SMART PARKING SYSTEM USING IBM WATSON CLOUD

BY

G.SAMPURNA

A.SRIYA

A.RISHIKA

K.PRIYANKA

# **ABSTRACT**

In recent times the concept of smart cities have gained grate popularity. Thanks to the evolution of Internet of things the idea of smart city now seems to be achievable. Consistent efforts are being made in the field of IOT in order to maximize the productivity and reliability of urban infrastructure. Problems such as, traffic congestion, limited car parking facilities and road safety are being addressed by IOT. In this paper, we present an IOT based cloud integrated smart parking system. The proposed Smart Parking system consists of an on-site deployment of an IOT module that is used to monitor and signalize the state of availability of each single parking space. A mobile application is also provided that allows an end user to check the availability of parking space and book a parking slot accordingly. The paper also describes a high-level view of the system architecture. Towards the end, the paper discusses the working of the system in form of a use case that proves the correctness of the proposed model.

**PROBLEM STATEMENT:**

**Smart Parking System Using IBM Watson**

With the increase in vehicle production and world population, more and more parking spaces and facilities are required. Generally people are facing problems on parking vehicles in parking slots in a city.

DESCRIPTION:

In this project a new parking system called Smart Parking System is proposed to assist which enables the user to find the nearest parking area and gives availability of parking slots in that respective parking area. And it mainly focus on reducing the time in finding the parking lots and also it avoids the unnecessary travelling through filled parking lots in a parking area. Firstly, we can check the status of the parking slot by using sensors. These sensor values are sent to IBM Watson services and these values are stored in the database. We can create a User interface by using Node Red through which we can show the users the empty slots and the filled slots.

**PROJECT WORKING PROCESS:**

ULTRA SONIC SENSOR B

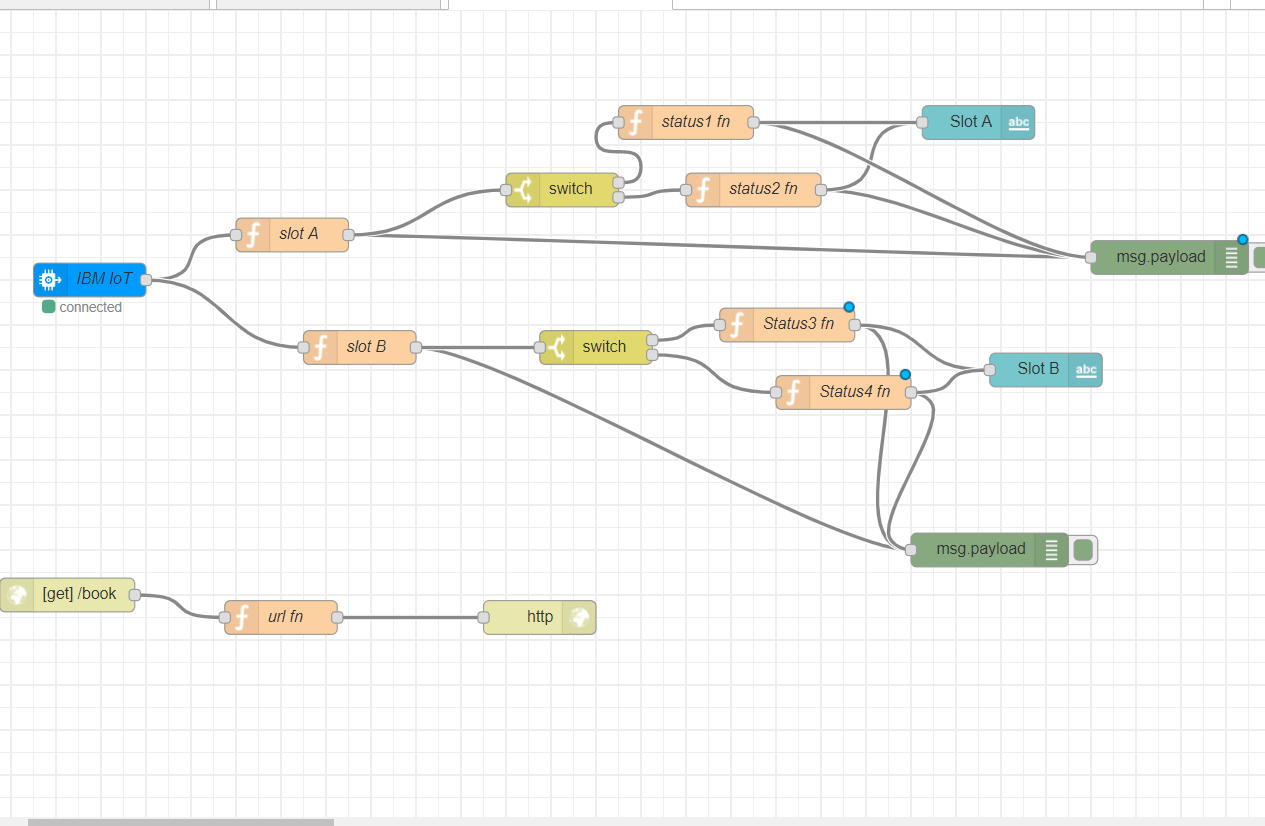
NODE MCU

ULTRA SONIC SENSOR A

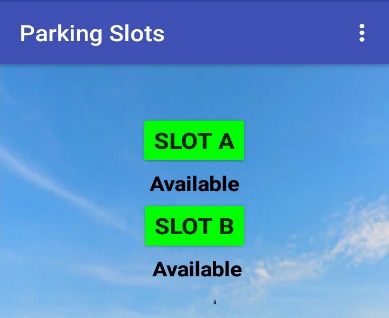
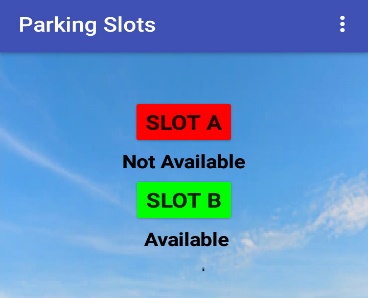
MIT APP

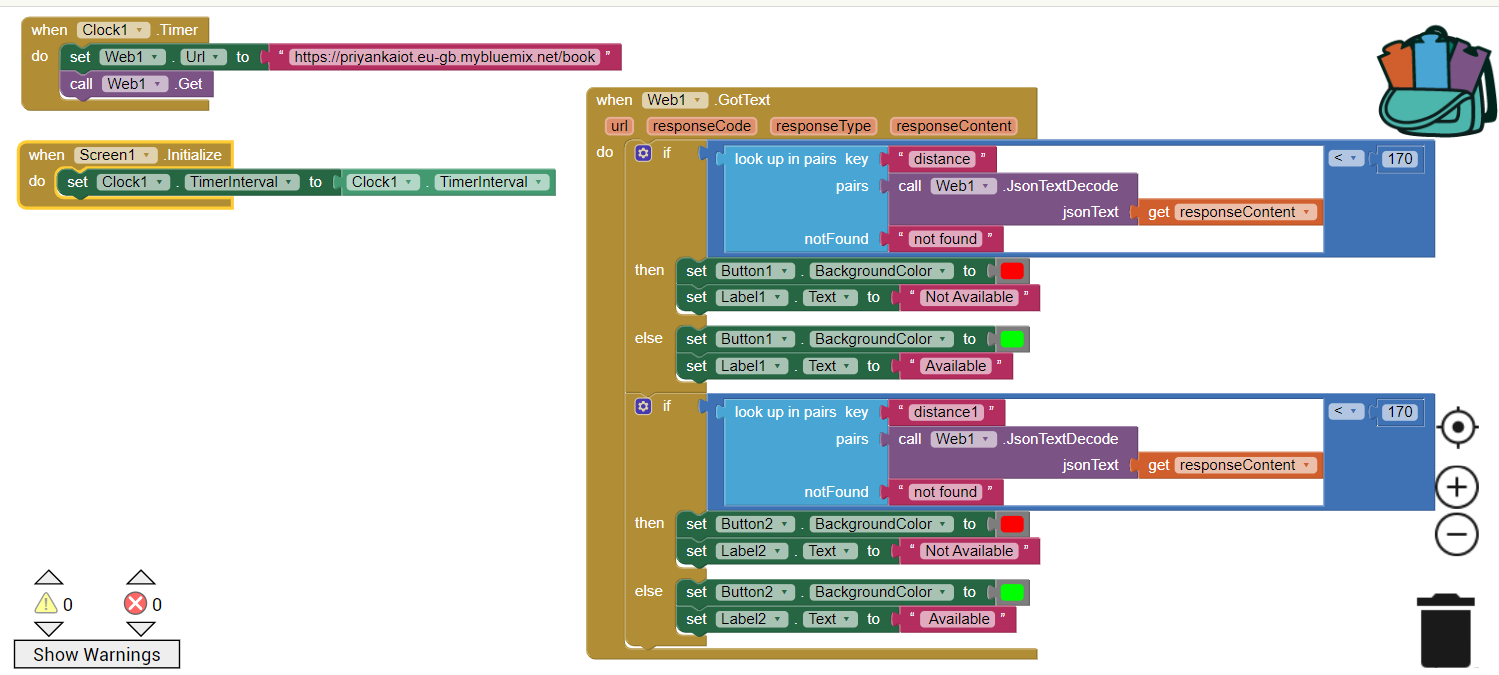
In this project Ultra sonic sensors are used to know the status of Parking Slots. Given two Ultra sonic sensors are connected to NODE MCU. On the basis of the distance a pulse is generated in the ultrasonic sensor to send the data to Node MCU or any other micro-controller. Node MCU publishes the data (distances) generated by the ultrasonic sensors to the IBM cloud. In the IBM Node Red Flow Editor based on the distance values sensed by ultrasonic sensor the status (Available or not) of Parking slot is displayed in Node Red UI. The status of parking slots is also displayed in MIT app.so we use the Web component of the App and set the web to the URL and get the status of slots from it.

**NODE RED FLOW EDITOR:**



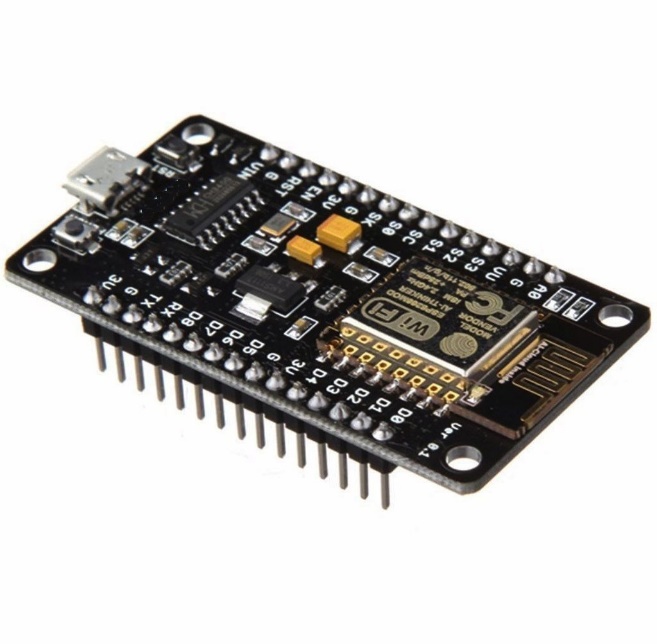
**MIT MOBILE APPLICATION:**





**COMPONENTS USED:**

**Hardware Components**

 **NodeMCU**: NodeMCU is an open source IOT platform. It includes firmware which runs on the ESP8266 Wi-Fi Module. and hardware which is based on the ESP-12 module. The term "NodeMCU" by default refers to the firmware rather than the development kits. The firmware uses the Lua scripting language.

**Specifications:**

**CPU:** ESP8266

**Memory:**128KB

**Power By:** USB

**Operating Voltage:**3.3v-5v

**GPIO:**13 pins

**IDE:** Arduino IDE

**ULTRASONIC SENSORS:** Actually we have to figure out the distance because the sensor itself simply holds it's "ECHO" pin HIGH for a duration of time corresponding to the time it took to receive the reflection (echo) from a wave it sent. The module sends out a burst of sound waves, at the same time it applies voltage to the echo pin. The module receives the reflection back from the sound waves and removes voltage from the echo pin. On the base of the distance a pulse is generated in the ultrasonic sensor to send the data to NodeMCU or any other micro-controller. The starting pulse is about 10us and the PWM signal will be 150 us-25us on the base of the distance. If no obstacle is there, then a 38us pulse is generated for NodeMCU to confirm that there are not objects detected.

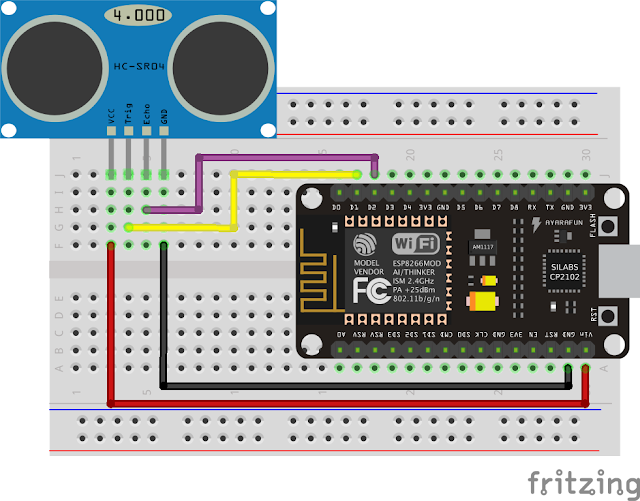
Before getting the reading of the HC-SR04 know about the calculation.

**FORMULA:**

D = 1/2 × T × C

where D is the distance, T is the time between the Emission and Reception, and C is the sonic speed.

(The value is multiplied by 1/2 because T is the time for go-and-return distance.)



**SPECIFICATION OF HC-SR04:**

Power supply : 5v DC

Ranging distance : 2cm – 500 cm

Ultrasonic Frequency : 40k

**Software Components:**

* Arduino IDE
* IBM Watson Cloud platform

**FINAL CODE:**

#include <ESP8266WiFi.h>--------

#include <PubSubClient.h>

//-------- Customise these values -----------

const char\* ssid = "Rishika";

const char\* password = "rishi123";

//#include "DHT.h"

//#define DHTPIN D2 // what pin we're condefine ORG "17qph2"

#define ORG "459p8w"

#define DEVICE\_TYPE "Iot\_project"

#define DEVICE\_ID "4109"

#define TOKEN "8328011328"

//-------- Customise the above values --------

char server[] = ORG ".messaging.internetofthings.ibmcloud.com";

char topic[] = "iot-2/evt/Data/fmt/json";

char authMethod[] = "use-token-auth";

char token[] = TOKEN;

char clientId[] = "d:" ORG ":" DEVICE\_TYPE ":" DEVICE\_ID;

WiFiClient wifiClient;

PubSubClient client(server, 1883,wifiClient);

const int trigPin = D1;

const int echoPin = D2;

const int trigPin1=D3;

const int echoPin1=D4;

// defines variables

long duration;

int distance;

long duration1;

int distance1;

void setup() {

pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output

pinMode(echoPin, INPUT);

pinMode(trigPin1,OUTPUT);

pinMode(echoPin1,INPUT);// Sets the echoPin as an Input

Serial.begin(9600); // Starts the serial communication

Serial.print("Connecting to ");

Serial.print(ssid);

WiFi.begin(ssid, password);

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

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.print("WiFi connected, IP address: ");

Serial.println(WiFi.localIP());

}

void loop() {

// Clears the trigPin

digitalWrite(trigPin, LOW);

//digitalWrite(trigpin1,LOW);

delay(1000);

// Sets the trigPin on HIGH state for 10 micro seconds

digitalWrite(trigPin, HIGH);

delay(1000);

digitalWrite(trigPin, LOW);

// Reads the echoPin, returns the sound wave travel time in microseconds

duration = pulseIn(echoPin, HIGH);

// Calculating the distance

distance= duration\*0.034/2;

// Prints the distance on the Serial Monitor

Serial.print("Distance: ");

Serial.println(distance);

//for slotb

digitalWrite(trigPin1, LOW);

//digitalWrite(trigpin1,LOW);

delay(1000);

// Sets the trigPin on HIGH state for 10 micro seconds

digitalWrite(trigPin1, HIGH);

delay(1000);

digitalWrite(trigPin1, LOW);

// Reads the echoPin, returns the sound wave travel time in microseconds

duration1= pulseIn(echoPin1, HIGH);

// Calculating the distance

distance1= duration1\*0.034/2;

// Prints the distance on the Serial Monitor

Serial.print("Distance1: ");

Serial.println(distance1);

PublishData(distance,distance1);

delay(5000);

}

void PublishData(int distance,int distance1){

if (!!!client.connected()) {

Serial.print("Reconnecting client to ");

Serial.println(server);

while (!!!client.connect(clientId, authMethod, token)) {

Serial.print(".");

delay(500);

}

Serial.println();

}

String payload = "{\"d\":{\"distance\":";

payload += distance;

payload+="," "\"distance1\":";

payload += distance1;

payload += "}}";

if (client.publish(topic, (char\*) payload.c\_str())) {

Serial.println("Publish ok");

} else {

Serial.println("Publish failed");

}

}