SMART GUEST IDENTIFIER WITH REMOTE ACCESS MANAGEMENT (IOT BASED PROJECT)

1.INTRODUCTION

1.1) Overview:

Sometimes we will be in our offices or we may go out and our loved ones and friends may visit us. We may miss them as we don't have any idea that someone has come to meet us. so by using smart guest identifier we can resolve this problem,with the help of visual recognition and IBM cloud we can detect the person and stores in the cloud.And by integrating both nodered and MIT with python code we can get the image to our phone.With that we can go with our wish.

1.2) Purpose:

Smart Guest Identifier With Remote Access Management is a device which is integrated at the entrance. There will be video streaming and whenever any person comes near the entrance. It will detect the person using IBM Watson visual recognition services. And captures a pic and then that picture is sent to the mobile application.

In the mobile application, the owner can see the person who is in front of their door. And they can also open the door through the mobile application if they want to give access to the visitors.And also if any unauthorised person is roaming at the door we can also get an alert message.

2)LITERATURE SURVEY

2.1) Existing problem:

The existing problem is we have the face detectors when we are at home to receive the person into home .if the person is unknown we don't open the door,but what if a person is at our door when we are outside of the home ?

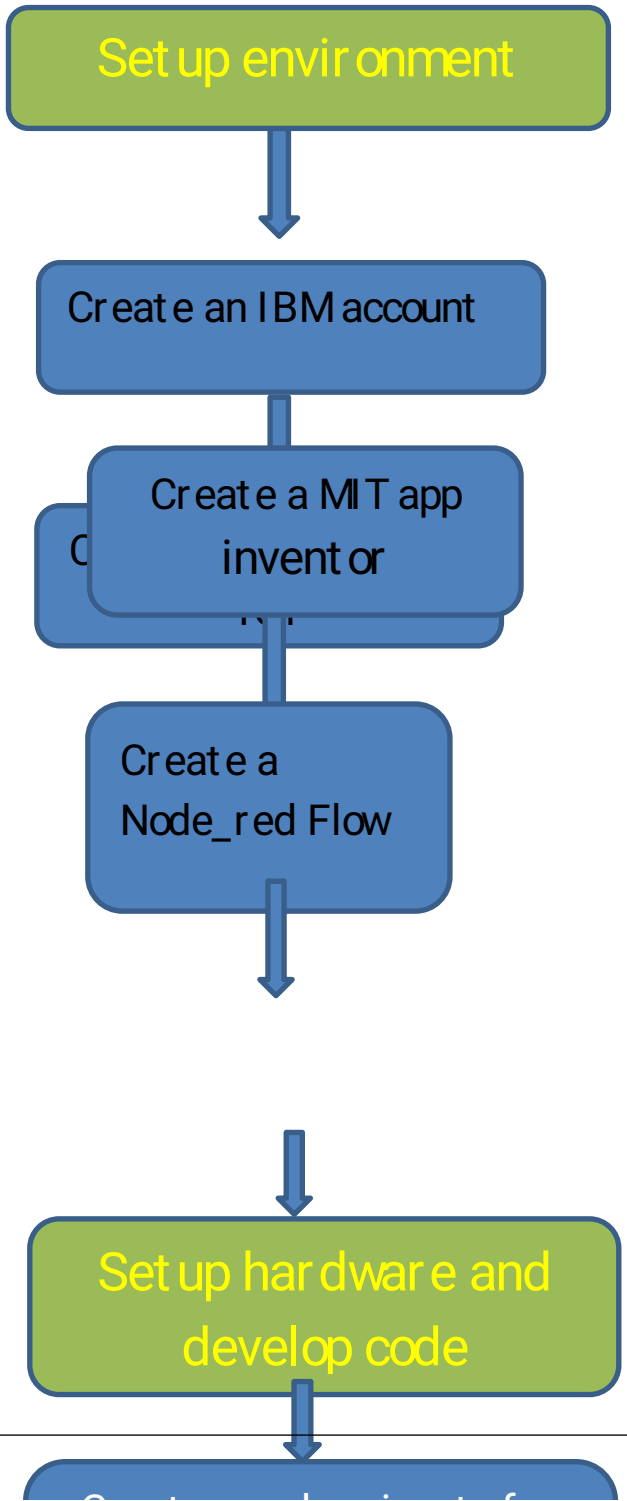
2.2) Proposed solution:

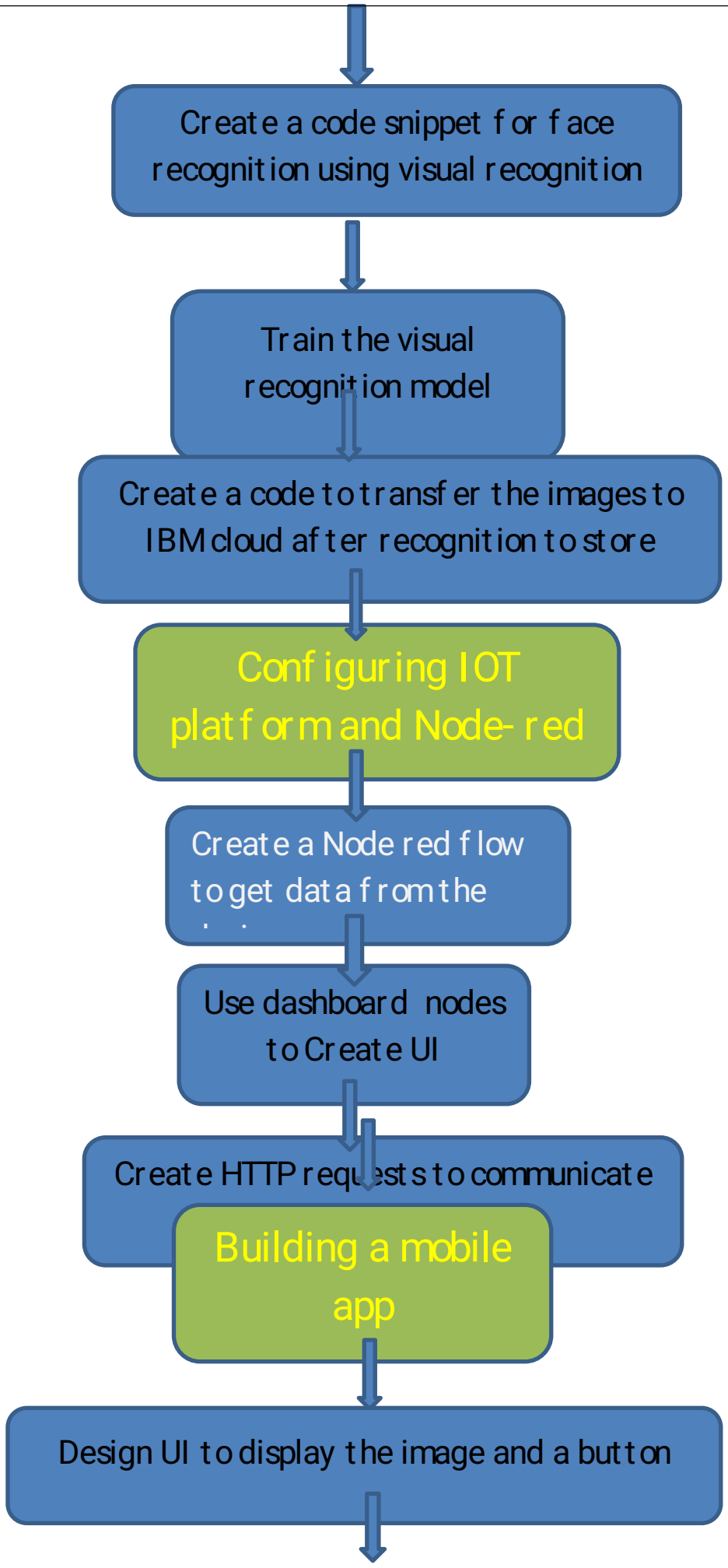
We came with a solution for the above problem by using visual recognition when a person

comes to our house the camera which placed at the door detects the image and send that captured image to the IBM cloud with the help of visual recognition.By integrating the Node_red and MIT with the python code we can get that captured image to our mobile.If the person is known we allow the person inside by clicking button open the door in MIT mobile app.

3) THEORITICAL ANALYSIS

3.1 Block diagram




Create a code snippet for face recognition using visual recognition
Train the visual recognition model
Create a code to transfer the images to IBM cloud after recognition to store
Configuring IOT platform and Node- red
Create a Node red flow to get data from the
Use dashboard nodes to Create UI
Create HTTP requests to communicate
Building a mobile app
Design UI to display the image and a button


Configure the application to
receive the data from cloud

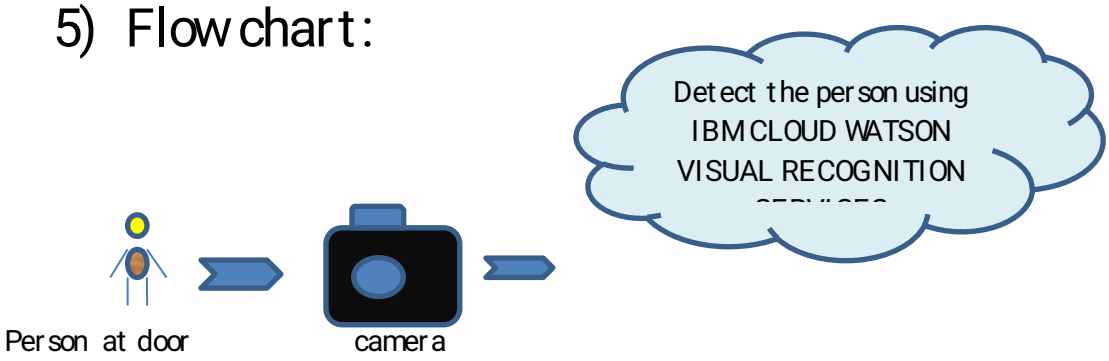
3.2) Hardware / software designing:

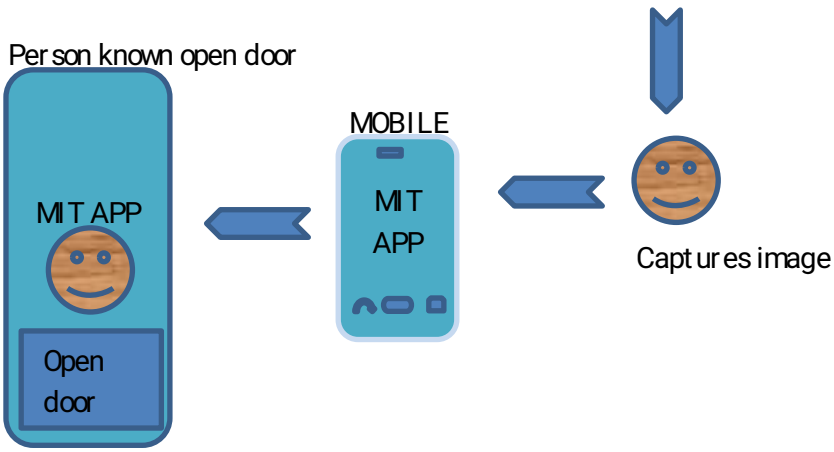
We use the IBM cloud I.e cloud.ibm.com to create an IOT platform.In that platform create a Node_red account which is flow based platform developed based on the NodeJS , consisting a node section in which we have plenty of nodes, a workbench section to create a flow and a console is there to display the commands.In our project we use inject node, debug node some function nodes,IBM in and out nodes,HTTP node.We also use MIT application to built a mobile app.Using visual recognition we get the image to the mobile.We use Python IDLE to run the code with which we integrate our Node_red and MIT to display the status of the buttons.

4)Experiment Investigation:

We start our investigation of our project from Google and with some reference links.At the beginning we find how to create an IBM account and node_red account and also know about MIT to build a mobile app.In that MIT we develop the blocks by using buttons ,web,text box.. We also work on the visual recognition to capture the face. Learn how to create HTTP requests and generation of Urls.After that we search for the python code to integrate node_red and MIT to get the image to the mobile from the cloud.And also prepare the code for the button status and to send emergency alert message. With help of the details provided by our mentor we proceed with our project.

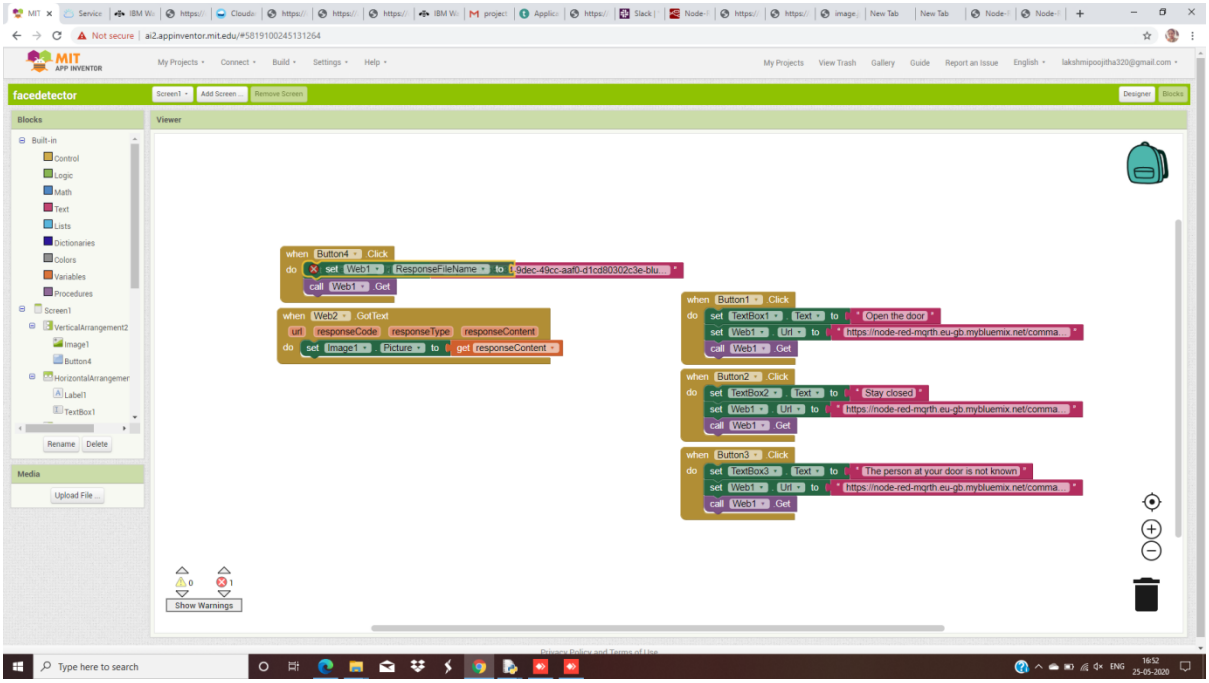
5) Flowchart:



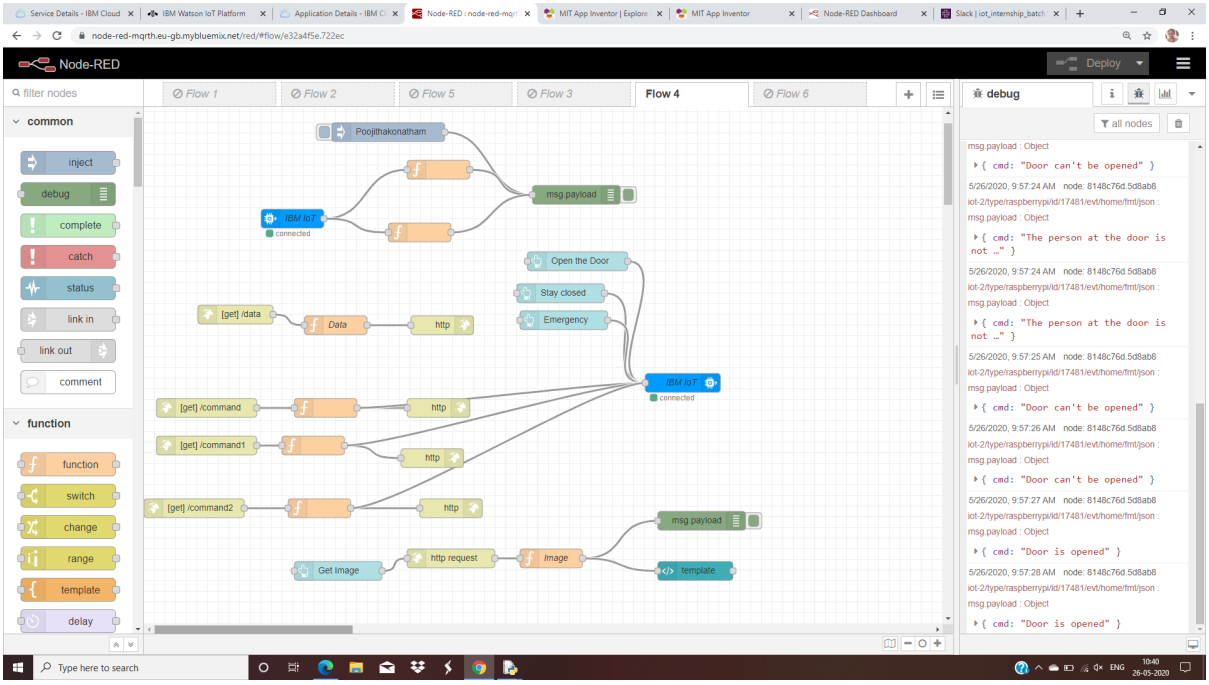

Person known open door
MIT APP
Open door
MOBILE
MIT APP
Captures image

6) Result :

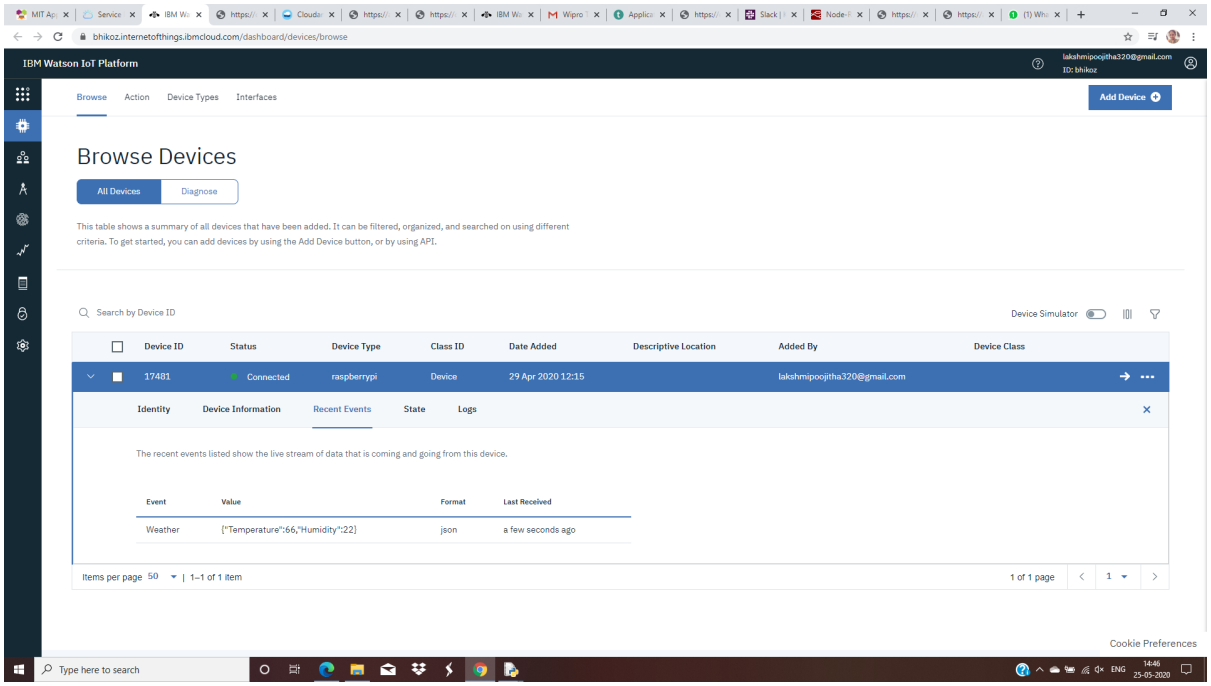
Blocks Construction



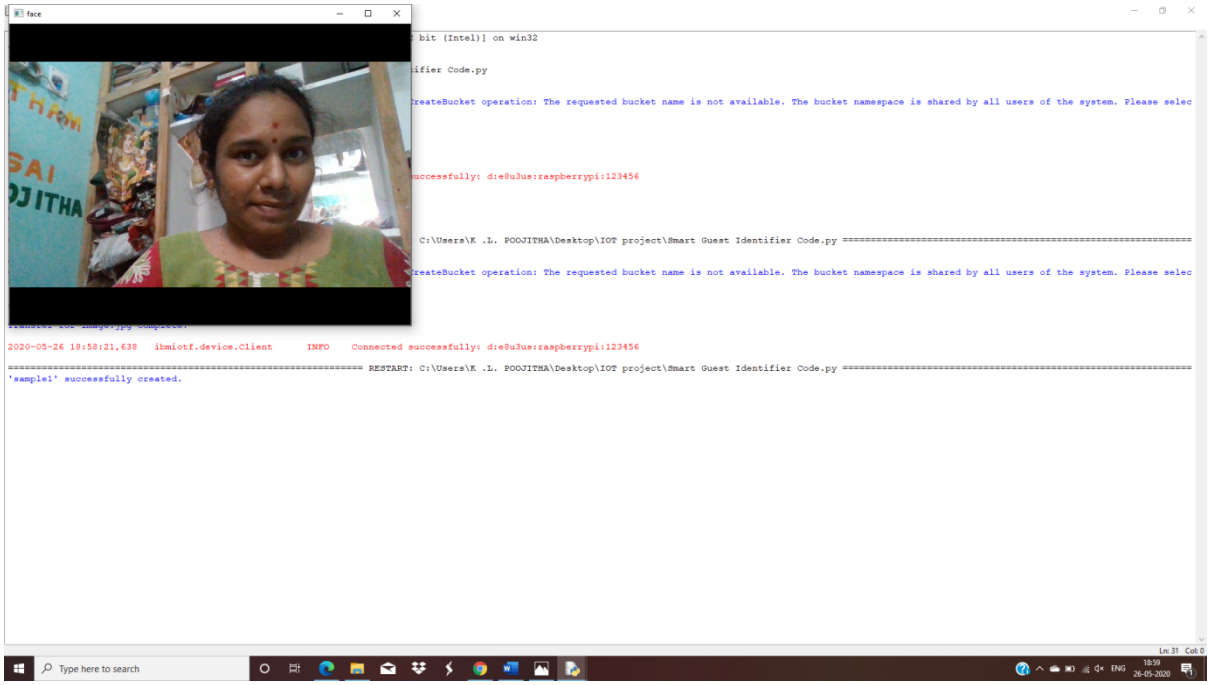
Node_red Flow



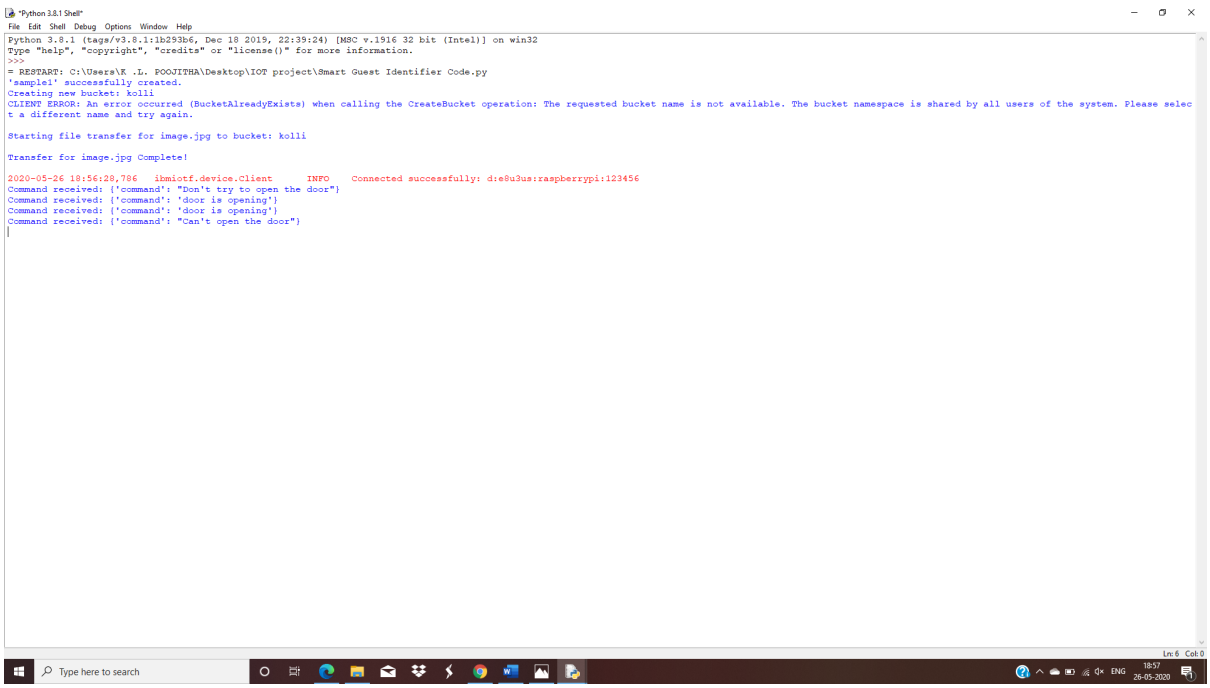
Status of the Device



Capturing the image



Output



7) Advantages:

a) System compatibility

b) Smart security

c) User-friendly and comprehensible attributes

Disadvantages:

The only disadvantage that we find in this project is ,we can not able to store the pictures that are captured before for the future recollection.

8) Applications:

a) Offices

b) In military secret rooms

c) Hospitals

d) Police stations

9) Conclusions:

Electronic visual display enabled by touchscreen technologies evolves as one of the universal multimedia output methods. By using iot platforms and mit we created this application

We can use Raspberrypi also it comes to hardware .

10) Future scope:

since remote screen sharing systems are also increasingly prevalent we propose a cross platform middle ware infrastructure which supports remote monitoring and control functionalities Based on remote streaming for networked intelligent devices such as smart phone ,computer and smart watch, etc. and home appliances such as smart refrigerator , smart air- conditioner and smart tv , etc.

11) Bibliography:

1.https://pypi.org/project/opencv-python/

wrapper package for opencv python bindings

2.https://thesmartbridge.com/documents/projects/IBMCloudObjectStorageandCloudtDB.pdf

code snippet for transferrig the images to the ibm cloud after recogintion to store

3.https://flows.nodered.org/node/node-red-dashboard

UI nodes installation

4.https://thesmartbridge.com/documents/projects/SmartHomeAutomationusingIBMCloud.pdf

configure the appication to send the button status to cloud

5.https://node-red-mqrth.eu-gb.mybluemix.net/ui

nodered dashboard for button UI

6.https://youtu.be/lMC55qgfYy4

7. https://partheniumproject.com

8. smartbridgeteachable.com

Appendix:(Sourse code)

```
import time
import sys
import ibmiotf.application
import ibmiotf.device
import random
```

```
import cv2
import numpy as np
import datetime

# ObjectStorage
import ibm_boto3
from ibm_botocore.client import Config, ClientError



# CloudantDB
from cloudant.client import Cloudant
from cloudant.error import CloudantException
from cloudant.result import Result, ResultByKey
import requests

import json
from watson_developer_cloud import VisualRecognitionV3

face_classifier=cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
eye_classifier=cv2.CascadeClassifier("haarcascade_eye.xml")
```

```
# Provide CloudantDB credentials such as username,password and url


client = Cloudant("e1437d2d-1d2d-4c02-87a6-2c979d96197e-bluemix",
"03a21003379ae3c42a6bcdf53de6b73af12c4d9082f9e35f53287e9a7038d495",
url="https://e1437d2d-1d2d-4c02-87a6-2c979d96197e-
bluemix:03a21003379ae3c42a6bcdf53de6b73af12c4d9082f9e35f53287e9a7038d495@
e1437d2d-1d2d-4c02-87a6-2c979d96197e-
bluemix.cloudantnosqldb.appdomain.cloud")

client.connect()


# Provide your database name

database_name = "sample1"

my_database = client.create_database(database_name)


if my_database.exists():

  print(f"'{database_name}' successfully created.")



img=cv2.VideoCapture(0)


while True:

    ret,frame=img.read()


    global imgname


    cv2.imshow("face",frame)

    imgname=datetime.datetime.now().strftime("%y-%m-%d-%H-%M-%S")
```

```
        cv2.imwrite(imgname+".jpg",frame)
        k=cv2.waitKey(1)
        #waitKey(1)- for every 1 millisecond new frame will be captured
        if k==ord('q'):
         #release the camera
              img.release()
         #destroy all windows
              cv2.destroyAllWindows()
              break
# Constants for IBM COS values
COS_ENDPOINT = "https://s3.jp-tok.cloud-object-storage.appdomain.cloud" # Current list avaiable at https://control.cloud-object-storage.cloud.ibm.com/v2/endpoints
COS_API_KEY_ID = "5xOFhgtbOeSNOnAzeZIOdnvDtc-0myLHgaWa2k1Lz9Da" # eg "W00YiRnLW4a3fTjMB-odB-2ySfTrFBIQQWanc--P3byk"
COS_AUTH_ENDPOINT = "https://iam.cloud.ibm.com/identity/token"
COS_RESOURCE_CRN = "crn:v1:bluemix:public:cloud-object-storage:global:a/d03d8edd8b6049d2817db9754f28605d:0ad8004b-2343-455f-aef2-3b42d11da5f4::" # eg "crn:v1:bluemix:public:cloud-object-storage:global:a/3bf0d9003abfb5d29761c3e97696b71c:d6f04d83-6c4f-4a62-a165-696756d63903::"
# Create resource
cos = ibm_boto3.resource("s3",
    ibm_api_key_id=COS_API_KEY_ID,
    ibm_service_instance_id=COS_RESOURCE_CRN,
    ibm_auth_endpoint=COS_AUTH_ENDPOINT,
    config=Config(signature_version="oauth"),
```

```
    endpoint_url=COS_ENDPOINT
)
def create_bucket(bucket_name):
    print("Creating new bucket: {0}".format(bucket_name))
    try:
        cos.Bucket(bucket_name).create(
            CreateBucketConfiguration={
                "LocationConstraint":"jp-tok-standard"
            }
        )
        print("Bucket: {0} created!".format(bucket_name))
    except ClientError as be:
        print("CLIENT ERROR: {0}\n".format(be))
    except Exception as e:
        print("Unable to create bucket: {0}".format(e))

create_bucket("kolli")

def multi_part_upload(bucket_name, item_name, file_path):
        try:
            print("Starting file transfer for {0} to bucket: {1}\n".format(item_name,
bucket_name))

            # set 5 MB chunks
            part_size = 1024 * 1024 * 5
```

```
        # set threadhold to 15 MB
        file_threshold = 1024 * 1024 * 15

        # set the transfer threshold and chunk size
        transfer_config = ibm_boto3.s3.transfer.TransferConfig(
            multipart_threshold=file_threshold,
            multipart_chunksize=part_size
        )
        # the upload_fileobj method will automatically execute a multi-part upload
        # in 5 MB chunks for all files over 15 MB
        with open(file_path, "rb") as file_data:
            cos.Object(bucket_name, item_name).upload_fileobj(
                Fileobj=file_data,
                Config=transfer_config
            )

        print("Transfer for {0} Complete!\n".format(item_name))
    except ClientError as be:
        print("CLIENT ERROR: {0}\n".format(be))
    except Exception as e:
        print("Unable to complete multi-part upload: {0}".format(e))
multi_part_upload("kolli", "image.jpg", imgname+".jpg")
json_document={"link":COS_ENDPOINT+"/"+"kolli"+"/"+"image.jpg"}
new_document = my_database.create_document(json_document)

# Provide your IBM Watson Device Credentials
```

```
organization = "e8u3us"
deviceType = "raspberrypi"
deviceId = "123456"
authMethod = "token"
authToken = "12345678"



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()


# Connect and send a datapoint "hello" with value "world" into the cloud as an event
```

```
of type "greeting" 10 times
deviceCli.connect()

while True:

        # hum=random.randint(10, 40)
        # print(hum)
        # temp =random.randint(30, 80)
        # Send Temperature & Humidity to IBM Watson
        # data = { 'Temperature' : temp, 'Humidity': hum }
        # print (data)
        def myOnPublishCallback():
          # print ("Published Temperature = %s C" % temp, "Humidity = %s %%" %
hum, "to IBM Watson")

          success = deviceCli.publishEvent("Weather", "json", data, qos=0,
on_publish=myOnPublishCallback)
          if not success:
             print("Not connected to IoTF")
             time.sleep(2)

        deviceCli.commandCallback = myCommandCallback

# Disconnect the device and application from the cloud
deviceCli.disconnect()
```