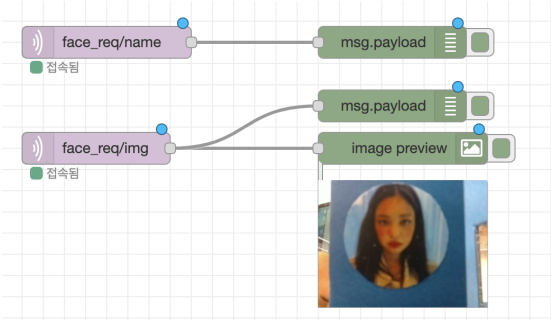

Fig. 23. Node-RED image received

Fig. 25. Smart home Door Control using a servo motor

IV. Results

Most of the parts worked successfully as the project was designed to be, and went even further by implementing more functions using many methods such as Node-RED or OpenCV library. The attempts to set the devices and a server as clients or a broker were successful. Node-RED has made it much easier to monitor and control the data that are received from the clients. Also, the Node-RED dashboard has enabled the operation and the visualization of the data. In a communication area, it was also successful progress to make the Mosquitto broker listen to the 8883 port and store the received data from sensors. Which are not expected when this project first started. All of these works were implemented into a prototype of a smart home IoT platform, and it worked successfully.

A. Limitations

The biggest limit of this project was that the Applications could not conduct MQTT protocol communication successfully. It is due to the SSL/TSL certification problem as stated in previous sections. However, many tries and research has been conducted to solve this issue. Below are the limitations for each part of the IoT system.

1) Devices: Arduino is suitable for controlling the devices and the sensors. It has Digital and Analog pins, so it makes it much easier to implement sensors that need analog input. Also, with its IDE, it is much convenient to import libraries, ports and make programming easier. However, it needs Wi-Fi or Bluetooth module to communicate with other clients and the server.

2) MQTT Broker: To transfer the data by WebSocket, the 8081 port that is an MQTT port over WebSocket is required. However, it was unable to make the Mosquitto broker listen to the 8081 port.

```
1629310657: Error: Websockets support not available.
1629310657: Error found at mosquitto.conf:411.
```

Fig. 26. Cannot listen the websocket port.

Therefore, two kinds of attempts were made.

a) Update version of libwebsockets: The Mosquitto 1.6.9 version has been installed. Therefore, there is a possibility that the version of libwebsockets currently installed on the server will not fit. However, the version of libwebsockets was already the newest version. So it could not solve this problem.

b) Install another version of Mosquitto: Found the version of Mosquitto that fits the libwebsockets version 2.4.2. Therefore, installed version 1.6.12 on the server, and It works successfully.

```
1629312817: mosquitto version 1.6.12 starting
1629312817: Config loaded from mosquitto.conf.
1629312817: Opening websockets listen socket on port 8081.
1629312817: Opening ipv4 listen socket on port 8883.
1629312817: Opening ipv6 listen socket on port 8883.
1629312817: mosquitto version 1.6.12 running
```

Fig. 27. Success to listen the 8081 port.

3) Web Application: The web app couldn't connect to the server using MQTT, so it was connected to the test server. I think I can do it with Django channels, but I ran out of time. There are only developers, so there is not enough design.

4) Mobile Application: Unable to connect to a real server using MQTT. An MQTT client Android application was created and connected to the test server. At this time, the connection was successful and both subscribe and publish functions could be implemented using tokens. However, when changed the value to a real server, there was no connection at all. An error log of 'java.net.SocketException: Connection

reset' appeared. The problem was that it is a TLS version problem during the SSL setting of Java HTTP communication. In this regard, three attempts were made.

a) Change TLS to TLSv1.2: It was said that changing the version would solve the problem, but impossible to change the version.

b) Install Java certificate on device and development environment: The Android version of the device is 'Pie'. So installed the 'ca.crt' file from Settings - Security and Location - Encryption and Credentials - Trusted Credentials on this device. Tried to install a certificate from the Java root certificate store (JAVA_HOME\lib\security\cacerts) using the 'keytool' command, but an error, 'java.io.FileNotFoundException: ca.crt' occurred. When looked up the root certificate, there was nothing. So created a new 'keystore' and tried to install the certificate, but the same error was repeated. In a last way, downloaded the 'Portecle GUI' tool and tried to install the certificate, but also got an error saying that the file could not be found and could not solve this problem.

V. Conclusion

In this project, an IoT platform was built to be implemented as an IoT smart home. Based on the IoT platform that enables end-to-end communication using the MQTT protocol, a lighting system, temperature, and humidity monitor, door control, and visitor alarm system using a face recognition camera were added. The result emphasizes its value in designing and composing a concept of an IoT platform from scratch and building it. As it is shown in the work that diverse libraries, services, or protocols can be used to implement new functionalities, this work gives good motivation to further implementations.

References

[1] A. Tamboli, "So… You Want to Build Your Own!" in *Build Your Own IoT Platform*, 3rd ed. Berkeley, CA, USA: Apress, 2019, ch. 1, pp. 1-2.

[2] T. Erić, S. Ivanović, S. Milivojša, M. Matić and N. Smiljković, "Voice control for smart home automation: Evaluation of approaches and possible architectures," *2017 IEEE 7th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, 2017, pp. 140-142, doi: 10.1109/ICCE-Berlin.2017.8210613.

[3] "What Is The "Internet of Things"?" last accessed 9 Feb 2015. [Online]. Available: https://www.postscapes.com/what-exactly-is-the-internet-of-things-infographic/

[4] J. Guth et al., "A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences," *Internet of Everything*, Oct. 2017, pp. 13-15, doi: 10.1007/978-981-10-5861-5_4.

[5] L. Nikityuk and R. Tsaryov, "Optimization of the Process of Selecting of the IoT-Platform for the Specific Technical Solution IoT-Sphere," *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC ST)*, 2018, pp. 401-405, doi: 10.1109/INFOCOMMST.2018.8632088.

[6] K. Agarwal, A. Agarwal and G. Misra, "Review and Performance Analysis on Wireless Smart Home and Home Automation using IoT," *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 2019, pp. 629-633, doi: 10.1109/I-SMAC47947.2019.9032629.

[7] "Eclipse Paho" last accessed 25 Feb 2019. [Online], Available: https://projects.eclipse.org/projects/iot.paho

[8] "RPi.GPIO 0.7.0" last accessed 21 Jul 2019. [Online], Available: https://pypi.org/project/RPi.GPIO/

[9] "Operating system images" last accessed 7 May 2021. [Online], Available: https://www.raspberrypi.org/software/operating-systems/raspberry-pi-os-32-bit

[10] "How much will a server cost?" last accessed 27 May 2021. [Online], Available: https://startups.co.uk/technology/how-much-will-a-server-cost/

[11] H. Hejazi, H. Rajab, T. Cinkler and L. Lengyel, "Survey of platforms for massive IoT," *2018 IEEE International Conference on Future IoT Technologies (Future IoT)*, Jan. 2018, pp. 3-5, doi: 10.1109/FIOT.2018.8325598.

[12] L. Anyi, A. Ali, M. Hua and D. Balakrishnan, "Automatic Verification of SSL/TLS Certificate for IoT Applications," IEEE ACCESS: 27038-27050.

[13] K. Alghamdi, A. Alqazzaz, A. Liu, and H. Ming, "An Automated Tool to Verify SSL/TLS Certificate Validation in Android MQTT Client Applications," *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, Mar. 2018, p. 95, doi: 10.1145/3176258.3176334.

[14] T. Malche and P. Maheshwary, "Internet of Things (IoT) for building smart home system," *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 2017, pp. 65-70, doi: 10.1109/I-SMAC.2017.8058258.

[15] "OpenCV 4.5.1" last accessed 22 Dec 2020. [Online], Available: https://opencv.org/opencv-4-5-1/

[16] "face-recognition 1.3.0" last accessed 20 Feb 2020. [Online], Available: https://pypi.org/project/face-recognition/