**TITLE: SMART PARKING SECURITY SYSTEM**

**by**

**NAME**: ABHYUDAY SINGH MANDLOI REGISTER NUMBER: 18BLC1159 **NAME**: PRERIT AGRAWAL REGISTER NUMBER:18BLC1156

A project report submitted to

**Dr. Bala Murugan M S**

**SCHOOL OF ELECTRONICS ENGINEERING**

**in partial fulfilment of the requirements for the course of**

**ECM4002 – IoT System Design**in

**B. Tech. ELECTRONICS AND COMMUNICATION**

**ENGINEERING**



**Vandalur – Kelambakkam Road**

**Chennai – 600127**

**NOVEMBER 2020**

**BONAFIDE CERTIFICATE**

Certified that this project report entitled “**SMART PARKING SECURITY SYSTEM”** is a bonafide work of **NAME: ABHYUDAY SINGH -REG.NO 18BLC1159 and NAME: PRERIT AGRAWAL REG.NO 18BLC1156** who carried out the Project work under my supervision and guidance for **ECM4002 – IoT System Design.**

**Dr. Bala Murugan M S**

**Associate Professor**

School of Electronics Engineering (SENSE), VIT University, Chennai

Chennai – 600 127.

**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.

**ACKNOWLEDGEMENT**

We wish to express our sincere thanks and deep sense of gratitude to our project guide **Prof. Bala Murugan M S,** Associate Professor, School of Electronics Engineering, for her consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Sivasubramanian. A,** Dean of School of Electronics Engineering, VIT Chennai, for extending the facilities of the School towards our project and for his unstinting support.

We express our thanks to our Head of the Department **Dr. Vetrivelan. P** for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

**ABHYUDAY SINGH MANDLOI PRERIT AGRAWAL**

**TABLE OF CONTENTS**

**SERIAL TITLE PAGE**

**NO. NO.**

ABSTRACT

ACKNOWLEDGEMENT 3-4

1. INTRODUCTION 6-7

* 1. OBJECTIVES AND GOALS 6
  2. APPLICATIONS 6
  3. FEATURES 7

1. DESIGN AND 8-14

IMPLEMENTATION

* 1. BLOCK DIAGRAM 7-10
  2. NODE ANALYSIS-NODE RED 11-14

1. 3.1 SOFTWARE –CODING AND 15-25

ANALYSIS

(SNAPSHOTS OF CODING

AND RESULTS)

1. CONCLUSION AND FUTURE 26-33

WORK

* 1. RESULT, CONCLUSION AND 26-29

INFERENCE

* 1. FUTURE WORK 30-33

1. 5.1 REFERENCES 34
2. PHOTO GRAPH OF THE PROJECT ALONG WITH THE

TEAM MEMBERS 35

1. **INTRODUCTION:**

* 1. **OBJECTIVE AND GOALS:**

In this project we have built a car recognition system using Node-RED. For this project we’ll be using a software called Plate recognization API(Automatic License Place Recognition) that has an API you can use to identify car plates and car models based on an image. We are using visual recognization associated with IBM Watson Cloud to detect the features of the car.

* 1. **APPLICATION:**

In this project, we use a PIR motion sensor to detect that the car arrives home. There are other sensors that may be more suitable to detect a car, for example:

Hall effect sensor: senses changes in magnetic field when the car is near;

Ultrasonic sensor: detects distance to an object;

Active infrared detectors: detects the presence of an object by detecting the reflection of infrared light. When the sensor detects motion, the camera module takes a photo. After that, the http sends a request to

PLATRECOGNIZATION API with the car photo to be identified. Then, the PLATE RECOGNIZATION API returns the car details like: plate number, model, color, and the confidence of the results. After identifying a car, we’ll do some verifications, and if we found an authorized car, we’ll trigger an event (that can be open the garage, for example).

Here’s what happens: after the car has been identified by PLATE

RECOGNIZATION, we’ll check if the license plate and the car model match. If they match, we’ll check if the car is in the list of authorized vehicles. If it is, we’ll trigger an event. For example: open the garage. After that, we wait a determined period of time until the car enters the garage. Then, you need to add several verifications to check if the car has already entered the garage. If yes, you can close the garage.

**1.3 FEATURES:**

There are two main things are used in this Project are 1. plate recognization API

2. IBM Watson cloud.

1. **ALPR (PLATE RECOGNIZATION API)**

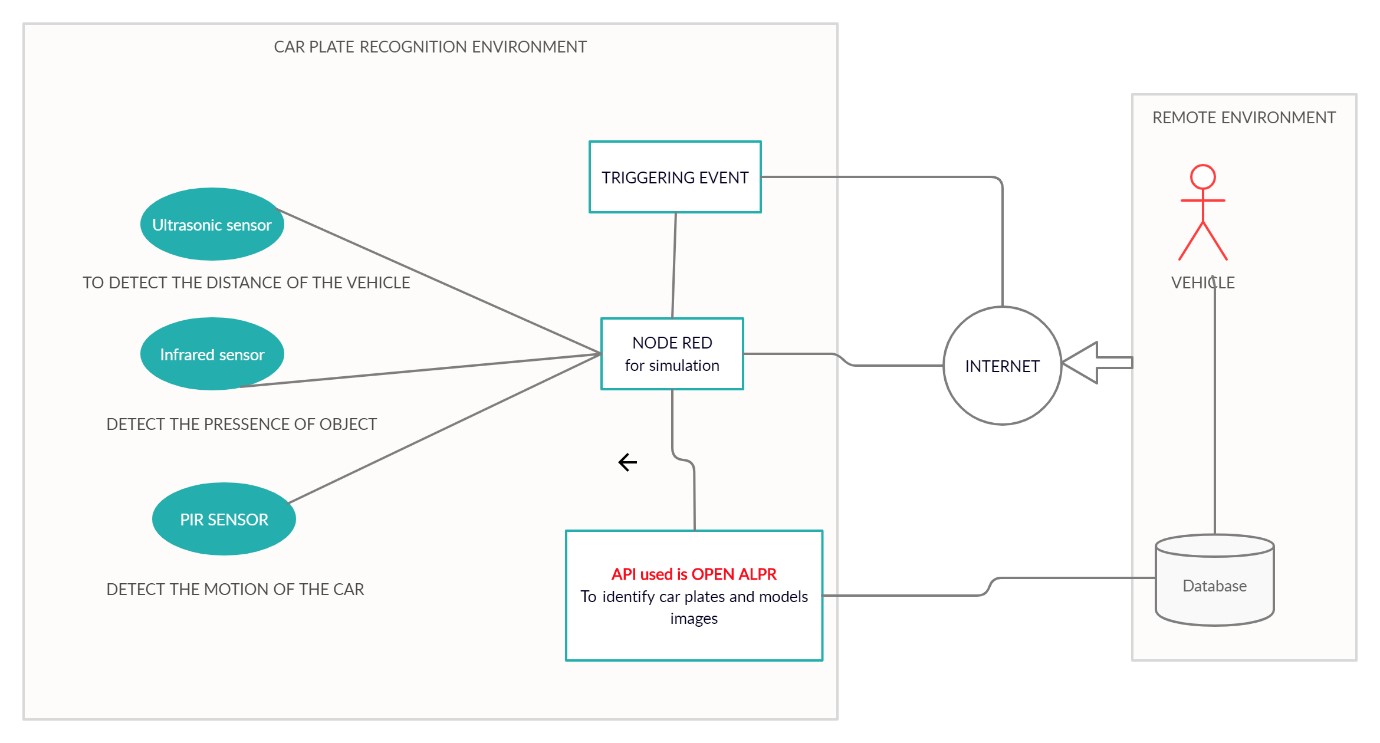
[ALPR](https://github.com/openalpr/openalpr) is an open source Automatic License Plate Recognition library written in C++ with bindings in C#, Java, Node.js, Go, and Python. We are using the website www.platerecognizer.com which uses Plate Recognizer Snapshot API. is a web service running in the cloud that analyzes images of vehicles and responds with license plate, model, color and much more.

1. **IBM WATSON**

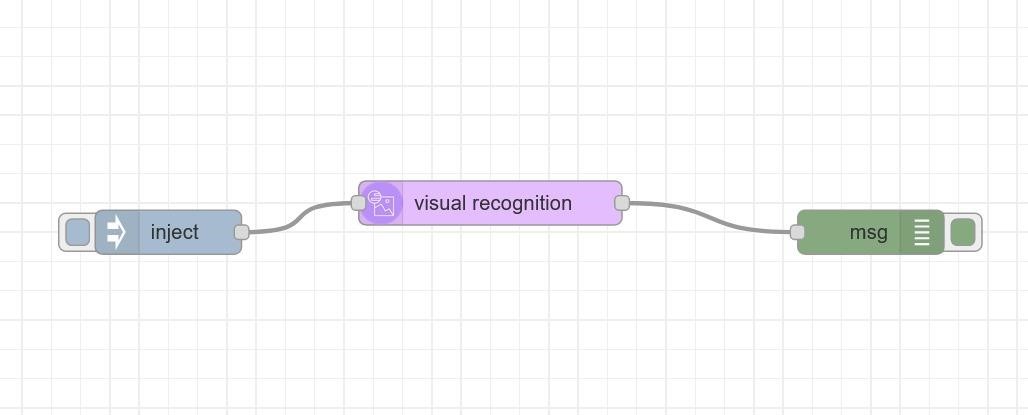
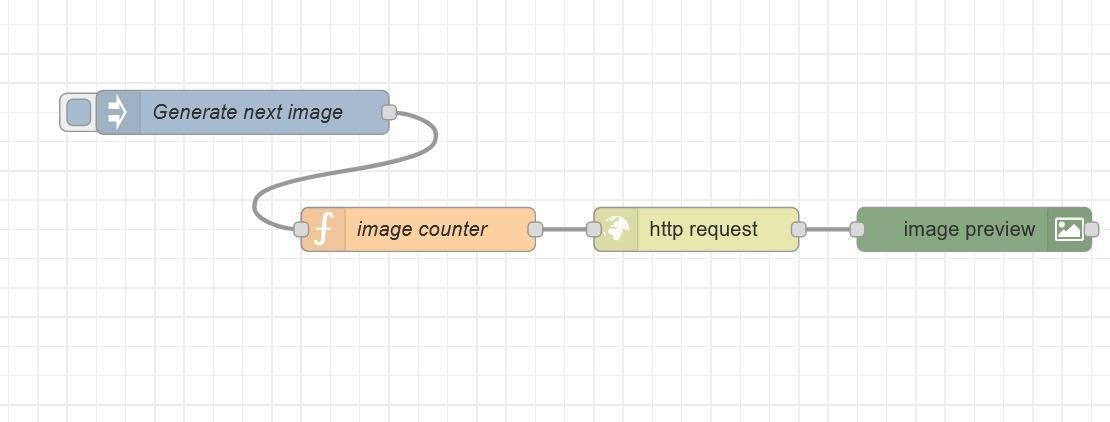
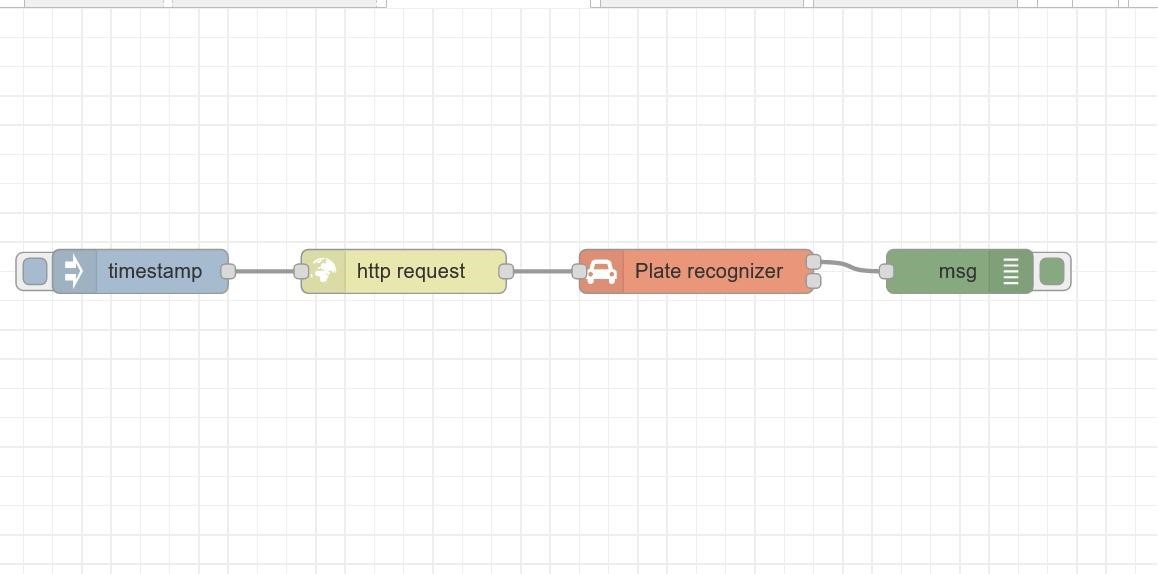
[Watson](https://www.geeksforgeeks.org/speech-to-text-using-ibm-watson-studio/) is AI from IBM. Created to form your business more intelligent and every worker your best worker. Watson features a range of advanced APIs, specialized tooling, and Software as a Service application. This implies that Watson is made for complex use cases and designed to integrate with platforms that experts utilize in their daily work. Ensuring seamless access to the knowledge you would like to form the right decisions.

1. **DESIGN AND IMPLEMENTATION:** 
   1. **BLOCK DIAGRAM:**

**USE CASE DIAGRAM:**



**NODE RED BLOCK DIAGRAM:**



* 1. **ANALYSIS:**

**USED NODES DETAILS:**

**INJECT NODE:**

Injects a message into a flow either manually or at regular intervals. The message payload can be a variety of types, including strings, JavaScript objects or the current time.

The Inject node can initiate a flow with a specific payload value. The default payload is a timestamp of the current time in millisecs since January 1st, 1970.

The node also supports injecting strings, numbers, booleans, JavaScript objects, or flow/global context values.

By default, the node is triggered manually by clicking on its button within the editor. It can also be set to inject at regular intervals or according to a schedule.

It can also be configured to inject once each time the flows are started.

The maximum *Interval* that can be specified is about 596 hours / 24 days. However if you are looking at intervals greater than one day you should consider using a scheduler node that can cope with power outages and restarts.

**HTTP REQUEST NODE:**

Sends HTTP requests and returns the response. When configured within the node, the URL property can contain [mustache-style](http://mustache.github.io/mustache.5.html) tags. These allow the url to be constructed using values of the incoming message. For example, if the url is set to example.com/{{{topic}}}, it will have the value of msg.topic automatically inserted. Using {{{...}}} prevents mustache from escaping characters like / & etc. The node can optionally automatically encode msg.payload as query string parameters for a GET request, in which case msg.payload has to be an object.

**PLATE RECOGNIZER NODE:**

A node for recognizing license plates in images.

**Input field:**

The field of the input message which will need to contain the input image. By default msg.payload will be used. The image should be a binary Buffer or a base64 encoded string.

**Output field:**

The field of the output message where the recognition result will be stored (in JSON format). By default msg.payload will be used.

**API token:**

Create an account at [platerecognizer.com](https://platerecognizer.com/) and enter your private API token here.

**URL:**

Specify the URL of the recognition service, to allow different kind of setups:

* Use the official cloud service, which will be the default (and most used) option.
* Use a local installation (based on the SDK).  Use a local Docker container.

**Camera ID:**

Optionally specify the camera id, to send it to the recognition service.

**Status text:**

Specify how the recognition result needs to be displayed in the node status label:

* *None:* Show no recognition results.
* *Plate count:* Show the number of plates that have been recognised in the image.
* *Plates:* Show a (comma separted) list of the plates that have been recognized in the image.
* *Plates and scores:* Same as the previous option, but now the 'score' percentage is also added.

**Ignore images arriving during recognition:**

When selected images will automatically be skipped, when the previous image is still being recognized. When deselected multiple images can be recognized simultaneously.

**Predict vehicle make and model (MMC):**

When selected not only the plate will be recognized, but there will also be a prediction of the vehicle brand and type. CAUTION: this is only supported for some paid account types!

**Send separate message for each plate:**

When selected a separate output message will be send for each recognized license plate. If not selected a single output message will be send containing an array of ALL recognized license plates.

**DEBUG NODE:**

Displays selected message properties in the debug sidebar tab and optionally the runtime log. By default it displays msg.payload, but can be configured to display any property, the full message or the result of a JSONata expression.

[Details](http://127.0.0.1:1880/)

The debug sidebar provides a structured view of the messages it is sent, making it easier to understand their structure.

JavaScript objects and arrays can be collapsed and expanded as required. Buffer objects can be displayed as raw data or as a string if possible.

Alongside each message, the debug sidebar includes information about the time the message was received, the node that sent it and the type of the message. Clicking on the source node id will reveal that node within the workspace.

The button on the node can be used to enable or disable its output. It is recommended to disable or remove any Debug nodes that are not being used.

The node can also be configured to send all messages to the runtime log, or to send short (32 characters) to the status text under the debug node.

**FUNCTION NODE:**

A JavaScript function to run against the messages being received by the node.

The messages are passed in as a JavaScript object called msg.

By convention it will have a msg.payload property containing the body of the message.

The function is expected to return a message object (or multiple message objects), but can choose to return nothing in order to halt a flow.

The **Setup** tab contains code that will be run whenever the node is started. The **Close** tab contains code that will be run when the node is stopped.

[Details**.**](http://127.0.0.1:1880/)

*Sending messages*

The function can either return the messages it wants to pass on to the next nodes in the flow, or can call node.send(messages). It can return/send:

* a single message object - passed to nodes connected to the first output
* an array of message objects - passed to nodes connected to the corresponding outputs

If any element of the array is itself an array of messages, multiple messages are sent to the corresponding output.

If null is returned, either by itself or as an element of the array, no message is passed on.

**IBM WATSON VISUAL RECOGNIZATION NODE:**

This flow builds a very simple web page / form that prompts the user to create a Watson Visual Recognition Custom Classifier. The web form requires a name for the custom classifier, prompts the user to upload a training set of >10 images of an object and >10 images of a negative training set.

The flow then uploads the images, creates two zip files and finally calls the [Watson Visual Recognition Custom Classifier](https://cloud.ibm.com/apidocs/visual-recognition#create-a-classifier) API.

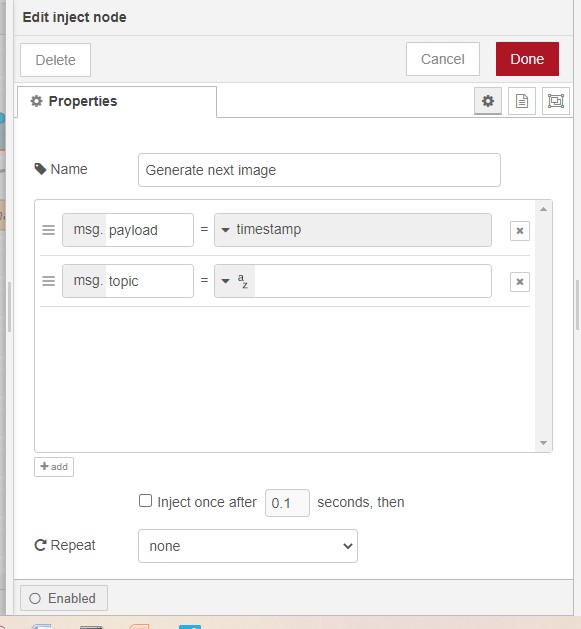
To test the Visual Recognition model, the form optionally prompts for an image URL to be analyzed.

To test the Visual Recognition model, the form optionally prompts for an image to upload to be analyzed.

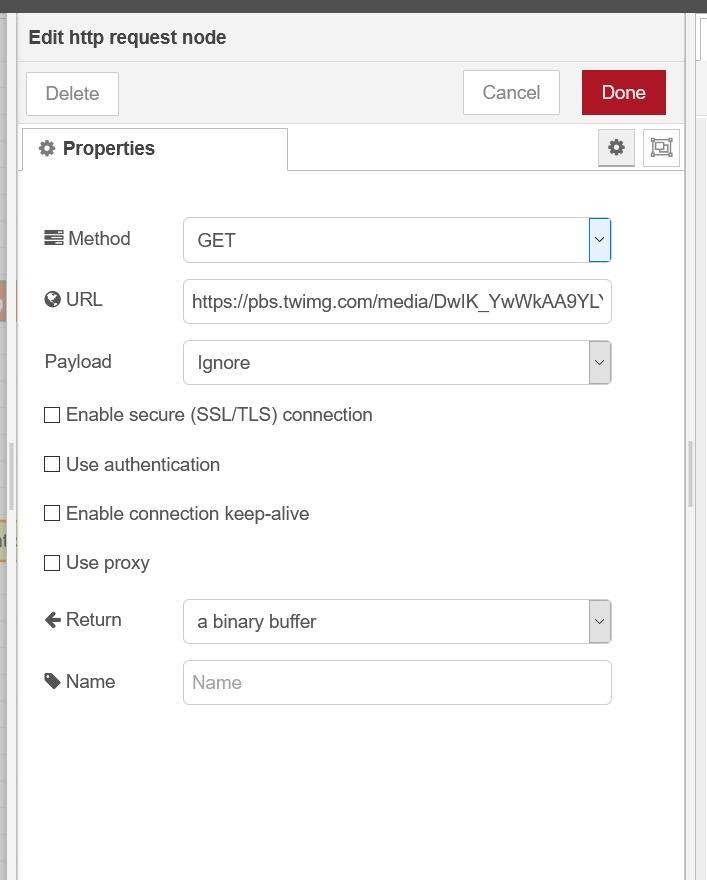
1. **SOFTWARE –(CODING ANDANALYSIS / IMPLEMENTATION\_):** 
   1. **NODE RED IMPLEMENTATION:**

**INJECT NODE PROPERTIES**

:



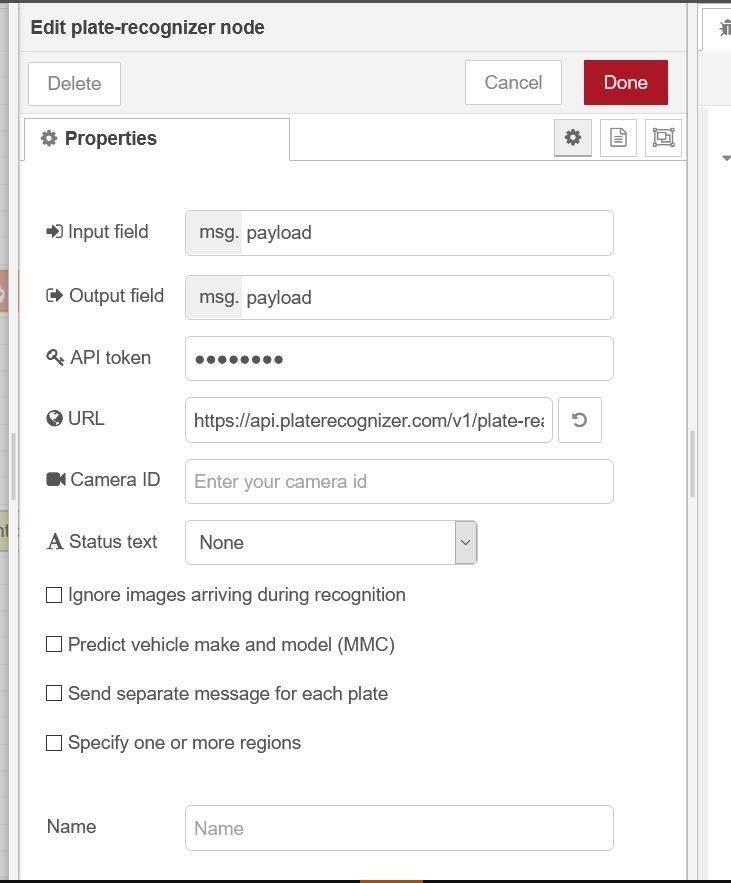
**HTTP REQUEST NODE PROPERTIES:**



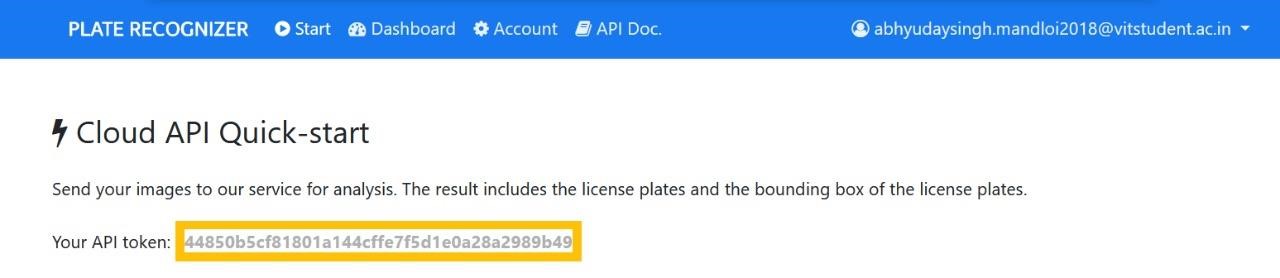
**FUNCTION NODE PROPERTIES:**



**PLATE RECOGNIZER NODE PROPERTIES:**

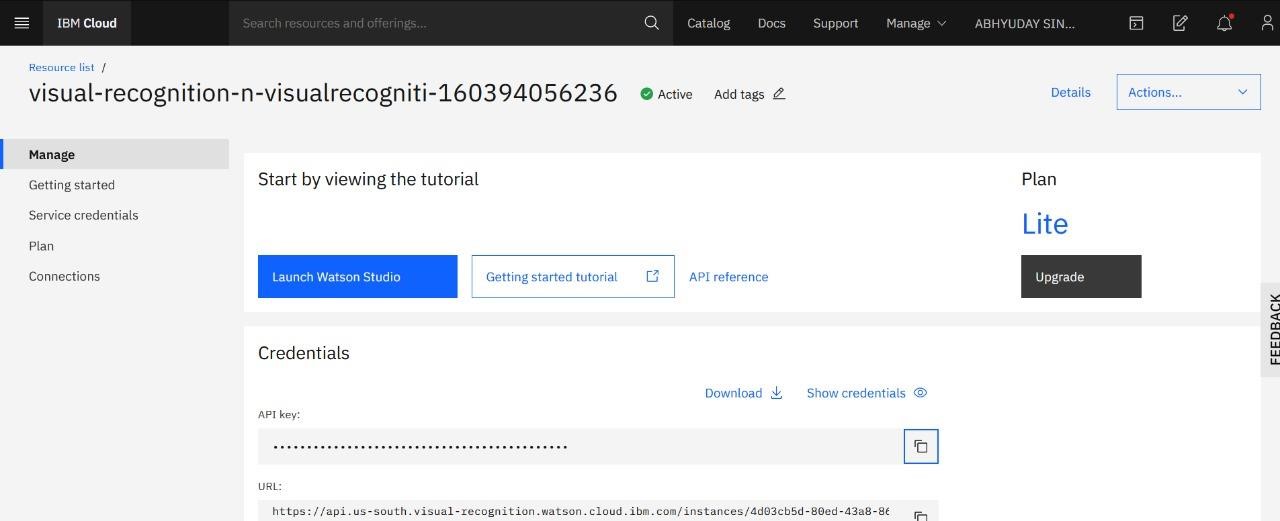


**GENERETING API TOKEN IN PLATE-RECOGNIZER API:**

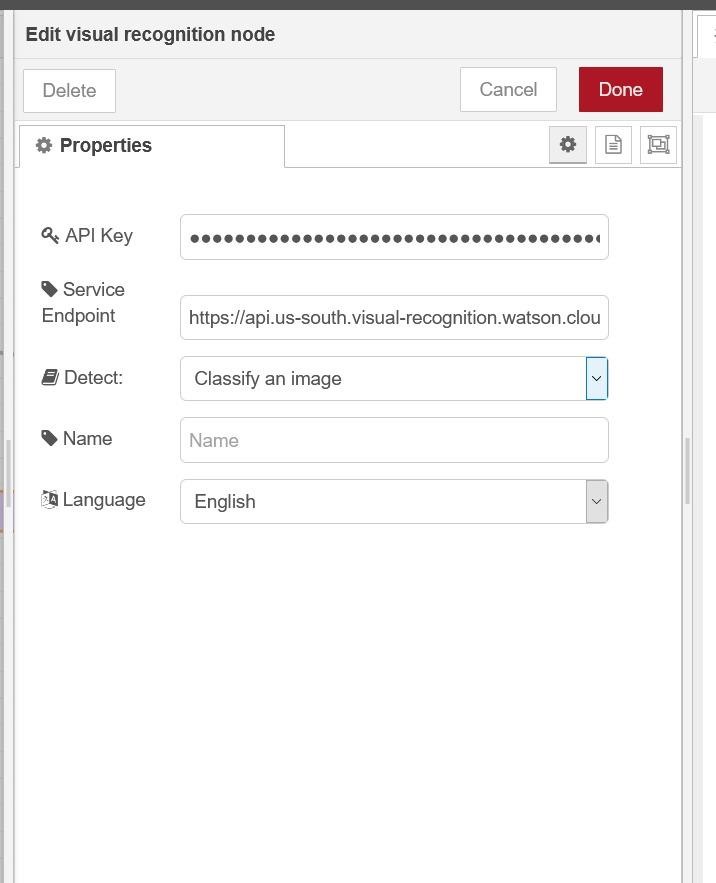


**GENERETING API TOKEN WATSON LINKED VISUAL RECOGNIZER**

**API:**

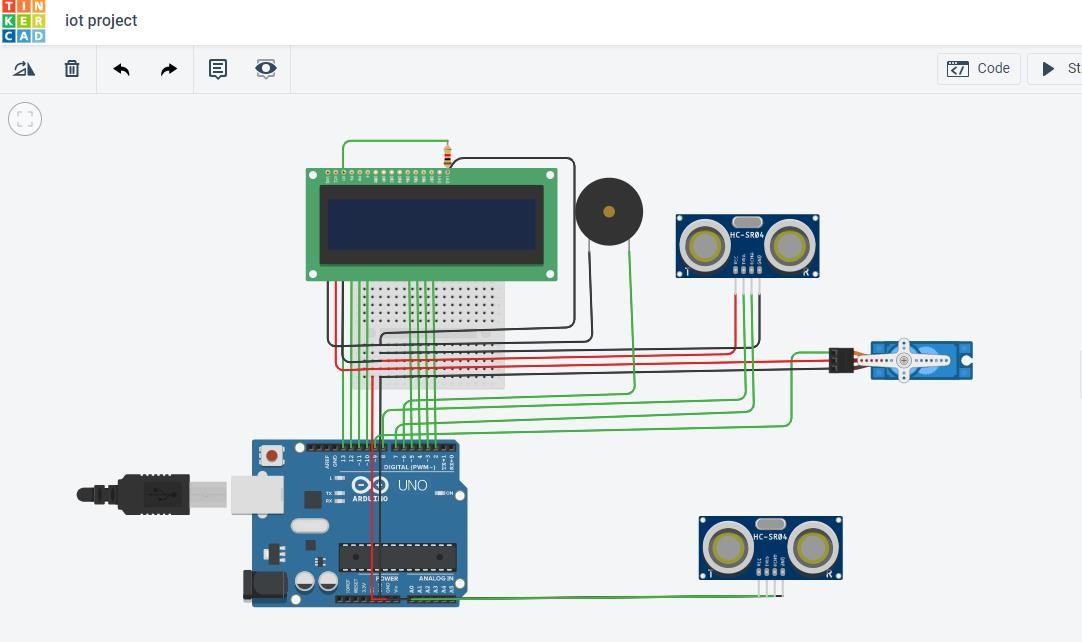


**VISUAL RECOGNITION NODE PROPERTIES:**



* 1. **TINKERCAD IMPLEMENTATION:**

**CIRCUIT DIAGRAM:**



**COMPONENTS:**



**CODING:**

#include<LiquidCrystal.h>

#include<Servo.h> LiquidCrystal lcd(12,11,10,5,4,3,2); int backLight=13; int trig=8; int echo=7; int trig1=A0; int echo1=A1; Servo myservo; float timeduration; float distance; float timeduration1; float distance1; int count=15; void setup() { myservo.attach(9); pinMode(backLight,OUTPUT); digitalWrite(backLight,HIGH); lcd.begin(16,2); lcd.clear();

Serial.begin(9600); pinMode(trig,OUTPUT); pinMode(echo,INPUT); pinMode(trig1,OUTPUT); pinMode(echo1,INPUT); pinMode(6,OUTPUT);

} void loop() { digitalWrite(trig,LOW); delayMicroseconds(2); digitalWrite(trig,HIGH); delayMicroseconds(10); digitalWrite(trig,LOW); timeduration= pulseIn(echo,HIGH); distance=0.034 \* timeduration/2 ;

Serial.print("DisTANCE IN cm:"); Serial.println(distance); digitalWrite(trig1,LOW); delayMicroseconds(2); digitalWrite(trig1,HIGH); delayMicroseconds(10); digitalWrite(trig1,LOW); timeduration1= pulseIn(echo1,HIGH); distance1=0.034 \* timeduration1/2 ; Serial.print("DisTANCE1 IN cm:"); Serial.println(distance1); lcd.setCursor(0,0); lcd.print("SMART PARKING");

if(distance<=30 && count>=0)

{ lcd.setCursor(0,1); lcd.write("AVAIL SLT:");

lcd.print(count); Serial.println(count); myservo.write(90); delay(5000); lcd.clear(); myservo.write(0); count=count-1; if(count==0)

{

digitalWrite(6,HIGH); lcd.setCursor(1,1);

lcd.print("NO AVAIL SLT"); delay(2000);

}

}

if(distance1<=30 && count<15)

{ lcd.setCursor(0,1); lcd.write("AVAIL SLT:"); lcd.print(count); myservo.write(90); delay(5000); lcd.clear(); myservo.write(0); count++;

}

digitalWrite(6,LOW);

}

**APPLYING MACHINE LEARNING ALGORITHM:**

**(MULTIPLE LINEAR REGRESSION)**

Multiple Linear Regression is an extension of Simple Linear regression as it takes more than one predictor variable to predict the response variable. We can define it as:

*Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable* **Some key points about MLR:** o For MLR, the dependent or target variable(Y) must be the continuous/real, but the predictor or independent variable may be of continuous or categorical form.

* Each feature variable must model the linear relationship with the dependent variable.
* MLR tries to fit a regression line through a multidimensional space of datapoints.

**EQUATION**

In Multiple Linear Regression, the target variable(Y) is a linear combination of multiple predictor variables x1, x2, x3, ...,xn. Since it is an enhancement of Simple Linear Regression, so the same is applied for the multiple linear regression equation, the equation becomes:

Y= b<sub>0</sub>+b<sub>1</sub>x<sub>1</sub>+ b<sub>2</sub>x<sub>2</su b>+ b<sub>3</sub>x<sub>3</sub>+...... bnxn Where, **Y= Output/Response variable**

**b0, b1, b2, b3 , bn....= Coefficients of the model.**

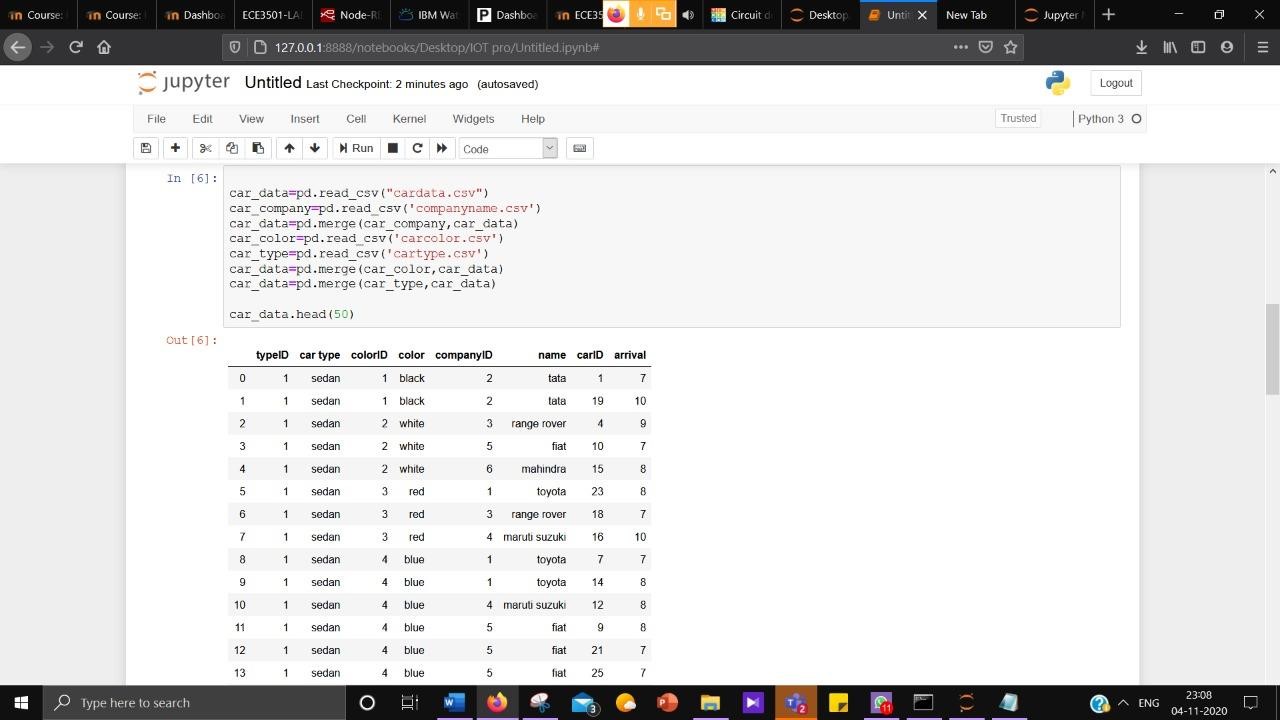
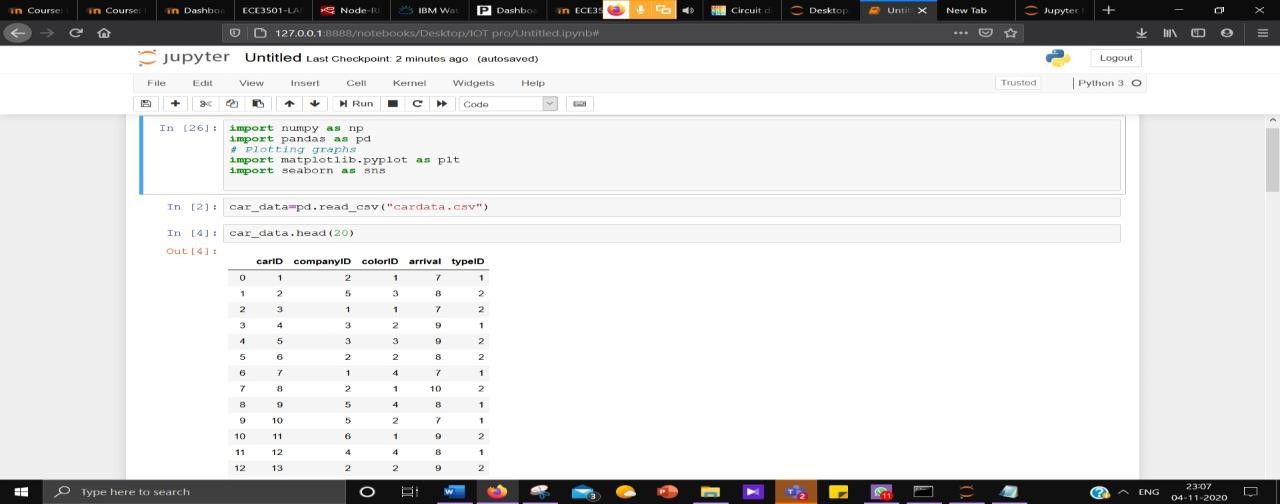
**x1, x2, x3, x4,...= Various Independent/feature variable**

**ASSUMPTION:**

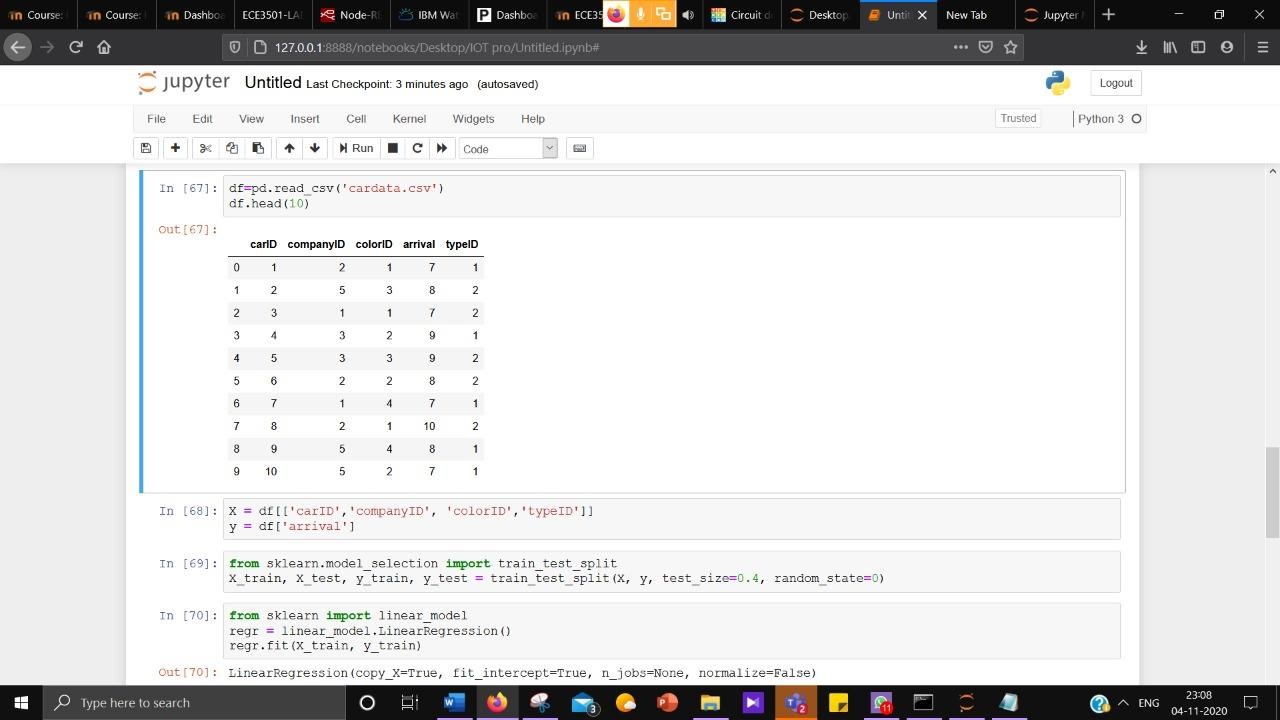
* A **linear relationship** should exist between the Target and predictor variables.
* The regression residuals must be **normally distributed**.
* MLR assumes little or **no multicollinearity** (correlation between the independent variable) in data.

**IMPLEMENTATION OF MLR USING PYTHON:**

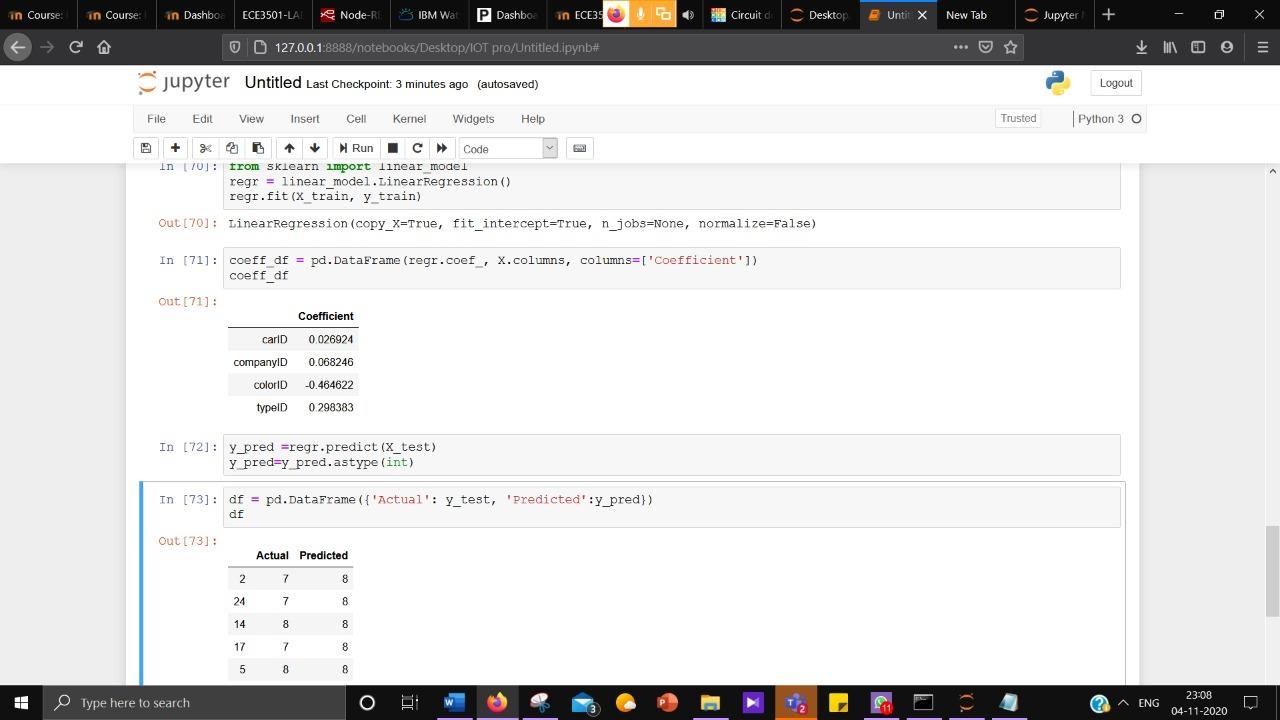
**Step-1: Data Pre-processing Step:**

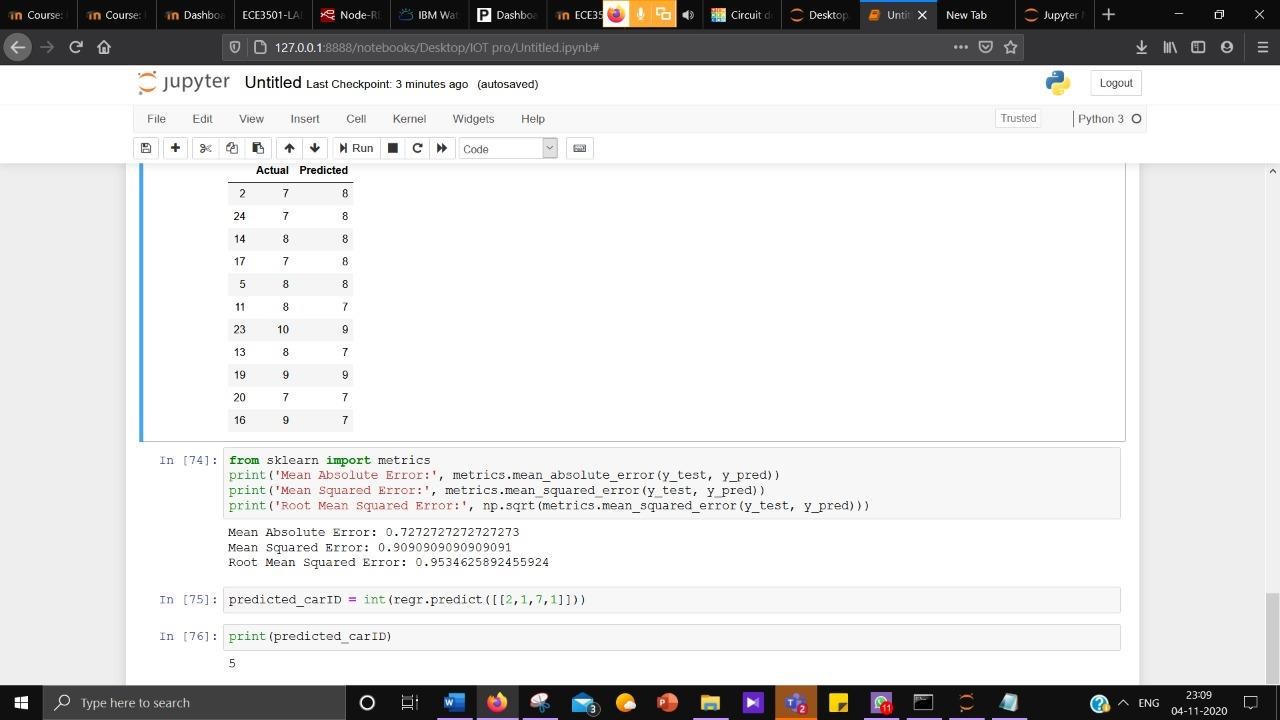


**Step-2: Fitting MLR model & the training set:**



**Step-3: Fitting MLR model & the prediction set:**



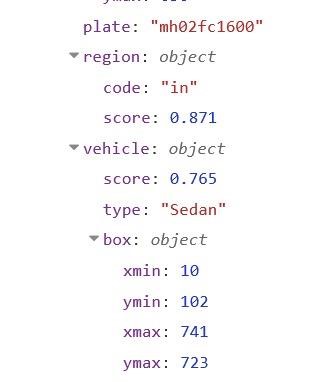
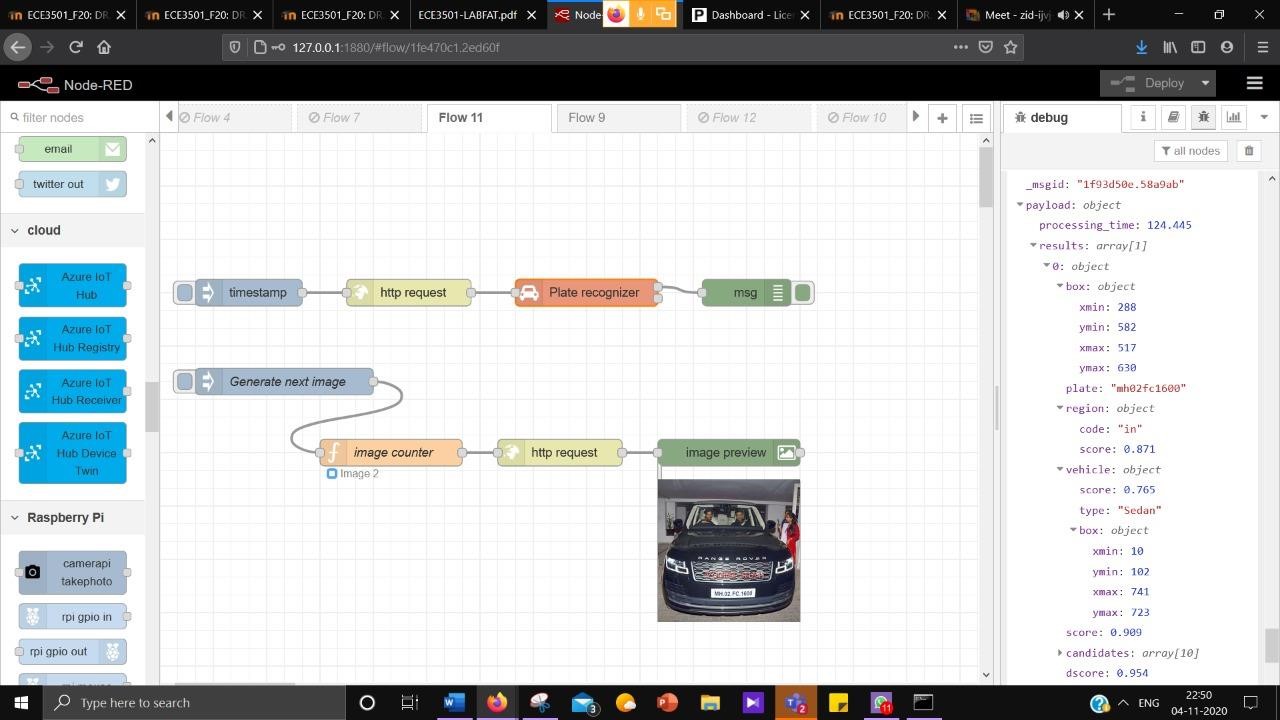


**4.CONCLUSION AND FUTURE WORK:**

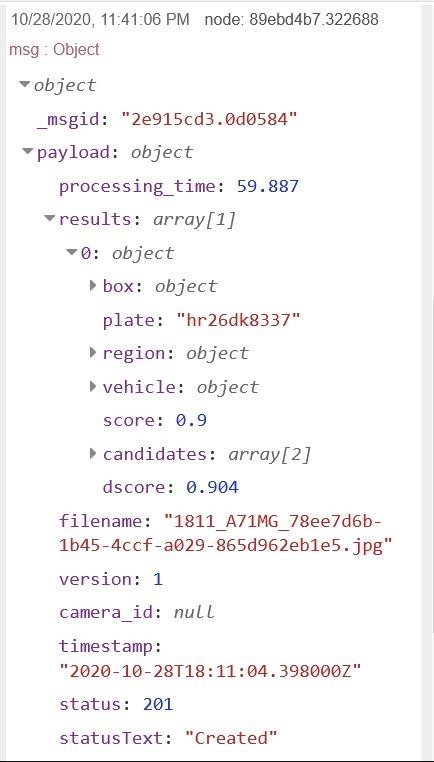
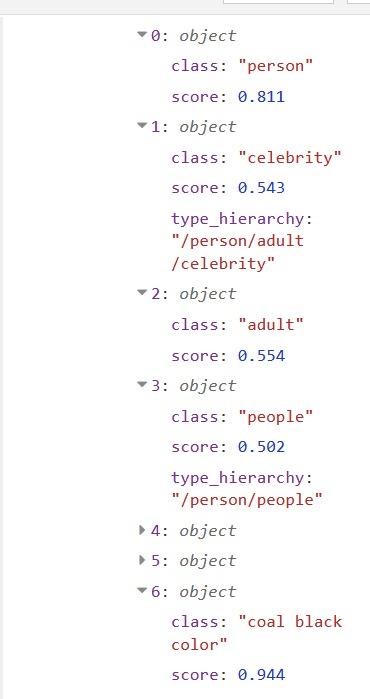
**4.1 RESULT:**

**NODE RED RESULTS:**

**PLATE RECOGNIZATION API OUTPUT FOR IMAGE PREVIEW:**

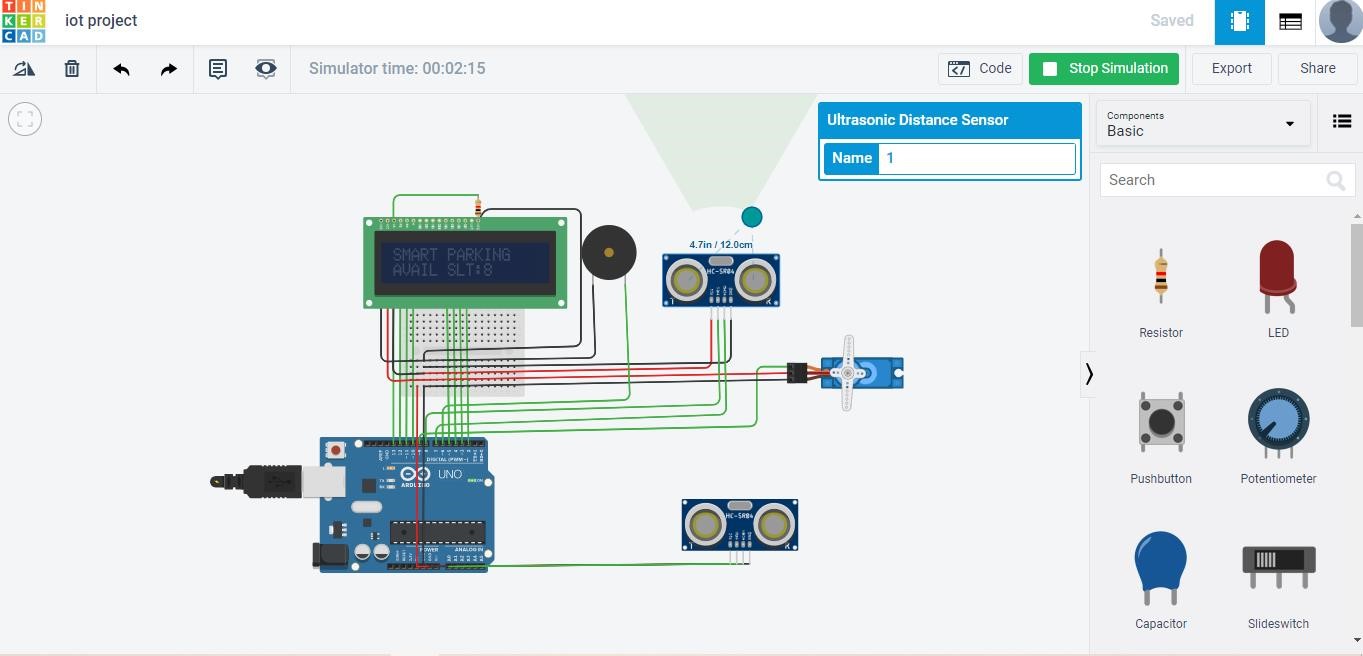


**AFTER USINF VISUAL RECOGNITION NODE ASSOCIATED WITH IBM WATSON:**

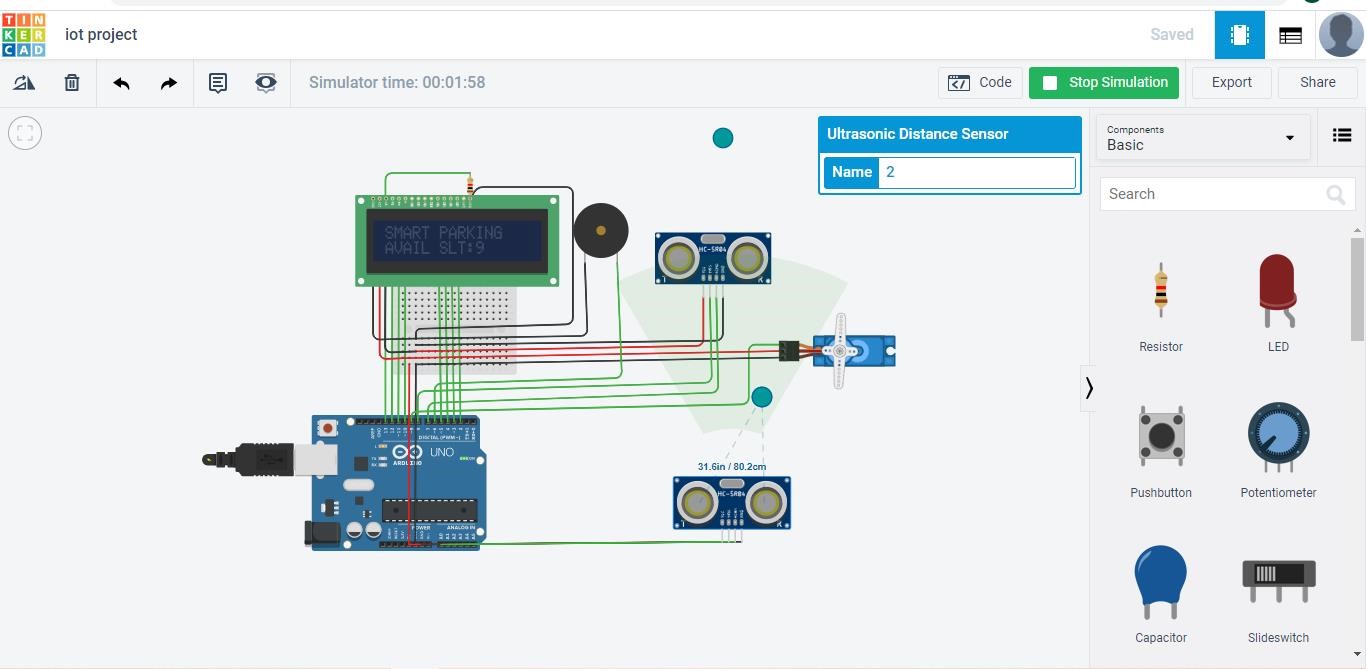


**TINKERCAD RESULTS:**

**WHEN CAR IS APPROACHING TO THE GARAGE- SLOT OF 15 CARS DECREASES TO 8**



**WHEN A CAR IS LEAVING FROM THE GARAGE –A LOT OF 15 CARS INCRESES FROM 8 TO 9**



**4.2 CONCLUSION:**

The concept of Smart Cities have always been a dream for humanity. Since the past couple of years large advancements have been made in making smart cities a reality. The growth of Internet of Things and Cloud technologies have give rise to new possibilities in terms of smart cities. Smart parking facilities and traffic management systems have always been at the core of constructing smart cities. In this paper, we address the issue of parking and present an IoT based Cloud integrated smart parking system. The system that we propose provides real time information regarding availability of parking slots in a parking area. Users from remote locations could book a parking slot for them by the use of our mobile application. The efforts made in this paper are indented to improve the parking facilities of a city and thereby aiming to enhance the quality of life of its people.

**4.3 INTERFERENCE:**

This study has proposed a parking system that improves performance by reducing the number of users that fail to find a parking space and minimizes the costs of moving to the parking space. Our proposed architecture and system has been successfully simulated and implemented in a real situation. The results show that our algorithm significantly reduces the average waiting time of users for parking. Our results closely agree with those of our proposed mathematical models. The simulation of our system achieved the optimal solution when most of the vehicles successfully found a free parking space. The average waiting time of each car park for service becomes minimal, and the total time of each vehicle in each car park is reduced. In our future study, we will consider the security aspects closely of our system as well as implement our proposed system in large scales in the real world.

**4.4 FUTURE WORK :**

1. GSM module can be added to send all information directly to user.
2. GPS module to detect the current location of car to increase the security and advance working of parking system.
3. Face ID and biometric can be added to increase security of system.
4. Better microprocessor can be used to increase the speed of system.
5. Automatic empty slot detection can be implemented in parking premises.
6. Better ML algo can be implied to predict the incoming and outgoing time of vehicle.

**5. REFERENCE:**

1. Rico, J., Sancho, J., Cendon, B., & Camus, M. (2013, March). Parking easier by using context information of a smart city: Enabling fast search and management of parking resources. In Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on (pp. 1380-1385). IEEE.
2. Zheng, Y., Rajasegarar, S., & Leckie, C. (2015, April). Parking availability prediction for sensor-enabled car parks in smart cities. In Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2015 IEEE Tenth International Conference on (pp. 1-6). IEEE.

**6. BIODATA**

Name: ABHYUDAY SINGH MANDLOI

Name

E

-

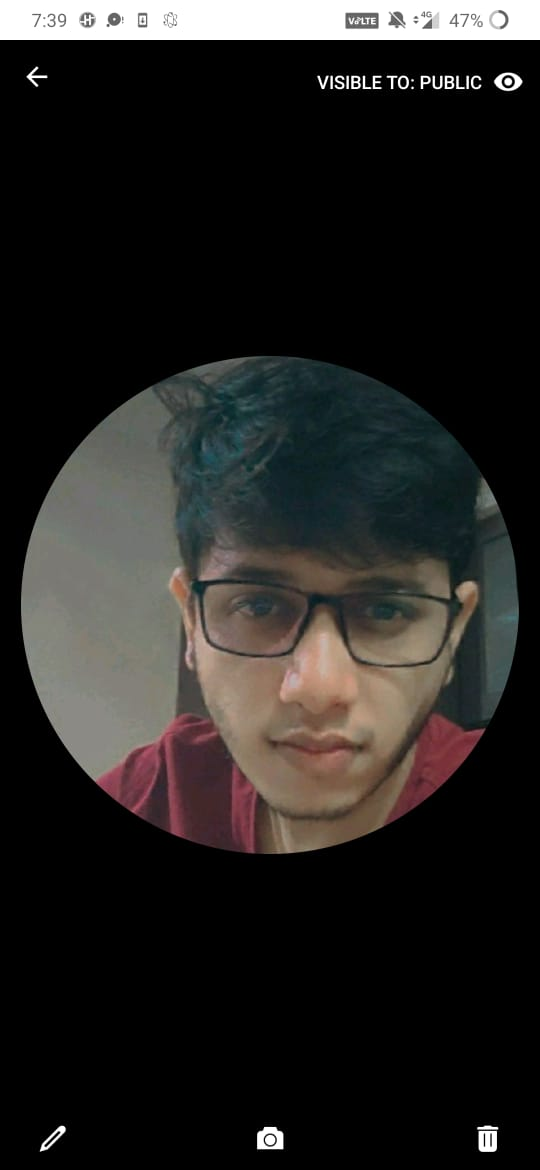
mail



Mobile Number: 9826990507

E-mail: abhyudaysingh.mandloi2018@vitstudent.ac.in

Permanent Address: Indore, Madhya Pradesh



Name: PRERIT AGRAWAL

Mobile Number:7000185123

E-mail: prerit.agrawal2018@vitstudent.ac.in

Permanent Address: RAIPUR

1. **WORKING PROTOTYPE**

**VIDEO link of woking model:** [**https://drive.google.com/file/d/1F1vsqg-v-RZSSyF7GfDdqoKVVnGct5\_V/view?usp=sharing**](https://drive.google.com/file/d/1F1vsqg-v-RZSSyF7GfDdqoKVVnGct5_V/view?usp=sharing)

**GOOGLE DRIVE WITH ALL DATASET , SIMULATION & ML IMPLEMENTATION:**

[**https://drive.google.com/drive/folders/1UjPJIf\_2-91-vO4SdAAYsnU2jVphe7NF?usp=sharing**](https://drive.google.com/drive/folders/1UjPJIf_2-91-vO4SdAAYsnU2jVphe7NF?usp=sharing%20)

--------------------------------------THANK YOU----------------------------------------