|  |  |  |
| --- | --- | --- |
| **Smart car parking System** | | |

**PROBLEM DEFINITION:**

The project involves integrating IoT sensors into public transportation vehicles to monitor ridership, track locations, and predict arrival times. The goal is to provide real-time transit information to the public through a public platform, enhancing the efficiency and quality of public transportation services. This project includes defining objectives, designing the IoT sensor system, developing the real-time transit information platform, and integrating them using IoT technology and Python.

**DESIGN THINKING:**

**Smart car parking:**

The proposed smart parking lot circuit will be equipped with several sensors, inexpensive microcontrollers and Wi-Fi module using which a car / any vehicle owner can check if there is a vacant space in a parking lot using his / her phone or tablet or even on computer.

The number of vacant spaces in the smart parking lot can be viewed from anywhere in the world using a URL link or the user can scan a QR code. The scanned / shared URL can be browsed on any web browser to know how many empty parking spot exist in real time.

O**bjective**

Project Objectives: Define specific objectives such as real-time parking space monitoring, mobile app integration, web browser and efficient parking guidance. IoT Sensor Design: Plan the design and deployment of IoT sensors in parking spaces to detect occupancy and availability. Real-Time Transit Information Platform: Design a mobile app interface that displays real-time parking availability to users. Integration Approach: Determine how Arduino uno will collect data from sensors and update the mobile app.

**Benefit of smart car parking system:**

An IoT-based car parking system using Arduino can help drivers find available parking spots. The system uses an infrared sensor to detect the presence or absence of a car. It can also display the status of parking slots, including how many are filled and how many are empty**.**

**Some benefits of an IoT-based car parking system using Arduino include:**

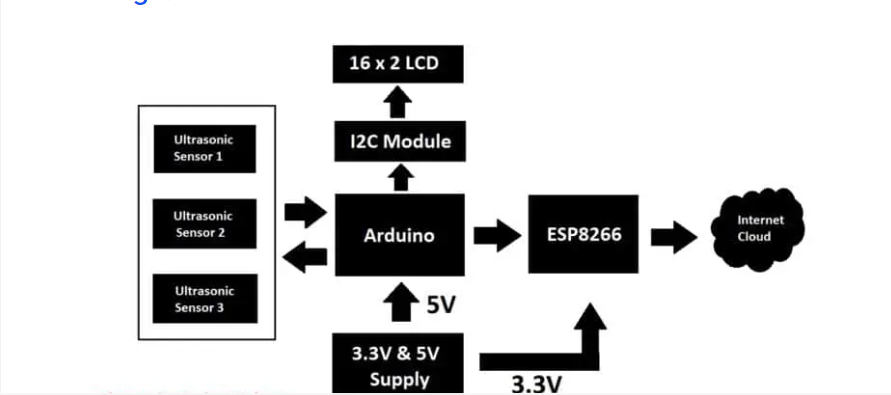
Saving time and reducing hassle for drivers.

Helping drivers avoid traffic congestion.

Allowing drivers to search for and reserve parking spots remotely through their smartphones.

Informing drivers of parking availability in advance.

**FLOW DIAGRAM:**



**System Model:**

The circuit we are going to build will be based on the above architecture. An inexpensive Arduino board is going to be the brain of the project.

 A 16 x 2 LCD is utilized for displaying the number of vacant spots locally (without internet). An I2C module is utilized for driving the LCD with just four wires so that GPIO pins can be saved for interfacing the sensors and other modules.

There are three ultrasonic sensors for detecting 3 cars / vehicles on the parking spot, we are using ultrasonic sensors instead of IR based sensors because if the parking lot is situated outdoors, infrared light from sunlight may interfere with IR sensors and may give incorrect detection of the vehicle, whereas ultrasonic sensor acts like a mini radar and environmental factors affecting its functionality is minimal.

An ESP8266 Wi-Fi module is used for internet connectivity which sends the parking lot’s data to a cloud server where general public can view the data in real time. A power supply module is utilized which provides 5V and 3.3V for Arduino, ultrasonic sensors and ESP8266 Wi-Fi Module.

The internet cloud service we are going to use is called “Thingspeak” where the parking lot’s data to be sent, stored and displayed in real time. This concludes the block diagram.

**Power supply unit:**

A power supply module takes 9V to 12V DC from a wall adapter and converts in to 5V and 3.3V outputs, the 5V output from power supply module is directly connected to 5V pin of Arduino and GND of power supply is connected to GND of Arduino.

Similarly, 3.3V from the power supply unit is connected to 3.3V Vcc of ESP8266 (it operates strictly on 3.3V and 5V will kill the module), the ground of power supply is connected to ground of ESP8266.

Make sure you that have inserted the voltage select jumpers correctly.

**Generic ESP8266 Wi-Fi module:**

This project utilizes a generic ESP8266 Wi-Fi module for internet connectivity. The ESP8266 is actually a miniature microcontroller board and just like Arduino the ESP8266 need a program code to perform its intended function.

It uses UART protocol to communicate with Arduino board; the baud rate we are going to set for UART is 115200 bits per second.

**Ultrasonic Sensor HC – SR04:**

The sensor we are going to use for detecting a parked vehicle on its parking spot is called HC – SR04 which is an ultrasonic sensor module.

The ultrasonic sensor module generates ultrasonic sound at around 40 KHz, these sound waves are inaudible to human beings and propagate through air and if the ultrasonic sound wave hits an obstacle, it reflects back to sensor just like radars.

If a car or any vehicle is parked, the ultrasonic sound waves hit the parked vehicle and the sensor module detects the reflection and thus existence of a vehicle on a parking spot is detected.

The ultrasonic sensor module has four pins, Vcc, GND, trigger and echo. The Vcc is connected to 5V supply and GND is connected to GND of the supply.  When we apply “HIGH” signal to trigger pin for 10 microseconds, the module generates ultrasonic sound from one of the transducers, when the sound wave hit back the other transducer, the echo pin gets “HIGH” and this signal is detected by Arduino.

The time taken between generating and detecting the sound wave is calculated and thus a parked vehicle is detected.

**LCD display module 16 x 2:**

In this project we are using a 16 x 2 LCD display for displaying parking lot’s data locally without the need for internet. The LCD is driven by an I2C adapter module to reduce the number of wires to four; otherwise you need to connect up to 16 wires to Arduino just to drive the display. If the LCD occupies most of the I/O pins, then there won’t be any pins left for the sensors.

The I2C module has 16 pins at the output and just four at the input: Vcc, GND, SDA and SCL. The SDA and SCL are I2C bus pins which are connected to A4 and A5 pins of Arduino respectively and it operates on 5V.

You can control the contrast of the display by adjusting the trim pot on the I2C adapter module. This concludes about the circuit diagram.

**How to setup your Thingspeak account?**

We are using a (free) cloud service called Thingspeak where we will send parking lot’s data to share it with public.

**How to operate the machine?**

* After completing the circuit setup and Thingspeak account setup, power the circuit ON.
* Here we have three parking spots and each parking spot’s status is displayed along with the number of free parking spots.
* When you bring an obstacle like a junk box near to the ultrasonic sensor, the machine will count it as an occupied parking spot and this information will be updated to Thing speak as well as on the display.
* “OK” status signifies that a car / vehicle is occupied and “NO” signifies the parking spot is empty.

**Thing speak public view on smart phone:**

The URL of the public view page is converted into a QR and this can be done using any online URL to QR code converter tool. This QR code can be placed at the parking spot or anywhere else, so that the users can scan and bookmark the URL and when they are visiting the parking spot, users can open the link and see how many vacant spaces exist and if this parking lot is full they can move to another.

Future:

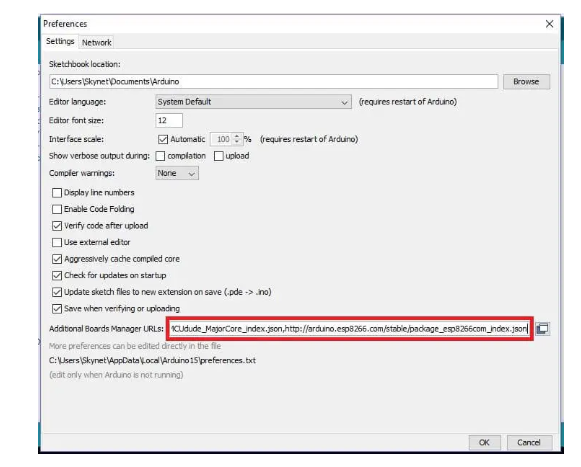
**Website Access**: Additionally, display the QR code on website, so anyone visiting the website can access the QR code for scanning. This provides an alternative way for users to access parking information.

**upload code to ESP8266?**

To upload the given code to ESP8266 we need 2 things:

1. ESP8266 board package (Software).
2. An ESP8266 programmer (Hardware).

**Download ESP8266 board package?**

* Copy this link: <http://arduino.esp8266.com/stable/package_esp8266com_index.json>
* Now open Arduino IDE and click on **File > Preferences**.
* A window will open like this
* 
* Type “ESP8266” on the box as shown and you will get installation option, select the **latest version** and click install.
* Now the IDE will download the necessary packages and this could take more than 5 minute to complete.
* Now go to **Tools > Board > ESP8266 boards > select “Generic ESP8266”.**
* Now, copy the given ESP8266 program code and paste it on to Arduino IDE software.
* Now press compile button (Green tick button). The compilation of code may take more than couple of minutes and be patient. If the compilation failed please check whether have you selected the “Generic ESP8266 Module” in the board option or not.
* After successful compilation of code, now it’s time to upload the code to ESP8266.

**2)** **How to upload code to ESP8266:**

To program ESP8266 we need a programmer module

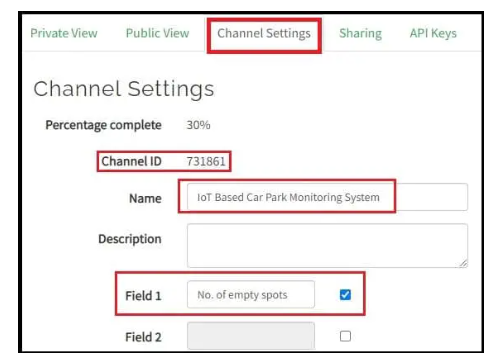
Insert the ESP8266 on the programmer module as shown below and insert it to your PC’s USB port and press upload.

Once the code is successfully uploaded you will see the following info, now you may insert the ESP8266 to the main circuit:

**3) setup your Thingspeak account?**

We are using a (free) cloud service called Thingspeak where we will send parking lot’s data to share it with public.

* First you need to sign up for Thingspeak:
* Enter the credentials it asks for and create a new channel and do the following to your new channel:



* Go to **channel settings** and enter the things as shown above and take note of your channel ID which we need to enter it in the program code.
* Scroll down and **click save** to save the changes.
* Now click on **API keys tab** and you will see your keys as illustrated below. API keys are responsible for writing and reading the data to your Thingspeak account.
* Go to **channel settings** and enter the things as shown above and take note of your channel ID which we need to enter it in the program code.
* Scroll down and **click save** to save the changes.

Now click on **API keys tab** and you will see your keys as illustrated below. API keys are responsible for writing and reading the data to your Thingspeak account.

* Take note of your “write API key” which needs to be entered in the program code and read API key is not used in this project.
* Now go to sharing tab and click on “share channel view with everyone”, this makes your channel visible to those who have the URL of “public view” page.

**Program code for Arduino:**

#include <LiquidCrystal\_I2C.h>

#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11);

LiquidCrystal\_I2C lcd(0x27, 16, 2); // set the LCD address to 0x27 for a 16 chars and 2 line display

const int trig\_1 = 2;

const int echo\_1 = 3;

const int trig\_2 = 4;

const int echo\_2 = 5;

const int trig\_3 = 6;

const int echo\_3 = 7;

float distanceCM\_1 = 0, resultCM\_1 = 0;

float distanceCM\_2 = 0, resultCM\_2 = 0;

float distanceCM\_3 = 0, resultCM\_3 = 0;

long Time\_1, Time\_2, Time\_3;

float car\_1, car\_2, car\_3;

float Dist\_1 = 8.0, Dist\_2 = 8.0, Dist\_3 = 8.0;

int total = 0, timer\_cnt = 0;

void setup()

{

mySerial.begin(115200);

pinMode(trig\_1, OUTPUT);

pinMode(trig\_2, OUTPUT);

pinMode(trig\_3, OUTPUT);

pinMode(echo\_1, INPUT);

pinMode(echo\_2, INPUT);

pinMode(echo\_3, INPUT);

digitalWrite(trig\_1, LOW);

digitalWrite(trig\_2, LOW);

digitalWrite(trig\_3, LOW);

lcd.init();

lcd.backlight();

lcd.setCursor(0, 0);

lcd.print(" IoT CAR PARK");

lcd.setCursor(0, 1);

lcd.print(" MONITOR SYSTEM");

delay(2000);

lcd.clear();

}

void loop()

{

total = 0;

car\_1 = sensor\_1();

car\_2 = sensor\_2();

car\_3 = sensor\_3();

lcd.setCursor(0, 0);

lcd.print("CAR1:");

if (car\_1 <= Dist\_1)

{

lcd.print("OK ");

}

else

{

total += 1;

}

if (car\_1 > Dist\_1) lcd.print("NO ");

lcd.print("CAR2:");

if (car\_2 <= Dist\_2)

{

lcd.print("OK ");

}

else

{

total += 1;

}

if (car\_2 > Dist\_2) lcd.print("NO ");

lcd.setCursor(0, 1);

lcd.print("CAR3:");

if (car\_3 <= Dist\_3)

{

lcd.print("OK ");

}

else

{

total += 1;

}

if (car\_3 > Dist\_3) lcd.print("NO ");

lcd.print("FREE:");

lcd.print(total);

if (timer\_cnt >= 50)

{

mySerial.print('\*');

mySerial.print(total);

mySerial .println('#');

timer\_cnt = 0;

}

timer\_cnt += 1;

delay(200);

}

float sensor\_1(void)

{

digitalWrite(trig\_1, HIGH);

delayMicroseconds(10);

digitalWrite(trig\_1, LOW);

Time\_1 = pulseIn(echo\_1, HIGH);

distanceCM\_1 = Time\_1 \* 0.034;

return resultCM\_1 = distanceCM\_1 / 2;

}

float sensor\_2(void)

{

digitalWrite(trig\_2, HIGH);

delayMicroseconds(10);

digitalWrite(trig\_2, LOW);

Time\_2 = pulseIn(echo\_2, HIGH);

distanceCM\_2 = Time\_2 \* 0.034;

return resultCM\_2 = distanceCM\_2 / 2;

}

float sensor\_3(void)

{

digitalWrite(trig\_3, HIGH);

delayMicroseconds(10);

digitalWrite(trig\_3, LOW);

Time\_3 = pulseIn(echo\_3, HIGH);

distanceCM\_3 = Time\_3 \* 0.034;

return resultCM\_3 = distanceCM\_3 / 2;

}

**Program code for ESP8266:**

#include "ThingSpeak.h"

#include <ESP8266WiFi.h>

//------- WI-FI details ----------//

char ssid[] = "SSID"; //SSID here

char pass[] = "PASSWORD"; // Password here

//--------------------------------//

//----------- Channel details ----------------//

unsigned long Channel\_ID =123456; // Your Channel ID

const char \* myWriteAPIKey = "ACBDE12345"; //Your write API key

//-------------------------------------------//

const int Field\_Number\_1 = 1;

String value = "";

int value\_1 = 0;

WiFiClient client;

void setup()

{

Serial.begin(115200);

WiFi.mode(WIFI\_STA);

ThingSpeak.begin(client);

internet();

}

void loop()

{

internet();

if (Serial.available() > 0)

{

delay(100);

while (Serial.available() > 0)

{

value = Serial.readString();

if (value[0] == '\*')

{

if (value[2] == '#')

{

value\_1 = value[1] - 0x30;

}

}

}

}

upload();

}

void internet()

{

if (WiFi.status() != WL\_CONNECTED)

{

while (WiFi.status() != WL\_CONNECTED)

{

WiFi.begin(ssid, pass);

delay(5000);

}

}

}

void upload()

{

ThingSpeak.writeField(Channel\_ID, Field\_Number\_1, value\_1,myWriteAPIKey);

delay(15000);

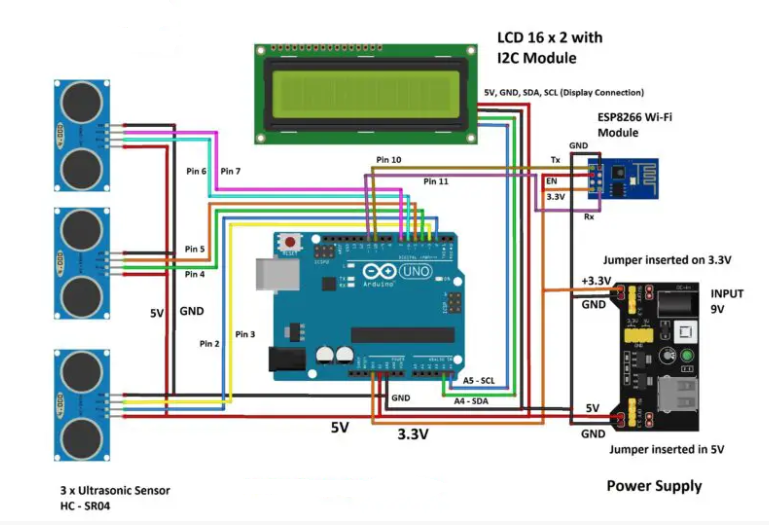
value = "";

}

**Circuit diagram for IoT based car park monitoring system:**

The proposed smart parking lot circuit will be equipped with several sensors, inexpensive microcontrollers and Wi-Fi module using which a car / any vehicle owner can check if there is a vacant space in a parking lot using his / her phone or tablet or even on computer.

The number of vacant spaces in the smart parking lot can be viewed from anywhere in the world using a URL link or the user can scan a QR code. The scanned / shared URL can be browsed on any web browser to know how many empty parking spot exist in real time.

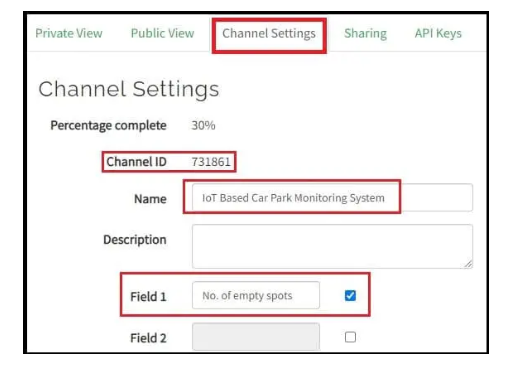


The internet cloud service we are going to use is called “Thingspeak” where the parking lot’s data to be sent, stored and displayed in real time. This concludes the block diagram.

****How to setup your Thingspeak account?****

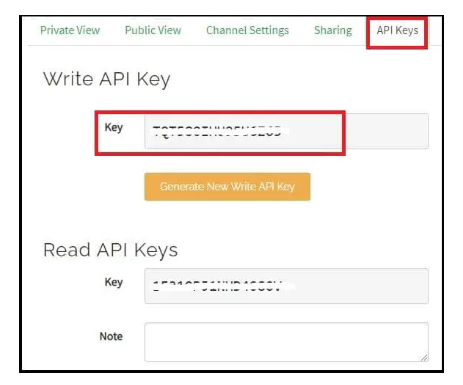
We are using a (free) cloud service called Thingspeak where we will send parking lot’s data to share it with public.

* First you need to sign up for Thingspeak:
* Enter the credentials it asks for and create a new channel and do the following to your new channel:

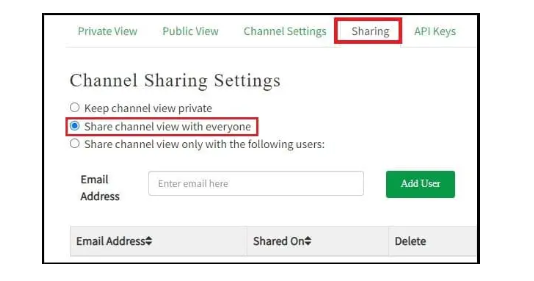


* Go to **channel settings** and enter the things as shown above and take note of your channel ID which we need to enter it in the program code.
* Scroll down and **click save** to save the changes.
* Now click on **API keys tab** and you will see your keys as illustrated below. API keys are responsible for writing and reading the data to your Thingspeak account.
* Go to **channel settings** and enter the things as shown above and take note of your channel ID which we need to enter it in the program code.
* Scroll down and **click save** to save the changes.

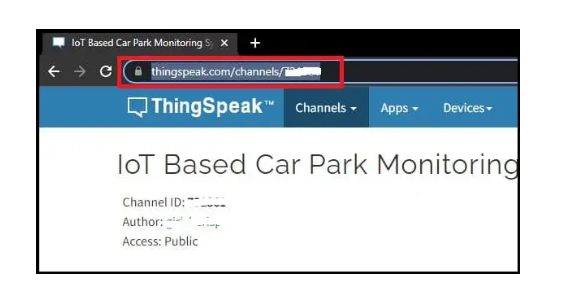
Now click on **API keys tab** and you will see your keys as illustrated below. API keys are responsible for writing and reading the data to your Thingspeak account.



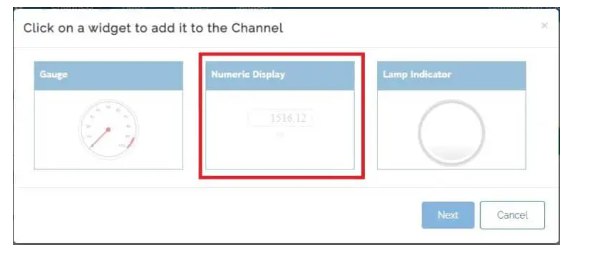
* Take note of your “write API key” which needs to be entered in the program code and read API key is not used in this project.
* Now go to sharing tab and click on “share channel view with everyone”, this makes your channel visible to those who have the URL of “public view” page.

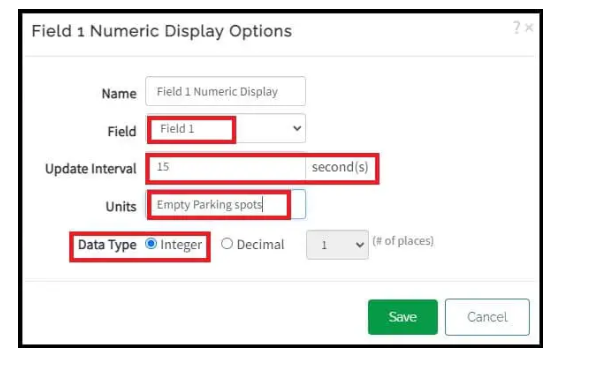


* Click on **public view tab** and you will see an empty graph field and the **URL of this page can be shared to the public**.

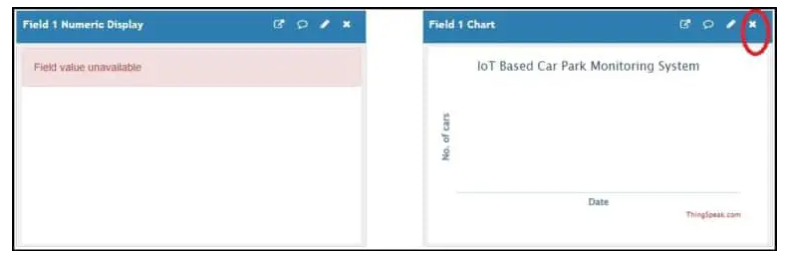


* Now on the public view tab we are going to setup a number widget where public can view the number of vacant spots on the parking lot, click on “add widget”.



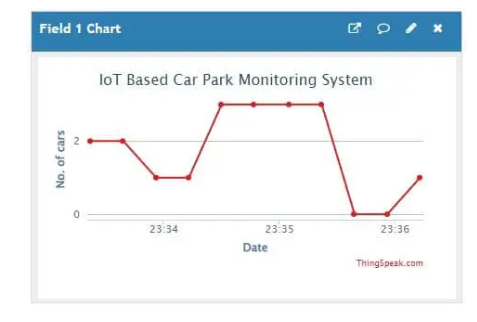


* Now you will see a new widget where number will be displayed once we send data. Close the “field chart” by clicking on ‘X’, the public just need to know the number of vacant parking spots and not the parking history.

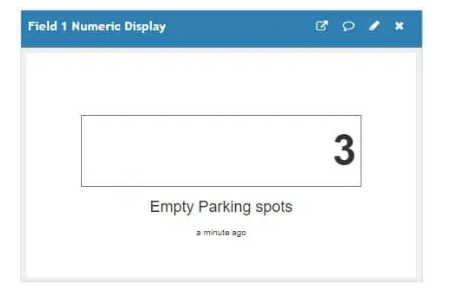


* Click on **private view tab**, you see an empty field chart; this is where you can view the history car parking history and this is not visible to public. This concludes on Thingspeak account setup.

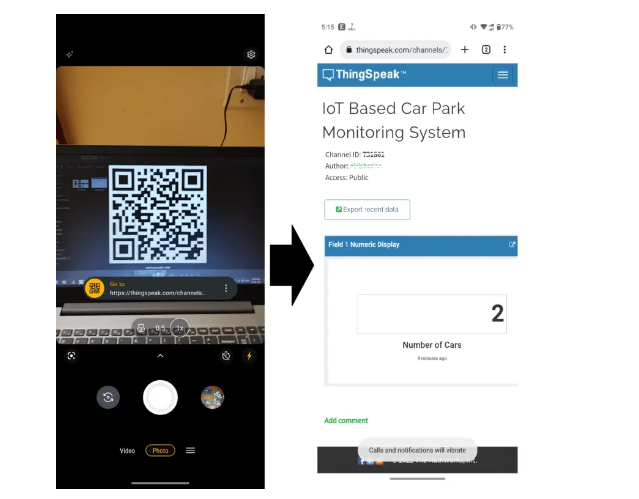
**Thingspeak Private View:**



****Thingspeak public view:****



****Thingspeak public view on smart phone:****



The URL of the public view page is converted in to a QR and this can be done using any online URL to QR code converter tool. This QR code can be placed at the parking spot or anywhere else, so that the users can scan and bookmark the URL and when they are visiting the parking spot, users can open the link and see how many vacant spaces exist and if this parking lot is full they can move to another.