

# **PROJECT REPORT**

## **TITLE: SMART PARKING SYSTEM FOR SMART CITIES**



**SUBMITTED BY:**

**KARTHIK BHARADWAJ R M**

# **1. INTRODUCTION**

## **1.1 Overview:**

Smart Parking System in Smart Cities project is about a new parking system called Smart Parking System. This 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 focuses on reducing the time in finding the parking lots. And also it avoids the unnecessary traveling through filled parking lots in a parking area. We can check the status of the parking slot by using sensors.

With the advent of Internet of Things, the concept of smart cities can be readily achievable. An extensive research is ongoing in the field of Internet of Things to increase the quality of services. It offered in cities and to improve the productivity and reliability of urban infrastructure.

IoT is addressing the most common problems faced in cities like availability of car parking and traffic jams. This paper presents an Internet of Things based Parking system for Smart Cities. The proposed parking system contains an IoT module deployed on-site for managing the available parking spaces. A platform provided in the form of portal for booking the parking spaces.

## **1.2 Purpose:**

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 IoT and Cloud technologies have given 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.

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.

## 2. LITERATURE SURVEY

### 2.1 Existing Problem:

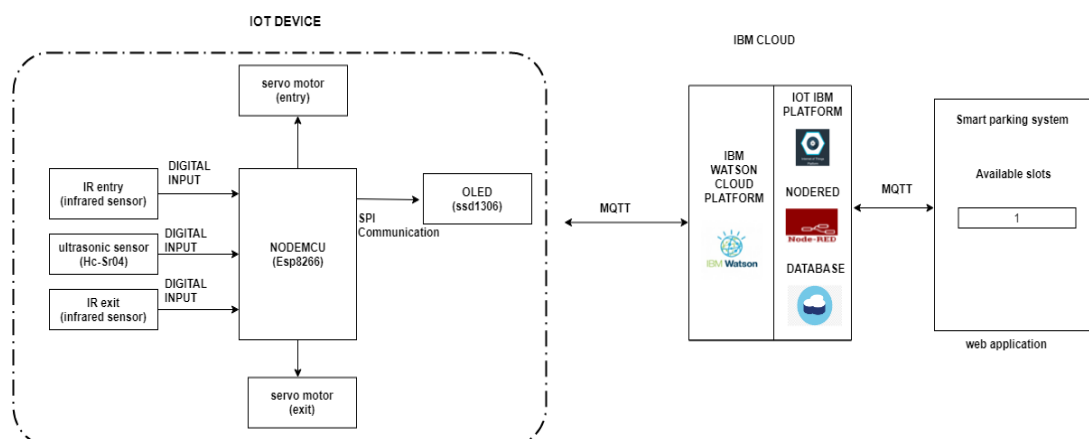
Traffic management and car parking on modern cities continues to be a problem both for citizens and for city officials. The increasing number of vehicles flowing into the city drain the existing scarce parking resources, and the increase in time spent looking for a parking spot leads to more congestions, parasitic traffic, whilst augmenting fuel consumption and air pollution.

### 2.2 Proposed Solution:

This project proposes a smart car parking system that will assist users to solve the issue of finding a parking space and to minimise the time spent in searching for the nearest available car park. In addition, it provides users with roads traffic congestion status.

## 3. THEORITICAL ANALYSIS

### 3.1 Block Diagram:



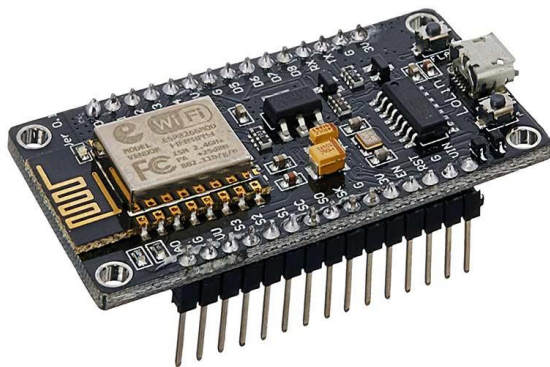
## 3.2 Hardware and Software Designing

### Hardware Components used:

- NodeMCU (ESP8266)
- IR Sensor
- Ultrasonic Sensor
- Servo Motor

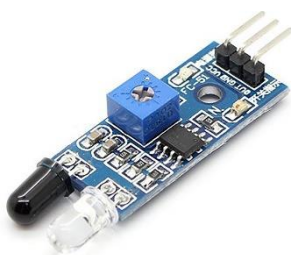
### NodeMCU (ESP8266):

NodeMCU is an open-source Lua based firmware and development board specially targeted for IoT based Applications. It includes firmware that runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module.



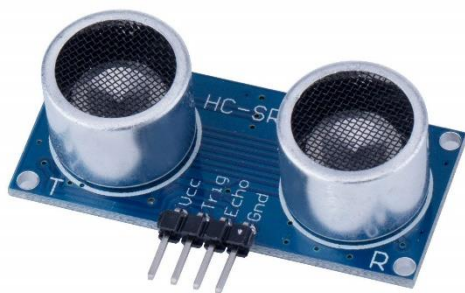
### IR Sensor:

An infrared (IR) sensor is an electronic device that measures and detects infrared radiation in its surrounding environment. There are two types of infrared sensors: active and passive. Active infrared sensors both emit and detect infrared radiation. Active IR sensors have two parts: a light emitting diode (LED) and a receiver. When an object comes close to the sensor, the infrared light from the LED reflects off of the object and is detected by the receiver. Active IR sensors act as proximity sensors, and they are commonly used in obstacle detection systems (such as in robots).



### **Ultrasonic Sensor:**

An ultrasonic sensor is an electronic device that measures the distance of a target object by emitting ultrasonic sound waves, and converts the reflected sound into an electrical signal. Ultrasonic waves travel faster than the speed of audible sound (i.e. the sound that humans can hear). Ultrasonic sensors have two main components: the transmitter (which emits the sound using piezoelectric crystals) and the receiver (which encounters the sound after it has travelled to and from the target).



### **Servo Motor:**

A servo motor is a rotary actuator or a motor that allows for a precise control in terms of the angular position, acceleration, and velocity. Basically it has certain capabilities that a regular motor does not have. Consequently it makes use of a regular motor and pairs it with a sensor for position feedback. A typical servo motor comprises of three wires namely- power, control, and ground. The shape and size of these motors depends on their applications.

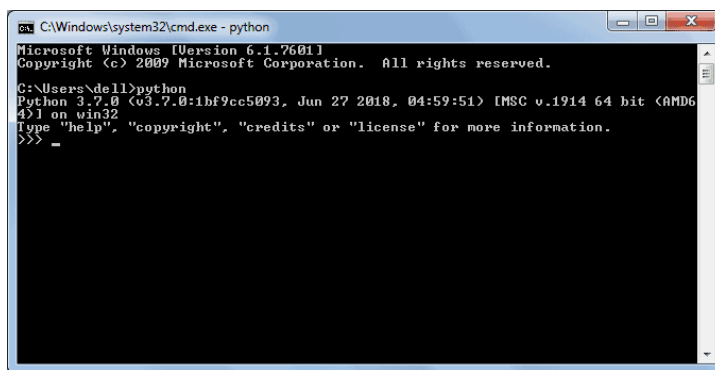


## Software Used:

- Python Shell
- IBM Cloud
- Node-RED
- MIT App Inventor

## Python Shell:

Python provides a Python Shell (also known as Python Interactive Shell) which is used to execute a single python command and get the result. Python Shell waits for the input command from the user. To open the Python Shell on Windows, open the command prompt and press enter.



```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\dell>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> _
```

## IBM Cloud:

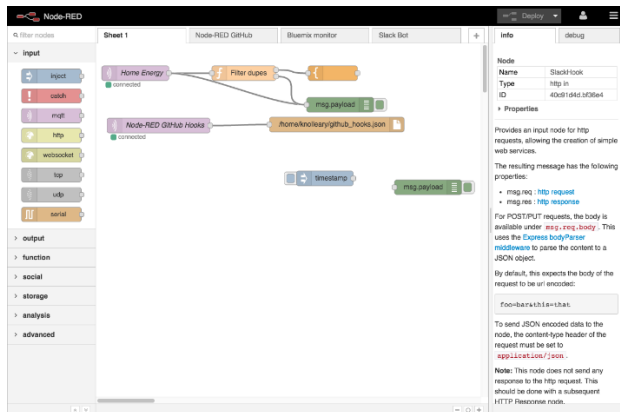
IBM Cloud is a suite of cloud computing services from IBM that offers both Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). With IBM Cloud IaaS, organisations can deploy and access virtualized IT resources such as compute power, storage and networking- over the internet.



# IBM Cloud

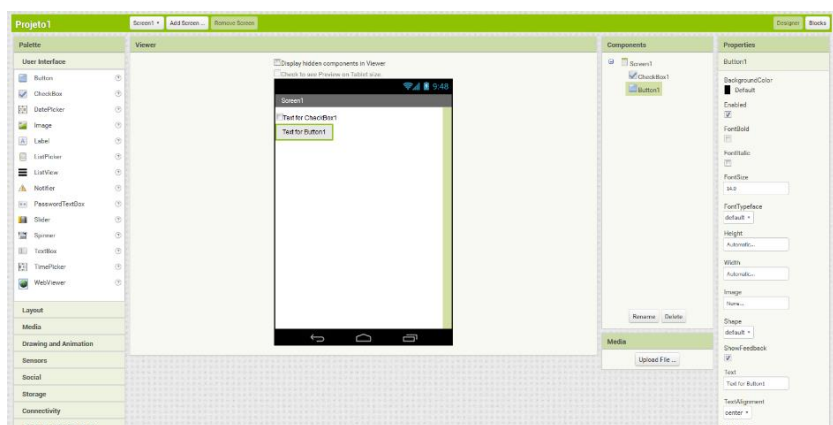
## Node-RED:

NodeRED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways. It provides a browser based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single-click.



## MIT App Inventor:

MIT App Inventor is an online platform designed to teach computational thinking concepts through development of mobile applications. We can create applications by dragging and dropping components into a design view and using a visual blocks language to program application behaviour.

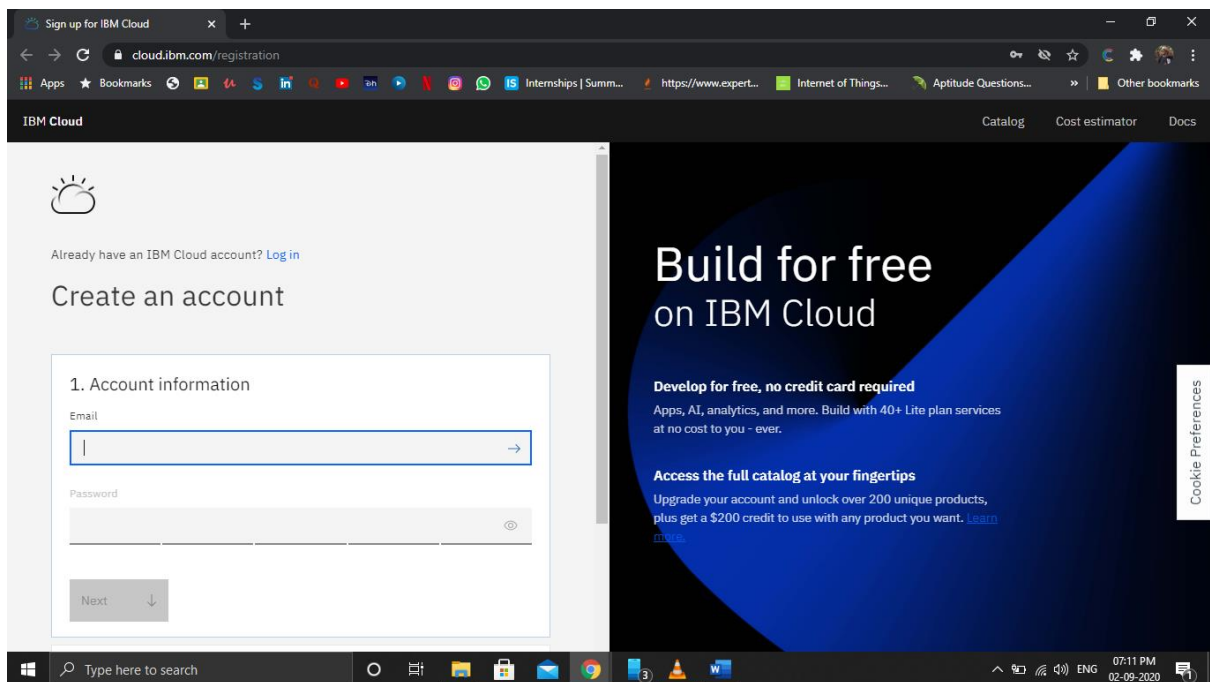


## 4. EXPERIMENTAL INVESTIGATION

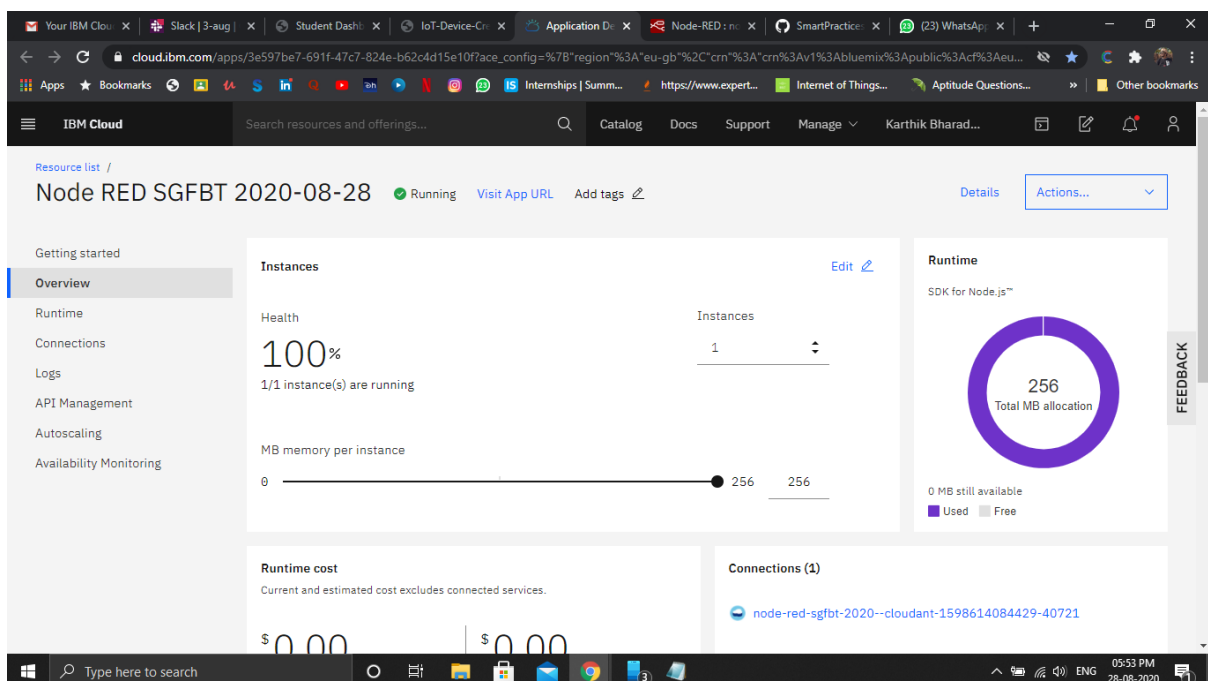
The IoT based Cold Smart Parking system is structured as below:

### ❖ Setup Environment:

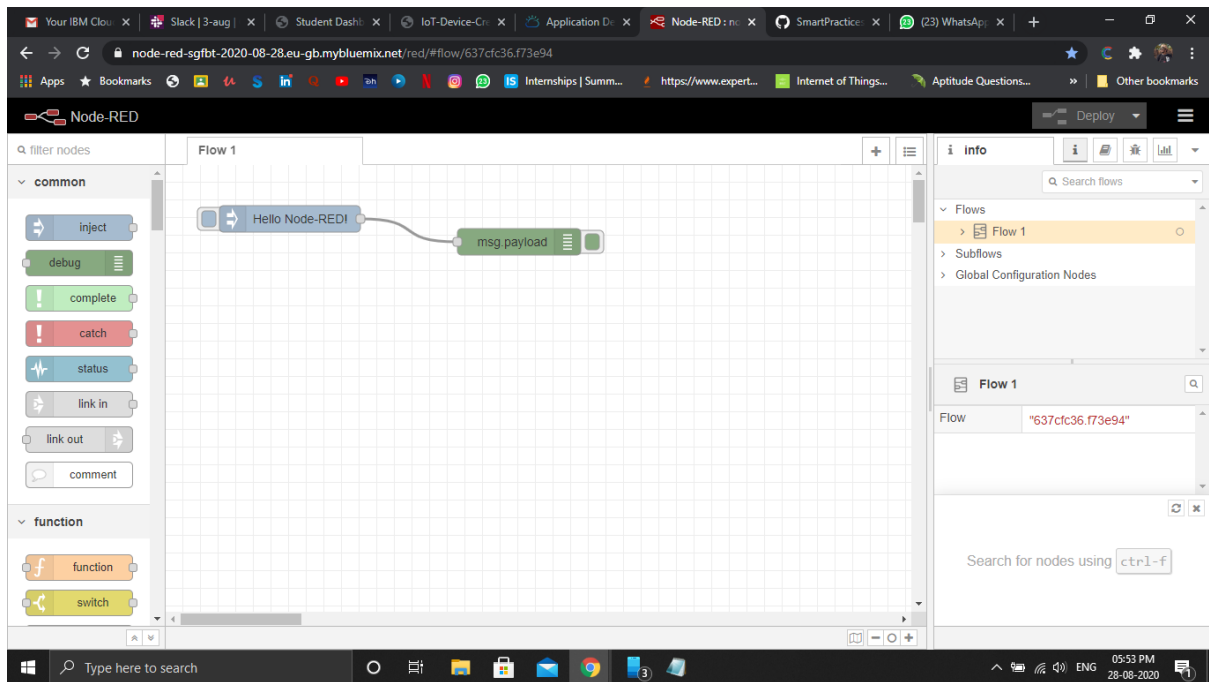
#### 1. Create an IBM Account



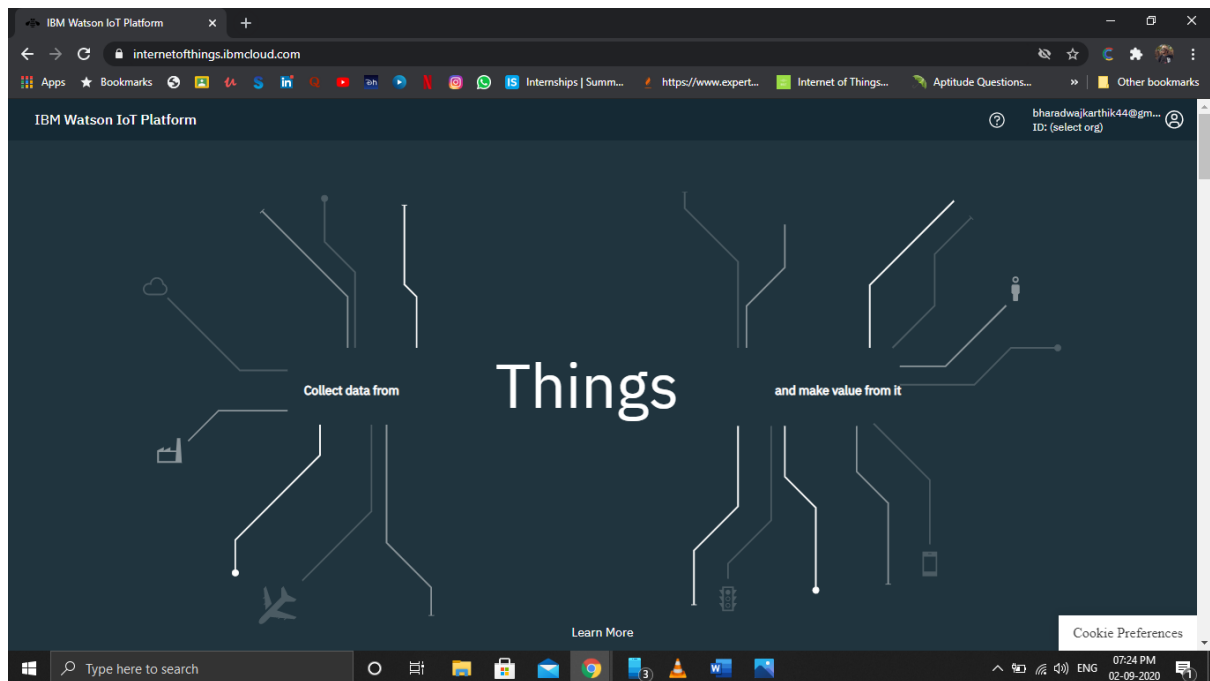
#### 2. Create a Node-Red Application

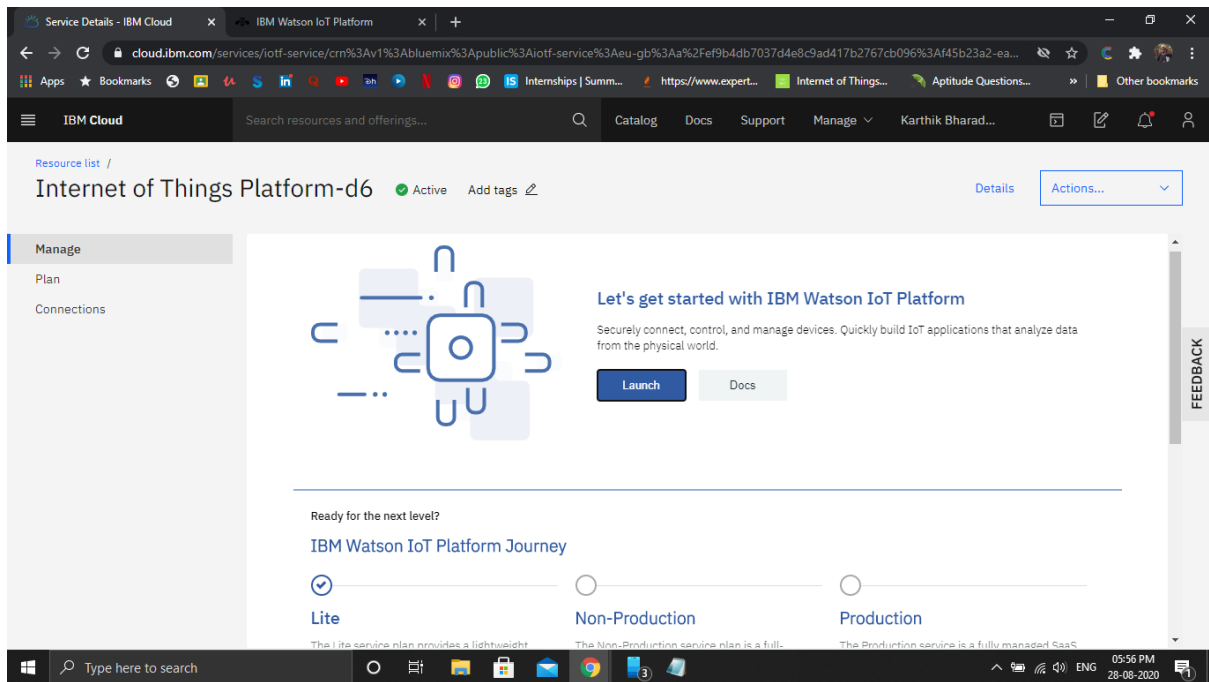






### 3. Create An IBM Watson IoT Platform





## ❖ Setup Hardware and Develop the Code:

### 1. Code Snippet for publishing data using MQTT Communication

```
Python 3.8.5 Shell - C:/Users/RMKB/OneDrive/Desktop/smartpark.py (3.8.5)
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import time
import sys
import ibmiotf.application
import ibmiotf.device
import random
#Provide your IBM Watson Device Credentials
organization = "00ougak"
deviceType = "NodeMCU"
deviceId = "mmul23"
authMethod = "use-token-auth"
authToken = "smarthome123"

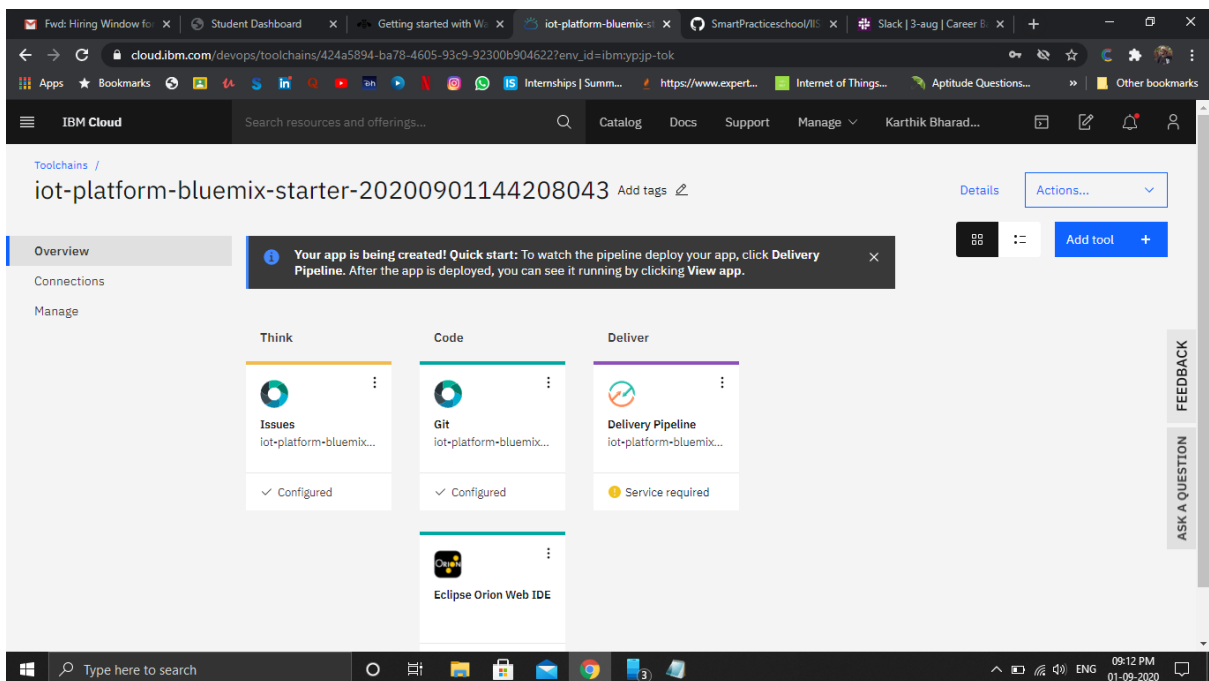
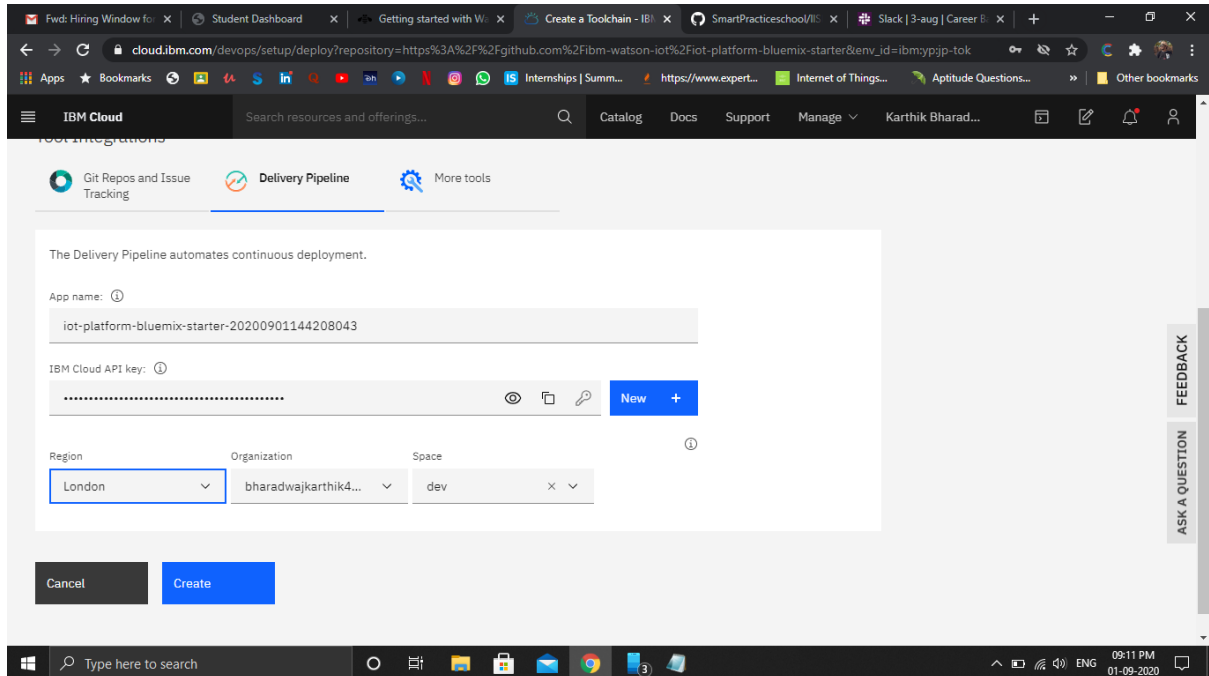
def myCommandCallback(cmd):
    print("Command received: %s" % cmd.data) #Commands

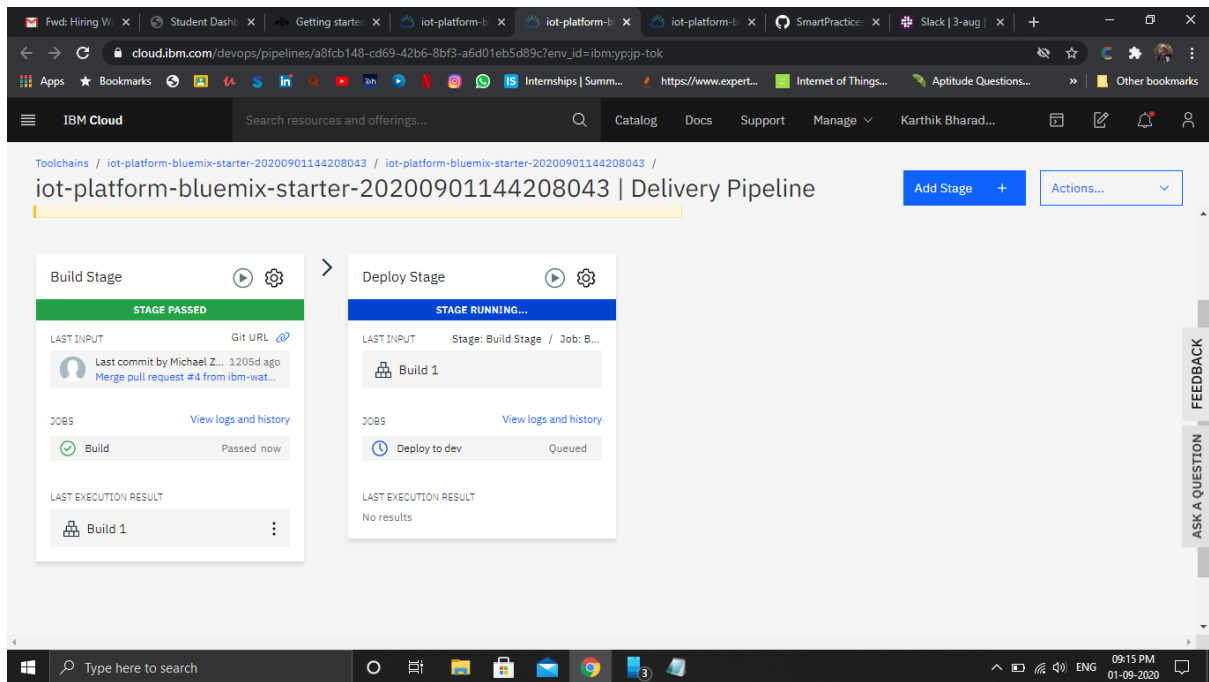
try:
    deviceOptions = {"org": organization, "type": deviceType, "id": deviceId, "auth-method": authMethod, "auth-token": authToken}
    deviceCli = ibmiotf.device.Client(deviceOptions)
    #.....

except Exception as e:
    print("Caught exception connecting device: %s" % str(e))
    sys.exit()
```

## ❖ Building a Web:

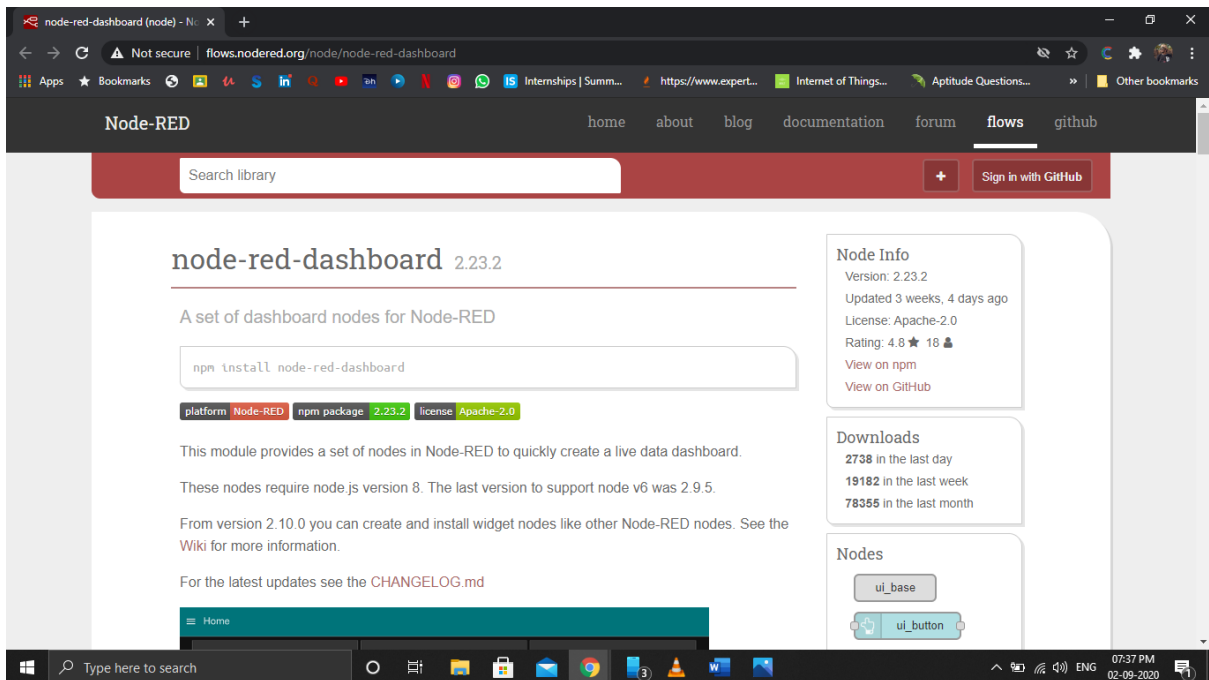
### 1. Create a node-red flow to get data from the device

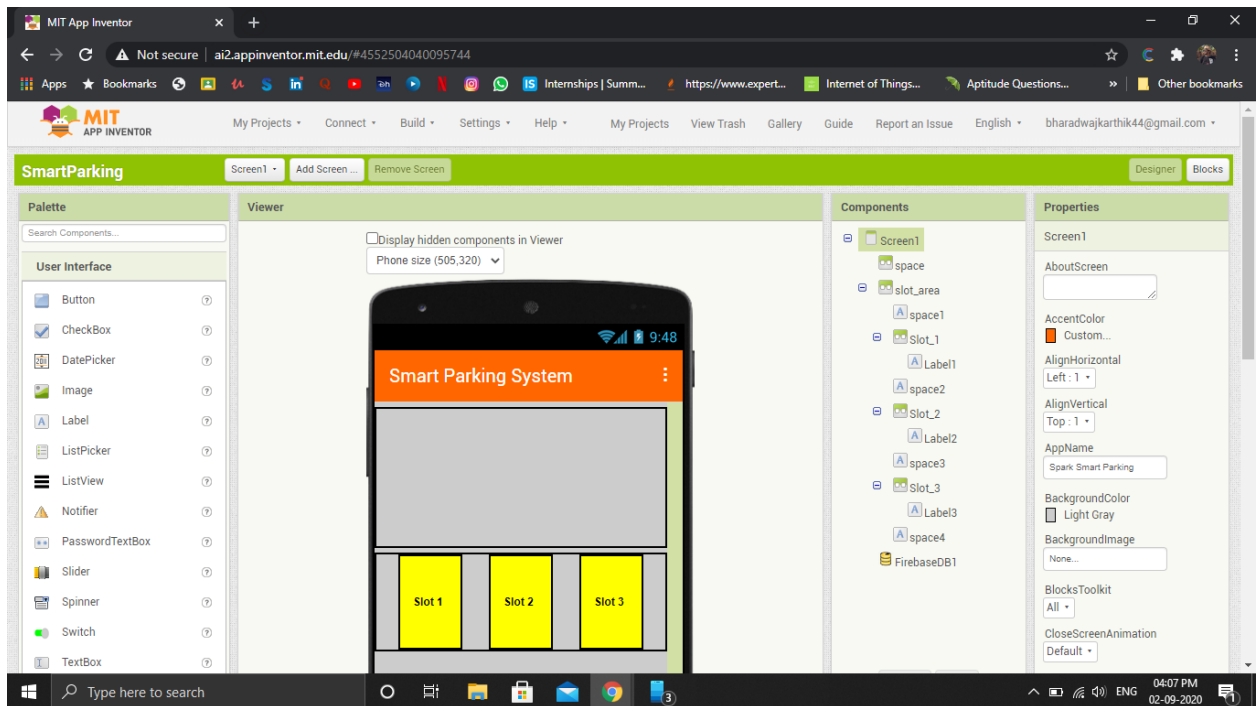
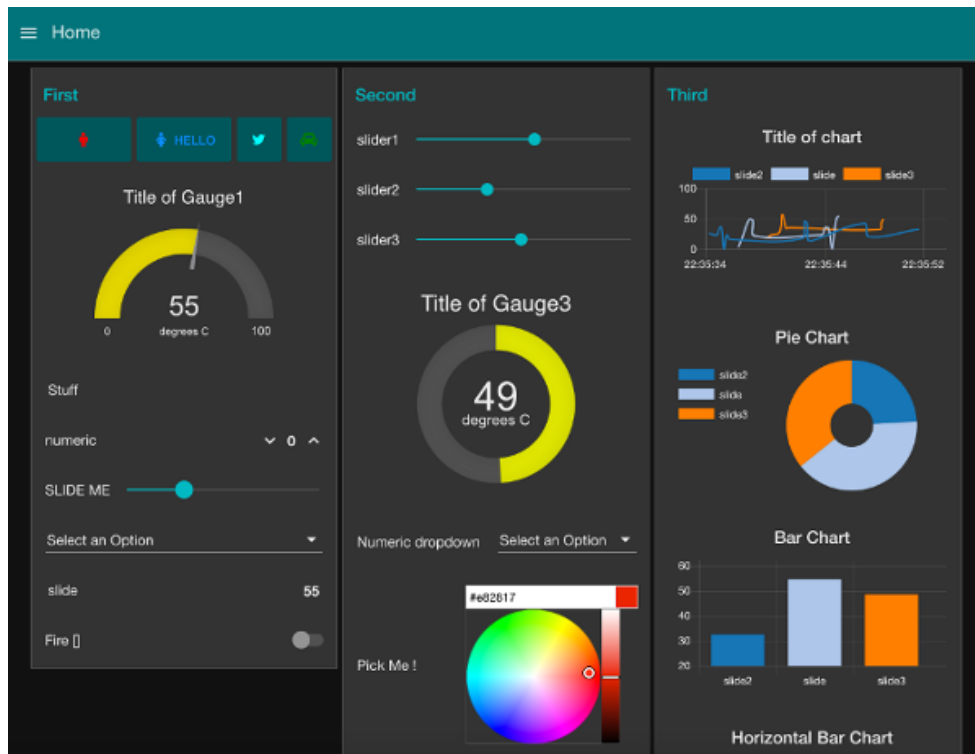




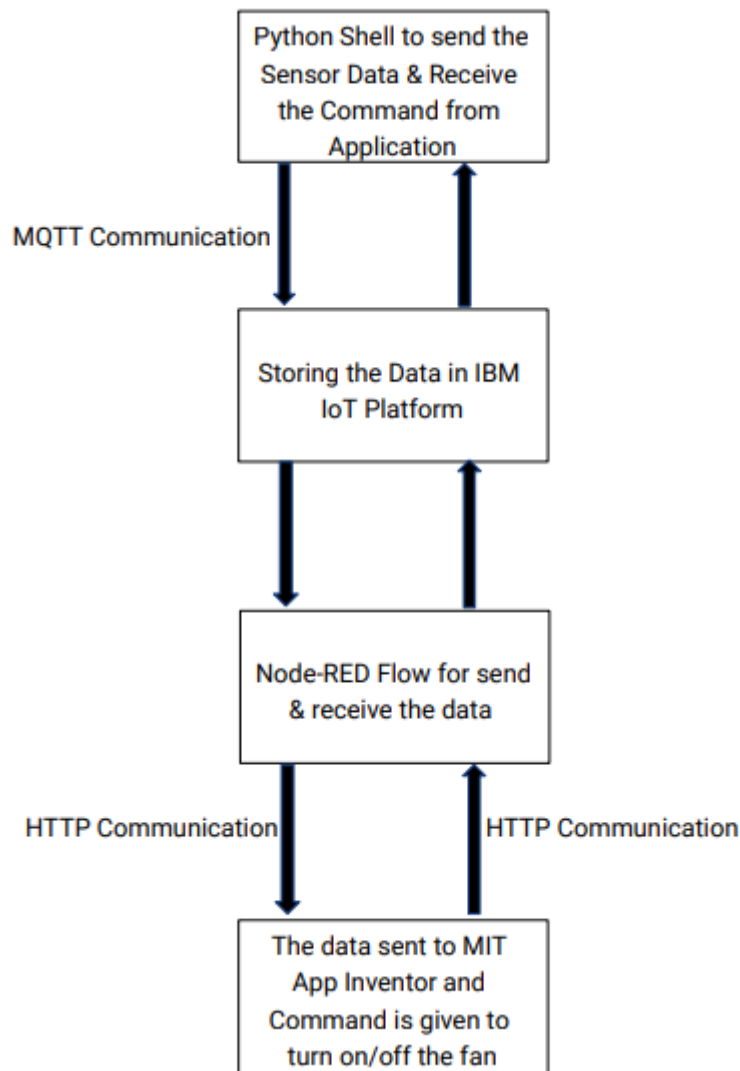
## ❖ Use dashboard nodes for creating UI (Web App):

### 1. UI Nodes Installation





## 5. FLOWCHART



## 6. RESULT

The Smart Parking System for Smart cities using IoT was successfully built where the data was sent using the MQTT communication protocol to the IBM IoT Platform Cloud then to the Node-RED flow, from Node-RED using HTTP Communication protocol to the Application where the user can monitor the parking using the Mobile Application.

## **7. ADVANTAGES AND DISADVANTAGES**

### **Advantages of Automated Parking Facilities:**

- \* There is a greater sense of security due to the fact that patrons do not actually walk to and from their own space.
- \* It is highly feasible for extremely small sites that are unable to accommodate a conventional ramped parking structure.
- \* There is high parking efficiency.
- \* There is no need for driving while looking for an available space.
- \* Emissions are greatly brought down and reduced.
- \* The patrons wait for their car in a highly controlled environment.
- \* There are less chances for vehicle vandalism.
- \* There is a minimal staff requirement if it is used by known parkers.
- \* It is possible that the retrieval time is lower than the combined driving/parking/walking time in conventional ramped parking structures.
- \* There is an easier facade integration since there are no ramping floors or openings in exterior walls.

### **Potential Disadvantages Of Automated Parking Facilities:**

- \* There is a greater construction cost per space (but this may be offset by the chance for lesser land costs per space and the system manufacturers say that the operating and maintenance cost will be lower as compared to a conventional ramped parking structure).
- \* Use of redundant systems will result in a greater cost.
- \* It may be a bit confusing for unfamiliar users.
- \* It is not recommended for high peak hour volume facilities.
- \* There may be a fear of breakdown.
- \* There is an uncertain building department review and approval process.
- \* It requires a maintenance contract with the supplier.

## 8. APPLICATIONS

Smart Parking involves the use of low cost sensors, real-time data and applications that allow users to monitor available and unavailable parking spots. The goal is to automate and decrease time spent manually searching for the optimal parking floor, spot and even lot. Some solutions will encompass a complete suite of services such as online payments, parking time notifications and even car searching functionalities for very large lots. A parking solution can greatly benefit both the user and the lot owner. Here are some of the top benefits:

- **Optimized parking** – Users find the best spot available, saving time, resources and effort. The parking lot fills up efficiently and space can be utilized properly by commercial and corporate entities.
- **Reduced traffic** – Traffic flow increases as fewer cars are required to drive around in search of an open parking space.
- **Reduced pollution** – Searching for parking burns around one million barrels of oil a day. An optimal parking solution will significantly decrease driving time, thus lowering the amount of daily vehicle emissions and ultimately reducing the global environmental footprint.
- **Enhanced User Experience** – A smart parking solution will integrate the entire user experience into a unified action. Driver's payment, spot identification, location search and time notifications all seamlessly become part of the destination arrival process.
- **New Revenue Streams** – Many new revenue streams are possible with smart parking technology. For example, lot owners can enable tiered payment options dependent on parking space location. Also, reward programs can be integrated into existing models to encourage repeat users.
- **Integrated Payments and POS** – Returning users can replace daily, manual cash payments with account invoicing and application payments from their phone. This could also enable customer loyalty programs and valuable user feedback.
- **Increased Safety** – Parking lot employees and security guards contain real-time lot data that can help prevent parking violations and suspicious activity. License plate recognition cameras can gather pertinent footage. Also, decreased spot-searching traffic on the streets can reduce accidents caused by the distraction of searching for parking.
- **Real-Time Data and Trend Insight** – Over time, a smart parking solution can produce data that uncovers correlations and trends of users and lots. These trends can prove to be invaluable to lot owners as to how to make adjustments and improvements to drivers.



- **Decreased Management Costs** – More automation and less manual activity saves on labor cost and resource exhaustion.
- **Increased Service and Brand Image** – A seamless experience can really skyrocket a corporate or commercial entities brand image to the user. Whether the destination is a retail store, an airport or a corporate business office, visitors will surely be impressed with the cutting edge technology and convenience factors.

## 9. CONCLUSION

The implementation of the smart parking system being presented, its efficiency in alleviating the traffic problem that arises especially in the city area where traffic congestion and the insufficient parking spaces are undeniable. It does so by directing patrons and optimizing the use of parking spaces. The system benefit of smart parking go well beyond avoiding time wasting. Developing smart parking system within the city also solves the pollution problem. Thus it is recommended to implement Smart Parking System in Smart Cities.

## 10. FUTURE SCOPE

The smart parking industry continues to evolve as an increasing number of cities struggle with traffic congestion and inadequate parking availability. While the deployment of sensor technologies continues to be core to the development of smart parking, a wide variety of other technology innovations are also enabling more adaptable systems—including cameras, wireless communications, data analytics, induction loops, smart parking meters, and advanced algorithms.

## 11. BIBLIOGRAPHY

- ❖ <https://iopscience.iop.org/article/10.1088/1742-6596/1339/1/012044/pdf>
- ❖ <https://www.ijeat.org/wp-content/uploads/papers/v9i1/A1963109119.pdf>
- ❖ [https://www.researchgate.net/publication/329686583\\_IoT\\_Based\\_Smart\\_Parking\\_System](https://www.researchgate.net/publication/329686583_IoT_Based_Smart_Parking_System)
- ❖ <https://iotdesignpro.com/projects/iot-based-smart-parking-using-esp8266>
- ❖ <https://www.digiteum.com/iot-smart-parking-solutions>

```
smart parking code.py - C:/Users/RMKB/OneDrive/Desktop/smart parking code.py (3.8.5)
File Edit Format Run Options Window Help

import cv2
import csv
VIDEO_SOURCE = 1

cap = cv2.VideoCapture(VIDEO_SOURCE)
suc, image = cap.read()
cv2.imwrite("frame0.jpg", image)
cap.release()
cv2.destroyAllWindows()
img = cv2.imread("frame0.jpg")
r = cv2.selectROIs('Selector', img, showCrosshair = False, fromCenter = False)
rlist = r.tolist()
def drawRectangle(img, a, b, c, d):
    sub_img = img[b:b + d, a:a + c]
    edges = cv2.Canny(sub_img, lowThreshold, highThreshold)
    pix = cv2.countNonZero(edges)

    if pix in range(min, max):
        cv2.rectangle(img, (a, b), (a + c, b + d), (0, 255, 0), 3)
        spots.loc += 1
    else:
        cv2.rectangle(img, (a, b), (a + c, b + d), (0, 0, 255), 3)
    def callback(foo):
        pass
cv2.createTrackbar
cv2.namedWindow('parameters')
cv2.createTrackbar('Threshold1', 'parameters', 186, 700, callback)
cv2.createTrackbar('Threshold2', 'parameters', 122, 700, callback)
cv2.createTrackbar('Min pixels', 'parameters', 100, 1500, callback)
cv2.createTrackbar('Max pixels', 'parameters', 323, 1500, callback)
class spots:
    loc = 0
    with open('data/rois.csv', 'r', newline='') as inf:
        csvr = csv.reader(inf)
        rois = list(csvr)

rois = [[int(float(j)) for j in i] for i in rois]
VIDEO_SOURCE = 1
cap = cv2.VideoCapture(VIDEO_SOURCE)

while True:
    pass
```

```
smart parking code.py - C:/Users/RMKB/OneDrive/Desktop/smart parking code.py (3.8.5)
File Edit Format Run Options Window Help

cv2.createTrackbar
cv2.namedWindow('parameters')
cv2.createTrackbar('Threshold1', 'parameters', 186, 700, callback)
cv2.createTrackbar('Threshold2', 'parameters', 122, 700, callback)
cv2.createTrackbar('Min pixels', 'parameters', 100, 1500, callback)
cv2.createTrackbar('Max pixels', 'parameters', 323, 1500, callback)
class spots:
    loc = 0
    with open('data/rois.csv', 'r', newline='') as inf:
        csvr = csv.reader(inf)
        rois = list(csvr)

rois = [[int(float(j)) for j in i] for i in rois]
VIDEO_SOURCE = 1
cap = cv2.VideoCapture(VIDEO_SOURCE)

while True:
    spots.loc = 0

    ret, frame = cap.read()
    ret2, frame2 = cap.read()
    min = cv2.getTrackbarPos('Min pixels', 'parameters')
    max = cv2.getTrackbarPos('Max pixels', 'parameters')
    lowThreshold = cv2.getTrackbarPos('Threshold1', 'parameters')
    highThreshold = cv2.getTrackbarPos('Threshold2', 'parameters')

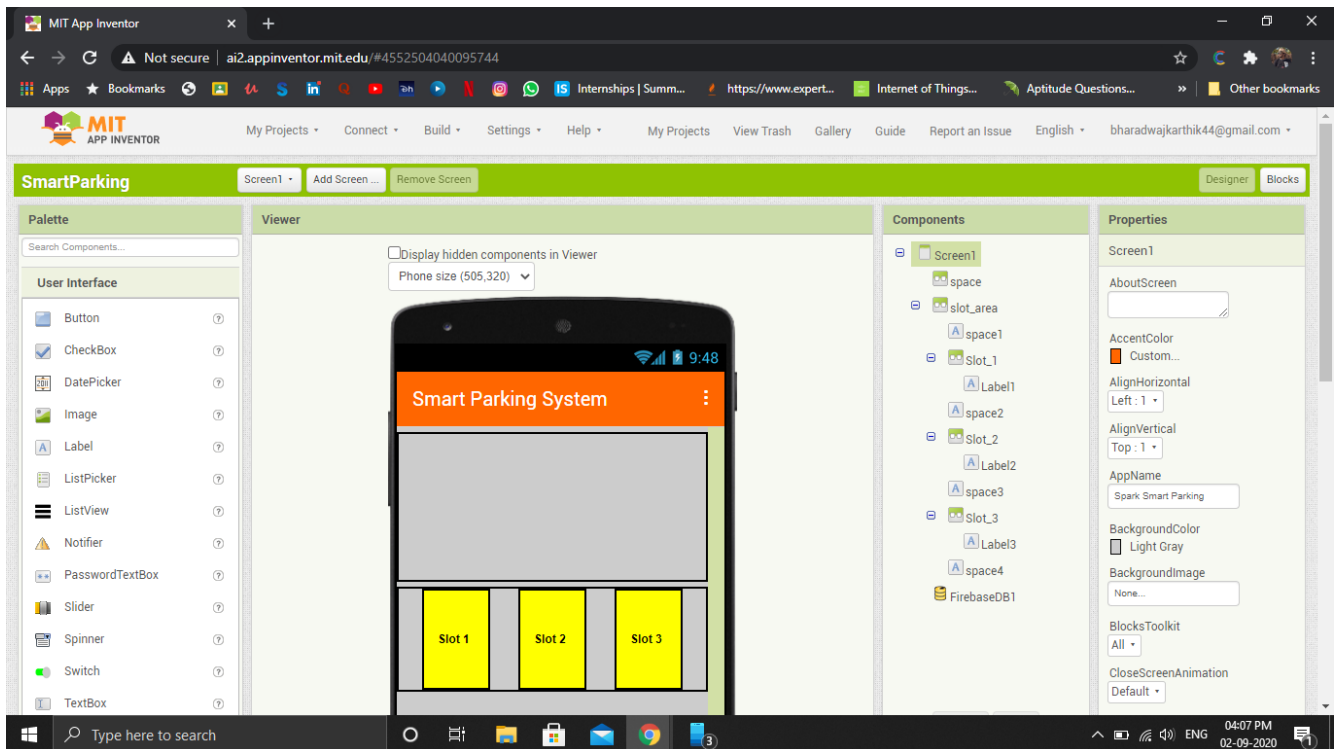
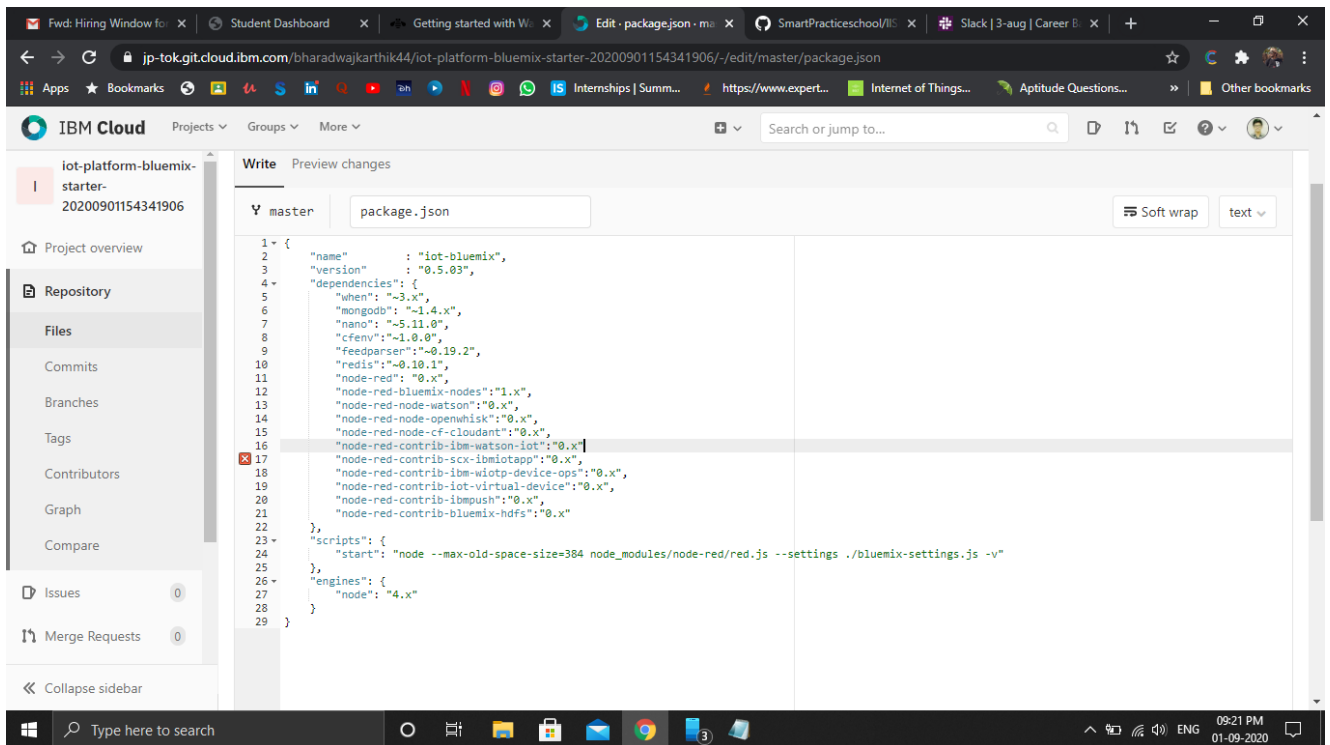
    for i in range(len(rois)):
        drawRectangle(frame, rois[i][0], rois[i][1], rois[i][2], rois[i][3])

    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(frame, 'Available spots: ' + str(spots.loc), (10, 30), font, 1, (0, 255, 0), 3)
    cv2.imshow('Detector', frame)

    canny = cv2.Canny(frame2, lowThreshold, highThreshold)
    cv2.imshow('canny', canny)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```



**THANK YOU**