



---

# SMART PARKING SYSTEM

---

Swe30011 – IoT Programming



MAY 24, 2019  
VASILIOS DASOPATIS

## IoT Smart Parking System

Introduction Of Application And Components.....	1
Hardware details.....	1
Software Being Used And Code explained + Data Management. ....	3
Python.....	3
Bash Script .....	8
Apache2 & PHP7 INSTALL .....	8
PHP / HTML / CSS / AJAX / MAIL-1.4.1 + NET_SMTP + Pear Package Manager.....	8
MariaDB.....	9
Arduino .....	10
Description and details of the APIs and Webservers you are using .....	12
IoT Networking – details of the potential IoT network including the motivation for choosing the protocol and network diagram .....	12
IoT Cloud computing – Potential cloud computing techniques for your IoT application .....	13
Potential Machine Learning and Advanced Data Analytics techniques that can be added to your IoT project .....	13
IoT Security challenges and solutions for your IoT project .....	14
Challenges, Limitations, and future work .....	14

## Introduction Of Application And Components

For my individual assignment, I have decided to construct a Smart Parking system.

This parking system is designed to track an individual's parking time. The system also displays if the parking spot is taken or is available via two LED'S, Red is taken Green is available, these LED'S will be fixed to the roof or a mounting mechanism on top of the Car Spot.

A Key feature of this system is that the price due for a user who has parked is automatically calculated by the following formula (Time\_In\_Minutes \* .166) This can be changed at a later date to any fee.

Another Major Feature is it has a Database with users Number plates, Email and Names, This table is queried by an administrator to send off emails to anyone who has parked. The email consists off the Number plate of their car and the price that is due.

I have also implemented a Current Weather bar which also tells you if it will be busy or not today at the car park.

## Hardware details

Hardware Name	Specs / Description	Justification
---------------	---------------------	---------------

Arduino Compatible Dual Ultrasonic Sensor Module	<p>Model: HC-SR04</p> <p>Pins:</p> <p>VCC = 5V power</p> <p>TRIG = Trigger Input (Digital Pin)</p> <p>ECHO = Echo Input (Digital Input)</p> <p>GND = Ground</p> <p>Measures Distance Up To 4.5M away using ultra sonic waves.</p> <p>I has a 15Degree field of view.</p>	For a smart parking system, we do not need a 100% highly accurate distance measuring system, we need a system that will check if there is something in between it and the floor. That's why we use the HC-SR04 Ultra sonic sensor.
Male To Male Cables	Model: WC6024	Used to connect the sensor modules to the Arduino via the breadboard
Arduino Uno	<p>Model: XC-4410</p> <p>8bit CPU</p> <p>16MHz clock speed</p> <p>2kb ram</p> <p>32kb flash storage</p>	Cheap and powerful enough board with GPIO and POWER pins needed to complete this assignment. It is very power efficient and can be powered via USB
Breadboard	<p>Model: PB8815</p> <p>Distribution holes (Power): 200</p> <p>Terminal Holes (Connections): 630</p>	<p>I could have used a much smaller breadboard</p> <p>Such as the PB8820 it has 300 connections and 100 power points, but I had my larger one on hand so I decided to use it.</p>
Defused LED RED	<p>Model: ZD0100</p> <p>Colour: Red</p> <p>2.3V power</p> <p>180Ohm resistor for 5VDC</p>	<p>Very cheap single colour led that requires LOW power.</p> <p>Perfect for setting up many different sensors and keeping the price down.</p> <p>Something to note, I used a 220Ohm resistor since that is all I had on me, worked perfectly fine.</p>
Defused LED GREEN	<p>Model: ZD0120</p> <p>Colour: Green</p> <p>2.3V power</p> <p>130Ohm resistor for 5VDC</p>	<p>Again it's a VERY cheap single colour led that requires a small amount of power.</p> <p>Again I used a 220ohm resistor because its all I had, in saying this it doesn't make to much of a difference</p>
220OHM resistor	<p>Model: RR0556</p> <p>Used with the LED'S</p>	<p>These are the resistors I had in my Arduino set, so I decided to use them as I didn't want to go out and buy a new set of resistors.</p> <p>They are really cheap (55cents for 8) and easy to implement.</p>
Raspberry Pi	<p>Model: XC9000</p> <p>5V Input</p> <p>1x LAN ethernet port</p> <p>4X USB Type A ports</p>	<p>A Much more powerful board for the data storage / processing of the website.</p> <p>This Arduino is the middle man between the Arduino and any processing that needs to be done / the cloud.</p>

	1X micro SD storage slot 1X HDMI 1GB RAM 1.2GB Quad core CPU Bluetooth V4.0 Low energy	I used this because its relatively cheep and can implement up to 4 different sensors at the once for future expansion. It has WIFI and ETHERNET connectivity so we can set it up wirelessly to a network of devices. It can have a VERY large storage inside of it which is perfect for saving data.
Printer Cable	No model number.	This is used for the serial connection between the Arduino to the Raspberry pi The length can vary depending on how far the Arduino is from the Raspberry PI.
Power Cable for raspberry pi	No Model Number	A Dedicated power cable for the raspberry pi

## Software Being Used And Code explained + Data Management.

I will start with the data management part first, it will be covered in the code part of this segment of the report.

Firstly we are sending code form the Arduino to the raspberry pi using JSON, this makes communication SO much easier. We send a json object that is serialized on the Arduino, receive and decode it on the raspberry pi.

In terms of the database, in the segment below I have the 3 table I use in my MariaDB and all the querys explained so I wont repeat myself.

### Python

**DEPENDANCIES: mysql.Connector, MariaDB**

**Python is in charge of retrieving data, processing it and then inputting the data into a database.**

```

1 import mysql.connector as mariadb
2 from mysql.connector import Error
3 from mysql.connector import errorcode
4 import sys
5 import serial
6 import json
7 import random
8 import time
9
10 vacancy = None
11 OutPut = None
12 parkedduration = None
13 parked = None
14 parkedprevious = 0
15 ArrayLength = 0

```

#### Lines 1-8

These are simple import statements, we want to use different modules from different installed packages.

The main ones to note are MySQL Connector, Serial, Json, random and time. The others are for console output and debugging

Lines 10-15: Simple variable setup with default values, I do this so I can access them globally thought my code

```

17 #DB connection variables
18 sys.stdout.write("-----Starting Database Connection-----\n" )
19 connection = mariadb.connect(user="pi", password="root", database="individual")
20 cursor = connection.cursor(buffered=True)

```

#### Lines 17 – 20

I setup a basic start of database connection output on line 18

**Line 19 and 20** are the 2 connection statements. 19 is the actual connection with the username / password and the database we are using and line 20 is the connection cursor.

```

22 #Check that all our tables exist, and set them up if they dont
23 try:
24     query = "CREATE TABLE IF NOT EXISTS CurrentlyParked (Parked int(1) NOT NULL, Primary KEY(Parked));"
25     cursor.execute(query)
26     query = "CREATE TABLE IF NOT EXISTS ParkedDuration (ID int(10) AUTO_INCREMENT, ParkTime DOUBLE(6,2) NOT NULL," /
27             "NumberPlate varchar(6) NOT NULL, Primary KEY(ID));"
28     cursor.execute(query)
29     query = "CREATE TABLE IF NOT EXISTS ParkingUsers (NumberPlate varchar(6) NOT NULL, Email varchar(30) NOT NULL," /
30             "Name varchar(15) NOT NULL, Primary KEY(NumberPlate));"
31     cursor.execute(query)
32     query = "INSERT IGNORE INTO ParkingUsers (NumberPlate, Email, Name)VALUES" \
33             "('{1CD2SD','wardude202@hotmail.com', 'BillyDas'),' \
34             '({1LS2MD','billy.das@hotmail.com', 'BillyDasWork'),' \
35             '({0LD9KD','10115315@student.swin.edu.au', 'Student Email'),' \
36             '({6DF4HJ','SpamBam1234@hotmail.com', 'SpamEmail'),' \
37             '({8KS5LK','ShopAndShipIt@hotmail.com', 'ShopAndShipIt'});";
38     print(query)
39     cursor.execute(query)
40     connection.commit()
41 except:
42     print("Tables Created And Data Inserted")
43 # Serial communication

```

**Lines 24-37** are the initial setup query's to the MYSQL database.

The tables and what they mean will be described later in the report.

We want to create 3 tables if they don't exist and we also want to insert some dummy data into the table, in my case I put my personal emails as dummy data as they must be valid, they are required from the WEB UI.

**Line 23, 41 and 42** are just part of a try statement, this will try execute the code, if it has an exception in our case it always should unless the tables are dropped, it will trigger a print in the terminal

```
43 # Serial communication
44 try:
45     ser = serial.Serial('/dev/ttyACM0', 115200)
46     ser.flushInput()
47     ser.flushOutput()
48     print("Serial Uplink Established")
49 except:
50     print("Yo we couldnt connect to the arduino")
51
52 #Checks if there is a car parked
```

#### Lines 44-50:

These are setting up the communication with the Arduino, we use 115200 baud for serial communication as that's what the sensor communicates best with.

We create a serial connection, flush the input and output so we don't have old data, if this query stack fails it will fire an exception saying you couldn't connect to the Arduino.

```
52 #Checks if there is a car parked
53 def get_parked_status():
54     global OutPut
55     try:
56         Temp = ser.readline()
57         Temp = Temp.decode('utf-8')
58         OutPut = json.loads(Temp)
59     except (json.decoder.JSONDecodeError, UnicodeDecodeError):
60         print("Error With Input")
61     global vacancy
62     vacancy = OutPut["Vacancy"]
63     time.sleep(1)
64     return vacancy
```

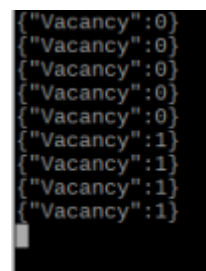
#### Lines 52-64:

These following lines are in charge of getting the current parked status.

I created a little function that will return the Vacancy, this is received via serial here is an example of what the serial output looks like, we have 0 for when the car spot is vacant and 1 when it is taken.

The way we receive data is it is sent in a JSON format, so we read the json output from the Arduino and decode it, if this fails we get an error "Error With Input"

On **Line 61**, I reference the global variable vacancy, making sure I can change it globally, I set the variable to the output that we just read and sleep the pi for 1 second. Followed by returning the variable we just set.



```
{"Vacancy":0}
{"Vacancy":0}
{"Vacancy":0}
{"Vacancy":0}
{"Vacancy":0}
{"Vacancy":1}
{"Vacancy":1}
{"Vacancy":1}
{"Vacancy":1}
```

```

68 while True:
69     get_parked_status()
70     if vacancy == 1:
71         query = "UPDATE CurrentlyParked set parked = 1 where parked = 0"
72         cursor.execute(query)
73         connection.commit()
74         print("Car Is Parked")
75         if parkedprevious == 0:
76             global parkedprevious
77             global starttime
78             starttime = time.time()
79             parkedprevious = 1
80     elif vacancy == 0:
81         query = "UPDATE CurrentlyParked set parked = 0 where parked = 1"
82         cursor.execute(query)
83         connection.commit()
84         print("Car Is Not Parked")
85         if parkedprevious == 1:
86             global parkedduration
87             endtime = time.time()
88             parkedduration = endtime - starttime
89             parkedduration = parkedduration / 60
90             parkedduration = str(parkedduration)
91             print("Parked For " + parkedduration + " Minutes")
92             parkedprevious = 0
93

```

This next code segment is VERY long so bare with me on the explanation.

#### Line 69:

This is where we call the function I created previously (**Lines 53-64**), we get the output and then begin an if statement on **line 70**.

If the **Vacant variable is set to 1** we query the database, and set the CurretnlyParked table, parked row to 1. we then print in the output Car Is Parked (this is to help debugging)

The next thing we do is check a variable called Parked Previous the first time re run the code this will automatically be set to 0, this is because if they user is in the car spot they want to start the timer.

So we run though the parked previous if statement, we reference the 2 variables (start time and parked previous)

Start time is then set to the current time and parked previous is set to 1.

#### Line 80:

This line is if the **Vacant variable is set to 0** we query the database, and set the CurretnlyParked table parked row to 0, and we print Car Is Not Parked.



We then check if the parked previous is set to 1, PLEASE NOTE. If the user IS parked and then leaves, this is when this part of the if statement will trigger. We reference the parkedduration variable, we then get the Current time and set it to endtime.

Okay so this is where we calculate how long the customer has been parked for, earlier we set the start time and we just set the end time, so we subtract the end time by the start time and re set that as the parkedduration variable. #NOTE# this is in seconds, so we divide the result by 60 to turn it into minutes and then set it as a string so we can insert it into the database. We also print out how long the user has parked for and lastly set the parkedprevious variable to 0

```
92     if parkedduration != None:
93         global parkedduration
94         comparison = float(parkedduration)
95         if not(comparison < .1):
96             global ArrayLength
97             query = "SELECT NumberPlate FROM ParkingUsers"
98             cursor.execute(query)
99             NumberPlates = cursor.fetchall()
100             ArrayLength = len(NumberPlates)
101             NumberChosen = random.randint(0,ArrayLength)
102             NumberChosen = NumberChosen - 1
103             NumPlateChosen = NumberPlates[NumberChosen]
104             NumPlateChosen = "".join(NumPlateChosen)
105
106             try:
107                 query = "INSERT INTO ParkedDuration (ParkTime, NumberPlate) VALUES" \
108                     "(" + parkedduration + "," + NumPlateChosen + ")"
109                 cursor.execute(query)
110                 connection.commit()
111             except mysql.connector.Error as error :
112                 print("Failed to work".format(error))
113                 print("Placed Park Time In Database")
114                 parkedduration = None
115     cursor.close()
```

Last segment of code!

#### Line 92 – 95:

Okay so this code is triggered then the parkedduration IS NOT set to none. After this is it gets the parkedduration and converts it to a float and sets it to a variable called comparison.

If checks if this is NOT less than .1 (minutes), this can be changed to any time you like.

This if statement is in charge of the MINIMUM limit people have to park for before they are entered into the database

#### Lines 96-104:

This part of the code is just a proof of concept, as we can't have real users and real numberplates we do is we use the dummy data I put into the database on **Line 32**. What we do is query the database for the length of it (how many user entries exist) we get the length of the array (which is the result) we then pick a random number between 0 and the array length and then subtract one from it the reason we subtract one is because a database auto incrementing table starts at 1 and an array starts at 0 so we must subtract one. We then select the numberplate they chose.

#### Line 106-115



We insert into the database the parked duration and the numberplate of the user. If we get an error it will output an error saying “failed to work”. It will then output Placed park time into database, set parkedduration to none and then close the connection to the database.

### Bash Script

```
#!/bin/bash
while :
do
python3 -W ignore comms.py
done
```

This is a simple bash script loop that calls the python script coms.py and ignores any errors.

### Apache2 & PHP7 INSTALL

Firstly we run the command, **sudo apt-get install apache2**

Secondly we then run the command **sudo apt install php7.0**

I also make web page in /var/www/html called index.php with the following code <?php phpinfo(); ?>

That’s how you install the APACHE2 web server and PHP7 code language.

### PHP / HTML / CSS / AJAX / MAIL-1.4.1 + NET\_SMTP + Pear Package Manager

Unfortunately the files are too long to take screenshots of so I will summarise each file for you.

**Temperature.php** – This file is a simple file that gets the current temperature in Melbourne, and displays it, it also displays a message depending on the weather. If the temperature is over 25 degrees Celsius it will display the message Its Probably Going To Be Busy Today, if it below 25 degrees It will say It Shouldn’t Be Busy Today

**SpotStatus.php** – This file connects to the database and retrieve the currently parked values (0 or 1), if the value is 1 it will print Out Spot Taken else it will say Spot Is Free (this is using AJAX)

**ReceptEmail.php** – Okay so this file is rather long, Firstly this page has bootstrap to it for the visuals, we connect to the database, we then begin to print out a table with headings, **Number Plate / Park Time and lastly Fee**, once this has been printed out we run a query to get all the park users Numberplates and park times, we only print out the last 10 results and also calculate the FEE (the fee is time in minutes \* .166, this turns out to be \$10/hr.

After this we have a form where it has a text field where you enter a numberplate.

After this we have an error checker where it checks if the Numberplate variable from the form is set and is not blank, if this is true we run a query where we get the ParkingUsers details where their numberplate matches in the database. For each of these results we run ANOTHER query that gets the Numberplate of the user. We also calculate the FEE again.

## MAIL SYSTEM

this part is the fun part.

So this following part (lines 82-114) are for the Mail Server setup and sending of the mail.

So firstly I have a stock message that is sent to the user we get the Numberplate and the price of the user depending on what numberplate was entered into the form.

We then set WHO is sending the email [billydasuni@gmail.com](mailto:billydasuni@gmail.com) and we also set the Email subject "Parking Receipt", in an array we set the details (Who is it the email from, who is it going to (this is also dependant on which numberplate the admin puts into the form) and lastly the subject.

Next we setup the SMTP server for the EMAIL. I use the Simple Mail Transfer Protocol as it's the easiest to setup and use, and is supported by Gmail. So I set the SMTP server to Gmail. The HOST is `ssl://smtp.gmail.com` (Gmail's host servers) the port for emails is 465 and the email has authentication, so we set this to true and enter the username and password.

Next we send the email! Quite simply we send the email with the details entered and the result (if it sent or had an error) is set to the variable \$mail, this is then checked and an error message or successful message is printed depending on the output.

**Price.php** – This page simply connects to the database and grabs the latest 10 park details from the database and prints them out with the fee they need to pay in a table (used with ajax)

**LastDuration.php** – This page gets the LATEST park duration in the carp parking spot (the length) and prints it out.

**Index.html** – Just a nice clean intro page with my details.

**DataBaseCon.inc** – Just a file that contains the database connection details, if I ever need to change them I can change them here instead of each of the files separately.

**CarStatusPage.php** – This page prints out the temperature.php page, it prints out a table called Parking Status which is the Spot Status and the Duration of the last park. It also prints out the Print.php table

**Ajax.js** – This is how we reload the data dynamically. The 3 pages that use ajax is SpotStatus.php, LastDuration.php and Price.php with a 1 second delay

## MariaDB

Firstly here are our 3 tables we use.

```

MariaDB [individual]> describe ParkedDuration;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| ID    | int(10) | NO | PRI | NULL | auto_increment |
| ParkTime | double(6,2) | NO | | NULL | |
| NumberPlate | varchar(6) | NO | | NULL | |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)

MariaDB [individual]> describe ParkingUsers;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| NumberPlate | varchar(6) | NO | PRI | NULL | |
| Email | varchar(30) | NO | | NULL | |
| Name | varchar(15) | NO | | NULL | |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.02 sec)

MariaDB [individual]> describe CurrentlyParked;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Parked | int(1) | NO | PRI | NULL | |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

ParkedDuration, this table contains a unique ID of each time someone parks, how long they parked for and their number plate.

ParkingUsers: this is the table of each user who is able to park, it has their NumberPlate, email and name.

CurrentlyParked.

One entry, either a 1 or 0

1=spot taken 0=Spot Free

Now you know what each table is for, I can show you how to install MariaDB.

Firstly we run the command **sudo apt-get install mariadb-server** this will install the database.

Next we run **sudo mysql** this will connect to it

Lastly this is the command you run to make a new database user with all privileges.

**GRANT ALL PRIVILEGES ON \*.\* TO 'username'@'localhost' IDENTIFIED BY 'password';**

## Arduino

```

#include <ArduinoJson.h>

//Variables needed
#define TRIG 12
#define ECHO 13
#define USMAX 3000
double timeset = 0;
#define GREEN 11
#define RED 10
//Defines the JSON object.
const int capacity = JSON_OBJECT_SIZE(2);
StaticJsonDocument<capacity> doc;

//Initial Sstartup (baud and port setup function)
void setup() {
  Serial.begin(115200); //open serial port
  PortSetup(); //set up ultrasonic sensor
}

//PIN SETUP
void PortSetup(void) {
  pinMode(ECHO, INPUT);
  pinMode(TRIG, OUTPUT);
  pinMode(GREEN, OUTPUT);
  pinMode(RED, OUTPUT);
  digitalWrite(TRIG, LOW);
}

```

This first big of code is the base setup of the Arduino, we have the pins defined, the object setup for the JSON serial communication and the Port Setup function I made.

```

//Program Loop
void loop() {
  int distance; //variable to store distance
  distance=usonic(11600)/58; //distance in cm, time out at 11600us or 2m maximum range

  //how we check the distance of the sensor. (set 20 to anything you like MAX IS 449)
  //This also contains the data set into the Json Object.
  if (distance < 20) {
    doc["Vacancy"].set(1);
    digitalWrite(REDF, HIGH);
    digitalWrite(GREEN, LOW);
  } else {
    doc["Vacancy"].set(0);
    digitalWrite(REDF, LOW);
    digitalWrite(GREEN, HIGH);
  }
  //wait a bit so we don't overload the serial port
  delay(1000);

  //sends the serial data.
  serializeJson(doc, Serial);
  Serial.println("");
}

```

This part of the code is the one reliable for the DISTANCE of the sensor (how far does the car have to be before it will say the spot is taken) as you can see its currently 20 which is 20CM this can be changed up to a max of 449cm.

In the IF loop you set the car spot status (1 is taken 0 is free)

We have a delay of 1 second aswell so we don't overload the serial communication

Lastly we serialise the JSON data and send it.

```

//ULTRA SONIC SENSOR READING DATA TAKEN FROM JAYCAR
long usonic(long utimeout) { //utimeout is maximum time to wait for return in us
  long b;
  if(digitalRead(ECHO)==HIGH){return 0;} //if echo line is still low from last result,
  digitalWrite(TRIG, HIGH); //send trigger pulse
  delay(1);
  digitalWrite(TRIG, LOW);
  long utimer=micros();
  while ((digitalRead(ECHO)==LOW)&&((micros()-utimer)<1000)){ //wait for pin state to c
  utimer=micros();
  while ((digitalRead(ECHO)==HIGH)&&((micros()-utimer)<utimeout)){ //wait for pin state
  b=micros()-utimer;
  if(b==0){b=utimeout;}
  return b;
}
}

```

**THE FOLLOWING CODE WAS TAKEN FROM JAYCAR. This is the code jay car provide us for the sensor, I take NO credit for this code.**

This code basically gets the delay between each pulse the sensor sends out, it also has some error checking, such as if no reaction is received. That's the if (b==0) {b=utimeout} that checks if no reply was sent.

## Description and details of the APIs and Webservers you are using

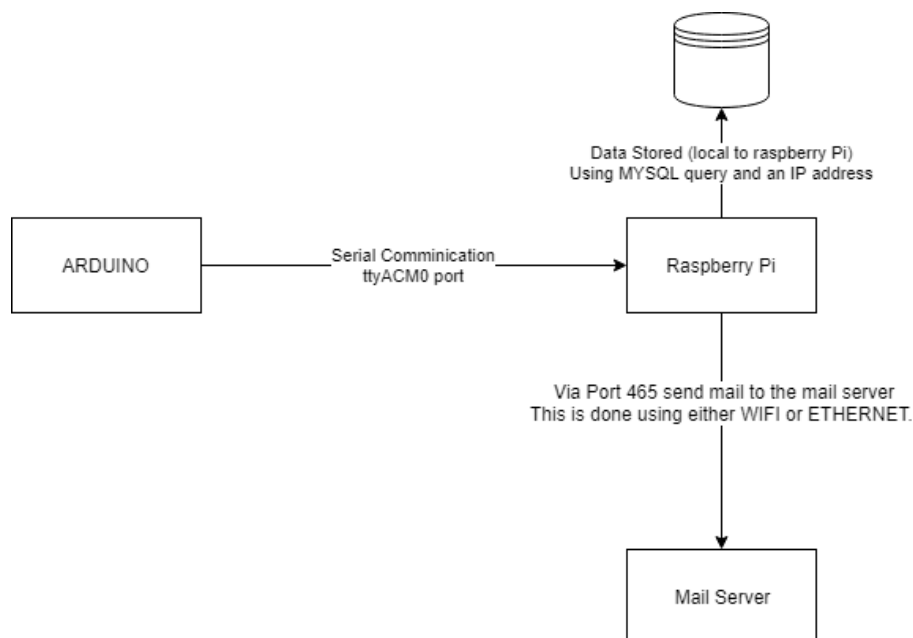
I use the Open weather maps API which is free, I get the temperate of Melbourne CURRENTLY and print It out as well as show a statement on the judging of how busy it will be on the day.

I DID SPEND 4 days trying to get a MAP to be displayed locally of the closes car parks, the way I was trying this was I used google maps as the base and the google places to get the data of the closes car parks within 1.5km of the user, I got the users location by running a bit of code in PHP that gets the users latitude and longitude position which is then sent via the GET method to the browsers URL where the google maps JavaScript function will get the result, this part was stupidly hard and I couldn't get it working unfortunately, feel free to email me for more details if you like, I can show you my google API account status and history.

## IoT Networking – details of the potential IoT network including the motivation for choosing the protocol and network diagram

So firstly the EMAIL protocol we use was SMTP (Simple mail transfer protocol) the reason I use this is well because in the name it states it's a simple protocol to use, easy to implement and send emails, the port I used was 465 as that's the default SMTP port that IANA (INTENET ASSIGNED NUMBERS AUTHORIT) Set.

I have done my best to attach a diagram of what the network CURRENTLY looks like. I will discuss future plans in a later part of the report.



I chose this project in about week 3, I wrote about it in my individual report and wanted to build this system as it would give me a clear ground base knowledge into many of the systems, I was thinking of controlling my air conditioner but as I have NO incite into the protocols / signals that my aircon needs / receives I decided to scrap that idea in fear of not getting anything to work

## IoT Cloud computing – Potential cloud computing techniques for your IoT application

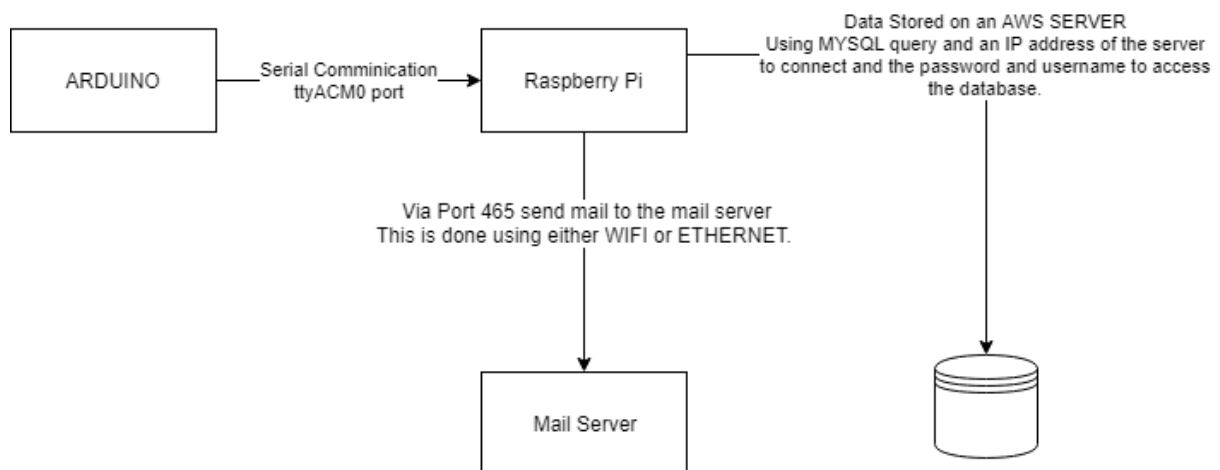
Due to learning the content rather late my plan was originally to host the MYSQL database on amazons AWS platform, I would run a MySQL database, I would use a general / public license, I would also have to create a username and password as well get the IP address of my database and change the details in the python file and the DataBaseCon.inc file.

I would simply create the exact same tables on the AWS cloud storage, I don't require much if any computing for this project.

Aside from this I could possibly move the ENTIRE web setup to the same service, just I'm not sure if this would be necessary as hosting your own server at the apartment complex would be easier and cheaper, all you need is a bit of bandwidth for a few users to check the carpark. BUT it is ALWAYS possible to host on an AWS server at any time, all you need to do is copy the web pages into the website host, you'd need to make sure it can still connect to the database as well.

If this project was a MUCH larger scale project where its multiple businesses that would log into the service and use over 50 or so parking spots, then that's where I would host it on a cloud service as they has FAR more bandwidth.

Here is a theoretical diagram of what this new system would look like.



## Potential Machine Learning and Advanced Data Analytics techniques that can be added to your IoT project

As a really far stretch, we could eventually implement a system that checks the date and the events running such as Christmas or specific public holidays where it is more likely to be busy, this can then output an email or a message to all the users who have used the parking spot in the last 2 months saying "hey its probably going to be busy because X Y Z events" as for machine learning I'm not so sure this part would probably be a mix of machine learning and data analytics.

AS A LARGE SIDE NOTE.

This project is ideal with automatic numberplate recognition from a camera from EACH car parking spot. This is then used for each user as they park. This project was more of a proof of concept.

## IoT Security challenges and solutions for your IoT project

Someone could access the database via penetrating the security and seeing peoples personal details such as the emails / and numberplates that equate to the emails. We would want some sort of encryption in the communication between the database and the raspberry pi (either AWS or LOCALLY)

Possibly someone could create a fake numberplate and hold it in the parking spot for a weeks on end, this would charge that user A LOT of money, figuring out a system to sort this issue out would be vital.

## Challenges, Limitations, and future work

The first MAJOR thing is the limitations, setting up numberplate recognition / having an automated system for that as well as not having a variety of vehicles to test that system on would be a major limiting factor

Setting up that system in the future would be ideal as the system would be trailed and run.

Challenges was setting up a consistent work environment for my solution to work, if this system was to be an official product I would need to figure out a way to built it in a small contained unit.

May I just reinforce. THIS SYSTEM is a proof of concept as it will work.