3.2 Software Flowchart

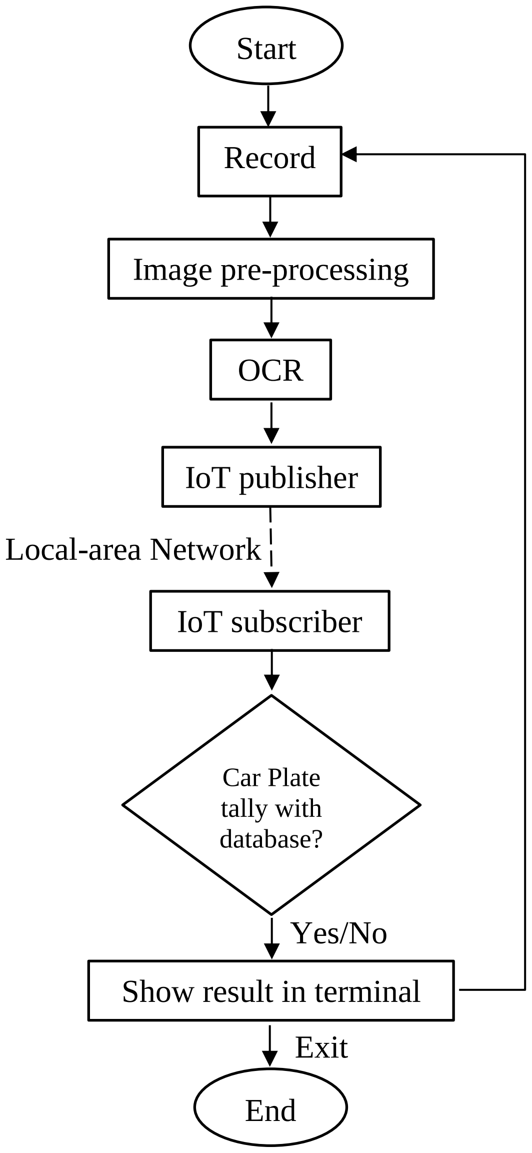


Figure 3.3: Software Flowchart

The software is categorized into 3 main parts. The image processing part, which involves MATLAB to do image treatment and OCR. This part is likely to enhance the recorded image so it can improve OCR performance.

The second part is the IoT part, where the app will extract information from MATLAB and Database. This part involves MQTT for publishing and subscribing purposes. This app can be separated into two working folders, by means, it can operate on two different computers. Note that this can only work if both of the computers are connecting to the same local-area network (LAN). One is to extract information from MATLAB through an excel file, then publish them on the broker. Another one is to subscribe to the message and request information from the database according to the message.

The third part is the database part. Registered vehicle's information will have to be stored inside the database to enable future extraction. This can be done by using Influx query in influx.exe.

The methodology of doing the parts mentioned above will be mentioned in detail along with possible errors faced during the setup process in the following paragraph. The image below shows the overview of the project.

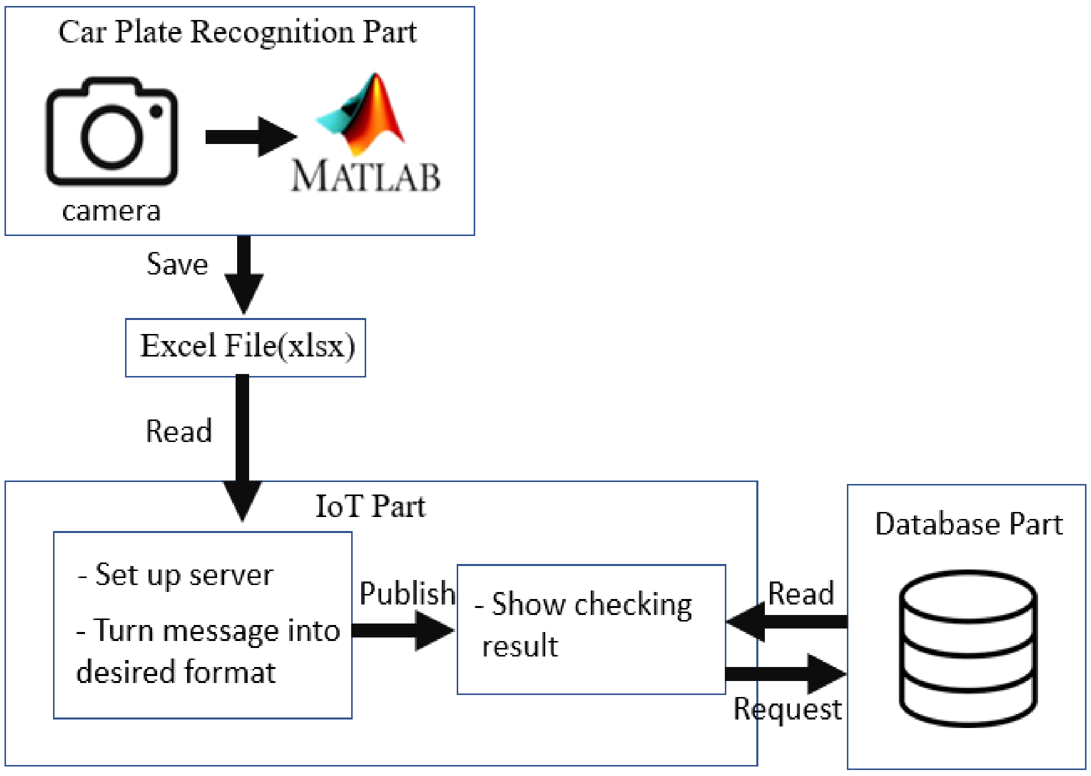


Figure 3.4: Overview of Project

3.3 Car Plate Recognition Part Methodology

A mobile phone camera is connected to a laptop via Bluetooth. Then, by using MATLAB the recording sent from the phone is snapped and undergoes image treatment before sending it to OCR.

The first step is converting the image from RGB to gray color using ‘rgb2gray(I)’ function. Then, extract the gray thresh information from the converted image using ‘graythresh(I)’, where the gray color is given value from 0 to 1. 0 stands for white and 1 stands for black. The next step is binarization, which is converting the gray image into a black and white image using ‘~imbinarize (I, threshold)’ function. In this case, the ‘~’ sign in front of the function stands for negative, which means according to grayscale from 0 to 1, 0 to 0.49 will become black while 0.5 to 1 will be turned into white. This process is to ease the OCR process by making the characters black and significant. Figures below show the comparison between the original image and the image that is ready to go for the OCR process.

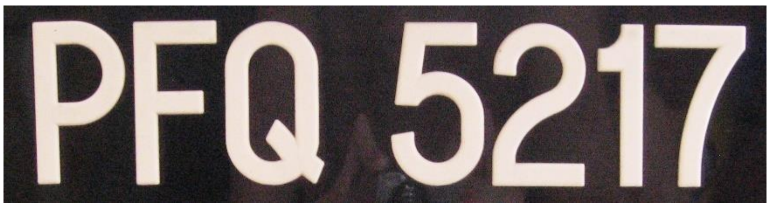


Figure 3.5: Original Image before Image Treatment

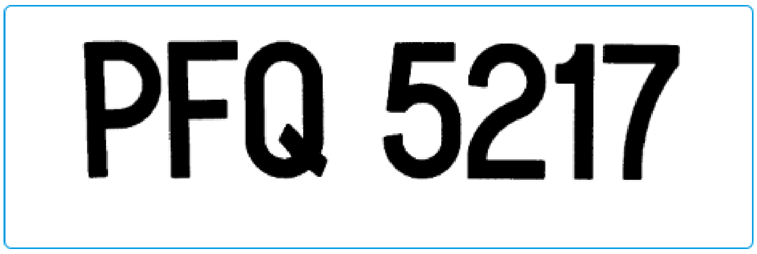


Figure 3.6: Output Image after Image Treatment

The image will then go through an OCR process by using the ‘ocr’ function in MATLAB. The result will be converted into text and saved in a Microsoft Excel file (xlsx format).

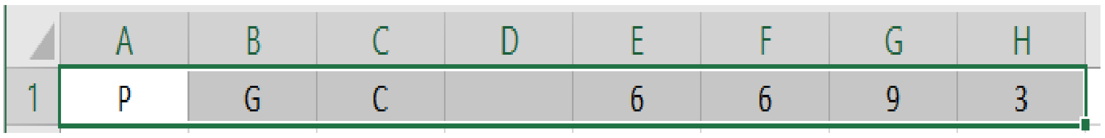
3.4 IoT Part Methodology

IoT part is the main part of this project as we are implementing this system on ALPR. So, message transferring and good communication between two or more entities are very crucial.

3.4.1 Excel Format

In this project, an approach to upload the result of OCR to Thing Speak IoT platform before but the platform itself can only allow transferring message that contains number only, which is not ideal for uploading car plate characters because car plate usually consists of the alphabet too. It is designed for data aggregation and analytic, therefore it is not suitable for this project. There are no current available IoT tools that can work with MATLAB to send a message through the internet, therefore another alternative is used, which is through xlsx format.

An app is created using Node.js to read the data from the excel file. This excel file will update every 3 seconds to ensure the data is up to date in case the MATLAB OCR recognized any character from the camera. This xlsx folder cannot be opened while the MATLAB is writing data. The image below shows the recorded result saved from MATLAB and stored in an excel file.

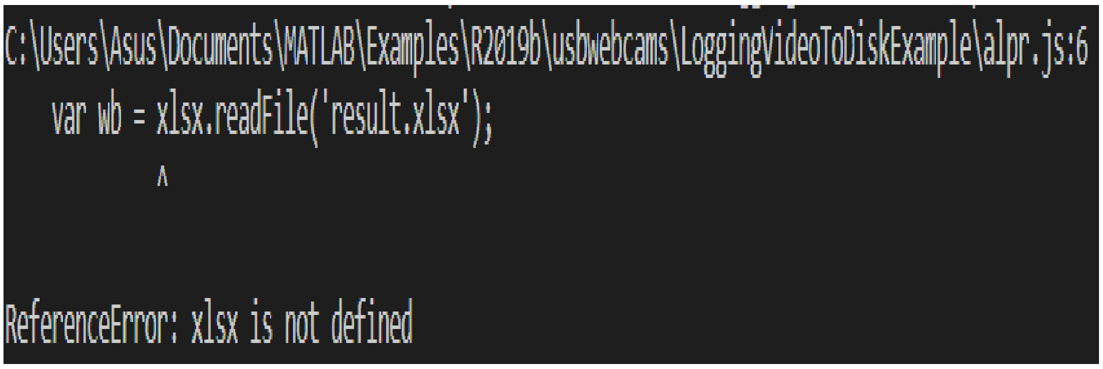


	A	B	C	D	E	F	G	H
1	P	G	C		6	6	9	3

Image 3.7: The recorded data in excel file

3.4.2 Including Protocols

To enable xlsx-node protocol, the documentation had to be downloaded by opening a terminal in the working folder. Then, including the functions or protocols in the source code, else the app cannot extract data from the excel folder because xlsx function will become an error. The API will show an error while debugging because this program does not recognize the function if the protocol or documentation is not included. The image below shows the result of running the app to extract data from excel without calling the function.



```
C:\Users\Asus\Documents\MATLAB\Examples\R2019b\usbwebcams\LoggingVideoToDiskExample\alpr.js:6
   var wb = xlsx.readFile('result.xlsx');
            ^

ReferenceError: xlsx is not defined
```

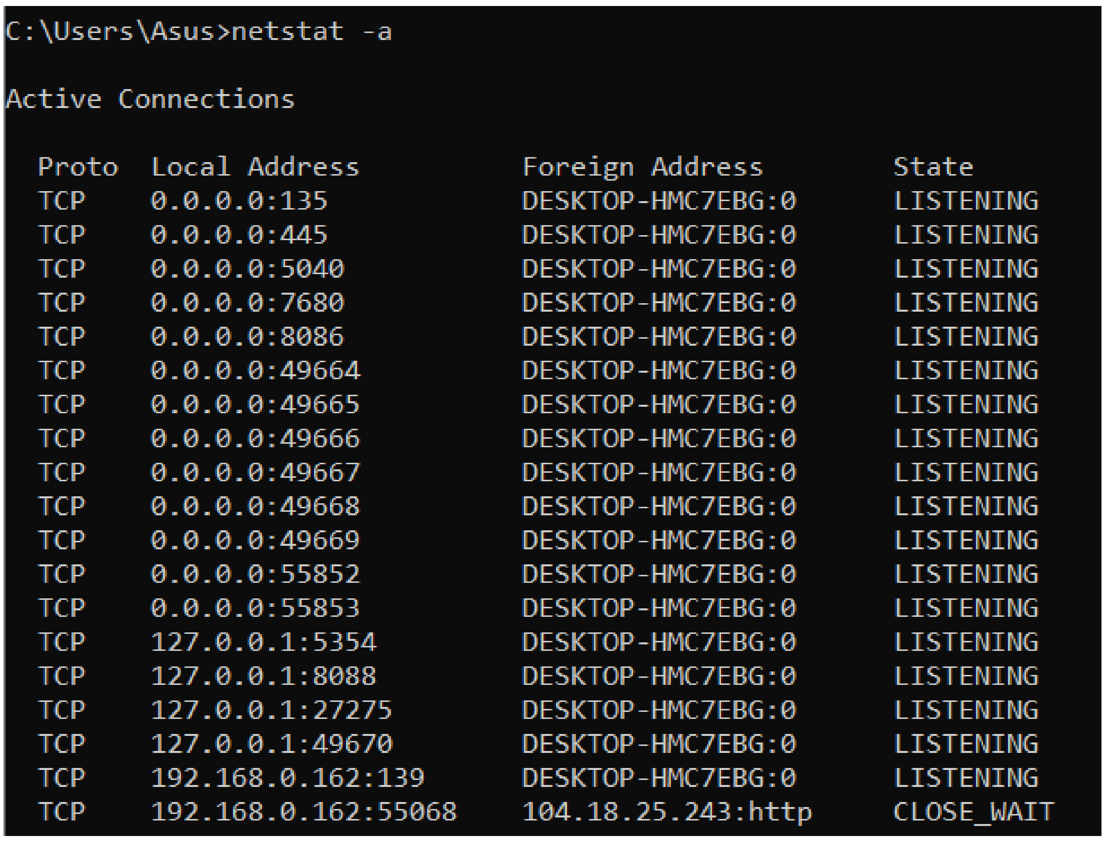
Image 3.8: Error: xlsx function is not defined

Therefore, to enable communication between Influx database, the app, and MATLAB, different functions had to be included in the source code after downloading them in command prompt. We also need to include ‘Mosca’ protocol for broker set up purpose, MQTT for publishing and subscribing messages, Influx-node for operating database and xlsx-node for extracting information from excel file to the app

3.4.3 Setting up a broker

A broker is set up before performing message transferring. Without the broker, MQTT alone will not work because there is no server to perform subscribing and publishing messages. In this project, the broker is set up on a localhost, which is on the laptop’s IP address, in this case, it is 192.168.0.162.

After a port is used to set up the broker on this laptop. A computer port act as an interface between computer and computer or computer with external devices. If a port is not available, it cannot be used to set up the broker. Therefore, to check a broker, one can open a terminal and type in ‘ipconfig’. Afterward, type “netstat -a”. A list of port numbers will come out along with their current state. The figure below shows the list of port numbers after executing the command on the laptop. Referring to the figure below, the port number will be shown at the end of the ‘Local Address’. For example, the port in the first line is 135, following by 445, 5040, and so on.

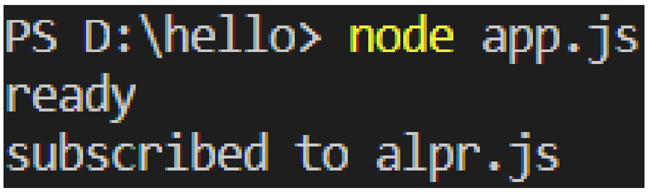


```
C:\Users\Asus>netstat -a

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            DESKTOP-HMC7EBG:0      LISTENING
  TCP    0.0.0.0:445            DESKTOP-HMC7EBG:0      LISTENING
  TCP    0.0.0.0:5040           DESKTOP-HMC7EBG:0      LISTENING
  TCP    0.0.0.0:7680           DESKTOP-HMC7EBG:0      LISTENING
  TCP    0.0.0.0:8086           DESKTOP-HMC7EBG:0      LISTENING
  TCP    0.0.0.0:49664          DESKTOP-HMC7EBG:0      LISTENING
  TCP    0.0.0.0:49665          DESKTOP-HMC7EBG:0      LISTENING
  TCP    0.0.0.0:49666          DESKTOP-HMC7EBG:0      LISTENING
  TCP    0.0.0.0:49667          DESKTOP-HMC7EBG:0      LISTENING
  TCP    0.0.0.0:49668          DESKTOP-HMC7EBG:0      LISTENING
  TCP    0.0.0.0:49669          DESKTOP-HMC7EBG:0      LISTENING
  TCP    0.0.0.0:55852          DESKTOP-HMC7EBG:0      LISTENING
  TCP    0.0.0.0:55853          DESKTOP-HMC7EBG:0      LISTENING
  TCP    127.0.0.1:5354         DESKTOP-HMC7EBG:0      LISTENING
  TCP    127.0.0.1:8088         DESKTOP-HMC7EBG:0      LISTENING
  TCP    127.0.0.1:27275        DESKTOP-HMC7EBG:0      LISTENING
  TCP    127.0.0.1:49670        DESKTOP-HMC7EBG:0      LISTENING
  TCP    192.168.0.162:139      DESKTOP-HMC7EBG:0      LISTENING
  TCP    192.168.0.162:55068    104.18.25.243:http     CLOSE_WAIT
```

Figure 3.9: List of Port Connection

According to the MQTT official website [16], the broker can only be set up on 5 ports, 1883 (unencrypted), 8883 (encrypted), 8884 (encrypted but client certificate required), 8080 (on WebSockets, unencrypted), 8081 (on WebSockets, encrypted). In this case, the ‘Mosca’ server was set up on port 1883.

After the broker is set up, the application will show “ready” in the terminal to verify that the broker is ready for subscribing and publishing tasks. Figure below shows the terminal message after a broker is set up and ready after running the program.

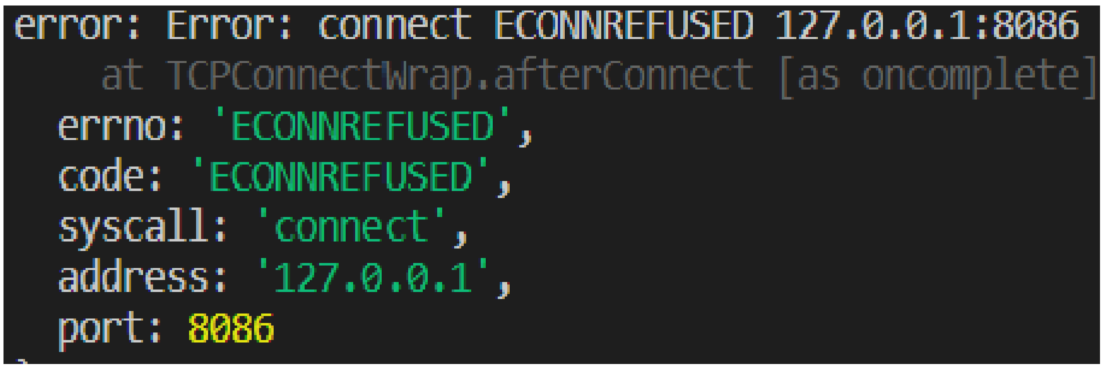


```
PS D:\hello> node app.js
ready
subscribed to alpr.js
```

Figure 3.10: Terminal Message to Show Broker is Ready

3.4.3.1 Error Setting Up Broker on Non-listening Port

If a port on a non-listening state, the broker could not be set up on this port. Else the application will show an error. Figure below shows error if a broker is set up on a non-listening port. In this case, the application could not detect port 8086 because it is not listening. The port is not available or already in use.



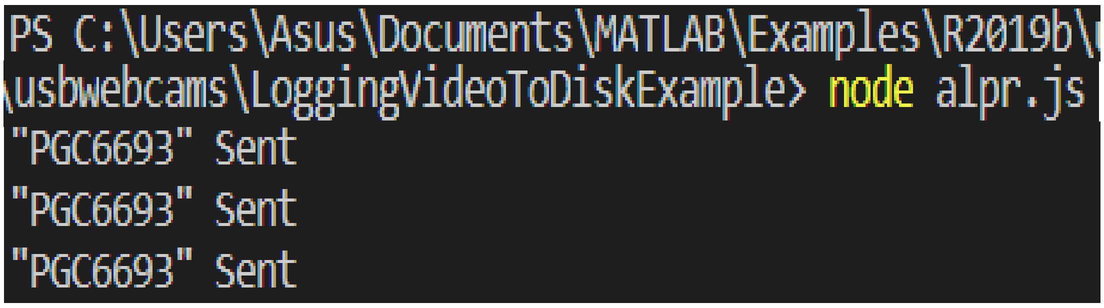
```
error: Error: connect ECONNREFUSED 127.0.0.1:8086
    at TCPConnectWrap.afterConnect [as oncomplete]
  errno: 'ECONNREFUSED',
  code: 'ECONNREFUSED',
  syscall: 'connect',
  address: '127.0.0.1',
  port: 8086
```

Figure 3.11: Error Setting Up Broker

3.4.4 Publishing Message - MQTT

After extracting OCR result from excel to the app, the app rearranged the characters from [‘P’,’G’,’C’,‘ ’,’6’,’6’,’9’,’3’,‘ ’,’ ‘] into [‘PGC6693’]. The excel-folder has to be closed on any tab of the computer else it cannot be read to the app. The characters are combined into a single string. Spaces or unwanted signs are eliminated to match the format of car plate characters saved in the database. The string is then published with a topic, in this project the topic name is ‘myTopic’. A terminal message is set to inform the user the message is published. The purpose of showing the result in the terminal is to inform the user that the published message is sent, else the user would not be informed. The message will be directed to ‘{string_final}’ in the code, as an object to make it dynamic. This will cause the result in terminal changes according to extracted data from excel.

To make sure the message published is correct, an instruction is written to show the published message on the terminal. Image below shows the result in the command prompt or terminal after the app is running in the working folder.



```
PS C:\Users\Asus\Documents\MATLAB\Examples\R2019b\
\usbwebcams\LoggingVideoToDiskExample> node alpr.js
"PGC6693" Sent
"PGC6693" Sent
"PGC6693" Sent
```

Figure 3.12: Terminal Message after Publishing Topic

3.4.5 Subscribing Message – MQTT

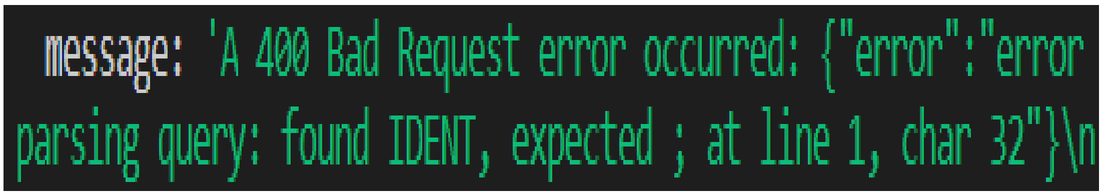
This app will publish a message to the broker so on the other side, the app will subscribe to the message by directing through the topic. The subscribed message will then be used to extract information from Influx Database. When the topic is subscribed, a terminal message “subscribed to alpr.js”, will be shown to verify the subscription towards the topic. Please refer to Figure XX: Terminal Message to Show Broker is Ready.

In this case, the topic name is simply “myTopic”, but it had to be exactly the same as the published topic, considering spaces, symbols, and big or small letters.

After subscribing, the app receives a line of string. Note that only string or array could be sent through MQTT.

3.4.6 Correcting JSON format

The subscribed topic will be parsed into an object from string. The purpose of turning this message into a JSON object is to ease the use of Influx Database's query function inside the app. A string format of the message will be rejected. Image below shows the result shown in the terminal if the message is not in the right format to extract data from Influx database.



```
  message: 'A 400 Bad Request error occurred: {"error":"error
parsing query: found IDENT, expected ; at line 1, char 32"}\n
```

Figure 3.13: Error while extracting Information from Influx Database Using String

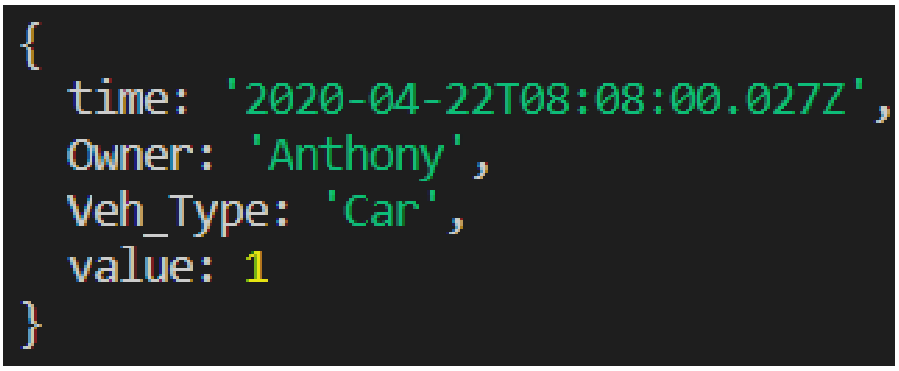
3.4.7 Extract Information From Database

After parsing the received message into a JSON object, the app selects and reads specific information from Influx Database. By doing this the extracted message is dynamic. Any changes at the subscribed message will change the requested information from the database.

Influx query protocols enable this app to access Influx database from this app internally by using Influx's query language. Note that Influx.exe has run in the background to enable the accessing of the database. The extracted information from the database is then changed into string so that it can be printed out on the terminal to be verified.

3.4.7.1 Managing Plain Result from Database

Influx database is a time-series database, in this case, the data insert time for the vehicle will be extracted along when a car plate information is requested. If the vehicle is recorded twice in the database, the output will show two messages. To avoid this, the app will rearrange them and only take in the first registered information. Image below shows the sole output if the information extracted is not managed. This information cannot be rearranged or used in other functions inside the app because this information is not changed into a JSON object.



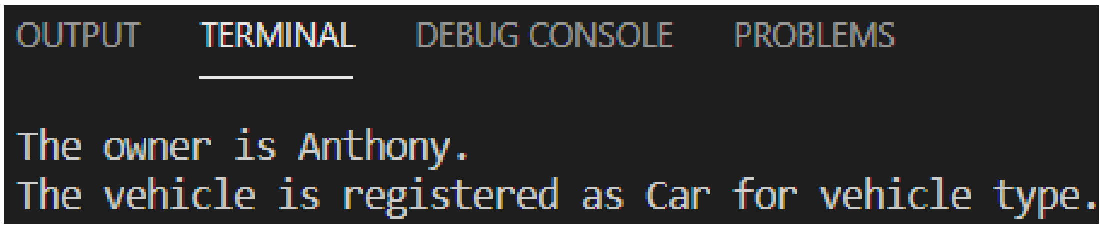
```
{
  time: '2020-04-22T08:08:00.027Z',
  Owner: 'Anthony',
  Veh_Type: 'Car',
  value: 1
}
```

Figure 3.14: The Complete Output if extracted information is not Managed

3.4.7.2 Setting up a condition for vehicle check

If the car plate number and its information is not stored in database, the app will give an output of empty arrays on the terminal which is unclear for the user. To make the result clear while the app is checking for availability, a condition is programmed. If the database gives an empty array as feedback, then it means the vehicle is not registered, the terminal will show a line of the message, “Warning! The vehicle is not recorded in database.” Else the app will show registered vehicle’s information. In this case, the shown information is vehicle owner’s name and vehicle type. “console.log” function is to determine what to be shown inside the terminal or command prompt area. Image below shows the source code of setting a condition to differentiate if a vehicle is registered inside the database.

The source code is crucial to make the terminal output clear. In this project, it will select important information only and rearrange them into clear and simple sentences to be shown in the terminal. The coding without parsing the extracted data and rearranging will give output in a relatively more complicated format. Image below shows the result on the terminal.



```
The owner is Anthony.
The vehicle is registered as Car for vehicle type.
```

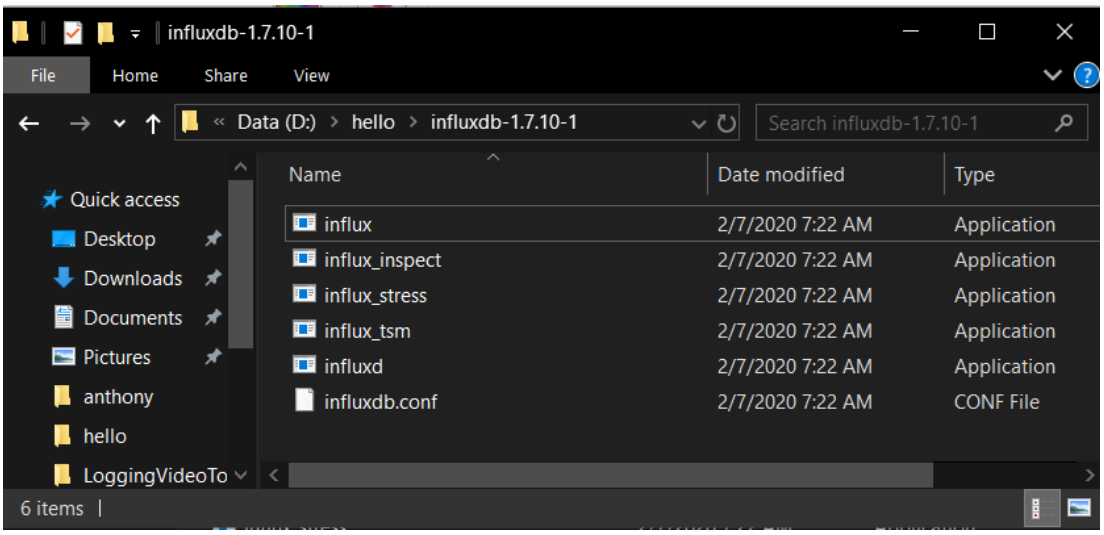
Figure 3.15: Desired Result from Influx Query

3.5 Database Part Methodology

The database is set up to store vehicle information for this project. A database could be based on local or cloud. In this project, the database is based on the laptop, which is local.

3.5.1 Database Set Up

The way to write data into the database is through “influx.exe” application. The program is created while installing Influx database on the computer. Image below shows how the folder looks like. The first application is the influx.exe.



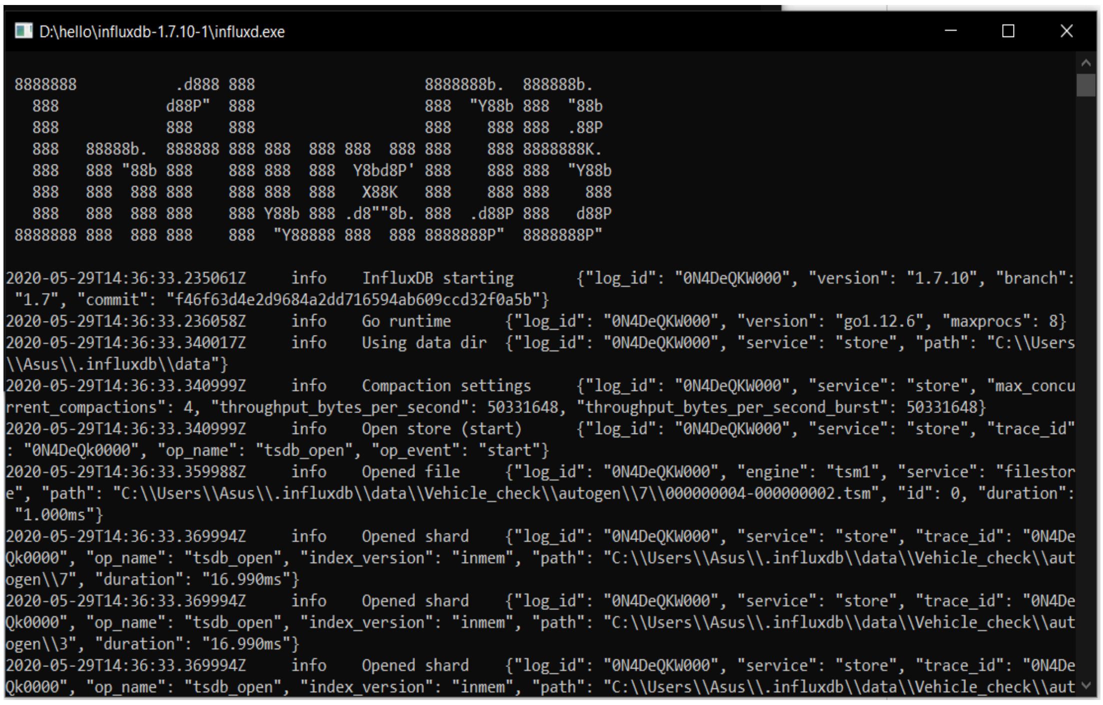
Name	Date modified	Type
influx	2/7/2020 7:22 AM	Application
influx_inspect	2/7/2020 7:22 AM	Application
influx_stress	2/7/2020 7:22 AM	Application
influx_tsm	2/7/2020 7:22 AM	Application
influxd	2/7/2020 7:22 AM	Application
influxdb.conf	2/7/2020 7:22 AM	CONF File

Figure 3.16: Influx database folder

Another alternative to access Influx database through Node.js. The ‘influx-node’ protocol could be installed. It allows users to write data through ‘influx.query’ function or set up a database through running a source code instead of giving sets of instruction using the influx-query protocol.

This alternative is not convenient because every time a user wanted to insert data, value, and information need to be changed here and re-run the source code again. It is not user friendly for an administrator who does not know much about the coding.

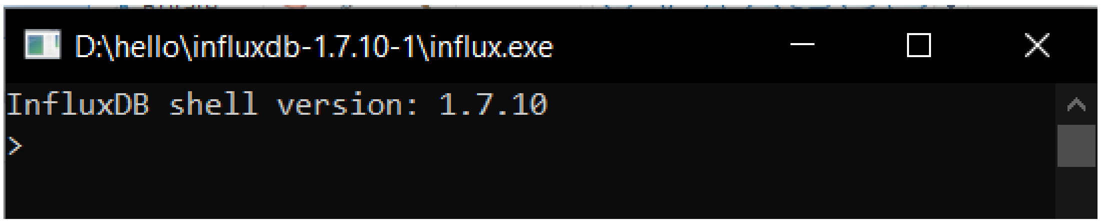
Other than that, before accessing through “influx.exe”. The database needs to be running in the background. In this case, it is “influxd.exe”. Afterward, instructions are executed through the Influx-query language inside of “influx.exe”. Without running the database in the background, or the influxd.exe, “influx.exe” will crash or close the window on its own. This program has to be running in the background when Node.js is running, else the database will not be accessed. Image below shows how influxd.exe looks like while running.



```
D:\hello\influxdb-1.7.10-1\influxd.exe

 8888888           .d888 888                   8888888b.  888888b.
   888            d88P"  888                   888  "Y88b 888  "88b
   888            888    888                   888    888 888  .88P
   888   88888b.  888888 888 888  888 888  888 888    888 8888888K.
   888   888 "88b 888    888 888  888  Y8bd8P' 888    888 888  "Y88b
   888   888  888 888    888 888  888   X88K   888    888 888    888
   888   888  888 888    888 Y88b 888 .d8""8b. 888  .d88P 888   d88P
 8888888 888  888 888    888  "Y88888 888  888 8888888P"  8888888P"

2020-05-29T14:36:33.235061Z     info    InfluxDB starting       {"log_id": "0N4DeQKW000", "version": "1.7.10", "branch":
 "1.7", "commit": "f46f63d4e2d9684a2dd716594ab609ccd32f0a5b"}
2020-05-29T14:36:33.236058Z     info    Go runtime      {"log_id": "0N4DeQKW000", "version": "go1.12.6", "maxprocs": 8}
2020-05-29T14:36:33.340017Z     info    Using data dir  {"log_id": "0N4DeQKW000", "service": "store", "path": "C:\\Users
\\Asus\\.influxdb\\data"}
2020-05-29T14:36:33.340999Z     info    Compaction settings     {"log_id": "0N4DeQKW000", "service": "store", "max_concu
rrent_compactions": 4, "throughput_bytes_per_second": 50331648, "throughput_bytes_per_second_burst": 50331648}
2020-05-29T14:36:33.340999Z     info    Open store (start)      {"log_id": "0N4DeQKW000", "service": "store", "trace_id"
: "0N4DeQk0000", "op_name": "tsdb_open", "op_event": "start"}
2020-05-29T14:36:33.359988Z     info    Opened file     {"log_id": "0N4DeQKW000", "engine": "tsm1", "service": "filestor
e", "path": "C:\\Users\\Asus\\.influxdb\\data\\Vehicle_check\\autogen\\7\\000000004-000000002.tsm", "id": 0, "duration":
 "1.000ms"}
2020-05-29T14:36:33.369994Z     info    Opened shard    {"log_id": "0N4DeQKW000", "service": "store", "trace_id": "0N4De
Qk0000", "op_name": "tsdb_open", "index_version": "inmem", "path": "C:\\Users\\Asus\\.influxdb\\data\\Vehicle_check\\aut
ogen\\7", "duration": "16.990ms"}
2020-05-29T14:36:33.369994Z     info    Opened shard    {"log_id": "0N4DeQKW000", "service": "store", "trace_id": "0N4De
Qk0000", "op_name": "tsdb_open", "index_version": "inmem", "path": "C:\\Users\\Asus\\.influxdb\\data\\Vehicle_check\\aut
ogen\\3", "duration": "16.990ms"}
2020-05-29T14:36:33.369994Z     info    Opened shard    {"log_id": "0N4DeQKW000", "service": "store", "trace_id": "0N4De
Qk0000", "op_name": "tsdb_open", "index_version": "inmem", "path": "C:\\Users\\Asus\\.influxdb\\data\\Vehicle_check\\aut
```

Figure 3.17: Running “influxd.exe” in Background

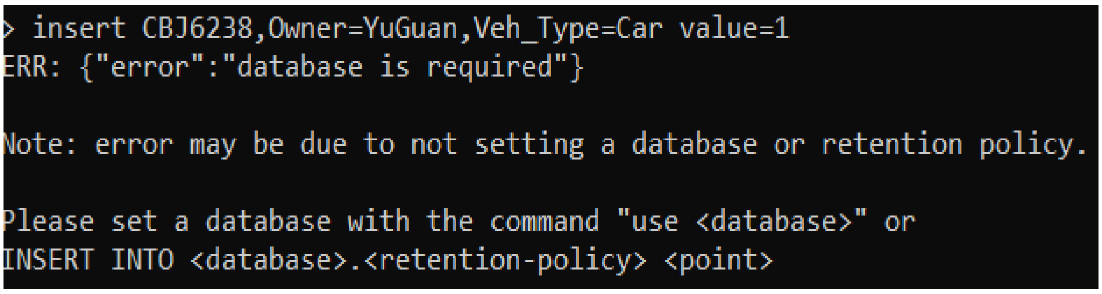
Meanwhile, after you open “influx.exe”, you can write or read from this database. Image below shows the first impression of the window after running influx.exe. The version of Influx Database is shown and ready to run instruction.



```
D:\hello\influxdb-1.7.10-1\influx.exe
InfluxDB shell version: 1.7.10
>
```

Figure 3.18: “influx.exe” first impression.

A database named “Vehicle_check” is built for this project. Writing any data will not work unless a database is chosen to access it. Image below shows the error if a data is inserted without setting a database.

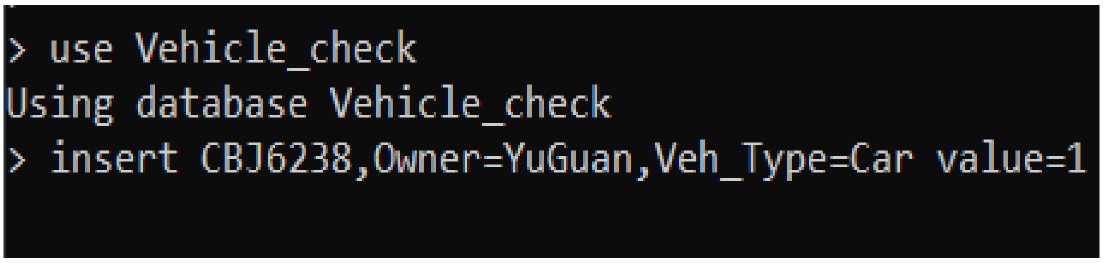


```
> insert CBJ6238,Owner=YuGuan,Veh_Type=Car value=1
ERR: {"error":"database is required"}

Note: error may be due to not setting a database or retention policy.

Please set a database with the command "use <database>" or
INSERT INTO <database>.<retention-policy> <point>
```

Figure 3.19: Error: database is required.

3.5.2 Writing Information Into Database

After setting a database, data could be inserted into the database. In this project, important criteria to record while registering a vehicle is the owner’s name, vehicle type and, car plate number. Image below shows a sample data is inserted into the database.



```
> use Vehicle_check
Using database Vehicle_check
> insert CBJ6238,Owner=YuGuan,Veh_Type=Car value=1
```

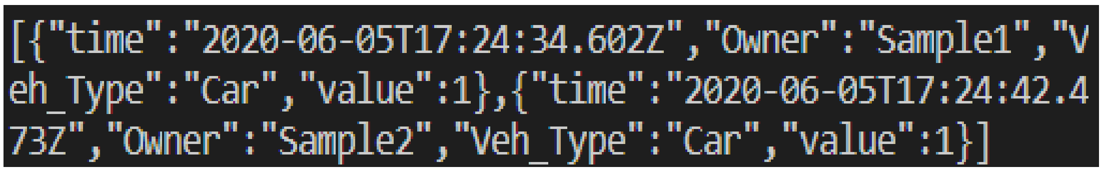
Figure 3.20: Inserting Data into database

Note that there will be no message or notification to notify that this line of query is executed. The data will be written into the database after executed.

3.5.2.1 Duplicated Messages

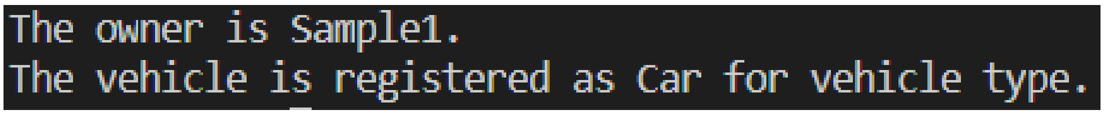
If the same query is executed mistakenly again, the record will duplicate, which means there will be two similar information inside the database. Or in the other hand, the same vehicle information is registered more than once or updated again. In this case, whenever the vehicle’s information is requested, the original result feedback given by the database is duplicated too.

To resolve this issue, feedback from the database had to be re-arranged again. This again pinpointed the importance of re-arranging database feedback results. This means if two or more messages are given through database feedback, the program only takes in the first feedback message to be withdrawn and show in the terminal as the desired message. Refer to Figure 3.15: Desired Result from Influx Query. Figure below shows the terminal message when there is duplicated information of a sample vehicle and the next shows how the app re-arrange the message and only shows the first one as output.



```
[{"time":"2020-06-05T17:24:34.602Z","Owner":"Sample1","V
eh_Type":"Car","value":1},{"time":"2020-06-05T17:24:42.4
73Z","Owner":"Sample2","Veh_Type":"Car","value":1}]
```

Figure 3.21: Duplicated feedback

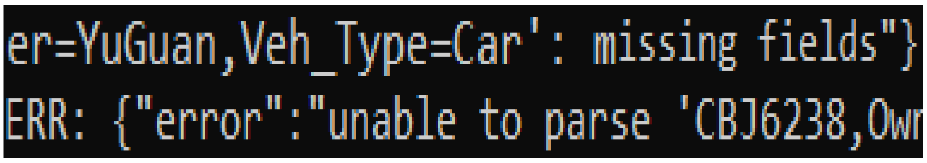


```
The owner is Sample1.
The vehicle is registered as Car for vehicle type.
```

Figure 3.22: Message Re-arranged

3.5.2.2 Influx Database Error Writing Data

The reason for setting value equals 1 in the end of the query is because Influx database does not allow inserting “null” or empty value for its parameter. Image below shows error inserting data if the value is empty or not mentioned. Refer to Figure 3.20: Inserting Data Into Database for the differences.

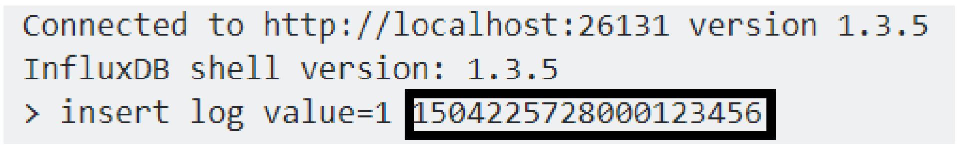


```
er=YuGuan,Veh_Type=Car': missing fields"}
ERR: {"error":"unable to parse 'CBJ6238,Owr
```

Figure 3.23: Error inserting data without value

3.5.2.3 Inserting Timestamp

There will be times that the system is not available or in maintenance. A specific timestamp needed to be inserted manually to make sure the registration time is accurate. Since Influx Database is a time-series based database, any information inserted will be given a current timestamp. If the desired timestamp is needed to be inserted manually, it must be separated from the field(s) by a space. Other than that, it must be in Unix time and are assumed to be in nanoseconds.[10] The figure below shows the sample query to insert a data with specified timestamp in Unix format. The timestamp is highlighted with a black rectangular box.



```
Connected to http://localhost:26131 version 1.3.5
InfluxDB shell version: 1.3.5
> insert log value=1 1504225728000123456
```

Figure 3.24: Sample Query To Insert Specified Timestamp

3.6 Program Installation

MATLAB Installation

1. Download the product to your computer, then locate and click the setup.exe.
2. Select ‘Get Add-ons’ from the ‘Add-ons’ drop-down menu from the MATLAB desktop. The Add-on files are in the “MathWorks Features” section.

Visual Studio Code Installation

1. Download the product to your computer, then locate and click the **setup.exe**.

Node.js and protocol Installation

1. Download Node.js from its official website, then locate and click the **setup.exe**. Afterward, open a command prompt then enter “node -v” to verify if Node.js is installed inside the working folder you wanted to do the project.
2. Open a command terminal on your working folder. Then type ‘npm install mosca –save’ and wait for the download to complete. Then type ‘npm install mqtt –save’ and wait for the download to complete.
3. Open a command terminal on your working folder. Then type ‘npm install –save influx’ and wait for the download to complete.
4. Open a command terminal on your working folder. The type ‘npm install -g json’ and wait for the download to complete.
5. Open a command terminal on your working folder. Then type ‘npm install exceljs’ and wait for the download to complete.

Microsoft Excel Installation

1. Login to you student email account on the device you wish to install Microsoft Office.
2. Navigate to Office 365 screen.
3. Click ‘install’ under the Install Office session. If requested to sign in after installation, use your student email to login instead of inserting a license key.

Influx Database Installation

1. Download the product to your computer
2. Locate and click the “setup.exe”. In the influx folder
3. Double click the “influxd.exe” to run the database
4. Double click the “influx.exe**”** to insert command.